On the Presence Information of Nodes in Mobile Ad-hoc Networks

Inaugural-Dissertation

zur
Erlangung des Doktorgrades der
Mathematisch-Naturwissenschaftlichen Fakultät
der Heinrich-Heine-Universität Düsseldorf

vorgelegt von

Tran Thi Minh Chau

aus Vietnam

June 2009

Aus dem Institut für Informatik
der Heinrich-Heine-Universität Düsseldorf

Gedruckt mit der Genehmigung der
Mathematisch-Naturwissenschaftlichen Fakultät der
Heinrich-Heine-Universität Düsseldorf

Referent:	Prof. Dr. Martin Mauve Heinrich-Heine-Universität Düsseldorf
Koreferent:	Prof. Dr. Stefan Conrad Heinrich-Heine-Universität Düsseldorf

Tag der mündlichen Prüfung: 17.07.2009

Abstract

While mobility in the sense of node movement has been an intensively studied aspect of mobile ad-hoc networks (MANETs), another aspect of mobility has not yet been subjected to systematic research: nodes may not only move around but also enter and leave the network. In fact, many proposed protocols for MANETs exhibit worst-case behavior when an intended communication partner is currently not present. This thesis addresses the problem of detecting the presence of nodes in MANETs and provides lightweight solutions to the problem.

As our major contribution, we introduce soft-state presence detection. It uses a Bloom filter-based beaconing mechanism to aggregate and distribute information about the presence of network nodes. This soft-state approach decays information over time, enabling the removal of no-longer-present nodes from aggregates, which is not possible with standard Bloom filter operations. Privacy issues and design alternatives of this approach are also discussed.

While being a lightweight presence detection, the soft-state approach provides an additional feature of estimating hop-distance to nodes at the cost of extra bandwidth usage. We introduce phase-based presence detection that improves upon the soft-state approach by significantly reducing the overhead required to decide whether a node is present. This approach makes use of standard Bloom filters in combination with a loose synchronization mechanism that solves the problem of information removal.

Finally, we take one step further to investigate the presence detection problem in the stricter sense of a node being present, i.e. whether communication with the node is possible, and adapt our presence detection approaches to solve it. We also developed `xWhoisthere`, an instant messenger that monitors presence status, including reachability, of friend nodes. The software can be deployed as an independent group-aware application supporting collaboration among members or simply as an instant messenger that requires no Internet connection.

Zusammenfassung

Während die Mobilität im Sinne von Knotenbewegung einen intensiv studierten Aspekt mobiler Ad-hoc-Netze (MANET) darstellt, ist eine andere Art der Mobilität bisher vernachlässigt worden: Knoten können sich nicht nur innerhalb des Netzes bewegen, sondern auch neu in das Netz kommen oder das Netz verlassen. In der Tat zeigen die vorgeschlagenen Protokolle für MANETs meist Worst-Case-Verhalten, wenn ein Kommunikationspartner zurzeit nicht anwesend ist. Aus diesem Grund beschäftigt sich diese Arbeit mit der Frage, wie auf leichtgewichtige Weise die Anwesenheit von Knoten in MANETs berprüft werden kann.

Der zentrale Beitrag der Arbeit besteht in der Entwicklung eines Ansatzes zur Präsenzerkennung. Als Grundlage dienen dabei Bloom-Filter, um Informationen über die Anwesenheit von Netzknoten effizient zu sammeln und zu verteilen. Gewöhnliche Bloom-Filter unterstützen jedoch nicht das Entfernen von Elementen aus einer gespeicherten Menge. Um dieses Problem zu lösen, wurden Bloom-Filter durch einen Soft-State-Mechanismus erweitert. Weiterhin wurde untersucht, wie die Anwesenheitsinformationen vor Missbrauch geschtzt werden können.

Der Soft-state-Ansatz liefert zusätzlich Informationen ber die Distanz zu einem anwesenden Knoten. In einigen Anwendungen ist dies eine nützliche Information. Allerdings stellt sich die Frage, ob ein noch effizienterer Mechanismus zur Präsenzerkennung möglich ist, welcher diese Information nicht bereitstellt. Der zweite Beitrag der Arbeit besteht daher in der Einführung eines Algorithmus zur Phasen-basierten Präsenzerkennung, welcher in vielen Fällen die notwendige Datenrate des Soft-State-Ansatzes deutliche unterschreitet. Dieser Ansatz macht von gewöhnlichen Bloom-Filtern in Kombination mit einem losen Zeitsynchronisation Gebrauch, um Knoten die das System verlassen haben, aus den Präsenzinformationen zu entfernen.

Schließlich wird das Problem der Präsenzerkennung im strengen Sinn untersucht. Hier gilt ein Knoten nur dann als anwesend, wenn man eine bidirektionale Kommunikation mit ihm durchführen kann. Es wird eine Verfahren vorgeschlagen, welches die beiden oben erwähnten Ansätze so erweitert, dass nur Knoten als anwesend betrachtet werden, mit denen eine bidirektionale Kommunikation möglich ist.

Die Ergebnisse der Arbeit wurden analytisch untersucht, mit Simulationen bewertet und schließlich in einer konkreten Anwendung implementiert. So enstand der Instant Messanger `xWhoisthere`, der den Präsenzstatus und die Erreichbarkeit von Freunden berwacht. Die Software wurde für die Durchführung einer Reihe von Realwelt-Experimenten verwendet, welche die Ergebnisse der analytischen und simulativen Evaluation bestätigen.

Acknowledgments

Although only my name appears on the cover page, this dissertation was made possible with the contribution of many people. I am grateful to them all.

First of all, I would like to express my gratitude to my advisor, Prof. Martin Mauve, for his great inspiration, guidance, and support throughout my PhD study. His infectious enthusiasm has been a great driving force through my research years at the University of Düsseldorf. I have been amazingly fortunate to have an advisor like him and I hope that one day I would become as good an advisor to my students as Martin has been to me. I would also like to thank Prof. Stefan Conrad for being my co-advisor and for his support and constructive suggestions.

I am also thankful to Prof. Jörg Rothe and Prof. Phan Minh Dung, who was my advisor during my Master study in AIT, for those valuable discussions on problems related to cryptography.

This is a great opportunity to express my respect to Björn Scheuermann, Wolfgang Kieß, Christian Lochert, Jedrzej Rybicki, Michael Stini, Markus Koegel, and all the other colleagues at the Computer Networks and Communication Systems group at the University of Düsseldorf for their discussions and their most valuable feedback on my ideas and papers. My special thanks go to Björn for helping me through the painful process of writing papers and to Wolfgang for great advice on real-world experiments. Without their discussions, I would have had much harder times ironing out my fuzzy ideas. They both also helped considerably with their proof-reading of this dissertation. My experiments with PDAs would not have been possible without student helpers. Among them are Thomas Ogilvie and other programmers of the EXC software.

My thanks also go to Marga Potthoff and Sabine Freese, who guided me through the bureaucracy inside as well as outside the university, and to Christian Knieling for maintaining our wonderful networking environment and helping me with the software and hardware I needed to carry out my experiments.

I am grateful to the Deutscher Akademischer Austausch Dienst for the financial support for my study in Germany. I am deeply indepted to the people who work hard for the money and the opportunity that people like me are granted.

My sincere gratitude is also to German people, who have made my stay in Germany a wonderful time. When I go back home, I will take with me the love for this beautiful country and the admiration of this great nation.

This thesis is dedicated to my extended family and friends, who have always been the source of love and support. I cannot find the right words to thank my parents, my husband and child, who held me up with their endless love.

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

AODV	Ad-hoc On-demand Distance Vector routing
ARP	Address Resolution Protocol
CIDR	Cluster-based Inter-Domain Routing
DSDV	Destination-Sequenced Distance Vector
DSR	Dynamic Source Routing
EXC	EXperiment Control
GLS	Grid Location Service
HLS	Hierarchical Location Service
ICMP	Internet Control Message Protocol
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
MAC	Medium Access Control
MANET	Mobile Ad-hoc NETwork
NHDP	NeighborHood Discovery Protocol
OLSR	Optimized Link State Routing protocol
PDA	Personal Digital Assistant
SANET	Sensor and Actuator NETwork
TDBF	Time-Decaying Bloom Filter
TTL	Time-To-Live

A stupid man's report of what a clever man says is never accurate,
because he unconsciously translates what he hears
into something that he can understand.
Bertrand Russell

Chapter 1

Introduction

Node mobility is a key challenge in mobile ad-hoc networks. As a consequence, the impact of a dynamic network topology on medium access, network and transport functionality has been studied extensively. However, there is another aspect of mobility besides having to deal with a dynamic network topology. Mobility also implies that nodes may enter and leave the network at any time. This can happen either physically by entering or leaving the network area, or logically by switching the networked device on or off. In contrast to supporting dynamic topologies, the impact of varying node presence has not yet been systematically studied, although it can affect the performance of a network significantly.

One example—by far not the only one, but a particularly good one—is reactive routing in MANETs with protocols like DSR [JM96] or AODV [PR99]. These have been designed and evaluated under the premise that all communication partners to which a route is to be established are actually present in the network. Routes are found by flooding route requests. Flooding is repeated if no answer arrives within some time interval. Thus, if the intended communication partner is not present, route discovery causes a maximum amount of unnecessary network traffic. The problem could be avoided if there was a way to tell whether some potential communication partner is currently present or not. While proactive routing protocols would certainly be able to provide this service, they additionally spend resources to track all other mobility-induced topology changes. This overhead was the key reason to develop reactive routing protocols in the first place. We therefore argue that a presence detection scheme should be lightweight: it should track only the presence of nodes and not the state of all links.

Many other protocols and applications for mobile ad-hoc networks could likewise profit from presence detection. Further examples are mechanisms for service discovery, where one wants to find a provider of a certain service, or location services used for geographic routing. Both could benefit from a way to check whether the subject of the query is present at all, before actually attempting to locate it.

This thesis addresses the problem of detecting presence of nodes in MANETs and provides lightweight solutions to the problem. Our main contributions are as follows:

1. We point out the problem of lightweight presence detection as an important open research problem in the context of MANETs.
2. We propose a solution to the problem using soft-state Bloom filters, assess the performance of this solution in various regards: analytically, by means of simulation and test-bed.
3. We devise a phase-based solution which saves a significant amount of network bandwidth using a more space-efficient method to remove old information.
4. We extend the problem of presence detection to reachability detection, adapt the proposed algorithms to solve the new problem, and develop an instant messenger application to demonstrate our solution.

Our core contribution is presented in Chapter 3, where we propose a soft-state lightweight presence detection service for mobile ad-hoc networks. It enables nodes to check whether other nodes are present within a given hop-count radius. This soft-state approach is based on a space-efficient approximate set membership data structure called Bloom filter [Blo70]. Essentially, this provides a lossy compression of presence information in order to minimize communication overhead. In our approach, the nodes periodically send beacon messages announcing the compressed presence information that they have gathered so far. They integrate the information received from neighbors into their own knowledge base and include it in their next announcement. Stale presence information about nodes that have left will automatically vanish.

Next, we observe that the soft-state approach not only solves the problem of lightweight presence detection, but also has a side effect that allows nodes to estimate their distance to other nodes at the cost of more bandwidth required for the exchange of soft-state Bloom filters. By using standard Bloom filters combined with a loose mechanism of phase synchronization to deal with the removal of information, we significantly reduce bandwidth cost to trade off with the nodes' ability to estimate distances. The result is the phase-based presence detection proposed and assessed in Chapter 4, an even more lightweight solution to the problem of presence detection.

While the problem of presence detection concerns only the physical presence of nodes in the network, there are applications that are interested in whether communication to a node is possible. Communication ability to a node does not always follow the fact that a node is present in the network. Hence, in Chapter 5, we extend the problem of presence detection and address the question of reachability, i.e. whether communication to a node is possible. The algorithm we propose has been implemented in a real-world application, `xWhoisthere`, which is a group-aware application that demonstrates the effectiveness of our approach to detecting node reachability.

In short, the thesis is structured as follows. Chapter 2 gives an overview of related work. Chapter 3 presents our soft state approach to presence detection. The more lightweight phase-based approach is described in Chapter 4. Thereafter, Chapter 5 presents our solution to reachability detection. Finally, Chapter 6 follows with concluding remarks.

Chapter 2

Related Work

Although detecting the presence of nodes is a largely unexplored field, not only in the context of MANETs, there are a number of research directions that deal more or less directly with the presence of nodes in wireless networks. This chapter surveys related work in the area of obtaining information about the set of nodes that are active in the network.

As already mentioned, routing protocols for mobile ad-hoc networks are able to determine the presence status of a node. However, reactive protocols like AODV [PR99] or DSR [JM96], which do not take the initiative for finding a route to a destination until required, induce very high, unnecessary network traffic in the case of a route discovery to a non-present destination. Thus, they might actually benefit from an additional presence detection service. In contrast, proactive routing protocols like OLSR [JMC$^+$01], a link-state based algorithm, can immediately determine the presence of a node. However, as stated in Chapter 1, this comes at the cost of tracking the complete network topology, too. Due to the continued maintenance of all routes by periodically flooding the network with updates of network topology, proactive routing protocols need to constantly exchange a great amount of information. Thus, they do not scale very well and are not suited for highly dynamic network environments. A presence detection system in combination with reactive routing is able to combine the benefits of both reactive and proactive approaches: there is no expensive search for non-present communication partners in reactive routing, while the high effort for maintaining many unused routes is also avoided. Since the presence information is far less dynamic than the topology of the network, the maintenance effort is significantly smaller.

An example of routing protocols that benefit from presence information is Cluster-based Inter-domain Routing (CIDR) [ZCG09] [1], a protocol that supports scalability and robustness to mobility in routing between dynamic clusters of network nodes. CIDR requires periodic communication between nodes within a network cluster to discover and advertise cluster membership. Though the authors also proposed to use Bloom filter in membership management and advertisement, they have not yet a specific mechanism to propagate membership advertisement apart from a suggestion to use DSDV-like protocols for the purpose. Once again, DSDV distance information contains more than just presence information and thus is more expensive in terms of bandwidth. Meanwhile, our lightweight approach to presence detection perfectly fits the task of membership discovering and managing that CIDR requires.

Close relatives to presence detection are location services for geographic routing, which map the address of a node to its geographic position. Examples are homezone-based systems [Sto99, GH99], the Grid location service (GLS) [MJK$^+$00], and the hierarchical location service (HLS) [KFWM04]. However, a location service provides significantly more information about a node than just telling whether it is present or not, namely its current position. This information is much more dynamic than pure presence information. Therefore, the cost to keep it up-to-date and to look up in such a service is much higher. Like reactive routing, most location services exhibit their worst-case behavior in terms of effort in the case of a request for information about a non-present node. Thus, location services might actually benefit from an additional presence detection service.

Some approaches to presence detection in wireless systems have been successful applications on their own. An example is the matchmaker Lovegety [Iwa98]. Users give the devices some information on what they have in mind, such as “talk” and “karaoke”, the Lovegety then alarms them when it detects a nearby user of the opposite sex who is a mutual match, so as to help the users involved either in finding or avoiding each other. A number of research projects deal with the exchange of presence information via single-hop wireless communication, with or without infrastructure. Hummingbird [HFW99], a custom-designed mobile device that alerts users when they are in the physical vicinity of each other, was designed to support group awareness and collaborations without depending on an infrastructure. Social Net [TMRL02] is a interest-matching application that broadcasts a user presence, detects those who are nearby, and infers interests shared by users based on patterns it records over time of physical proximity between people. Serendipity [EP05] is also a matching application. Running on mobile phone,

[1]The paper was published in 2009, two years after our first paper on the topic of presence detection ([TSM07])

it combines Bluetooth proximity detection with a central server containing user profiles to detect nearby users who share similarities and alert its user. However, none of these systems considers presence detection over multiple wireless hops. The same can be said about protocols that are able to discover one- or two-hop neighborhood. NHDP[CDD08] is an example of such a protocol.

Finally, [LW06] presents a design for an instant messaging system for sparse mobile ad-hoc networks called SPEED. Instead of the presence of nodes, the authors consider the dissemination of presence states of users, such as “available”, “busy”, or “do not disturb”, with the assumption that users’ devices stay in the network even if a user is “non-present”. SPEED distributes user presence information via periodical announcements and requests, both of which are flooded in the network. This design is well-suited for an instant messaging service. However, for our purposes a much more lightweight solution is necessary. Interestingly, like many of the previously discussed protocols, SPEED generates significant overhead in the case of users whose devices are not present in the network. Therefore, it might in fact profit from additional node presence detection in such cases.

Chapter 3

Soft-state Presence Detection

In this chapter, we propose a soft-state lightweight presence detection service for mobile ad-hoc networks. It enables nodes to check whether other nodes are present within a given hop-count radius. Our approach is based on a space-efficient approximate set membership data structure called Bloom filter [Blo70]. Essentially, this provides a lossy compression of presence information in order to minimize communication overhead. In our approach, the nodes periodically announce the compressed presence information that they have gathered so far in beacon messages. They integrate the information received from neighbors into their own knowledge base and include it in their next announcement. Stale presence information about nodes that have left will automatically vanish.

The cost of aggregating presence information is a small number of so-called false positives, where nodes are wrongly considered to be present. Note that this is not critical for typical applications of a presence detection service. When such a service is used, for instance, to avoid unnecessary route discoveries with reactive routing, the cost of a false positive is an unnecessary discovery attempt. This is exactly what happens without presence detection. Thus, in the rare error case, the system behaves like one without presence detection.

The soft state approach proposed in this chapter not only solves the problem of lightweight presence detection, but also has a side effect that allows nodes to estimate their distance to other nodes. As shown by Gupta and Kumar [GK00], the per-node capacity of an ad-hoc network decreases dramatically with the average distance between communication partners. Therefore, it is desirable that applications and algorithms for mobile ad-hoc networks are able to determine the distance between communication partners before attempting to exchange data.

In this chapter, Section 3.1 presents the algorithm we propose for lightweight presence detection, as well as optional enhancements to the algorithm. Section 3.2 evaluates our approach both analytically and by means of ns-2 simulations in terms of false positive rates, the accuracy of the distance estimates, and the speed of information propagation. In Section 3.3, we discuss an application of our approach for avoiding unnecessary overhead in reactive MANET routing and present corresponding simulation results. Section 3.4 presents our experiment results with a testbed, demonstrating that the algorithm works in real-world environment as well as in simulations. Finally, the privacy issue is addressed in Section 3.5 before the chapter concludes.

This chapter is based on two papers that have been published: [TSM07] and [TSM09a].

3.1 Scalable Presence Detection

We assume that each node has some static, unique ID. This could, for example, be a MAC- or statically assigned IP address.

The most naive approach for a presence detection mechanism would be simply distributing a list of all available node IDs throughout the network. This could be done, e. g., by transmitting the IDs of the nodes via beacon messages. The obvious problem of this approach is that the amount of data distributed to each node would increase linearly with the number of nodes in the network, which increases the network load accordingly. This is not appropriate for a lightweight service.

In order to avoid this problem, we propose to aggregate presence information. Generally, aggregation can be lossy or lossless. Lossless aggregation is efficient if there is some structure in the entries to be aggregated. For instance, IP routing table entries can be efficiently aggregated without losses, because the addresses are organized hierarchically. For nodes present in a MANET, such a structure is typically not given. Thus, we propose to use a lossy aggregation scheme.

When performing lossy aggregation of presence information, two types of errors may occur. Either a node may be reported as being present while it is not, or it may be reported as being absent while it is in fact present. The former situation is called a false positive, the latter a false negative. Given the application of presence detection in mobile ad-hoc networks, a false negative would “hide” an actually present target node, which is generally not acceptable. A low rate of false positives, on the other hand, is often quite tolerable: in the rare case of a false positive, the applications would simply behave as if there was no presence detection service active, e. g., a routing protocol would

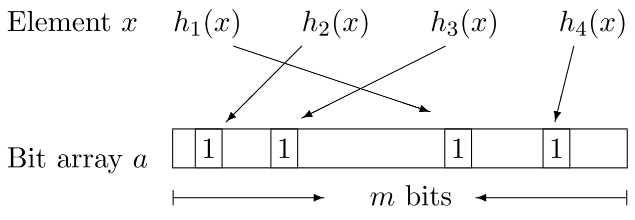


Figure 3.1: A Bloom filter.

attempt to set up a route to a node not present in the network. The actual absence of the node, and thus the occurrence of a false positive will become clear if communicating with the node fails. Therefore, for presence detection a scalable data structure that supports lossy aggregation of presence information without introducing false negatives is needed. Bloom filters are such a data structure.

3.1.1 The Bloom Filter

A Bloom filter [Blo70], is a data structure that represents a set $S = \{s_1, s_2, \ldots, s_n\}$ of n elements to support membership queries. It is described by an array a of m bits, which are initially set to 0. k independent hash functions $h_1, \ldots, h_k$ are used, each maps every possible item in the set to a uniformly distributed value in the range $\{1, \ldots, m\}$. Any hash function with good random distribution and outputs long enough for the Bloom filter size can be selected.

There are two basic operations that can be performed on a Bloom filter. New elements can be added, and the presence of an element can be queried. To add a new element s, which is a node ID in our specific case, the bits at positions $h_1(s), h_2(s), \ldots, h_k(s)$ in a are set to 1, as depicted in Figure 3.1. In order to determine the presence of some node x, the bits at these positions are checked. If any of these is 0, then it is certain that x is not in S. Otherwise, it can be assumed that x is in S with some remaining probability of a false positive that occurs when an element is actually not in the set, but all respective bit positions have been set to 1 by adding other elements.

The union of two Bloom filters is calculated by a bit-wise OR operation. Hence, to disseminate presence information, nodes may periodically send beacons containing a Bloom filter of the node IDs they know are present. Upon receiving such a beacon, they can merge the received information with what they already know. Since beacons contain aggregated data and are sent only to neighboring nodes, presence information is disseminated without flooding the network. For such a scheme to work, the hash functions need to be agreed upon beforehand and then used by all nodes in the network.

However, in a presence detection system the removal of no longer present nodes is also necessary. The standard Bloom filter has the drawback that there is no method to delete a value once it has been added. Since it is entirely possible that a given bit position has been set to 1 during the addition of more than one element, simply deleting all bit positions that refer to the element that should be removed is not an option.

3.1.2 Presence Detection

If we ignore the specifics of Bloom filters for a moment, there are two alternative approaches to remove the presence of a node from an aggregate. The first is to remove it explicitly when the node leaves the network. This is called a hard state approach. Alternatively, the presence information for each node can decay over time and has to be refreshed periodically in order to remain in the aggregate. Using this technique yields a so-called soft state approach.

There are extensions of Bloom filters (e. g., the counting Bloom filter in [FCAB00]) that enable explicit removal of items. Those could be used as a basis of a hard state approach. However, a hard state approach faces two major challenges when used for presence detection in MANETs. First, it must be guaranteed that the event of a node leaving the network actually triggers the removal. Second, the information on this event needs to be distributed in the network. Both problems are very hard to solve in the given decentralized environment. We therefore use a soft state approach. This requires an extension of Bloom filters that enables decay and refresh operations.

Soft state Bloom filters

In order to support these, we modify the Bloom filter used in the aggregates. Each entry now consists of l bits instead of one, where l is typically small, e. g., $l = 3$ or $l = 4$. Each of these l-bit words stores a counter. These counter values can be interpreted as the "age" of the respective Bloom filter entry. A node initializes all counters with the maximum value $2^l - 1$. This maximum value indicates that the position of the Bloom filter is not set, it is equivalent to setting a bit position to 0 in the standard Bloom filter. Periodically, just before a beacon is sent, a node applies the hash functions to its own

ID. It sets all counters at the resulting positions to 0. All other counters that are not already at the maximum value are incremented by one. Thus, each node continuously announces its own presence with the age of 0 and ages all other presence information by 1 each interval.

The aging of the information means that entries of leaving nodes will eventually die out. Hence, it provides the removal of nodes from the Bloom filter if they are no longer present. As already discussed, it is also desirable to be able to put a limit on the distance to a target node for it to be considered present. Applications can thereby take the expected communication effort into account, respecting the inherent capacity limits of wireless multihop networking. More formally, this can be stated as: node x is regarded as present by some other node u if and only if the length of the shortest hop-count path from u to x does not exceed some threshold T_u. Thus, x should be regarded as non-present by u if either x is not at all in the network or it is too far away. Since the entries in our scheme age with every hop over which the information is propagated, their value—as a side effect—also provides an indication of the distance to sought-after nodes.

Our Bloom filter modification bears certain similarities to some of the many Bloom filter variants that have been discussed in the literature. For example in the routing context, in [RK02] and [GJW$^+$06] schemes are discussed which use a set of Bloom filters: there is one Bloom filter for each hop distance, containing the information on all nodes at that distance. In [LYKH06], a scheme for service discovery is proposed that stores the minimum distance to a service using Bloom filters. The Time-Decaying Bloom Filter (TDBF) as introduced in [CXI$^+$05, CIXU05] is used to continuously analyze a data stream like, e. g., web page hits. Like in counting Bloom filters [FCAB00], but unlike in our approach, the *number of occurrences* of an item is stored in counters at each bit position. TDBF then reduces the contribution of less recent occurrences by periodically decaying all counters.

Aggregate structure

We define the structure of an aggregate as follows. It consists of m l-bit entries $a_1, a_2, \ldots, a_m$, each of which is interpreted as an integer in the range $0, \ldots, 2^l - 1$. a_i represents the age of the Bloom filter's "bit" i. Initially, all the a_i are set to $2^l - 1$.

l needs to be chosen large enough to account for the maximum of all nodes' distance thresholds. Thus, if N is the set of nodes,

$$l > \max_{u \in N} \lceil \log_2 T_u \rceil . \tag{3.1}$$

Each node u can select and change its T_u at will, but l is fixed and constant for the whole network.

Timeout and refresh

To accommodate the new aggregate structure, the operations on the Bloom filter are modified accordingly. The most central one is the timeout and refresh operation. Each node performs it periodically. It consists of three steps:

1. Increment each a_i by one, if it is not already at the limit of $2^l - 1$.
2. Refresh the information about the node's own presence, by setting $a_{h_j(ID)} = 0$ for all $j = 1, \ldots, k$, where ID is the ID of the local node.
3. Broadcast the updated aggregate to the neighbors.

This algorithm results in each position eventually reaching the maximal value when it is no longer refreshed by some node. Therefore, a no longer present node will vanish from the aggregate.

Merge operation

When a node receives a beacon message, the information is merged into the local aggregate. Instead of the bit-wise OR for standard Bloom filters, we use a position-wise minimum operation, i. e., we set each Bloom filter position to the minimum of local and received ages.

Query operation

In order to determine whether some other node x is present, a node u checks its local aggregate at positions $h_1(x), h_2(x), \ldots, h_k(x)$. Let

$$t := 1 + \max_{1 \leq i \leq k} a_{h_i(x)} . \tag{3.2}$$

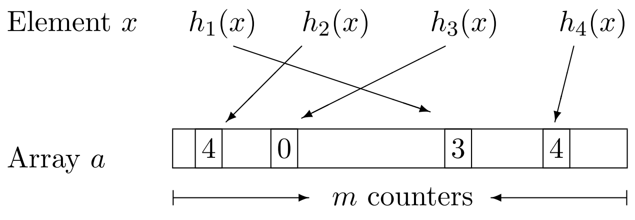


Figure 3.2: A soft state Bloom filter.

If $t = 2^l$, we conclude that x is not present. Otherwise, we say that it is *seen at a distance of t hops* from u, with some probability of a false positive. Depending on the choice of the threshold T_u, u considers x as present or not accordingly. By means of analysis and simulation, we will later show that this "seen distance" is in fact very close to the real minimum hop distance.

Figure 3.2 shows the entries corresponding to a sought-after node x. The age of the information about x is four, which means that x is seen at a distance of five hops. For instance, if $T_u = 10$ then x is said to be present. If $T_u = 3$ the node is said to be not present, as the distance at which x is seen exceeds the value of T_u.

3.1.3 Decoupling Information Decay and Beaconing Intervals

In the presented algorithm, the "age" of presence information increases with each beacon interval. In some situations, it might prove beneficial to increase the flexibility by decoupling the beaconing rate and the speed of information decay. This is possible, but it requires a small modification to the proposed algorithm.

In order to understand why this is the case, consider a situation where the aging of information is slowed down with respect to the beaconing in a naive way, by incrementing the counter values only every $b > 1$ beaconing intervals. Let A and B be two neighboring nodes; for simplicity, we neglect all other nodes and their beacons for the moment. Assume that some aggregate position a_i is currently at value 5 in both these nodes, but is no longer refreshed and should thus be subject to decay. A is first to age its counter values, and increments a_i to 6. But B's value of a_i is still 5, and hence A will very soon receive another beacon from B with $a_i = 5$, thus resetting A's aggregate. B will be next to age its own aggregate, but will in the very same way immediately be reset by A's next beacon. The reason for this misbehavior of the naive modification is that it may happen that a node repeats a counter value received from another node in its own broadcasted aggregate, without aging it at least once.

This problem can be overcome by introducing the following small extension to the soft state Bloom filter based presence detection protocol: whenever a beacon is sent *without* previously aging the counter values in the local aggregate, the sent-out *copy* of the aggregate must be decayed (and refreshed) prior to sending it to the neighbors. That is, if A's value of a_i is currently 5, and it sends a beacon without decaying its local copy of the aggregate, in its beacon it will broadcast the value $a_i = 6$. This ensures that a received counter value is never repeated un-decayed in own beacons, and hence avoids the described problem.

3.1.4 Aggregate Compression

Mitzenmacher showed that (standard) Bloom filters can be compressed to reduce their size [Mit02]. In theory, these compressed Bloom filters are a space-optimal representation of a set of items with a given false positive rate. On the one hand, this supports our choice of Bloom filters as the underlying data structure for our protocol. On the other hand, it points to an interesting additional option for further reducing the size of the presence detection beacons—or for achieving a lower false positive rate without increasing the size.

The counter values in soft state Bloom filters will typically not be equidistributed: not every counter value is equally likely. Therefore it is possible to save bandwidth by applying lossless compression to the beacons before they are transmitted. It turns out that arithmetic coding with an adaptive model as described by Witten et al. [WNC87] is very well-suited for this particular task.

The basic idea behind arithmetic coding is to divide the interval $[0, 1]$ first into subintervals according to the probability distribution of the possible values of the first character of the to-be-compressed input. For compressing soft state Bloom filters, the first character corresponds to the first Bloom filter position, i. e., to the first counter, which may take values in the range $0 \dots 2^l - 1$. The sub-interval corresponding to the counter's value is then chosen. It is again subdivided, matching the probability distribution for the second entry, and so on. The sender generates a binary representation of a value within the finally resulting (tiny) interval. The problem that arises when this is to be applied in practice is that the probabilities of the input values need to be known by both the encoder and the decoder. It has been shown that, given a suitable probability distribution model, arithmetic coding is able to compress down virtually to the entropy limit.

Witten et al. use arithmetic coding with a dynamic model. In their scheme, the input probability distribution is “learned” while processing the input. Basically, the algorithm keeps track of the input distribution in the data processed so far, continuously adjusting the model. This is very well-suited for the problem at hand, because all entries of a soft state Bloom filter do indeed exhibit the same input probability distribution. So, what has been learned during the compression of the first part of the filter is indeed a very good model for the remaining part. Another big benefit in particular on resource constained hardware is that both encoder and decoder can be implemented very efficiently with integer arithmetic. Both require only a constant, very small amount of memory, and encoding and decoding complexity is linear in the input length.

We propose to apply arithmetic coding as a kind of filter in our presence detection protocol: it comes into action when beacons are assembled (then, the transmitted soft state Bloom filter is passed through the arithmetic coding encoder and is thereby significantly reduced in size), and when beacons are received from other nodes (in this case, the decoder restores the originally transmitted soft state Bloom filter).

Since the compression with arithmetic coding is lossless, this technique does not have any direct impact on other aspects of the protocol itself or on which nodes are considered present or absent. Applying it comes at the cost of a slightly increased workload for the network nodes, for the compression and decompression of sent and received beacons; whether the achievable bandwidth savings outweigh this cost or not heavily depends on the particular devices and on the application scenario. However, as mentioned above, arithmetic coding can be implemented very efficiently even on resource-constrained devices, and it will thus often constitute an interesting extension.

3.2 Evaluation

In the previous section, we introduced an algorithm to disseminate presence information and to reliably remove no longer present nodes with a soft state approach. In this section, we assess the performance and suitability of the proposed scheme. In particular, we concentrate on five aspects: the reliability of the scheme in terms of the false positive rate, the accuracy of the seen distance, the effects of node movement, the speed of information propagation, and the effectiveness of compressing soft state Bloom filters with arithmetic coding.

3.2.1 False Positive Rate

For a practical application of presence detection, it is desirable to know the false positive rate: what is the probability of considering a non-present node to be present?

A false positive occurs when the bit positions corresponding to the sought-after node are all set by other added elements. In standard Bloom filters, the probability that a false positive occurs depends on three factors: the number of bit positions in the filter m, the number of hash functions k, and the number of elements n that are present in the set. The probability that a bit position is still zero after n elements with k bit positions each have been added is $(1-1/m)^{kn}$. Thus, the probability that all k bit positions of the sought-after node are one can be calculated as

$$\left(1-\left(1-\frac{1}{m}\right)^{kn}\right)^{k} \approx \left(1-e^{-kn/m}\right)^{k}. \tag{3.3}$$

So, if standard Bloom filters were used, the false positive rate would depend on the total number of nodes in the network. For our modified Bloom filter with decaying of information and counters at each filter position, however, the situation is slightly more complicated—and turns out to be significantly better.

Consider a node x that is regarded as present by u based on the local aggregate. As stated earlier, the maximum age at x's entries in the Bloom filter plus one is the distance in hops at which x is seen. Let this number like before be denoted by t. If x is actually not present *at this distance*, all of x's entries must have been set by other nodes. All these nodes must be at distance t hops or less. Let $n(t)$ denote the number of nodes within this maximum distance. Then, the probability that a non-present node is considered present at a distance of less than or equal to t is

$$P_{\text{fp}}(m,k,t,\rho) = \left(1-\left(1-\frac{1}{m}\right)^{kn(t)}\right)^{k} \approx \left(1-e^{-\frac{kn(t)}{m}}\right)^{k}. \tag{3.4}$$

Consequently, the probability of a false positive at distance $t \geq 1$ is

$$P_{\text{fp}}(m,k,t,\rho) - P_{\text{fp}}(m,k,t-1,\rho). \tag{3.5}$$

Being seen wrongly at distance t can either mean not being in the network at all, or being in the network but at a larger distance. These cases are hard to distinguish, since the Bloom filter positions that should contain information on x are all overridden by information on other nodes.

Note the implications of these effects: the probability of a false positive at distance t does not depend on the total number of nodes in the network, but only on the number of nodes within the t-hop neighborhood, and therefore on the local network density. This also means that the confidence in the presence of a node increases rapidly when it comes closer: the closer a node is seen, the higher is the chance that it is in fact there. This can be exploited by an application, by being more aggressive or spending more resources on contacting a node which is detected nearby and is therefore very likely to be actually present.

The results presented so far allow to estimate the false positive rate depending on the distance of some node and the number of t-hop neighbors. In practice, however, the latter will typically not be known, and it is hard to estimate. If, at run-time, a node intends to estimate the current false positive rate, however, this can also be done based on the node's local knowledge. Back to the initial point of our considerations, the probability of a false positive is the probability that all k filter entries corresponding to x are set by other nodes. Thus, given the status of the local aggregate, it is actually sufficient to know the probability that a randomly selected position is set.

Let $s(t)$ be the number of filter positions where the age stored in the counter value is less than t, i. e.,

$$s(t) := |\{i \mid 1 \leq i \leq m \text{ and } a_i < t\}| \,. \tag{3.6}$$

Then, the probability of a randomly selected position being set is $s(t)/m$, and consequently the probability of a false positive at distance t, which equals the probability of k randomly chosen positions being set, is simply given by

$$(s(t)/m)^k \,. \tag{3.7}$$

While estimating the same quantity, Formula 3.5 and Formula 3.7 are applicable to two different situations. The former calculates the expected false positive rates given a set of configuration parameters and thus can be used in tuning the approach's parameters to suit environment characteristics. On the other hand, using the latter, a node can rate the quality of specific positive answers from the presence detection service given the current status of its local aggregate. With such information, nodes can adapt the behavior of an application accordingly, depending on how likely a wrong answer is, and how big the potentially incurred overhead would be.

3.2.2 Accuracy of the Seen Distance

Let us now have a look at the relation between the seen distance and the true minimum hop distance between two nodes. That these two are related is intuitively clear: the soft state Bloom filter entries age by one over each hop, and hence the seen distance also increases by one every time the aggregate is handed over to the next node via a beacon. Due to the merging rule of the position-wise minimum, if information arrives over multiple paths, the lowest age (i. e., the lowest counter values, and hence the shortest path) will have precedence. However, there are effects that cause deviations between the seen distance at some point in time, and the true minimum hop distance.

In the proposed presence detection algorithm, each node periodically increments the age of the Bloom filter positions in its aggregate before sending a beacon. While waiting for the next beaconing interval to expire, updates are received from the neighboring nodes, potentially resetting the incremented positions to their previous values. Therefore, the Bloom filter entries go up and down periodically. Furthermore, if beacons are lost—either due to transmission errors or because of congestion or packet collisions—, this will lead to temporarily further increased counters: information gets through only from time to time. This influences the distance at which other nodes are seen; the seen distance is not always exactly the minimum hop count distance, but it will oscillate between the exact distance and slightly higher values.

The oscillations due to the alternating decay and refresh operations occur always, even in the case of an ideal medium without losses. Higher oscillation amplitudes are caused by packet losses.

Simulations using the ns-2 network simulator [ns2] were carried out to see to which extent the error rates effect on the fluctuation of reported pair distances. We placed 200 nodes randomly on a square area of 1500 meters side length. These parameters make sufficiently sure that the network is fully connected; they are similar to the settings used in other work on wireless multihop networks such as, for instance, [KK00] and [FZL$^+$03]. We used our presence detection algorithm with $m = 1024$ filter positions, $l = 4$ bits counter length, and a beaconing interval of three seconds. Ns-2's default transmission range of 250 meters was used. A randomized error model was used to make a varying fraction of beacon receptions, 0 %, 25 %, and 50 %, fail, in addition to medium-related losses like those caused by packet collisions. Figure 3.3 shows the influence of the true distance between two nodes and the beacon packet error rates on the oscillations. Each vertical bar stands for a shortest hop distance; it is an average over all node pairs with the respective distance. It shows for which fraction of time the distance has been reported correctly, as well as how long and how far it has been overestimated.

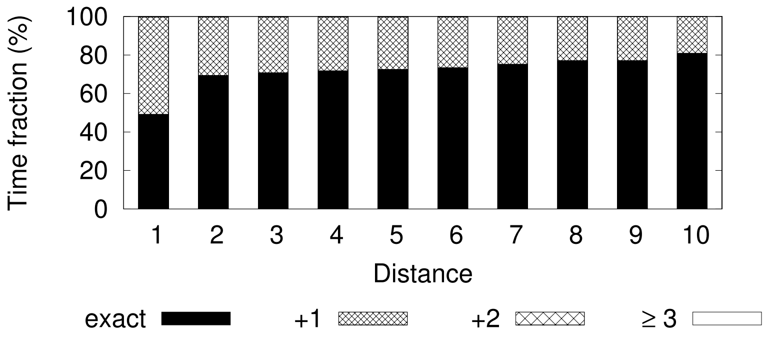
The results show that even high packet loss rates have surprisingly little effect on the oscillation of reported distances. For most of the time, either the reported distances between node pairs are correct, or they are overestimated by one, i. e., the decay operation has occurred and the Bloom filter positions have not yet been refreshed by a newly received aggregate. Distance estimates that are wrong by more than two hops are extremely rare: overestimates by three or more are barely visible in Figure 3.3 (c). The reason for this trait is that there often exist many alternative paths between a pair of nodes. They increase the chances for the information to get through and provide significant redundancy for the best path to be a fast one.

Whether the remaining oscillations are a problem essentially depends on the application's requirements. If an application requires higher stability of the seen distances, a simple way to achieve this is the following. In addition to the current local aggregate A, a node may store the aggregates from n previous broadcasting intervals, as they looked like just before incrementing the counters. When checking for the presence of node x, instead of using just the distance reported by the current aggregate, the minimum value from the current and all n previous aggregates is used. This technique yields very good results even and especially for small n, like $n = 2$ or $n = 3$. It comes at the cost of a slightly increased time until a no longer present node is considered absent, and until an actually increasing distance of a node is recognized. Both events will only be recognized after an additional delay of n beaconing intervals. This, however, will often be tolerable. The time until some newly arriving node is recognized as present does not change.

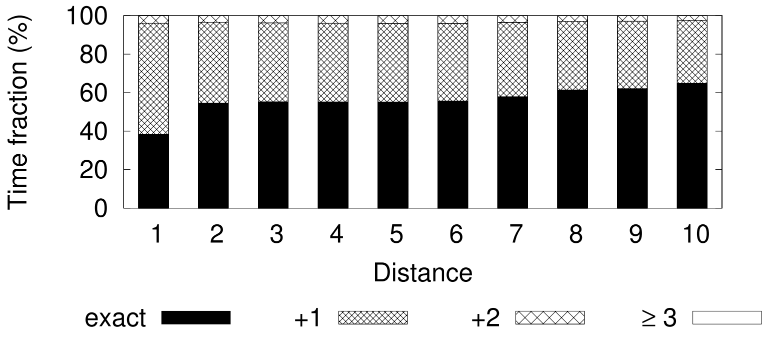
3.2.3 Node Movement

The movement of the nodes in a mobile ad-hoc network can influence the presence detection in two ways. First, the distances between nodes change over time, and there is some delay until these changes are correctly reflected in the presence detection aggregates. Furthermore, a node that moves to a different network area might "carry" presence information with it, making nodes from the area where it is coming from appear closer to the nodes in the new vicinity than they actually are.

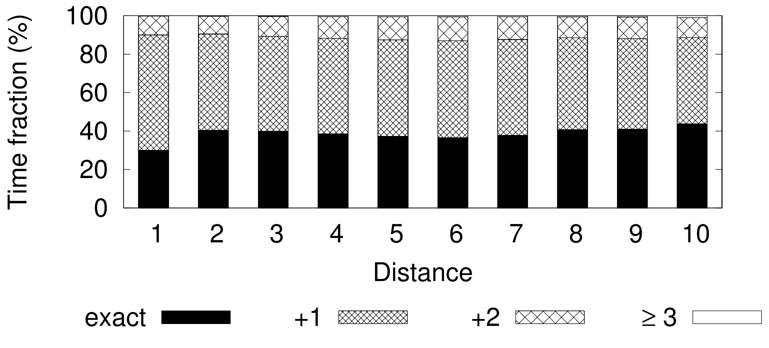
In order to assess the impact of these effects, we carried out another set of experiments with key parameters chosen as above, except that the nodes move according to the modified Random Waypoint mobility model without pause times and with different



(a) Beacon error rate 0.



(b) Beacon error rate 0.25.



(c) Beacon error rate 0.5.

Figure 3.3: Oscillations of distance estimates for varying beacon packet error rates.

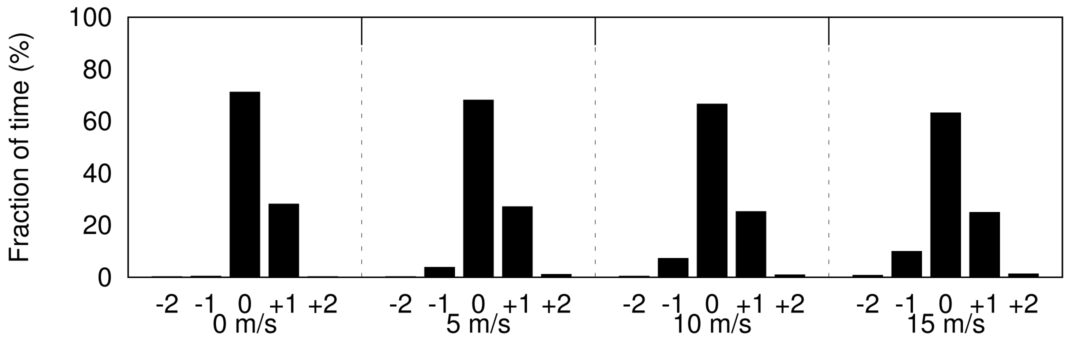


Figure 3.4: Deviations between seen distances and true shortest hop distances for different maximum node speeds.

maximum speeds. In order to overcome the well-known limitations [YLN03], we have used the modified version of the Random Waypoint model, initialized with the steady-state distribution. The results given in Figure 3.4 show that at all considered movement speeds, the shortest hop-distances are reported quite accurately. For this figure, we took 100 snapshots at random points in simulation time, calculated the true hop-distances for each node pair based on the current node positions, and compared them to the distances reported in the respective presence detection aggregates. With increasing mobility, there is an increase in underestimated distances, due to the reasons discussed above. However, even when node mobility is high, overestimated distances due to lost beacons and oscillations are much more common than underestimated ones. In the case of zero maximum speed, i. e., no mobility at all, there is a tiny number of cases in which the distance is underestimated; these are false positives that make a node appear closer than it is.

3.2.4 Speed of Information Propagation

Our scheme uses periodic beaconing to distribute presence information. Thus, a naive assumption on the propagation speed could be that the information travels one hop per beaconing interval, i. e., a node at a distance of d hops from a newly arrived node will notice its presence after d broadcasting intervals. Further investigation, however, quickly shows that this approximation is far too pessimistic.

The key reason why the dissemination of presence information is much faster in practice is that the beaconing cycles of the nodes are not synchronized. Assuming independent offsets of the beaconing times, the time between the reception of information by some node and the next beaconing cycle of that node is only *half* a beaconing interval on average. Therefore, on average, the propagation of the information along a chain of nodes with length d hops takes only $d/2$ broadcasting intervals. A dissemination delay of d broadcasting intervals is only the unlikely worst case.

In a topology that is more complex than a simple chain the results improve further: typically, there are many paths along which the information might propagate, and it is sufficient that it arrives along one of them. So, the time until a node x is recognized as present by some other node u is the minimum information propagation delay over all paths from x to u. Since the number of these paths quickly increases with increasing node density, it is reasonable to expect a propagation delay that is far below the worst case of d broadcasting intervals, and also less than the expected propagation delay along a single path of $d/2$ broadcasting intervals.

In order to illustrate this, we consider the situation of two nodes u and x that are two hops apart. We estimate the number of potential forwarders of the presence information between two such nodes. We assume a given node density $\rho > 0$ in the network area around the two nodes, and a circular one-hop neighborhood area of radius $r > 0$. Here, we sketch how an estimate of the delay of information propagation over two hops can be obtained. More details on the calculations can be found in the appendix.

Note that the distance d between two nodes that are two hops apart must be within $]r, 2r]$. It can be shown that the probability of having n nodes in the intersection of the radio ranges of two such nodes is

$$P(n \text{ nodes}) = \int_{r}^{2r} \frac{2\delta(\rho A(\delta))^n e^{-\rho A(\delta)}}{3r^2 n!} \, d\delta. \tag{3.8}$$

This can be used to calculate the probability of having n potential forwarders between two nodes that actually are two-hop neighbors (i. e., for which there exists at least one node that can directly communicate with both).

The expected time until a broadcast interval expires at the first out of n potential forwarders is $B/(n + 1)$, where B is the broadcasting interval length. Combining the results mentioned above (calculation details are given in Appendix A) yields an expected delay for the second hop's delay for a random pair of nodes u, x at two-hop distance

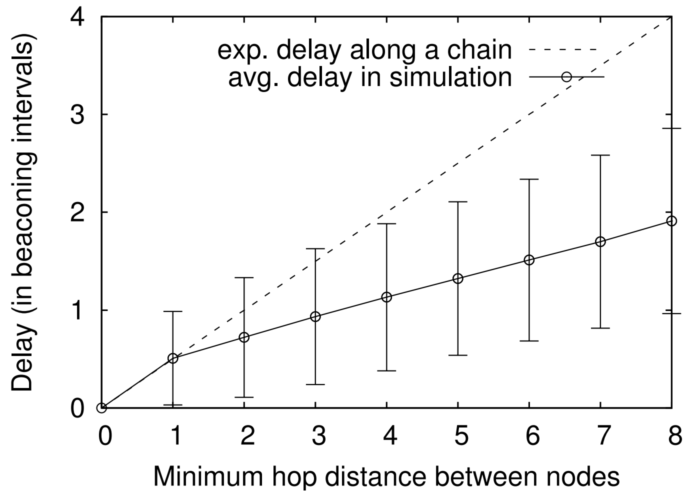


Figure 3.5: Average delay before a node is recognized as present.

of

$$\sum_{n=1}^{\infty} \frac{B \cdot \rho^n}{(n+1)!} \cdot \frac{\int_r^{2r} \delta (A(\delta))^n e^{-\rho A(\delta)}\, d\delta}{\frac{3}{2} r^2 - \int_r^{2r} \delta e^{-\rho A(\delta)}\, d\delta}. \tag{3.9}$$

In order to verify the predicted information dissemination speed, a simulation was carried out using the ns-2 network simulator [ns2]. We placed 200 nodes randomly on a square area of 1500 meters side length, and used our presence detection algorithm with $m = 1024$ filter positions, $l = 4$ bits counter length, and a beaconing interval of three seconds. The simulation uses IEEE 802.11 at 1 MBit/s. All local aggregates were initialized empty. Then, for each pair of nodes, the delay was measured until one node recognized the other one as present. Figure 3.5 shows the results of these simulations. The x-axis denotes the distance between the nodes, i. e., the minimum hop count. The y-axis then shows the average time until the presence information arrives, with 95-percentile error bars. The dashed line shows the expected dissemination speed along a single path.

It is evident that, as predicted by the theoretical arguments above, the dissemination speed is surprisingly high. For example, presence information travels over a distance of eight hops in less than two broadcasting intervals on average. This is because a high number of alternative paths are available, which increases the probability that for one of these paths the offsets of the nodes' periodic broadcasting are beneficially aligned, allowing for a quick information forwarding. For the first hop, there is only one single node which is able to forward the information, the source itself. At this point, the simulation results for the one-hop delay therefore match the delay predicted for a chain, $B/2$, exactly.

Evaluating (3.9) for the parameters of our simulation, i. e., for $\rho = 200/1500^2$ and $r = 250$, yields an expected delay for the second hop of 0.29 broadcasting intervals. In the simulation, the delay is even lower, around 0.23 broadcasting intervals. This is because it frequently happens that for a two-hop neighbor, an alternative path which is longer than two hops is even faster than all available two-hop paths. Such paths are not covered by (3.9).

So far, we have discussed the time until a node is considered present by other nodes after its arrival. The other interesting parameter is the time until it is no longer considered present after it has left. This, however, is straightforward: if the counters at the respective Bloom filter positions are no longer refreshed, they decay. Thus, after node x has left, it will be considered present by some other node u until the counters exceed u's threshold T_u, which can be expected to happen after $T_u - t + 1$ more beaconing cycles, if x is currently seen at distance t. During that time, x will be seen at an increasing distance.

We conducted another set of simulations, with similar parameters as above, in order to underline our results. We chose a distance threshold of 10. In these simulations, nodes enter and leave the network. Figure 3.6 shows the time until a node is considered present by all other nodes within 10-hop distance after it has arrived, and the time until it is recognized by all as no longer present when it leaves. It also shows the time until all other nodes detect that a node has left if the beaconing interval and the decaying speed are decoupled, as discussed in Section 3.1.3. Specifically, the decaying is slowed down by a factor of two in these simulations, in relation to the beaconing interval. The times are shown for different beaconing intervals with 99-percentile error bars. It can be seen that, as expected, both delays increase linearly with the beaconing interval length, and the delay until a no longer present node vanishes from the aggregate is generally longer than the delay until a newly arriving node is seen by the other nodes. As expected, the version with decoupled decaying exhibits a correspondingly longer delay until leaving nodes vanish from the aggregates: when aggregates are decayed only every other interval, the delay doubles.

The main lesson which one can learn from these results is that the propagation speed is significantly higher than what one would intuitively expect given a certain beaconing frequency. It is worth noting that these results hold for any information dissemination scheme that uses periodic broadcasting.

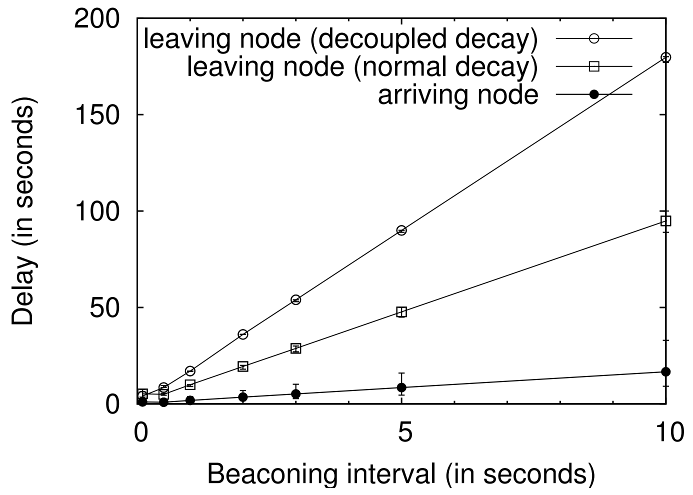


Figure 3.6: Time until information is received by all nodes.

3.2.5 Effectiveness of Arithmetic Coding Compression

In Section 3.1.4 we have introduced a mechanism to reduce the size of the beacons by piping the transmitted and received soft state Bloom filters through an arithmetic coding encoder or decoder, respectively. Now, we assess how effective this mechanism turns out to be in practice. For this purpose, we have implemented arithmetic coding based compression and applied it to the aggregates occurring in the presence detection service in different simulated scenarios.

We found that the size of the beacons can—depending on the density of the network—typically be reduced by 30–80 %, as shown in Figure 3.7. The figure shows—for a varying node density—the size of compressed 2048-entry filters, compressed 1024-entry filters, and uncompressed 1024-entry filters (represented by the dotted line). It can be seen in Figure 3.7 that compressed soft state Bloom filters with 2048 entries could under most circumstances easily replace uncompressed 1024-entry soft state Bloom filters without requiring more bandwidth. For practical applications, this implies that, with compression, we could reduce false positive rate by using Bloom filters of larger size at no additional bandwidth cost.

3.2.6 Choosing the Parameters

In the discussed algorithm, there are quite a number of parameters: the Bloom filter length m, the number of hash functions k, the distance thresholds T_u to be used, the

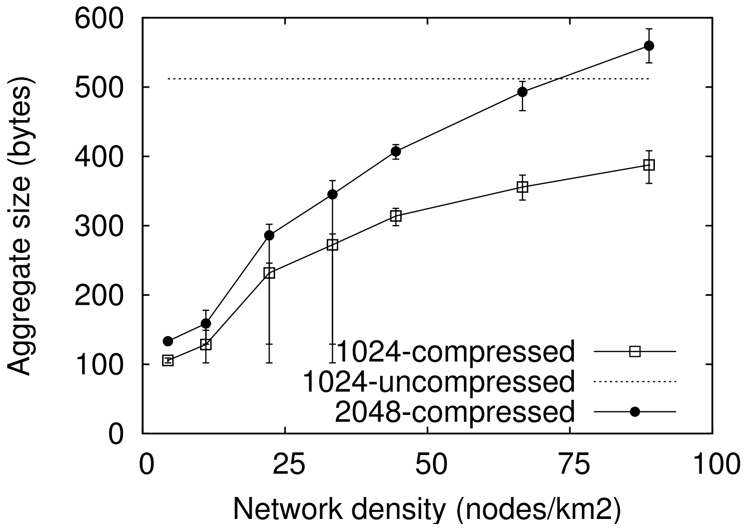


Figure 3.7: Average beacon size when compressed.

number of bits per Bloom filter position l, and the beaconing interval B. The best-suited values for these parameters depend on the application's requirements. Typically, one will find a tradeoff between bandwidth usage for beaconing, the delay of presence and disappearance detection, the false positive rate, and the achievable hop distance thresholds T_u. We consider it a very valuable property of our approach that it is possible to adjust this tradeoff in a very wide range, and thereby to tailor it for many specific application scenarios.

While it is relatively straightforward how one can find appropriate values for T_u, l, and B, the parameters m and k, which are directly related to the Bloom filter, deserve a little more attention. Actually, the optimal combination of m and k depends on the distance that is of main interest to the application: for given m, the optimal k for a minimum false positive rate is not the same for each distance. The analytical results regarding the false positives provide some hints. For a given Bloom filter length m and node count $n(t)$ within the considered t-hop radius, the false positive rate according to (3.4) is minimized for

$$k = \frac{m \cdot \ln 2}{n(t)}. \tag{3.10}$$

In practice, k must of course be chosen to be an integer.

Before we focus on a specific application for presence detection in the next section, let us now consider a concrete example network to illustrate what our results presented above actually mean in practice. Consider a mobile ad-hoc network using a 1 MBit/s channel (which is—considering today's wireless hardware—actually quite limited and therefore particularly challenging for a proactive, beacon-based protocol). We allow each node to spend 0.2 % of this bandwidth for presence detection beacons. Say that there is a total of 200 nodes in the network, which has a diameter of approximately ten hops. Let us furthermore assume that, for the considered application and network, a beaconing frequency of one beacon every three seconds suffices. So, the presence detection beacons may have a size of up to $3\,\mathrm{s}\cdot 2\,\mathrm{KBit/s} = 768$ bytes. We want to cover the whole network with the presence detection, so we may set $l = 4$. Therefore, we can use $m = 1400$ and still have plenty of space left for headers.

If we decide to optimize the false positive rate for longer distances, i. e., for the whole network, (3.10) suggests we use $k = 5$ hash functions. In this configuration, the expected false positive rate for a node at maximum search radius is 3.47 %. This will be fine for many applications. By spending slightly more bandwidth, 0.25 % instead of 0.2 %, and using $m = 1800$ and $k = 6$, the expected false positive rate is reduced to 1.33 %.

As we have pointed out in Section 3.2.1, the false positive rate also quickly decreases with a smaller search radius: if, for example, within some smaller radius there are only 100 nodes, there will only be 0.24 % false positives in the 0.2 % bandwith usage case, and 0.05 % when allowing to use 0.25 % of the bandwidth.

3.3 An Example Application

After having discussed the general features and applicability of presence detection in mobile ad-hoc networks, we now focus on one possible application: reactive routing. Finding a route to some destination node when a reactive routing protocol is used can be an expensive process, as it typically requires flooding a route request in the network. If the destination device is not present or too far away from the source device to be reached within the TTL limitation of the route request packet, much bandwidth is wasted. Furthermore, a route request is typically repeated if no answer arrives before some timeout expires, thus reactive routing exhibits worst-case behavior in the case of a non-present node.

In this section, we will analyze the presence detection service when it is used in conjunction with AODV routing [PR99]. We query the presence detection service at the source node before starting a route request for a new connection. If the destination node is considered present, the connection will be initiated as usual. Otherwise, the route request is delayed until the presence detection service indicates that the destination node is present.

We want to stress that this "model application" provides a hint on the possible effects of presence detection for a real application. It therefore complements the application-independent evaluation results from the previous section. The results shown here may not be arbitrarily generalized: the degree to which other applications benefit from presence detection depends on their cost associated with not having information on the presence of nodes and whether this outweighs the cost of the presence detection service.

The simulation study was conducted using ns-2 [ns2], with setups similar to those used before: 200 nodes in an area of 1500 by 1500 meters. IEEE 802.11 at 1 MBit/s network bandwidth is used; note that a low communication bandwidth is the worst case for a protocol that causes a constant beaconing load. The communication radius is 250 meters, with a 550 meter carrier sense range. Like above, the nodes move according to the modified Random Waypoint mobility model [YLN03]. The random speeds are in the range from 1 to 10 meters per second and the pause time is 20 seconds. All connections last 100 seconds and start at some random time between 10 and 190 simulation seconds. During a connection, the source node sends CBR traffic with four data packets per second, each with a payload size of 512 bytes. The results are averages over 25 scenarios, each with different traffic and movement patterns. Here, compared to the example given in the previous section, we can tolerate even some more false positives, because the negative impact is at most an unnecessary route discovery—ten percent seems tolerable. We trade the additional false positives off for a reduced beacon size and use $m = 1024$, $k = 4$, and a beaconing interval of three seconds. All plots in this section show 95 % confidence intervals.

3.3.1 Worst Case Performance

The worst case for the presence detection service is a situation where all network nodes are permanently present. In that case, the network will not profit from presence detection services, while they still consume bandwidth. Figure 3.8 shows the network performance, in terms of the packet delivery ratio, for an increasing number of connections. The negative impact of the presence detection is quite low, and sometimes there is actually a slightly better performance with presence detection.

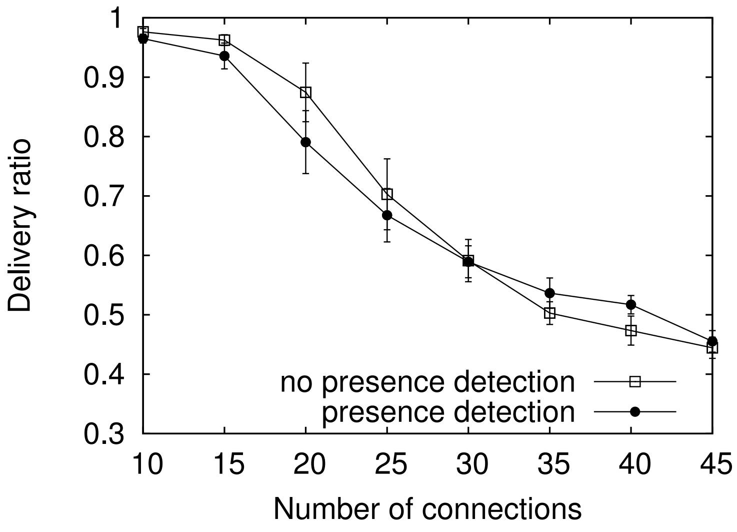


Figure 3.8: Worst case network performance.

The reason for these unexpected performance benefit with presence detection is a little subtle. When the network is so congested around some node that it cannot send its data packets, the node will not send any presence announcement until the situation improves. This is an acceptable behavior since there is no use for a node in a congested area to announce its presence: the node is not able to receive any more data packets anyway. When congested nodes are temporarily considered non-present, connections attempts to those nodes are delayed until they are again able to send beacons, i. e., they are effectively back in the network. Thus, employing presence detection has accidentally introduced some form of congestion control. We do not consider this to be a true advantage of presence detection, but it is certainly an interesting observation which may be exploited in future work.

3.3.2 Introducing Absent Nodes

Now we keep the number of “working” connections to present, available nodes fixed at 25. Figure 3.9 shows the effects on the network performance, again with 95% confidence intervals, when the number of additional connection attempts to non-present nodes increases. It can be seen that, without presence detection, more connection attempts to non-present nodes severely deteriorate the network performance. The performance with presence detection, on the other hand, does not show significant negative effects, no matter how many connections to non-present nodes are attempted.

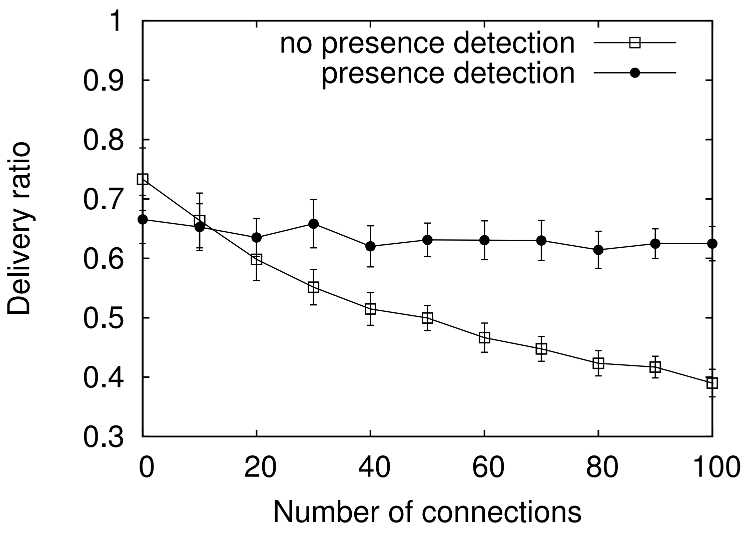


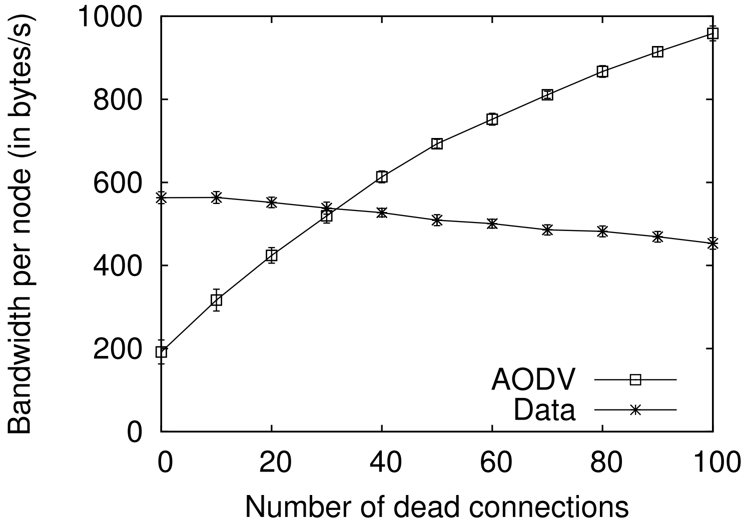
Figure 3.9: Packet delivery ratio as the number of connection attempts to non-present nodes increases.

Figure 3.10 compares the average bandwidth spent by a node in the network with and without presence detection, broken down to bandwidth used for routing packets, data packets, and presence detection beacons. The beaconing load in the case with presence detection is constant. It is obvious that, without presence detection, the bandwidth used for routing packets quickly increases, while the available bandwidth for application data decreases. Only a small portion of the bandwidth is actually effectively used.

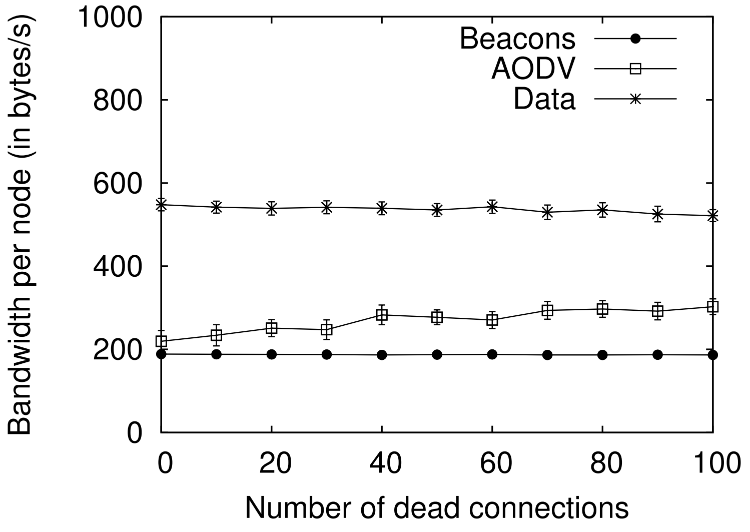
The situation with presence detection services running is much better. Since the presence detection service blocks the vast majority of connection attempts to non-present nodes, the bandwidth usage does not change significantly. There is only a minor increase in bandwidth used by AODV, which is due to false positives, causing some non-present nodes to appear as present.

Note that in a real network, the amount of connection attempts to non-present nodes heavily depends on the nature of the network and the application. Our results here show that it is possible to gain a benefit from exploiting presence information.

We also carried out the same simulation but using compressed aggregates for presence detection service. Figure 3.11 shows a small gain of network performance in terms of delivery ratio compared with the case when aggregates are not compressed. The gain can be explained by smaller bandwidth usage in the case of compressed Bloom filter as depicted in Figure 3.12.



(a) Without presence detection.



(b) With presence detection.

Figure 3.10: Bandwidth use per node as the number of connection attempts to non-present nodes increases.

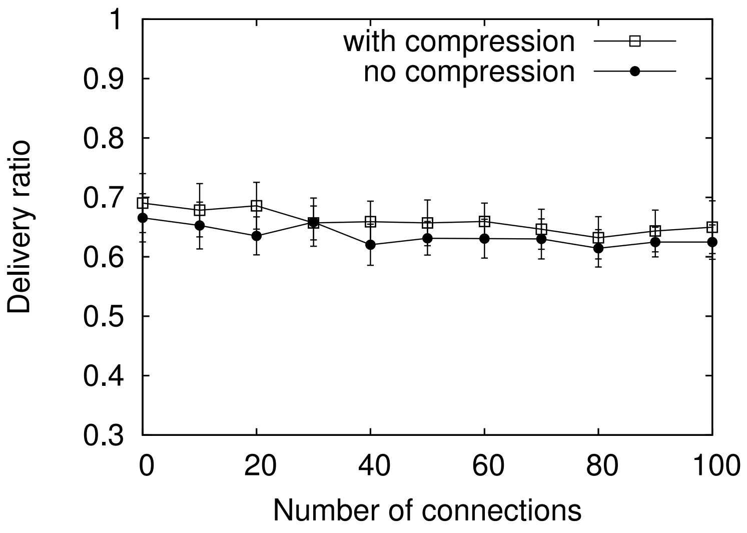


Figure 3.11: Delivery ratio in the case with compressed and uncompressed Bloom filters.

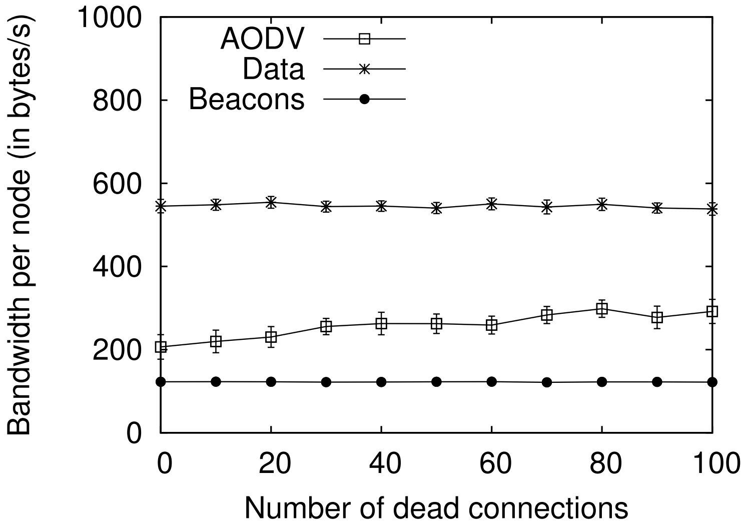


Figure 3.12: Bandwidth in the case with compressed Bloom filters.

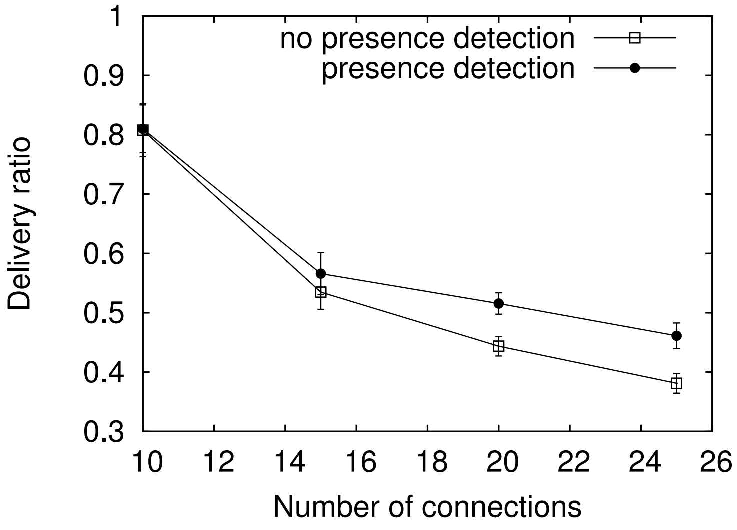


Figure 3.13: Delivery ratio with arriving and leaving nodes.

3.3.3 Arriving and Leaving Nodes

In this set of simulations, nodes actually enter and leave the network. One third of the nodes to which connections are attempted is permanently present, another third are switched on at some random time during the simulation, and the remaining nodes are initially present, but are switched off at some random time. This means that some connections run smoothly, some others can possibly be delayed until the destination is up, or they might be abruptly terminated because the destination is switched off while the connection is running. In Figure 3.13 it can once again be seen that the presence detection service helps to achieve a substantially better network performance.

3.3.4 Periodically Sleeping Nodes

Our last set of simulations emulates a situation in which nodes periodically go to sleep. This kind of periodic sleeping has, for example, been suggested for sensor and actuator networks (SANETs) [AK04]. During the simulation, the network nodes are switched on and off according to an exponential random model, in which the average up-time and down-time of nodes are 180 and 120 seconds, respectively. The only exception are the 25 senders, which are permanently on. As a result some connections run smoothly, some others can possibly be delayed until the destination is up, or they might be abruptly terminated because the destination goes to sleep.

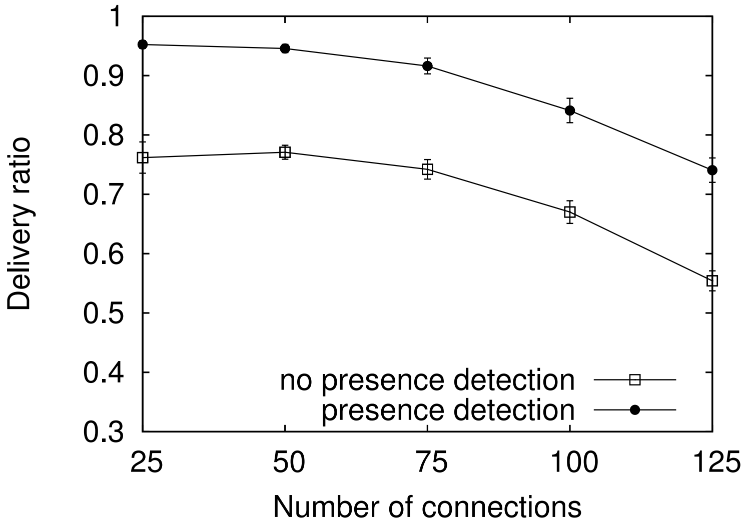


Figure 3.14: Delivery ratio with periodically sleeping nodes.

In Figure 3.14, it can once again be seen that the presence detection service helps to achieve a substantially better network performance. This can be explained by the numbers of AODV protocol packets being transmitted in the simulation, shown in Figure 3.15. In the case without presence detection the number of AODV packets sent is almost twice as large as is the case with presence detection.

3.4 Real-world Experiments

In the previous sections, we introduced an algorithm to detect the presence of nodes, assessed the performance and suitability of the proposed scheme, and evaluated the algorithm in an example application using the ns-2 network simulator. To accompany all that with a test-bed, we implemented a `whoisthere` service providing the basic functionality of presence detection. This section will describe the real-world experiments we then carried out to see whether the algorithm works as expected and whether there are any observations that are different from what had been seen in simulations.

In order to control and repeat the experiments, we used the experiment control software EXC [KOM08] on an IBM Thinkpad X40 laptop as the monitoring node. Other nodes executing `whoisthere` were Zaurus SL-6000 PDAs running OpenZaurus Linux version 3.5.4.2. The EXC software issued commands requesting nodes to switch on and off `whoisthere` according to specific experiment scenarios. Thanks to the central control

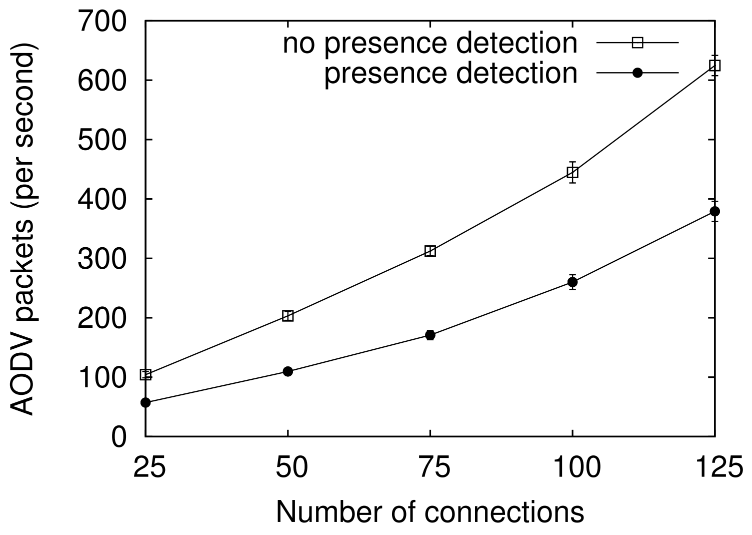


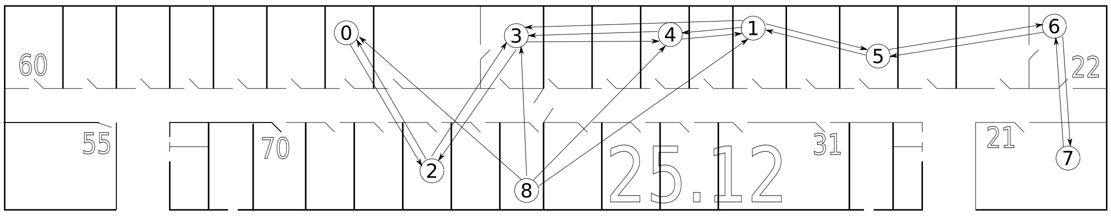
Figure 3.15: AODV packet count in the simulations with periodically sleeping nodes.

of timing, EXC provides us with two advantages. First, experiment scenarios can be repeated with precise timing. Second, clock differences between nodes create no problem in measuring the time between events, since nodes' local timestamps of events, as offsets to nodes' on-times, can all be converted to EXC time using EXC's local records of nodes' on-time.

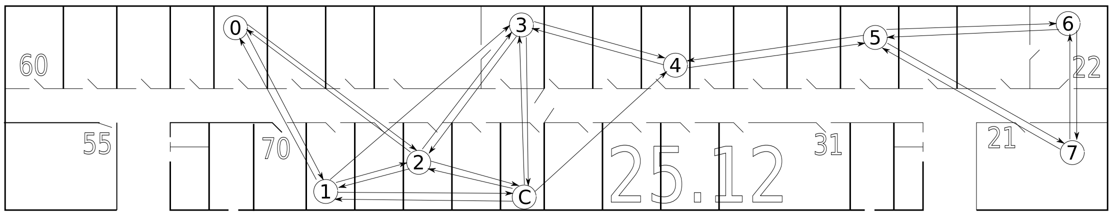
3.4.1 Static indoor

The sets of experiments presented in this section were designed to see how quickly nodes detect the presence of others. Here we investigated one of the most interesting aspects evaluated in Section 3.2: the speed of information propagation. In particular, we wanted to measure the delay over varying hop distances in order to see how well real-world experiments results match the simulations, noting that simulations and real-world are two different environments by nature and that we did not have many devices to form real-world networks as dense as we can in simulations.

To know roughly the hop-distances between nodes, we had to limit our experiments to static networks, since it would be extremely difficult to determine actual hop-distances in a network whose topology changes constantly. For each experiment, we used a network of 8-9 nodes distributed on the second floor of the computer science building. One node in the network switched on and off its presence detection service, thus 'disappeared' from the network when the software was switched off and 'entered' the network when



(a) Set 1: Network of 9 nodes including a controller node, 8.



(b) Set 2: Network of 8 nodes, excluding the controller node, C, who did not run the presence detection service

Figure 3.16: Ping snapshots of two static experiment sets.

the software was switched on. We measured the time until a newly 'appearing' node was seen as present by other nodes and compared this with the corresponding simulation results.

In the first set of experiments (Figure 3.16(a)), we arranged nodes in a chain-like topology. One node at the end of the chain was to switch its presence detection service on and off periodically. This is the worst case in the terms of propagation speed, since chain-like networks should enjoy little or no multi-path effect as has been seen in Section 3.2's simulations. Unlike in simulations, setting up a perfect chain in a real-world situation is very difficult as signals can sometimes reach over longer distances creating a temporary short-cut. In our experiments, we have, in fact, observed runs in which the network was not fully connected, while in other runs of the same scenario the shortest hop distance between two ends of the networks were as small as 5 hops (see Tables 3.1 and 3.2).

In the second set (Figure 3.16(b)), nodes were placed in a more random manner, and thus better connected. We expected to see higher propagation speeds than those of the first scenario, though both are unlikely to be as fast as those in simulations due to the limited density of the network.

	Distance reported														
Node	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	111	16	0	0	0	0	0	0	0	0	0	0	0	0
2	117	6	1	0	0	0	0	0	0	0	0	0	0	0	0
3	85	39	3	0	0	0	0	0	0	0	0	0	0	0	0
4	0	112	14	0	0	0	0	0	0	0	0	0	0	0	1
5	0	0	112	17	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	96	24	2	1	0	0	0	0	0	0	0	4
7	0	0	0	0	93	24	2	1	0	0	0	0	0	0	8
8	82	44	1	0	0	0	0	0	0	0	0	0	0	0	0

Table 3.1: Shortest hop distances to node 0, set 1, run 03

Figure 3.17 depicts the results of the two experiment sets, averaged over ten runs. The x-axis denotes the shortest hop-distances between nodes, i.e. the shortest distance recorded for each pair of nodes during all the experiments. The y-axis then shows the average time until the presence information of one node arrives at another. The dashed line shows the expected information propagation speed along a single path. Simulation results from Section 3.2.4 are also included in the figure for comparison.

Complementing to Figure 3.17, Tables 3.1 and 3.2 show the statistics of hop-distances to node 0 reported at the other nodes during two sample runs. During the experiments, nodes recorded hop-distances to node 0 at the end of each beacon interval. The value of the table cell at the intersection row i and column j is the number of times when the distance to node 0 is reported at node i as j hops.

We can see in Figure 3.17 that set 1's curve stays mostly above but very close to the line of theoretically expected delay over a chain of nodes, and so does set 2's curve to that of simulational random-topologied networks. The first reason why test-bed results are not as good as simulation results, as explained above, is that information dissemination in scarce networks, which is the case in our test-bed experiments, does not enjoy multi-forwarder effects as much as that in dense networks, which is the case in our simulations. Second, link quality in the real-world is usually not as consistent as in simulations. During experiments, the actual shortest path between a node pair is sometimes longer than the shortest path recorded over time. Table 3.2 shows a run in which the overall shortest paths (shown in Figure 3.17) failed far more often than they succeeded. Thus, taking these limitations into account, the average propagation speeds in real-world experiments are very close to our expectation, and the test-bed results are consistent with the simulation results.

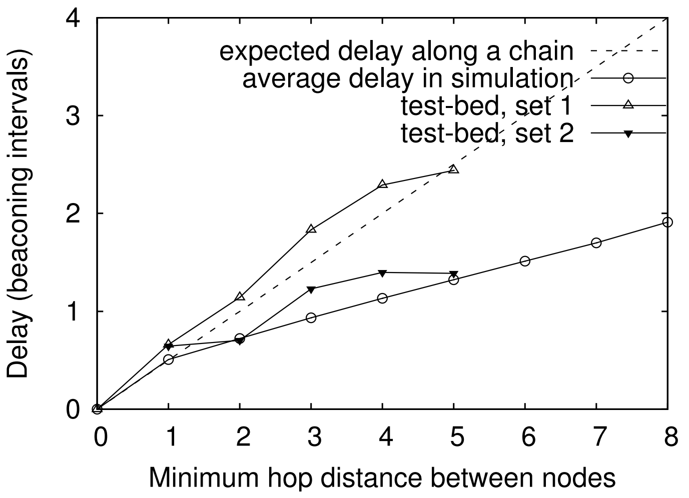


Figure 3.17: Test-beds: Average delay before a node is recognized as present.

Node	Distance reported														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	129	3	0	0	0	0	0	0	0	0	0	0	0	0	1
2	106	26	1	0	0	0	0	0	0	0	0	0	0	0	1
3	30	100	0	0	0	0	0	0	0	0	0	0	0	0	2
4	0	29	95	1	1	0	0	0	0	0	0	0	0	0	8
5	0	0	28	75	7	2	1	1	1	0	0	0	0	0	17
6	0	0	0	27	73	11	3	1	1	1	0	0	0	0	17
7	0	0	0	0	26	68	10	3	1	1	1	0	0	0	21

Table 3.2: Shortest hop distances to node 0, set 2, run 08

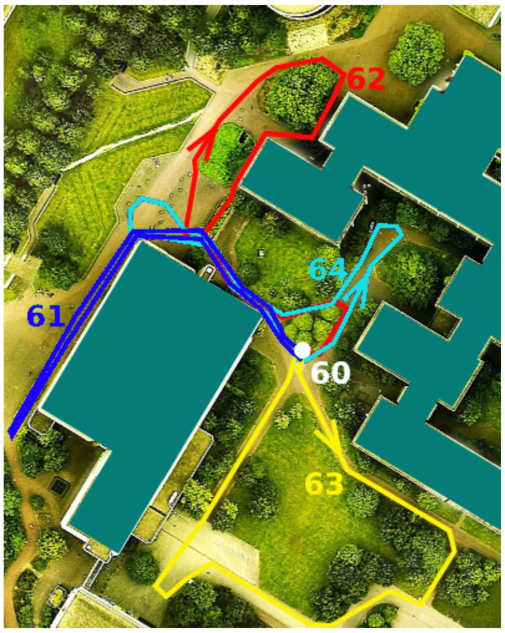


Figure 3.18: Movement of the four mobile nodes (61-64) and position of the stationary node 60 during mobile outdoor experiment.

3.4.2 Dynamic outdoor

In this set of experiments, we wanted to observe the presence detection service in dynamic networks, looking for any interesting behaviors resulting from changes of topology and connectivity that have not been seen before in our simulations.

Our mobile experiment consisted of five nodes: four mobile nodes (IDs 61-64) and one static node (ID 60), which also served as the monitor. Figure 3.18 shows the experiment's map and node movement. Nodes 61-64 started at the same position as node 60, then moved away in different directions before returning to the starting point at the end of the scenario.

Similarly to the static experiments discussed in Section 3.4.1, one of the network nodes, i.e. node 61, periodically switched on and off its `whoisthere` service, and we measured the time until the other nodes recognized node 61's appearance and disappearance. However, unlike in the static scenarios, nodes participating in this experiment moved all the time. The real hop-distance between nodes changed and nodes were sometimes disconnected because of either long distances or obstacles.

In this dynamic scenario, it is impossible to determine 'actual' hop-distances at a given point in time. Hence, it is impossible to construct a diagram showing delays over

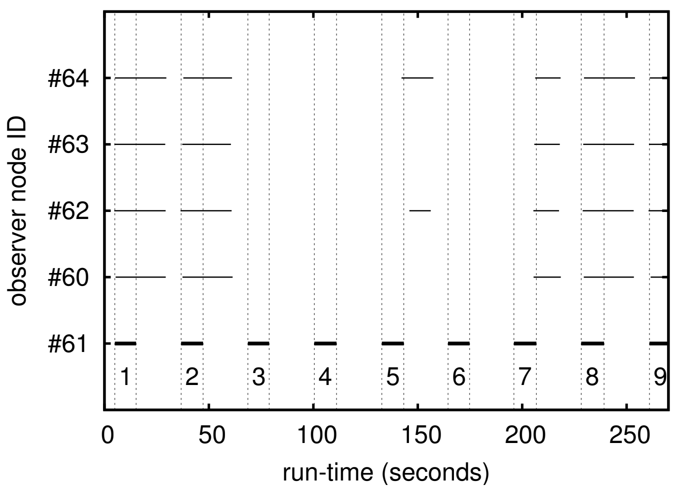


Figure 3.19: Presence information of node 61 at each node during run 1. Horizontal line segments represent periods when node 61 is "seen." Node 61's own segments represent actual on-times

distances similar to that in Figure 3.17. Instead, Figure 3.19 shows an example of how the presence of node 61 is perceived at the other nodes during the experiment. We can see in the figure that node 61's presence information usually took almost no time to reach the others and always took some time to die out after node 61 had gone off. That demonstrates the expected behavior of the presence detection service when nodes are well connected. The interesting parts are in the middle of the run, when nodes were farther away from each other and sometimes disconnected due to long distances or obstacles. At the third, forth and sixth on-times, node 61 was "on" but unseen by the others because its beacons could not reach any of them. At its fifth and seventh on-times, the long "delay" were actually not propagation delay but the time before node 61 got close enough to other nodes for its beacons to reach them.

This experiment again demonstrates that our approach to presence detection works well in real-world environments.

3.5 Privacy Concerns

The fact that nodes in a network all read Bloom filters from others and transmit their own Bloom filters to others leads to several issues concerning the accuracy and secrecy of the information the presence detection service provides. First, accepting beacons produced by strangers entails the possibility of receiving false information. Second, beacons being read by strangers could lead to the risk of presence information being revealed to unwanted parties.

In the first problem, there are false positives created by a node adding fake presence information or indiscriminately setting Bloom filter entries to low values, and false negatives caused by a node removing some presence information from its outgoing beacons by setting entries to high values. Because the merge operation by nature always registers the most positive presence information available and ignores the rest, and because presence information is disseminated in all directions and received via multiple paths (except in the rare case of a perfect chain of nodes), false negatives are quickly corrected and thus only rarely cause mis-information. Meanwhile, false positives cannot be corrected that way causing some non-present nodes to appear as present. Thus, in the worst case, an attack would be able to create the situation we had before introducing presence detection.

The second problem concerns node privacy. The presence status of a node is available to all nodes. For example, by examining an incoming beacon, a spying node can recognize the 'bit pattern' of the neighbor who sent the beacon, i.e. the set of Bloom filter entries that neighbor uses for its own presence announcement. In soft state Bloom filters, this pattern stands out as the set of entries with the freshest value. From then on, the spy can monitor the presence status of that node as long as they are in the same network.

In this section, we will tackle the second problem and propose a preliminary solution. Our objective is to prevent nodes from monitoring others without their permissions, while maintaining the effectiveness of the present detection service.

Let us call the nodes which are allowed to see the node u as "friends" of u. We assume that they can be trusted not to reveal u's secret keys and not to disclose u's presence status in any way. Otherwise, in the case that a "friend" is also a spy, even when presence detection service employs the ideally best mechanism against privacy invasion, the spy can always forward the secret information in a way that is far beyond the scope of a presence detection service, an example of which might be as simple as an email with the content "Watch out! The guy is around!"

To extract presence status of a node from a Bloom filter, it requires either the node's hash key(s) or, more directly, the list of Bloom entries the node reserves for its own presence announcements–its 'bit pattern'. Protecting node privacy means keeping both of them secret. The hash keys can be practically protected against all non-friend nodes by using encryption in the case of key transmissions, and by using good cryptographic hash functions as Bloom filter hash functions to prevent cryptanalytic attacks. However, the 'bit pattern' cannot be shielded from all. As explained in the example above, a non-friend neighbor can recognize 'bit patterns' from incoming beacons. Obviously, having been near a spy only once will lead to being spied on forever, unless the node changes its 'bit pattern' over time. In other words, hash keys must change over time.

Next comes the question of how the sought-after node changes hash keys without disrupting normal look-up operations of its friends, i.e. the second half of our objective stated above. Suppose the node should inform its friends by sending the new keys to all of them. However, there is a risk that some friends fail to receive the latest key announcement, because they were not around at the right time or the announcement itself got lost on the way, and, as a result, they will not be able to 'see' the node until after receiving the next announcement. If keys are announced frequently enough to reduce the risk, the cost will be high bandwidth usage, which defeats the idea of a lightweight presence detection service. Therefore, nodes should be able to independently calculate friends' hash keys instead of waiting for key announcements from friends and periodically sending one's own.

3.5.1 Hash Keys that Change over Time

We now describe in details our algorithm which deals with the privacy problem.

Each node u has two keys, which need to be securely distributed to its friends in advance:

- sk_u, a unique secret code, similar to node ID in normal cases,
- tk_u, period of time in seconds between two consecutive changes of hash keys,

The hash key node u uses for its own presence information is constructed from sk_u and t–a time-dependent factor to be described later–using a predesignated function that combines the pair, an example of which is the concatenation of the pair's bit-representations. Once the hash keys for Bloom filter have been calculated, the algorithms for refresh, merge, query operations are the same as described in Section 3.1.2.

Let c be the current local time at u, we define t at u as

$$\left\lfloor \frac{c}{tk_u} \right\rfloor \tag{3.11}$$

Calculations of t in refresh operation are as simple as that, as nodes always know their own local time.

However, in query operation, it is not as straightforward to determine t. Looking up u in an aggregate, a node v does not know the c value that u used to calculate t for the piece of information v is now trying to locate. One might think of using timestamps to fix the problem. However, as Bloom filters contain information from various nodes gathered at different points in time, each entry in a Bloom filter would need one timestamp of its own, which makes timestamp an unaffordable solution.

Therefore, v must try to "guess" the exact t that u so as to "get" the correct hash key to the presence information of u. Theoretically, t can be calculated as

$$\left\lfloor \frac{c - \tau - \delta}{tk_u} \right\rfloor \tag{3.12}$$

where c is the clock value when v receives the information, δ is the clock difference between u and v, and τ is the time needed for presence information to travel from u to v. However, δ is generally not known to v, neither is τ as it is expected to constantly change in a mobile ad-hoc network. Thus, while guaranteeing to give the correct answer, this theoretical method of calculating exact t is infeasible.

Nevertheless, it can help in estimating t using v's current clock value. Let Mc and Mt be the upper bound of δ and τ, respectively, whose values are to be estimated in advance with regards to the knowledge of the network in use. We have $-Mc \leq \delta \leq Mc$ and $0 \leq \tau \leq Mt$. It follows that $(c - Mt - Mc) \leq (c - \tau - \delta) \leq (c + Mc)$, or $\frac{c-Mt-Mc}{tk_u} \leq \frac{c-\tau-\delta}{tk_u} \leq \frac{c+Mc}{tk_u}$

Applying (3.12), we conclude that t must be an integer in the range

$$\left[\left\lfloor \frac{c - Mt - Mc}{tk_u} \right\rfloor, \left\lfloor \frac{c + Mc}{tk_u} \right\rfloor \right] \tag{3.13}$$

Given Mc and Mt, carrying out look-ups with each integer in that range, v cannot miss the presence information of u.

In short, whenever v wants to query presence information of u, instead of performing one Bloom filter look-up for u using one set of hash keys as in the original algorithm presented in Section 3.1.2, v will have to perform one or more look-ups with different hash key sets. If any of those look-ups gives a positive answer, v will conclude that u is present, otherwise, "not present" will be the answer.

3.5.2 Effects on False Positive Rate

The fact that more than one look-up might be required for each query of node presence raises the question of how badly it affects the accuracy of the query's answer, or more specifically, how it increases the false positive rate. Our analysis as well as simulations presented below will show that the cost is acceptable.

From (3.13), we have the number of necessary look-ups, i.e. the number of t's candidate values, (denoted as λ) is

$$\lambda := \left\lfloor \frac{c + Mc}{tk_u} \right\rfloor - \left\lfloor \frac{c - Mt - Mc}{tk_u} \right\rfloor + 1 \tag{3.14}$$

Since $\frac{c+Mc}{tk_u} \geq \left\lfloor \frac{c+Mc}{tk_u} \right\rfloor > (\frac{c+Mc}{tk_u} - 1)$ and $(1 - \frac{c-Mt-Mc}{tk_u}) > -\left\lfloor \frac{c-Mt-Mc}{tk_u} \right\rfloor \geq -\frac{c-Mt-Mc}{tk_u}$, by adding up corresponding parts of the two inequations, we have:

$(\frac{Mt+2Mc}{tk_u} + 1) > \left\lfloor \frac{c+Mc}{tk_u} \right\rfloor - \left\lfloor \frac{c-Mt-Mc}{tk_u} \right\rfloor > (\frac{Mt+2Mc}{tk_u} - 1)$

which is equivalent to

$$\left(\frac{Mt + 2Mc}{tk_u} + 2 \right) > \lambda > \frac{Mt + 2Mc}{tk_u} \tag{3.15}$$

Noting that λ must be an integer that is strictly less than the upper bound given in (3.15), it turns out that $(\frac{Mt+2Mc}{tk_u} + 2)$ is not too high an upper bound for λ. Let us take an example. Suppose u changes keys after every rather short period of ten minutes, i.e. $tk_u = 600$. Then the time it takes for presence information to travel across the network plus twice the clock difference between pairs of nodes, i.e. $Mt + 2Mc$, could be as much as ten minutes, or 600 seconds, without causing the number of required look-ups (λ) to exceed 2.

There is another issue that affects the false positive rate. After a node switches to a new key, until its presence information using the old key die out, the node appears to be registered multiple times. This increased number of set positions in the Bloom filter can lead to temporary higher rates of false positives.

To better show the cost of our privacy protection mechanism, we carried out simulations using the ns-2 network simulator. We placed network nodes randomly on a square area of 1500 meters side length, and used the presence detection algorithm with $m = 1024$ filter positions, $k = 4$ hash functions for Bloom filter, and a beaconing interval of 1.5 seconds. The network size varies from 50 to 300 nodes. We chose the upper bound

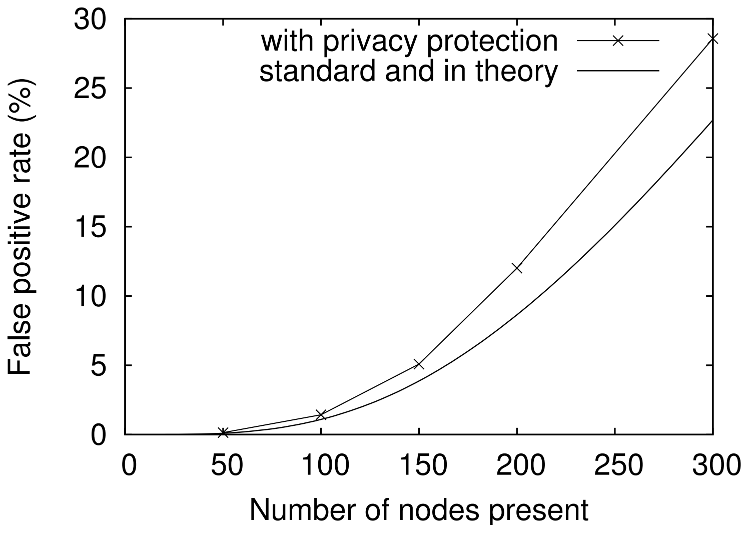


Figure 3.20: False positive rates. k = 4, m = 1024

of clock difference as $Mc = 300$ seconds, which is equivalent to five minutes; and the estimated maximal time needed for information to travel across the network as $Mt = 10$ seconds, which is quite generous an estimation given the statistics of information delay in Figures 3.5 and 3.6. Each node had a value tk randomly generated in the range [600, 3600] seconds, i.e. [10, 60] minutes. During each run, after presence information had been fully disseminated, 10,000 queries were issued on random node IDs regardless of whether the nodes were present in the network or not. False positive rates were then calculated from the results of these queries.

The results are presented in Figure 3.20 in comparison with the theoretical false positive rates of Bloom filters of the same size (Formula 3.3). We can see from the figure that the cost of the privacy protection is rather small, it increases only slightly the false positive rates of Bloom filters.

3.5.3 "Friend" Groups

To further the idea of "friends", a node could enjoy the flexibility of having more than one groups of friends by dedicating a separate key pair (sk, tk) to each group. This way, the node can actively choose to "appear" present to some groups while staying "non-present" to the others. In addition, the node can modify the member list of a friend group at will by setting a new key pair and send it to only the should-be members of the group dropping the others out. This feature of friend groups is readily supported by our algorithm.

3.5.4 Related Work

The idea of keys changing over time has been discussed in [CDM07]. While our approach allows nodes to have asymmetric relations with one another, the problem solved in the paper requires nodes in one clique to have symmetric relations sharing a common key. Although keys in [CDM07] also change over time and are calculated independently at different nodes, new keys are obtained just by applying hash functions to the current keys regardless of time difference between nodes. As a result, the mechanism described in that paper requires time synchronization among nodes or suffers resulting loss of signals, which is the very problem we have solved with our method of estimating t. Combined with it, the idea of hashing the key of one time slot to obtain the key for the next slot is, on the other hand, applicable to our problem without disallowing asymmetric node relations. However, the resulting t calculation will be computationally more expensive, especially in the case of high frequency of key changes.

3.6 Chapter Summary

In this chapter, we have presented a scalable solution to the presence detection problem, which is also able to estimate the number of hops required to reach a given node. A soft state variant of the well-known Bloom filter allows for an efficient aggregation of presence information. In particular, we extended Bloom filters to support soft state decay and refresh operations. Our algorithm aggregates presence information using these soft state Bloom filters. The aggregation comes at the cost of an adjustable amount of false positives, while it guarantees the absence of false negatives.

We investigated several key aspects of this approach, such as the speed of information propagation, the probability of false positives, and the bandwidth consumption by means of analysis and simulation. To underline the practical benefits that can be obtained by using presence detection, we also presented simulation results where presence detection was applied to the route discovery process of AODV. We were able to avoid issuing route requests to non-present destination nodes. We showed that the additional network load for the presence detection beacons is very limited and that substantial performance gains are possible if spurious flooding of the network with unsuccessful route discovery attempts can be avoided. Real-world experiments on PDAs also show that the presence detection works well both in static indoor and dynamic outdoor environments. In addition, the privacy issue was addressed and a mechanism was proposed to prevent spying on network nodes.

Chapter 4

Reducing the Overhead of Presence Detection

In Chapter 3, we have proposed a first mechanism that tackles the problem of detecting the presence of nodes. However, the mechanism provides more information than just node presence, it also yields a distance estimate to the respective node. This is a result of the employed data structure, a soft state variant of Bloom filters. This data structure provides a lossy compression of presence information and at the same time allows to remove old, timed-out information by aging information over time and distance. Distance estimates are a result of this particular data representation.

Although such hop distance estimates may be useful in some situations, it is not always necessary. It actually turns out that presence information can be represented in a substantially more compact form without distance information. This saves a significant amount of network bandwidth. The key to do so is a new, more space efficient method to remove old information. Here, we use a phase-based, coarse synchronization mechanism in order to periodically remove outdated presence information. We introduce a protocol that implements this mechanism and present both analytical and simulation results. Finally, in order to underline and concretize the effects of this reduced overhead presence detection mechanism, we apply it to the application scenario—reactive MANET routing—which was also used as the example application in Chapter 3. In both the general evaluation results and the application scenario, we compare the algorithm presented here with the presence detection mechanism previously proposed in Chapter 3.

In the following sections, we describe the approach with reduced overhead in Section 4.1, then evaluate it in Section 4.2. Section 4.3 contains the application study with reactive MANET routing. We finally conclude this chapter with a summary in Section 4.4.

The content of this chapter has been published in [TSM09b].

4.1 Algorithm

One primary objective of using a presence detection protocol in a network is to reduce unnecessary overhead, which may otherwise occur when communication with nodes or services is attempted, even though the intended communication partner is currently not available. For achieving this goal, it is of utmost importance that the overhead of the presence detection protocol itself is small—otherwise it will in many situations outweigh the benefits.

A way to achieve a small footprint of presence detection is to compress the presence information exchanged between the nodes. A central observation made in Chapter 3 is that presence information does not need to be absolutely accurate, as long as only a small number of *false positives* occur. In case of a false positive, a node is wrongly considered to be present. This is not critical for many typical applications: in the worst case, the cost of a false positive is an unnecessary attempt to contact a non-present node, just as if no presence detection were used. Thus, in that case, the system behaves like one without presence detection. False negatives, however, must not occur, because they would result in connection attempts not being made, even though the destination node is in fact present. Because a certain number of false positives is tolerable, while false negatives are unacceptable, Bloom filters [Blo70] are an interesting candidate for representing the set of currently present nodes in a very compact form.

The central problem that arises is to remove information about no longer present nodes from the aggregates. The solution proposed in Chapter 3, called soft state Bloom filter, achieves this goal but comes at the cost of substantially increasing the size of the standard Bloom filter. The additional information carried in the modified Bloom filter can be useful, because it allows to provides the querying node with an idea about the distance to the destination, if it is present. Often, though, such distance estimates are not necessary—and therefore the central question arises: is it possible to perform presence detection with *unmodified* Bloom filters, and still remove old information?

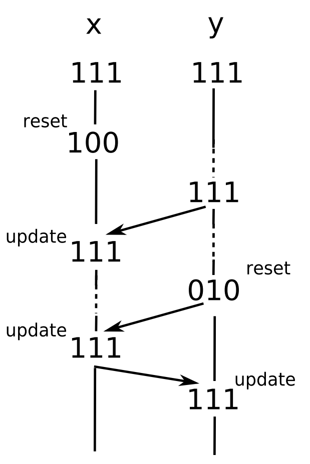
4.1.1 Phase Synchronization

Because the information in a soft state Bloom filter's positions ages while traveling through the network, the values of the counters at the bit positions corresponding to a node give an idea of the distance to this node. The cost of this information is, however, high: the algorithm proposed in Section 3.1.2 results in a size increase of the exchanged Bloom filters by a factor of l, in comparison to standard, unmodified Bloom filters. As it turns out, it is actually possible to achieve network-wide soft state behavior *without* the need to transmit soft state Bloom filters with counters at each position in the presence announcement beacons.

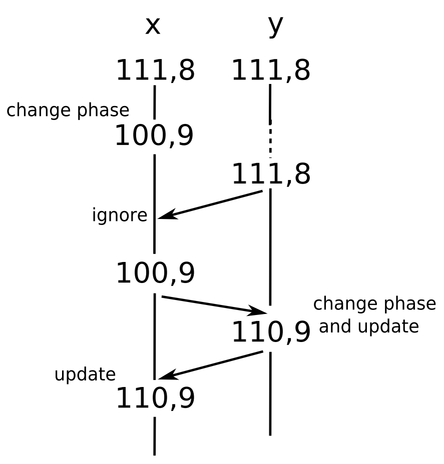
In order to get rid of the overhead for counters in each bit position, we return to exchanging *standard* Bloom filters in periodic beacons. As mentioned before, they do not allow to selectively remove information about single, specific, no longer present nodes. The general idea how old information can nevertheless be removed is surprisingly simple, at least at a first glance: we may periodically reset all bits in all nodes, and then start over by collecting information about all (still) present nodes again. However, two challenging problems make implementing such an approach more tricky than it initially seems.

The first such issue is related to synchronizing the process of resetting the Bloom filters in the nodes. If old information is removed asynchronously, it will not be permanently removed. To see why, consider the example shown in Figure 4.1(a). For simplicity's sake, instead of "real" Bloom filters, presence information of the three nodes x, y, and z is represented by a single bit each in our examples; in the example aggregates, x maps to the first, y to the second, and z to the third bit. In Figure 4.1(a), z has just left the network. Its presence has been (and is still) known by x and y, which can be seen from the third bit in their aggregates being set. x is first to reset its aggregate. However, with a naive reset, x may then receive a beacon from y that still contains the old information. As a result, not only the information about y's presence, but also the outdated information about z is recovered. If y later resets its own aggregate, the next beacon from x will again revive the information about z. In short, presence information of the no longer present node z will never die out.

Accurately synchronized clocks in all nodes could provide a solution to the problem by scheduling them to reset all the Bloom filters at the same time. However, perfect time synchronization is not realistic in many distributed real-word applications. So, instead of relying on time synchronization, we solve the problem by using a *phase synchronization mechanism.* Each node maintains a phase counter, which increments periodically. A node advances to the next phase after having operated in its current phase for a globally specified maximum phase duration (C beaconing intervals). Whenever such a phase transition occurs, the node resets its Bloom filter and starts over with collecting



(a) Naive resetting.



(b) Phase synchronization.

Figure 4.1: Phase synchronization for the removal of old information.

information about other nodes from scratch. The current phase ID is attached to every transmitted beacon. A phase will typically last a few seconds, and therefore a small (for instance, 32 bit) integer phase ID suffices even for a very long timespan, so that it does not significantly increase the beacon size. The phase ID in the beacons allows nodes to recognize and ignore beacons from neighbors which are less advanced in phases, making sure that old information—from phases with lower IDs—will not reappear in the aggregate.

Figure 4.1(b) depicts a situation similar to that in Figure 4.1(a), except that phase IDs are used. In the figure, x resets its aggregate, transitioning from phase 8 to phase 9. When it receives a beacon from y with phase ID 8, this information is ignored. As soon as y is also in phase 9, information is accepted again. Therefore, information on y's presence is accepted, but outdated information on the no longer present node z is reliably removed.

However, such an approach requires that the nodes in a network must somehow come to be in the same phase. We therefore let nodes in less advanced phases "catch up" with more advanced neighbor nodes, thereby (coarsely) synchronizing their phases. The principle we apply in this synchronization process is that every node catches up with its most advanced neighbor. Each node maintains that principle by examining the phase IDs in received beacons, and adjusting its own phase accordingly if the ID is higher than its own current value. In Figure 4.1(b), y performs such a transition when receiving a phase 9 beacon from x. Of course, such a neighbor-triggered phase transition

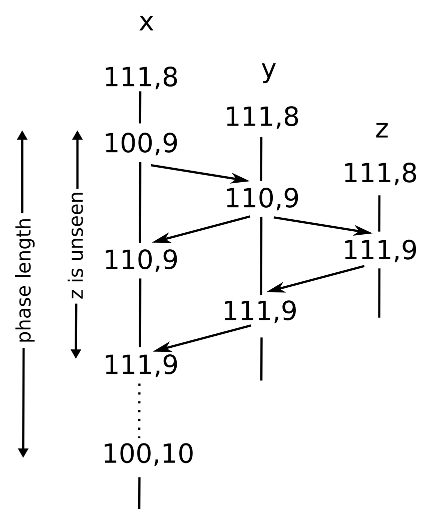


Figure 4.2: Temporary inconsistency.

also includes resetting the local aggregate, just like a timeout-triggered one. Note that an accurate synchronization—i. e., all nodes advancing to the next phase exactly at the same time—is *not* necessary, as long as the nodes remain within the same phase for long enough, so that presence information can be disseminated through the whole network. We will soon return to the issue of choosing phase length and beaconing interval appropriately in more detail.

4.1.2 Overcoming Temporary Inconsistencies

Temporary inconsistencies after a phase transition are the second central problem that needs to be solved in order to make presence detection with small, unmodified Bloom filters viable. Immediately after a phase transition, a node's aggregate is empty. Until information has been gathered again in the new phase, it is of no use for presence detection—recall that false negatives are to be avoided. It takes time before information from all present nodes reappears at each aggregate. An example is depicted in Figure 4.2: x is first to transition from phase 8 to phase 9, resetting its aggregate. There is a short period during which y is not seen, and an even longer time until z, which is not a direct neighbor of x, also reappears.

To overcome this, we use an additional, more durable local copy of presence information. We reuse the soft state Bloom filter described in Section 3.1.2 for this purpose. However, in contrast to the algorithm described in Chapter 3, the soft state Bloom filter is kept only locally. It is not transmitted over the network. Whenever information from a

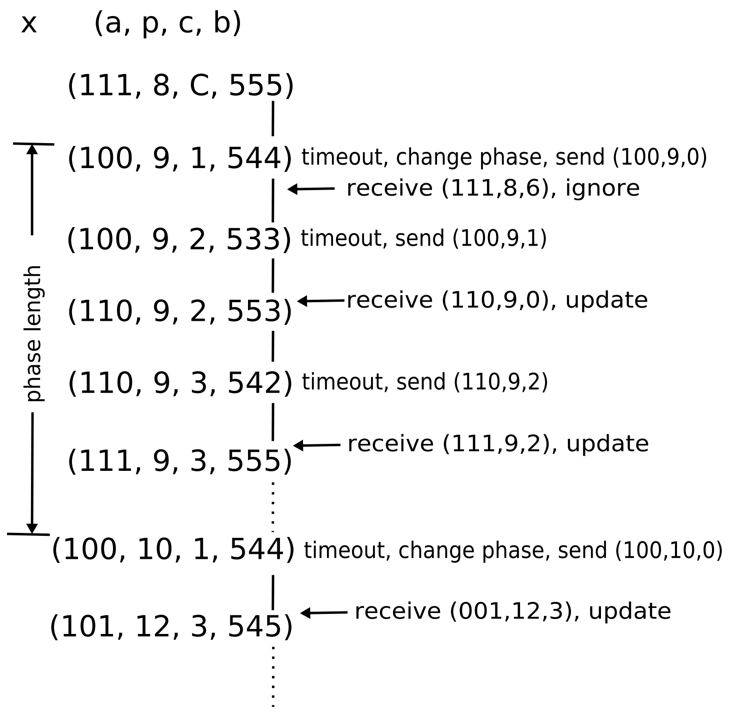

x
(a, p, c, b)
(111, 8, C, 555)
(100, 9, 1, 544) timeout, change phase, send (100,9,0)
receive (111,8,6), ignore
(100, 9, 2, 533) timeout, send (100,9,1)
(110, 9, 2, 553) receive (110,9,0), update
(110, 9, 3, 542) timeout, send (110,9,2)
(111, 9, 3, 555) receive (111,9,2), update
(100, 10, 1, 544) timeout, change phase, send (100,10,0)
(101, 12, 3, 545) receive (001,12,3), update
phase length

Figure 4.3: Close-up at one node.

received beacon is incorporated into the local aggregate, the corresponding positions in the local soft state Bloom filter are also refreshed. However, the soft state Bloom filter is not reset upon a phase transition. Instead, the information gradually decays where it is no longer refreshed and eventually fades out just as described above. The local soft state Bloom filter thus bridges the gaps after phase transitions, in some sense smoothing out the algorithm's behavior.

4.1.3 Detailed Algorithm Description

After having outlined and motivated the phase-based approach to remove old presence information from unmodified Bloom filter aggregates, we will now proceed by more formally introducing the complete protocol and data structures.

Data structure

For our phase-based presence detection protocol, each node locally maintains four data items: the current phase ID p of this node, the node's Bloom filter presence aggregate

a for the current phase (which is used for information exchange), the soft state Bloom filter b for local presence lookups, and the interval counter c indicating for how many beaconing intervals the current phase already lasts at this node.

Timeout

Periodically each node transmits a beacon. All nodes use the same beaconing interval length B, but their beaconing cycles do not need to be synchronized, i. e., they need not (and typically will not) generate beacons at exactly the same time. At the beginning of each beaconing interval, a node performs an operation consisting of four steps:

```
Decay the entries in the local soft state Bloom filter $b$.
if $c \geq C$ then {$c$ is at the threshold $C$}
  $p \leftarrow p + 1$
  $c \leftarrow 0$
  Empty $a$ and add own presence information to $a$
end if
Broadcast ($a$, $p$, $c$) to the neighbors.
Increment $c$.
```

Algorithm 4.1.1: Timeout and refresh

The beacons contain the current interval counter c because this allows for nodes to "catch up" when they receive beacons in which the interval counter c is higher than their own current value. This results in an even tighter synchronization.

Merge operation

The merge operation is performed whenever a presence detection beacon is received from a neighbor. It accomplishes two main functions: (1) it adds presence information received from neighbors who are either in the same phase or in a more advanced one, and (2) it synchronizes the node to the most advanced neighbor.

Upon receival of a beacon (p', a', c') from a neighbor, by examining the phase ID p', the node decides whether it should ignore the beacon, or catch up with the neighbor, or simply perform a normal update to its local information (p, a, b, c). The detailed procedure is shown in Algorithm 4.1.2.

Figure 4.3 depicts an example of state changes in response to events such as timeouts, phase changes, and received beacons, showing all the aggregates and counters in a node.

```
if $p > p'$ then {the sender is less advanced}
    {do nothing}
else if $p = p'$ then {both are in the same phase}
    $a \leftarrow$ merge_bloom_filters$(a, a')$
    $b \leftarrow$ merge_bf_into_softstate_bf$(b, a')$
    $c \leftarrow \max\{c, c'\}$
else {the sender is more advanced}
    $a \leftarrow a'$
    add own presence information to $a$
    $b \leftarrow$ merge_bf_into_softstate_bf$(b, a')$
    $c \leftarrow c'$
    $p \leftarrow p'$
end if
```

Algorithm 4.1.2: Merge operation

Query operation

In order to determine whether some node x is present, a node checks its local soft state aggregate b at positions $h_1(x), h_2(x), \ldots, h_k(x)$. If any of the bit positions corresponding to node x is not set (i. e., either it has never been set or it has already expired), it may be concluded that x is not present. Otherwise, x is considered present with some probability of a false positive.

4.2 Evaluation

In the previous section, we introduced a phased-based algorithm to disseminate presence information. In this section, we assess the performance and suitability of the proposed scheme. In particular, we concentrate on three aspects: the reliability of the scheme in terms of the false positive rate, how to choose the length of a phase, and the speed of information propagation. Where appropriate, comparison with the soft state approach from Chapter 3 is also presented.

4.2.1 False Positive Rate

A false positive occurs when the bit positions corresponding to the sought-after node are all set by other added elements. We use standard Bloom filters, so the probability of a false positive is well-known. It depends on three factors: the number of bit positions in the filter m, the number of hash functions k, and the number of elements n that are present in the set. The probability that a bit position is still zero after n elements with k bit positions each have been added is $(1 - 1/m)^{kn}$. Thus, the probability that all k bit positions of the sought-after node are one is

$$\left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^{k} \approx \left(1 - e^{-kn/m}\right)^{k}.$$

4.2.2 Choosing the Phase Length

The algorithm proposed here spreads information bi-directionally among nodes in the same phase, and uni-directionally otherwise, i. e., information from a more advanced node is accepted at a less advanced node, but not the other way around. However, once a uni-directional dissemination takes place, the less advanced node will at the same time synchronize itself with the more advanced neighbor. Consequently, a central issue is to choose an appropriate phase length: excessively long phases will delay the removal of outdated information from the network. On the other hand, if a phase is too short, a "complete" picture will not have been collected before the most advanced node again transitions to the next phase, resetting the aggregates.

It is straightforward to see that the worst case is the following simple scenario: assume the network has a chain topology with nodes labeled $0, 1, 2, \dots, d$. Let 0 be the most advanced node and d the least advanced node within the current phase, i. e., 0 will be the first to transition to the next phase. In order for 0 to receive presence information from d, the new phase must first propagate along the chain from 0 to d, i. e., every node $0, \dots, d-1$ must send a beacon. Once d is synchronized to the new phase, each node $d, \dots, 1$ must send a beacon in that order for the bits set by d announcing its presence to arrive at node 0. At this point in time, 0 must not yet have advanced to a new phase again. In other words, a phase length should be no shorter than the time needed to relay information in beacons from 0 to d and back.

Let u and v be neighboring nodes. Let $x_{u,v}$ be the time between u sending a beacon and v sending the next beacon after receiving the one previously transmitted by u. We may safely assume that (at least over shorter timespans) the relative offset between the periodic beaconing cycles of two nodes remains approximately the same, such that for each pair of nodes $x_{u,v}$ is constant. The time D needed to relay beacons from node 0 to d and back is

$$D = x_{0,1} + x_{1,2} + \ldots + x_{d-1,d} + \ x_{d,d-1} + \ldots + x_{2,1} + x_{1,0}.$$

Observe that $x_{u,v} + x_{v,u}$ is just the time between two successive beacons sent by u—i. e., it is the beaconing interval length, here denoted by B. By reordering we therefore obtain

$$D = \sum_{i=0}^{d-1} (x_{i,i+1} + x_{i+1,i}) = d \cdot B.$$

Consequently, the phase length C in beaconing intervals should at least be the network diameter in hops, plus potentially some additional time to account for possible beacon losses. Note that this is significantly shorter than what one could have expected at a naive first glance: d beaconing intervals actually suffice for information to travel over d hops forth *and* back again!

As explained in Section 4.1.2, to overcome the temporarily incomplete information in the Bloom filters after a phase transition, we use soft state Bloom filters locally as the more durable local copy of presence information. Its TTL parameter, determining how many beaconing intervals a soft state Bloom filter entry remains set without being refreshed, should be set to a value with the same lower bound as the phase length C.

4.2.3 Speed of Information Propagation

In the phase-based approach, information is propagated in the same manner as in the soft state approach presented in Chapter 3. Thus, it may be expected that the dissemination time should be the same as for the soft state approach, plus an additional delay of one beaconing interval for arriving nodes to get phase-synchronized with the rest of the network (this synchronization takes place as soon as a newly arriving node receives the first beacon).

The other interesting parameter is the time until a leaving node is no longer considered present. In the soft state approach, after node x has left, it will be considered present until the counters have completely decayed. If the presence information about x is currently t beaconing intervals old, this will happen after $TTL - t$ more beaconing cycles. In the phase-based approach, after x leaves, it will be removed first from the exchanged aggregate (the unmodified Bloom filter) when a node transitions to the next phase, and later from the node's local soft state Bloom filter when the TTL expires. Thus, depending on whether x left right at the beginning or more towards the end of a phase, the time until information about x is removed at another node varies in the range

$TTL \dots (C + TTL)$ beaconing intervals, plus or minus a time less than one phase length due to the fact that some nodes are more advanced within the phase than others.

As explicated above, both the phase length C and the TTL should be chosen slightly higher than the expected network diameter. Thus, the phase-based approach can be expected to react half as fast as the soft state approach, and the delay will be proportional to both the length of the beaconing interval and the network diameter.

We conducted simulations to verify this expectation using the ns-2 network simulator [ns2]. We placed 200 nodes randomly on a square area of 1500 meters side length, and used the phased-based presence detection algorithm with $m = 1024$ filter positions, and a beaconing interval of three seconds. As the maximal diameter of these networks is 10, we chose a phase length of 10 intervals, The simulation uses IEEE 802.11 at 1 MBit/s and 250 m radio range. 199 nodes were initialized with random phase IDs. After they have all synchronized and learned about the presence of each other, the 200th node enters the network and the delay until it was recognized as present by other nodes is measured. The node to be watched always starts with phase ID 0, so that it has to synchronize with neighbors before its presence is recognized.

Figure 4.4 shows the time until a node is considered present by all other nodes within 10-hop distance after it has arrived, and the time until presence information of leaving nodes vanished from the aggregates of all other nodes. Corresponding results of the soft state approach are included for comparison. The times are given for different beaconing intervals with 95-percentile error bars.

For arriving nodes, it can be seen that, as expected, the delay increases linearly with the beaconing interval length, and that the average dissemination time in the phased-based approach is one interval longer than that in the soft state approach. For leaving nodes, the simulation results also confirm the theoretical expectations.

4.3 Example Application

In the previous section we assessed the general properties of the algorithm analytically and in simulations. Node presence detection can, as stated before, be used in a large range of applications. The results presented in this section quantify its impact in one specific, exemplary one. Again we compare the results to those obtained with the soft state Bloom filter presence detection mechanism from Chapter 3.

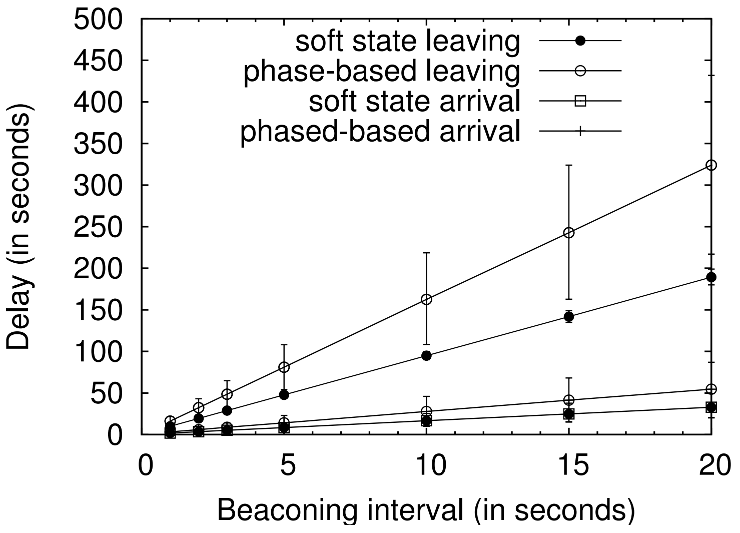


Figure 4.4: Time until information is received by all nodes.

We analyze both algorithms in the same exemplary application discussed in Section 3.3. In networks with AODV routing [PR99], we query the presence detection service at the source node before starting a route request for a new connection. If the destination node is considered present, the connection will be initiated as usual. Otherwise, the route request is delayed until the presence detection service indicates that the destination node is present. This avoids unnecessary route discovery attempts to currently non-present nodes.

In order to be able to investigate the impact of the network size, we used two classes of networks: (1) networks of 200 nodes in an area of 1500 by 1500 meters (we refer to those as "medium-size"), and (2) networks of 400 nodes in an area of 2500 by 2500 meters ("large-size"). The simulation study was conducted using ns-2 [ns2]. IEEE 802.11 is used at 1 MBit/s bandwidth. This is a relatively low value; however, note that a low network bandwidth is particularly hard for a beacon-based presence detection scheme: at higher total bandwidths, the fraction of the network capacity spent for presence detection beacons will be lower. We therefore chose to assess our scheme under these particularly difficult circumstances.

We use the same setting as that in Section 3.3. The communication radius in our simulations is 250 meters, with a 550 meter carrier sense range. The nodes move according to the random waypoint mobility model with random speeds in the range from 1 to 10 meters per second and a pause time of 20 seconds. Again, in order to overcome the well-known limitations of random waypoint [YLN03], we used the modified version of the model, initialized with the steady-state distribution. The beaconing interval is set to three seconds. All connections last 100 seconds and start at some random time between 10 and 190 simulation seconds. During a connection, the source node sends constant bit rate traffic with four data packets per second, each with a payload size of 512 bytes. The results are averages over 25 scenarios, each with different traffic and movement patterns. All plots in this section show 95 % confidence intervals.

We then examine the impact of additional load caused by additional connections to present and non-present nodes and by the present detection schemes.

4.3.1 Algorithm Parametrization

In simulations with medium-sized networks of 200 nodes, we re-used the soft state Bloom filter simulation results described in Section 3.3 and carried out equivalent phase-based approach simulations. Specifically, we choose the same Bloom filter parameters of $m = 1024$, $k = 4$, which produces the false positive rate is about 0.086. Since networks of these dimensions do typically never exceed a diameter of ten hops, four bits per soft state Bloom filter position was used in the soft state approach. In the phase-based approach, we set both the phase length and the TTL parameter of the local soft state Bloom filters to 10. As a result, the sizes of the aggregates in the beacons are 512 bytes for the soft state Bloom filter protocol and 128 bytes for the phase-based approach.

In the case of large-sized networks of 400 nodes, to have the same false positive rate of 0.086 as in medium-sized simulations, we used the Bloom filter parameters $m = 2048$ and $k = 4$ in both approaches. The soft state Bloom filter approach uses five bits per position to be able to detect nodes at distances of more than 15 hops in these bigger networks. For the phase-based approach, phase length and TTL are set to 25. As a result, the sizes of the aggregates used by the soft state and phase-based approaches are 1280 bytes and 256 bytes, respectively.

4.3.2 Worst case

The worst case for the presence detection service is a situation where all network nodes are permanently present. In that case, the network will not profit from presence detection services, while they still consume bandwidth.

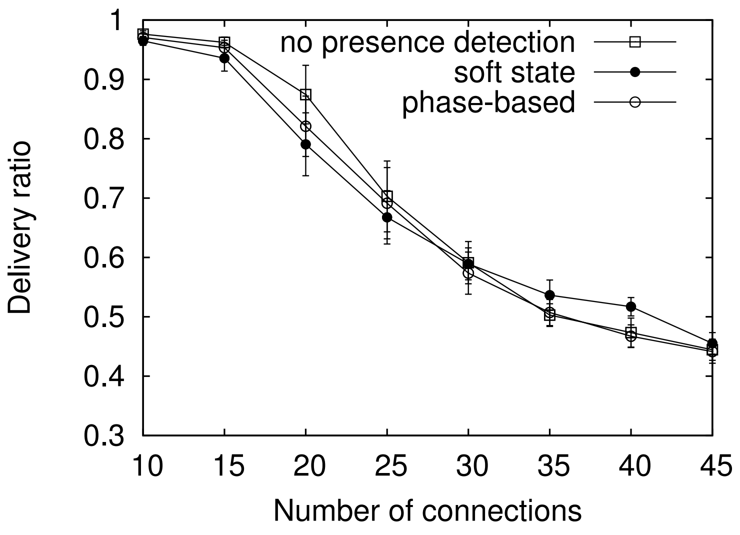
Figure 4.5 shows the performance of networks of medium and large size, in terms of the packet delivery ratio with 95% confidence intervals, for an increasing number of connections. Because all nodes are always present, we may not expect any benefit from presence detection. But we can assess the impact caused by the additional beaconing traffic in this worst case situation. For medium-sized networks, Figure 4.5(a) shows little negative impact of both approaches, the network performance in all three cases is roughly the same. The cost of both presence detection approach is so small that it does not add significantly to the congestion level of the network. However, in large-size networks (Figure 4.5(b)), the impact of the soft state approach becomes quite pronounced, due to the large beacon size.

4.3.3 Absent nodes

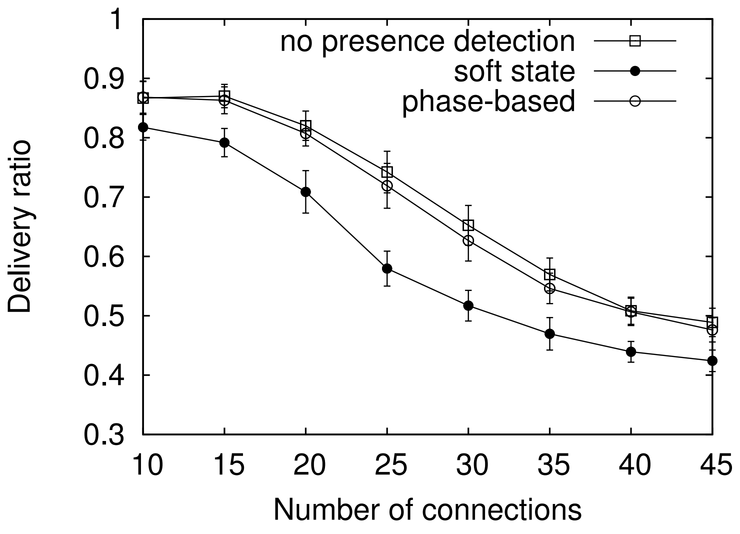
Now we keep the number of "working" connections to present, available nodes fixed at 25. Figure 4.6 shows the effects on the network performance, when the number of additional connection attempts to non-present nodes increases. It can be seen that, without presence detection, an increasing number of connection attempts to non-present nodes severely deteriorates the network performance. The performance with presence detection, on the other hand, does not show significant negative effects, no matter how many connections to non-present nodes are attempted. In medium-sized networks, the performance of both approaches is stable at a high level. For large networks the soft state approach's delivery ratio is 30 % less that the phase-based approach's. Again, the reason for this difference is the bandwidth spent on beaconing.

We compared the average bandwidth spent by a node for routing packets, data packets, and presence detection beacons, for the cases with and without presence detection. Without presence detection the bandwidth used for routing packets quickly increases, while the available bandwidth for application data decreases. Only a small portion of the bandwidth is actually effectively used. For large networks this is shown in Figure 4.8. For medium-sized networks the results are shown in Figure 4.7; the effect is very similar, though.

Both presence detection services are able to block the vast majority of connection attempts to non-present nodes. Thus, with presence detection, the bandwidth usage remains almost constant: with an increasing number of connections to non-present nodes there is only a minor increase in bandwidth used by AODV, caused by false positives. In medium-sized networks the bandwidth cost by the phase-based approach is already significantly lower than that of the soft state approach, a trait that becomes even more pronounced in large networks. In Figure 4.8(b), it is evident that with soft state Bloom

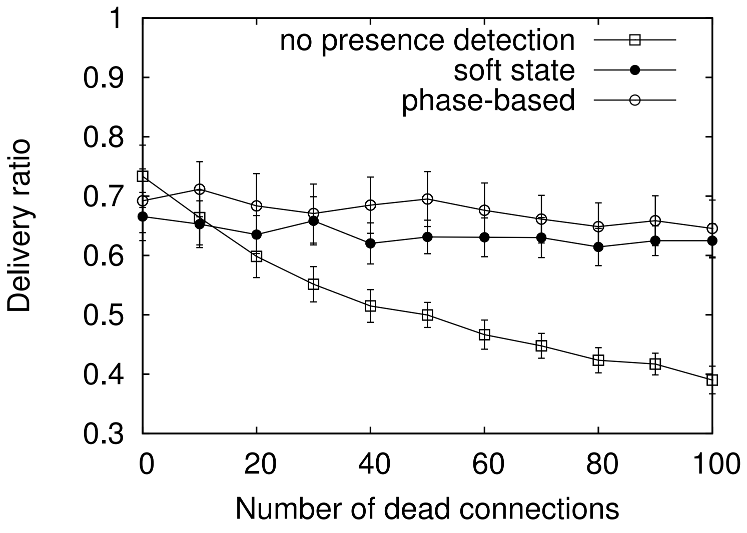


(a) Medium-sized networks.

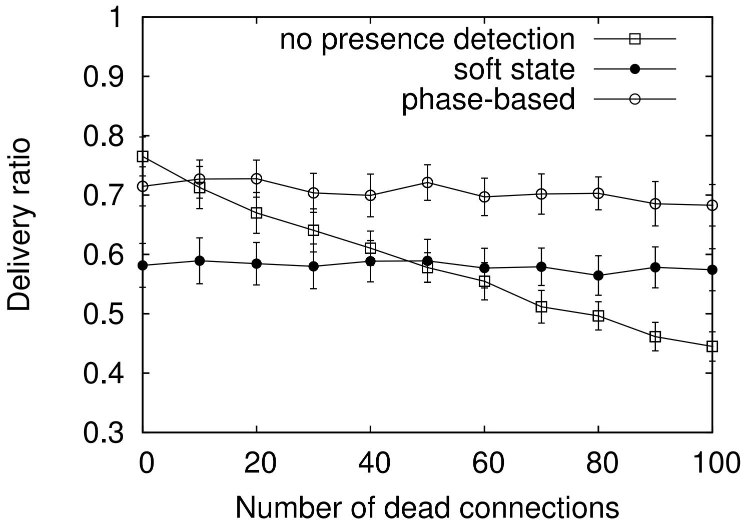


(b) Large-sized networks.

Figure 4.5: Packet delivery ratio as the number of connections increases, in a worst-case scenario.



(a) Medium-sized networks.



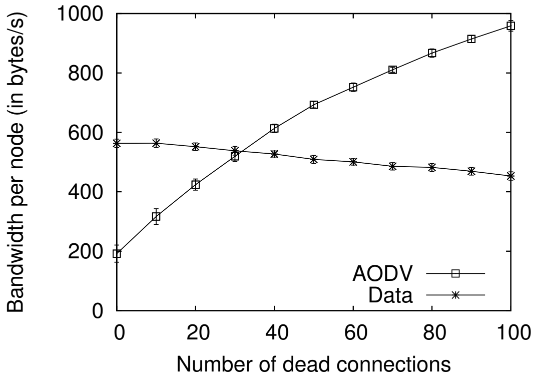
(b) Large-sized networks.

Figure 4.6: Packet delivery ratio as the number of connection attempts to non-present nodes increases.

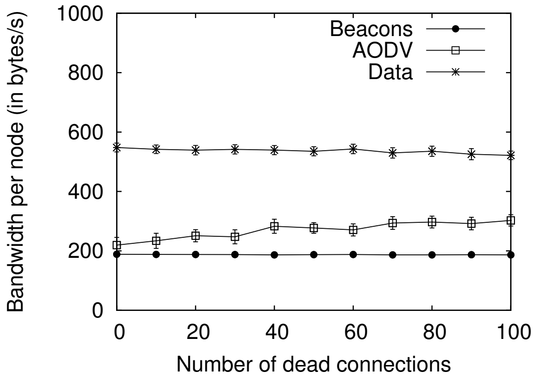
filter based presence detection, the average bandwidth each node spends on beaconing is in fact the largest part of the three types of traffic. This limits the network performance in terms of packet delivery ratio, as observed in Figure 4.6. The phase-based approach, on the other hand, maintains a low overhead for beacon information and can therefore deliver a much higher percentage of the data packets to their destination.

4.4 Chapter Summary

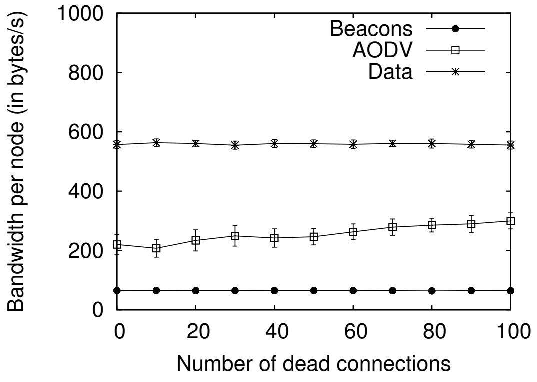
In this chapter we proposed an alternative presence detection scheme for MANETS. In comparison to the work in Chapter 3, it requires significantly less bandwidth for the distribution of presence information. The key idea is to rely on phases to remove information on nodes that are no longer present in the network. We investigated key aspects of the approach, such as the speed of information propagation, the probability of false positives, and the bandwidth consumption by means of analysis and simulation. To underline the practical benefits that can be obtained by using presence detection, and the advantages as well as disadvantages in comparison to prior work, we also showed simulation results from the same exemplary application of presence detection to the route discovery process of AODV.



(a) Without presence detection.

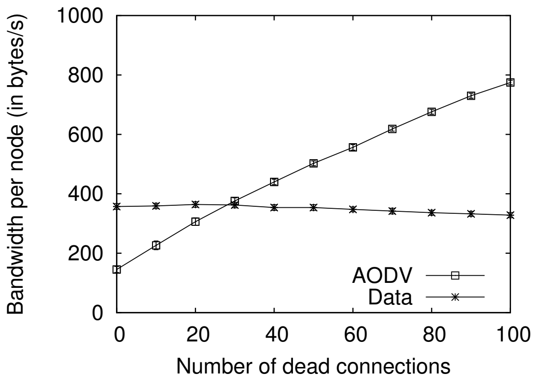


(b) With presence detection.

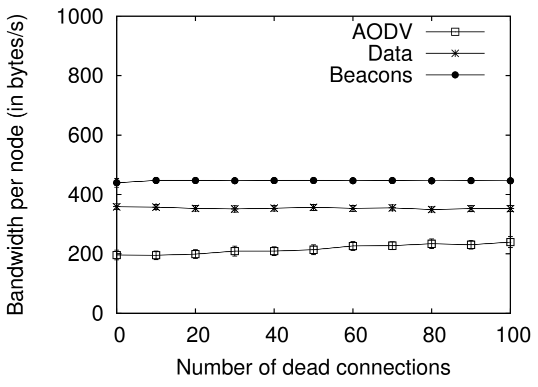


(c) Phase-based presence detection.

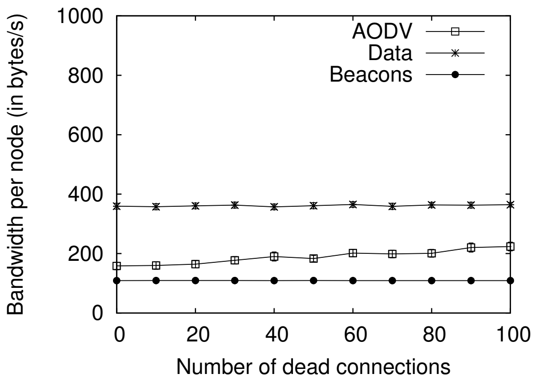
Figure 4.7: Medium-sized networks: bandwidth use per node as the number of connection attempts to non-present nodes increases.



(a) Without presence detection.



(b) Soft state presence detection.



(c) Phase-based presence detection.

Figure 4.8: Large-sized networks: bandwidth use per node as the number of connection attempts to non-present nodes increases.

Chapter 5

Reachability

In mobile ad-hoc networks, information on the presence of nodes can be perceived in two ways: physical locality and communication possibility. The approach we proposed in the previous chapters definitely solve the problem of detecting nodes that physically are in the network. But "seeing" a node being present in that sense does not necessarily mean being able to communicate with the node. That is because a node "sees" others essentially by receiving information from those nodes, which does not require its own information to be received by the other nodes. For example, if the links between two nodes u and v allow data to be transferred from u to v but not the other way, and if there is no other path from v to u, then v can see u but cannot communicate with u. Unlike in simulations, this situation of uni-directional links are not uncommon in real-world situations [MD02]. In our test-bed experiments, we have seen situations lasting dozens of seconds, in which all out-going packets from a node were delayed while the node was able to receive packets without any problems. The issue matters even more in the case of unicast communication when a bi-directional link is required for every hop along a communication path.

In this chapter, we extend the problem further to address the question of reachability, and propose an algorithm that solves the problem. The idea is similar to the presence detection approach described in Chapter 3, except that only bidirectional links are registered while uni-directional links are ignored, since communication is feasible across the former but not the latter. Upon receiving a beacon, a node would first check if the sender has recently received its beacons, i.e. whether communication in the other direction was possible. The beacon will be accepted and further processed as normal if the answer is yes, otherwise, it will be discarded. This way, presence information is disseminated across the network via bi-directional links only, and thus has the stricter meanings of 'reachability information'.

The solution we propose in this chapter is applicable both to the soft-state approach presented in Chapter 3 as well as the phase-based approach presented in Chapter 4. Hence, in this chapter, we focus on the soft state approach only. Details on adapting the phase-based approach to solve the problem of reachability detection can be found in Appendix B.

This chapter also describes our real-world experiments with the proposed solution implemented as a feature of an instant messenger application. The experiment results demonstrate that the algorithm works as expected.

5.1 Algorithm

We use Bloom filters to store reachability information in the same way as presence information in Chapter 3. To enable a node u to recognize beacons from neighbors who have received beacons from u, these neighbors must somehow store the information “I have received beacons from u” in their outgoing beacons. In other words, beacons broadcasted from a node must contain information about all the neighbors that the node has recently heard from. Standard approaches such as OLSR [JMC$^+$01], a link-state based routing protocol, store this information in a list of addresses of the neighbors. Their HELLO messages, as a result, may become very large in dense networks. In our problem, we need some lightweight data structure. Again, standard Bloom filters serve the purpose well. Each node will have such a Bloom filter stored locally. We call this a ‘neighbor aggregate’. Whenever a node receives a beacon, it will hash the sender address to the filter, which is to be included in the next beacon it broadcasts. As a result, by looking up its own address in an incoming beacon’s neighbor aggregate, a node can see whether the beacon’s sender has recently received its beacons.

Neighbor aggregate structure

We define the structure of the neighbor aggregate as a α-bit standard Bloom filter $nb_1, nb_2, \ldots, nb_\alpha$, whose entries are initialized to zero.

The size α and the set of hash functions to be used on neighbor aggregates must be agreed upon in advance by all nodes in the network, so that neighbor aggregates sent by a node can be understood by its neighbors.

Timeout and refresh

This is the operation performed periodically at each interval when a node announces its presence to its neighbors and starts a new beacon interval. This is also the time when the node tells its neighbors whether it heard anything from them during the last interval. With such information, the neighbors can see whether the links were bi-directional. Also, the local neighbor aggregates must be emptied periodically so as to remove the addresses of non-recently heard neighbors. The more frequent this 'refreshing', the stricter the meanings of 'recently'. We choose the strictest meanings of 'recently' as 'during the last beacon interval', thus, nodes empty the local neighbor aggregates at the beginning of each interval, i.e. in the timeout operation.

Modifications to the soft state or phase-base approach's algorithm include adding the neighbor aggregate to the beacon to be sent, then emptying the aggregate.

The soft state algorithm for timeout and refresh now is:

1. Increment each a_i by one, if it is not already at the limit of $2^l - 1$.
2. Refresh the information about the node's own presence, by setting $a_{h_j(ID)} = 0$ for all $j = 1, \ldots, k$, where ID is the ID of the local node.
3. Broadcast (a, nb) to the neighbors.
4. Empty nb.

Merge operation

This procedure is to be carried out when a node receives beacon (a', nb'). It registers the sender to the local neighbor aggregate and decides whether it should proceed with the merging of reachability Bloom filters.

1. Hash the beacon's sender address to the local neighbor aggregate nb.
2. Look up the node's own address in the beacon's neighbor aggregate nb'.
3. If the answer is negative, discard the beacon.
4. Otherwise, merge a' to a using position-wise minimum operation, i.e. set each a_i to the minimum of a_i and a'_i.

The addresses to be hashed into neighbor aggregates could be either IP or MAC addresses, which are readlily available in packet headers. To be consistent, nodes should agree upon which type of address to be used, although choosing one of them is simply a matter of convenience.

The look-up operation is performed solely on the main aggregate that stores reachability information. Therefore, it is the same as described in Section 3.1.2.

5.2 Evaluation

In this section, we assess the cost of the proposed reachability detection service in comparison with soft-state presence detection. Specifically, we will look at the aspects of bandwidth cost, false positive rate, and speed of information propagation.

5.2.1 Bandwidth

Compared to soft state presence detection, the reachability detection service needs more bandwidth to include neighbor aggregates in beacons. Though α needs to be chosen large enough to accommodate the expected number nodes in one-hop range, this value could be very small. For example, a 256-bit aggregate takes up only 32 bytes but can store 50 neighbors with a false positive rate of about 0.10. Compared to the size of the main soft-state Bloom filter taking up the most space of the beacon, this extra bandwidth requirement is rather insignificant and can be ignored.

5.2.2 False Positive Rate

Since two Bloom filters are involved in our reachability detection algorithm, there are two types of Bloom filter related false positives.

The first type are those of the soft-state reachability aggregate, which is the same as that in soft-state presence detection (see Section 3.2.1).

The second type are those in the neighbor aggregate, which is a standard Bloom filter as discussed in Section 4.2.1. Similar to the case of present information Bloom filters, false positives in neighbor Bloom filters are acceptable. The cost of such a false positive is that a uni-directional link is mistaken as bi-direction and as a result some present but out-of-reach node might be wrongly seen as reachable.

5.2.3 Speed of Information Propagation

The reachability detection service propagates information in the same manner as in the soft state approach presented in Chapter 3. Thus, it can be expected that the dissemination time should be the same as for the soft state approach, plus an additional delay of up to one beaconing interval for arriving nodes to receive beacons from neighbors for the first time before being accepted by them as reachable neighbors.

5.3 Real-world Experiments

In order to verify how well the algorithm works, we carried out a number of real-world experiments and compared the algorithm's detection results with the nodes' actual ability to communicate with one another. For all these experiments, networks with moving nodes were chosen because dynamic networks frequently show the gradual changes of connectivity, which result in asymmetrical links.

Note that a positive answer of our reachability detection algorithm to a query concerning a pair of nodes is based on actual two-way communication along a path connecting these two nodes. So by definition, a positive answer means true communication ability, except that this type of 'communication' was delayed at each hop along the path for significantly longer than in normal communication. In a sense, one could say that, in an ideal situation when packets are lost only due to distance or obstacles, our algorithm gives correct but delayed information about the past. We now wanted to compare them with information that is more recent.

5.3.1 Method of Assessment

One of the most interesting problems that arose was how to determine the baseline with which the algorithm's result can be compared, i.e. how to know for certain or almost certain if a pair of nodes can communicate with each other at a specific point in time.

In a network where distances between nodes change over time, it is very hard to determine whether a link works at a point in time without any packets actually being received or lost around that time. The sole information on the exact physical locations of the nodes at that time is not enough for the purpose, either. There are a number of other factors, such as someone opens a door, that would contribute to the success or failure of the would-be-sent packet.

Hence, we decided to test the 'reality' in real-time by sending actual packets to the node in question. Another question arose: how to send packets over multiple hops across a dynamic network. Static routing is not an answer since routes change over time due to nodes' movement. AODV, a well-known routing protocol for ad-hoc networks, is not a good choice, either. [1] In real world experiments, we often experienced situations when communication using broadcast ran smoothly but neighboring nodes could not ping each other [2] or AODV failed to re-establish a broken route despite the fact that all nodes were well within radio range of each other and had stopped moving. Moreover, these situations of false negatives lasted too long to be considered simply as noise of the would-be baseline. Figure 5.1 depicts one of our experiments with AODV, in which three nodes 1, 2, 3 were static and well-connected and node 0 was mobile. In every run, the route from node 0 to node 1 broke whenever node 0 reached position A in the diagram and never managed to recover despite node 0 staying at A in waiting for a long time. Meanwhile, communication between node 0 and node 1 using simple flooding worked well when node 0 continued in that direction up to position B, which is so far away from the node chain that it makes AODV's failure at point A unacceptable. Failure of routing using AODV while communication using broadcast runs smoothly is not equal to the failure of communication in general.

So, instead of AODV, we decided to use simple flooding to send probe messages to all nodes in the network discovering its current topology in terms of connectivity. Note that packet flooding in small scarce networks do not use up so much bandwidth that it would disrupt other types of communication in the network. In this simple flooding mechanism, upon hearing a probe for the first time, the receiver node immediately forwards that same probe message, and it will ignore later receptions of the probe. Although the bandwidth spent on beaconing and probing is rather low given the small size of the network, interference or rare incidents of collision are still possible and could result in some probe messages being lost. In order to ensure that such noise does not have significant effects on the overall results, each probe was sent three times from each node. During the experiments, each node logged the probes it received, and the graph formed by successful messages of one probing round is the snapshot of the network's connectivity at the time when the probing round was carried out. If the graph contains

[1] For these experiments, we used AODV-UU 0.9.5, the latest AODV implementation by CoreSoftware, Uppsala University.[aod]

[2] Such incidents show no pattern in terms of machines or network configurations and cannot be repeated. They can last very long but cannot be stopped at will. Wireshark logs show either of two situations: the pinger received ARP replies from the pingee but the pingee never heard any of the pinger's ICMPs that follow; the pinger never heard any of the pingee's ARP replies and kept sending ARP requests.

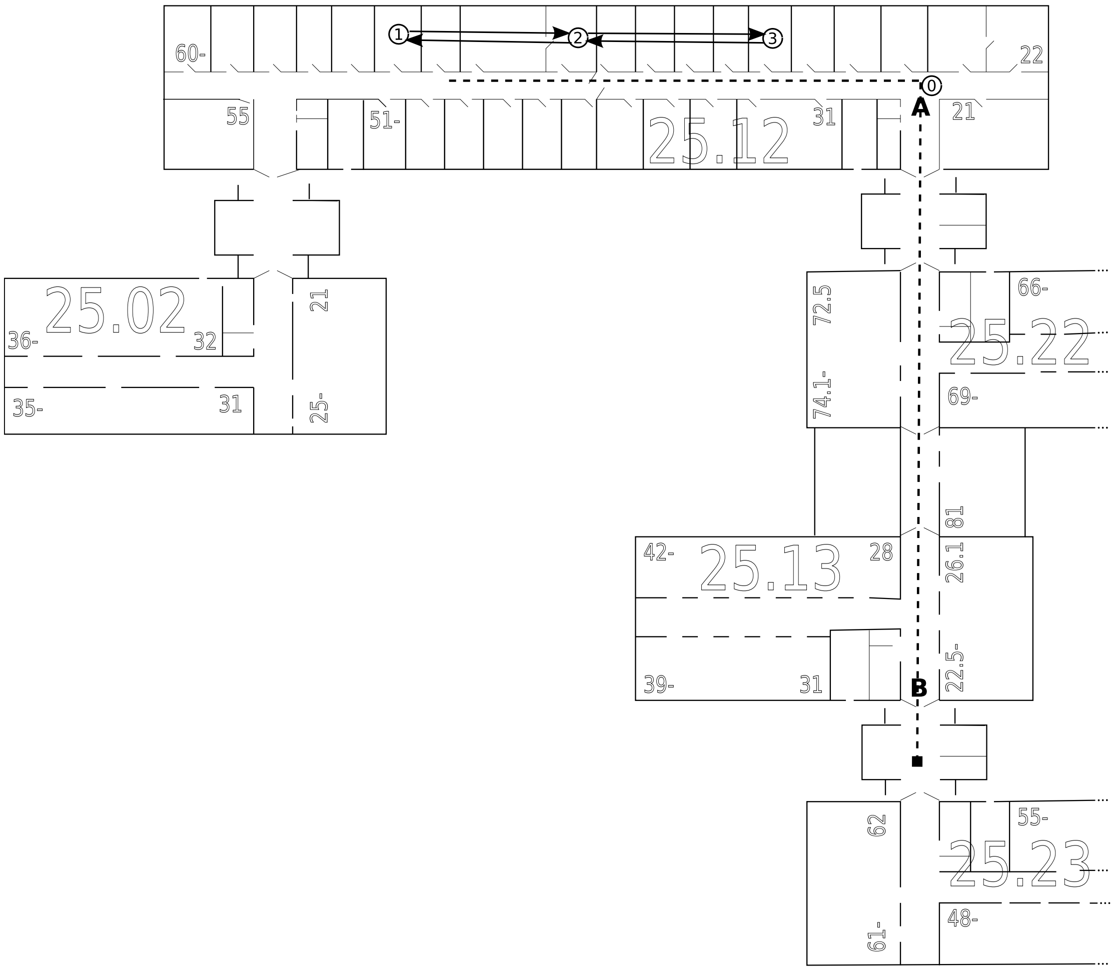


Figure 5.1: Experiments on multihop two-way communication with node 0. AODV worked up to point A. Simple flooding worked up to point B.

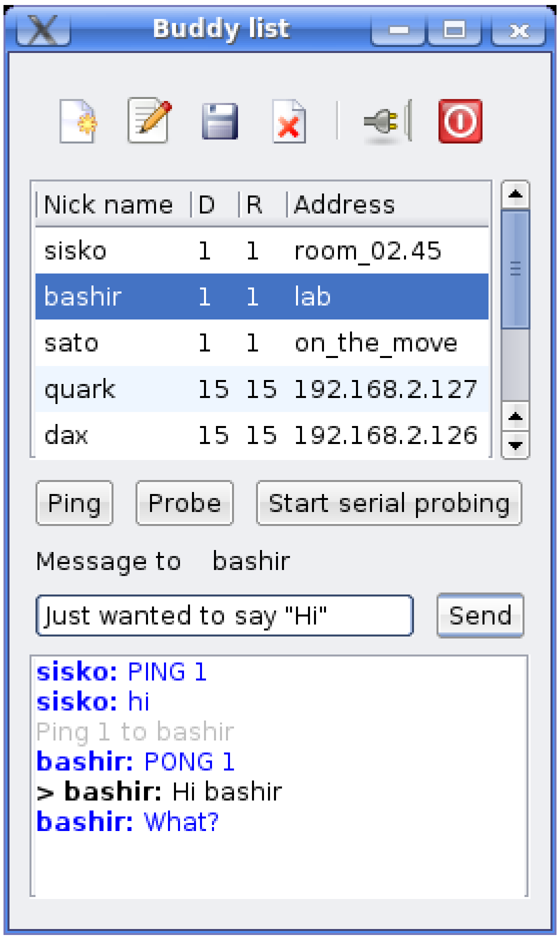
a bi-directional path to the destination node, we say that communication with the node is possible at that point in time.

The experiments were carried out in the computer science and adjacent buildings. The network consisted of four nodes: three Lenovo ThinkPad X61s (nodes 0, 1, 3) and one IBM ThinkPad X40 (node 2) with built-in wireless card, all running Kubuntu Linux operating system. Unlike the Zaurus SL-6000 PDAs that participated in Section 3.4 experiments, these systems provide much better compatibility and support for the programming library packages needed for the user-friendly application software that we used in these experiments. Three nodes (1, 2, 3) were static. They form a chain along the corridor of the computer science building (see examples in Figures 5.3(a) and 5.1). The fourth node (0) was carried by a person walking along the chain repeatedly leaving and re-entering the network at either end of it. Instead of having all the nodes moved around in different directions as in Section 3.4.2, by limiting mobility to one node, we were able to better anticipate node behaviors as well as link status so as to better relate node movements to changes in connectivity while assessing algorithm performance. During the experiments, a probing round is triggered by node 1 every 5 seconds. This rate was chosen high enough to get frequent snapshots of the constantly changing network, and, at the same time, low enough not to significantly disrupt beacons of the presence detection service whose rate was one beacon per second.

5.3.2 xWhoisthere Software

We implemented `xWhoisthere` (See Figure 5.2) as an instant messenger that provides presence and reachability detection service. Apart from the main purpose of serving our experiments in this chapter, the software can be readily deployed as a group-aware application that supports collaborations among group members.

Similar to `whoisthere` used in Section 3.4 experiments, each `xWhoisthere` user has an unique ID, or nick name, known to the others, which is to be hashed when someone looks up the user. Unlike `whoisthere`, `xWhoisthere` provides a graphical interface allowing users to monitor presence status and reachability of other users, as well as to send instant messages to one another. From the perspective of presence awareness, running on devices within a mobile ad-hoc network, `xWhoisthere` works perfectly as an Inter-Personal Awareness Device (IPAD) as proposed in [HFW99] but are more powerful than the Hummingbird described there, as `xWhoisthere` works over multiple hops. As an instant messenger, `xWhoisthere` requires neither an Internet connection



Nick name	D	R	Address
sisko	1	1	room_02.45
bashir	1	1	lab
sato	1	1	on_the_move
quark	15	15	192.168.2.127
dax	15	15	192.168.2.126

Message to bashir

Just wanted to say "Hi"

```
sisko: PING 1
sisko: hi
Ping 1 to bashir
bashir: PONG 1
> bashir: Hi bashir
bashir: What?
```

Figure 5.2: xWhoisthere screenshot.

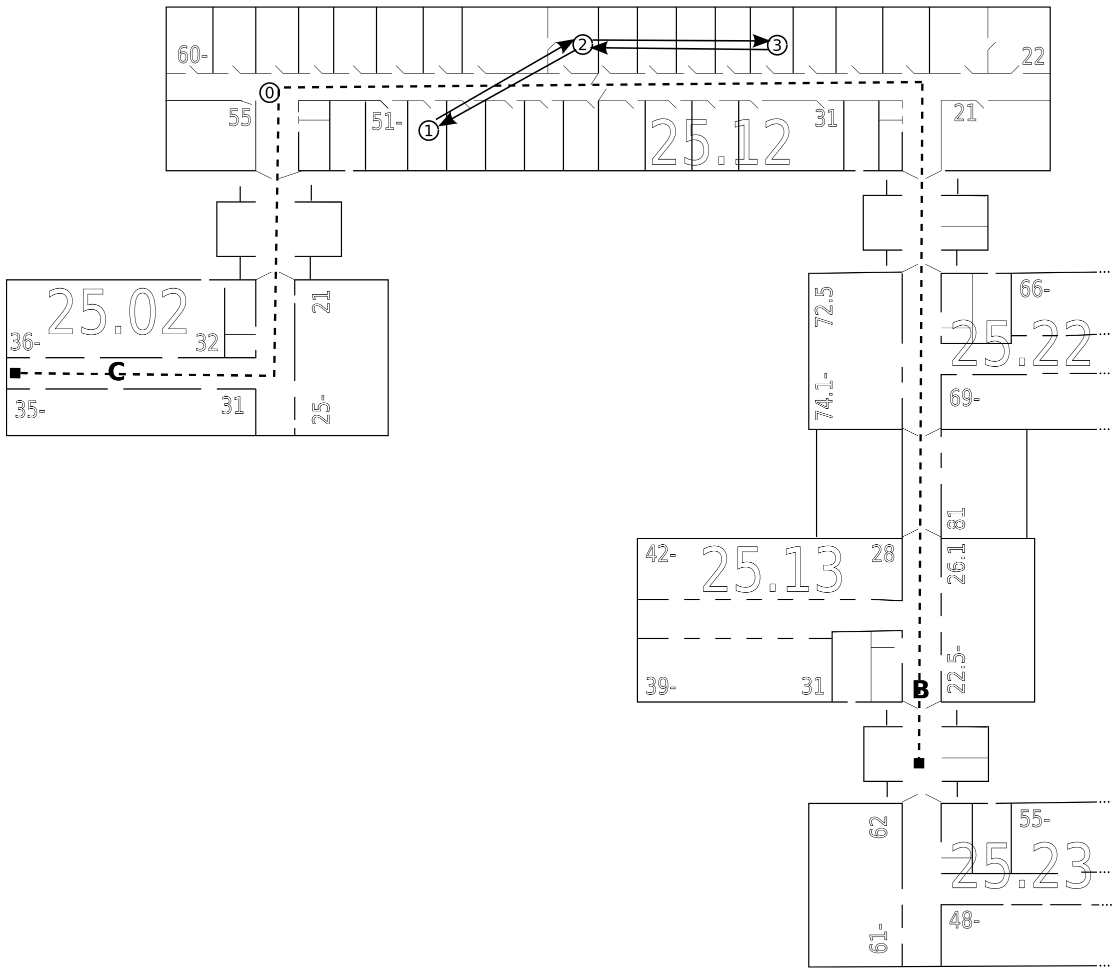
nor a network infrastructure. And most importantly with regards to the our problem of reachability, `xWhoisthere` constantly provides its user with estimates of hop-distances to other nodes, both two-way paths, in the case of reachability, and one-way paths, in the case of presence information. To serve the purpose of assessing our approach to detect reachability, the `xWhoisthere` used in experiments has special features including user-initiated network topology probing function described above, and automatic logging of all information that is important to our analysis, such as messages sent and received, for later off-line processing. Of course, these special-purpose features are otherwise useless and to be removed from non-experimental versions of `xWhoisthere`.

As specified by the algorithm, beacons are exchanged using broadcasts. Instant messages as well as network topology probes could be sent using either simple controlled flooding or unicasts together with AODV routing, depending on which version of `xWhoisthere` is in use. However, due to the misleadingly poor performance of unicasts together with AODV routing, as discussed in Section 5.3.1, we used controlled flooding for our main experiments whose results will be described in the next section. `xWhoisthere` was implemented using C++ together with GTK+.

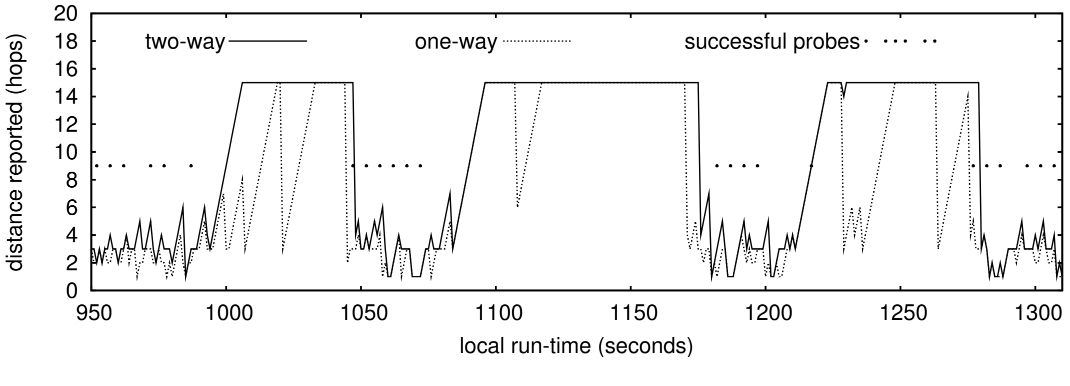
5.3.3 Experimental Results

We carried out a number of experiments with scenarios as described in Section 5.3.1 and obtained outcomes that are similar to each other. Figure 5.3 depicts one of the experiments and results of one segment of it. The outcome shown in the Figure 5.3(b) is data reported from node 3 while node 0 moved around the area in building 25.02 (position C in Figure 5.3(a)). This segment is typical of those interesting periods when the node 0 was close to the edge of the network moving continuously in and out of it. Depicted in Figure 5.3(b) is the information available at node 3 on: the reachability of node 0 (two-way hop-distance); presence information of node 0 (one-way distance); and probes covering both the nodes, i.e. successful two-way communication between the pair. High values of estimated distances mean node 0 is going (or already) out of touch, and low values mean presence or possibility of communication.

Given the fact that some probes can be lost due to collisions, which is a type of noise in logged data, we can see in the figure that periods of successful probes are consistent with the pattern of two-way distance: heading upwards to the maximal value after the end of a series of successful probes, and sharply going down around the time when the next series starts.



(a) Network layout and movement of node 0.



(b) Hop-distance from node 3 to node 0, reported at node 3 while node 0 moved back and forth around point C.

Figure 5.3: Sample experiment.

On the other hand, estimates of one-way distances is not as consistent to our baseline of topology probes. Node 0 was sometimes reported as being present in the middle of a non-communicable period. As the results of a presence detection algorithm, it is perfectly correct with regards to the meaning of 'present' as being physically in the network. However, it demonstrates that correct results of presence detection can be Bloom-filter-unrelated false positive answers to the problem of detecting reachability of nodes. Observing short periods of 'false positives' given by presence detection algorithm in Figure 5.3(b), one could argue that such periods are ignorable in applications. Nevertheless, we have learned from our experience with real-world experiments that network nodes can be arranged in a way that results in much longer and thus important periods of false positive answers.

There is an interesting observation in the case of a node leaving the network. In theory, there is a delay before `xWhoisthere` officially concludes that a node is no longer reachable, i.e. the time before the decaying estimated two-way distance to the node reaches the predefined threshold. In practice, however, a user could 'see' a node disappearing well before that conclusion of `xWhoisthere` by looking at recent changes of the estimated distance to the node. When the estimated two-way hop-distance to a node steadily increments with every beacon interval during a number of intervals, there is a very high chance that there is no longer a path to the node, or, in other words, the node is no longer reachable. With short beacon intervals, node mobility that sustains one communication path or another normally does not result in such fast and steady increase of hop distances.

5.4 Chapter Summary

In this chapter, we addressed the problem of detecting node reachability, i.e. whether two-way communication to a node is possible. We solved the problem using an information exchange scheme that allows presence information to be disseminated over bi-directional communication links only, thus ensuring two-way communication ability. This mechanism is adaptable to both soft state and phase-based approaches to presence detection.

We have also implemented `xWhoisthere`, an instant messaging application for mobile ad-hoc networks that detects presence as well as reachability of friends. This software can be readily deployed as a group-aware application that supports collaborations among group members.

Our real-world experiments using `xWhoisthere` demonstrates that the proposed mechanism to detect two-way communication ability works as expected. Observations made during the course of these experiments give rise to an idea for further research: the quality of the would-be communication to a node could be predicted from the pattern of changes in estimates of the hop-distance to the node.

Chapter 6

Conclusion

In this thesis, we investigated the problem of detecting whether specific nodes are present in a mobile ad-hoc network and provided scalable lightweight solutions to the problem.

Chapter 2 surveys research directions in the area of obtaining information about the set of nodes that are active in the network, dealing more or less directly with the presence of nodes in wireless networks.

In Chapter 3, we introduced the soft state solution to the presence detection problem, which also provides estimates of hop-distances between nodes. Our algorithm aggregates presence information using soft state Bloom filters, a variant of the well-known Bloom filter that allows for an efficient aggregation of presence information as well as the removal of old information. The aggregation comes at the cost of an adjustable amount of false positives, while it guarantees the absence of false negatives.

Also in Chapter 3, we investigated several key aspects of the soft state approach, such as the speed of information propagation, the probability of false positives, and the bandwidth consumption by means of analysis and simulation. In an application of our presence detection service to the route discovery process of AODV, our simulation results underline the practical benefits that can be obtained by using presence detection: substantial network performance gains are possible as unsuccessful route discovery attempts can be avoided while the additional network load for the presence detection beacons is very limited. Test-bed experiments on PDAs also show that the presence detection works as expected in various environments. In addition, the privacy issue was addressed and a mechanism was proposed to prevent spying on network nodes.

With the results of this chapter, we are convinced that presence detection is an important building block for wireless multihop networks. Service discovery, routing, and location services are just three examples where presence detection is vital to ensure an acceptable system performance if the presence of nodes changes over time.

Chapter 4 described an alternative approach to solve the problem of presence detection in MANETs: the phase-based presence detection scheme. Using standard Bloom filters in combination with a loose synchronization mechanism, this approach improves upon the soft state approach by significantly reducing the overhead that is required to decide whether a node is present. The key idea is to rely on phases to remove information on nodes that are no longer present in the network. Similarly to the previous chapter, key aspects of the phase-based approach, such as the speed of information propagation, the probability of false positives, and the bandwidth consumption were investigated by means of analysis and simulation in comparison with the soft state approach. Being more lightweight than the soft state presence detection, the phase-based approach shows more benefits in terms of network performance in large-sized networks, and hence would be a better choice for presence detection services in such networks.

In Chapter 5, we investigated the presence detection problem in the stricter sense, i.e. being present means communication ability. We proposed an algorithm to adapt the previously discussed approaches to solve the new problem. `xWhoisthere`, an instant messenger application integrated with a presence and reachability detection service, was developed to demonstrate that our solution works well in real-world environments.

Although we now conclude our thesis on the presence detection of nodes, observations made during the course of the experiments with our presence detection algorithms gave rise to several ideas for further research. First, the pattern of changes in hop-distance estimates given by the soft state approach could be useful in guessing the quality of the would-be communication to nodes. For example, small oscillations of estimated distance to a node could mean the stability of link quality, while a steady increase of hop-distance is the very likely sign of a no-longer-present node. Second, estimates of hop-distances provided by the soft state approach opens the possibility of communication without relying on a routing protocol, in which packets are directed using estimated distances to their destinations.

Appendix

Appendix A

Propagation Delay Calculations

In this appendix, we detail the calculations on the propagation delay of the presence information over two hops. We assume a constant network density $\rho > 0$ with randomly placed, equidistributed nodes, and a fixed communication radius $r > 0$. Let u and x be two randomly placed nodes. Under the condition that their distance δ is in $]r, 2r]$—a necessary condition for u and x being two hops apart—, it follows from basic geometric probability calculations that the probability density function of their distance is

$$f(\delta \mid r < \delta \leq 2r) = \frac{2}{3r^2}\delta. \tag{A.1}$$

The area in which potential forwarders are located is the lens-shaped intersection of the one-hop neighborhoods of u and x. The size of this area is

$$A(\delta) = 2r^2 \arccos \frac{\delta}{2r} - \delta\sqrt{r^2 - \frac{\delta^2}{4}}. \tag{A.2}$$

The number of nodes in this area is Poisson distributed. Thus, we obtain the probability of n nodes being in the intersection of the one-hop neighborhoods of u and x, if their distance is in $]r, 2r]$:

$$P(n \text{ forwarders}) = \int_{r}^{2r} \frac{2\delta(\rho A(\delta))^n e^{-\rho A(\delta)}}{3r^2 n!}\, d\delta. \tag{A.3}$$

Two nodes with distance in $]r, 2r]$ are two-hop neighbors if and only if there is at least one node in the intersection area of their one-hop neighborhoods. Then the probability of n potential forwarders, $n > 0$, is

$$P(n \text{ forwarders} \mid 2 \text{ hops}) = \frac{P(n \text{ forwarders})}{1 - P(0 \text{ forwarders})}. \tag{A.4}$$

We now turn towards the expected time until the first forwarder's broadcasting intervals expires, given that the offsets are independent. The time until the first forwarder broadcasts can be modelled as the minimum of n pairwise independent random variables which are all equidistributed in $[0, B[$, for beaconing interval B. The probability density of the minimum (in $[0, B[$) is

$$f_n(t) = \frac{n}{B}\left(1 - \frac{t}{B}\right)^{n-1}. \tag{A.5}$$

This can be shown by induction over n, exploiting the fact that $\min\{X_1, \dots, X_{n+1}\} = \min\{\min\{X_1, \dots, X_n\}, X_{n+1}\}$. Then the expected time until the first node sends a beacon is

$$\int\limits_0^B \tau \frac{n}{B}\left(1 - \frac{\tau}{B}\right)^{n-1} d\tau = \frac{B}{n+1}. \tag{A.6}$$

By combining (A.4) and (A.6) it results that the expected time for the second hop delay for any node pair u, x with two hops distance is

$$\begin{aligned}
&\sum_{n=1}^{\infty} \frac{B}{n+1} \cdot \frac{P(n \text{ nodes})}{1 - P(0 \text{ nodes})} \\
&= \sum_{n=1}^{\infty} \frac{B}{n+1} \cdot \frac{\int_r^{2r} \frac{2\delta(\rho A(\delta))^n e^{-\rho A(\delta)}}{3r^2 n!}\, d\delta}{1 - \int_r^{2r} \frac{2\delta e^{-\rho A(\delta)}}{3r^2}\, d\delta} \\
&= \sum_{n=1}^{\infty} \frac{B \cdot \rho^n}{(n+1)!} \cdot \frac{\int_r^{2r} \delta (A(\delta))^n e^{-\rho A(\delta)}\, d\delta}{\frac{3}{2}r^2 - \int_r^{2r} \delta e^{-\rho A(\delta)}\, d\delta}.
\end{aligned} \tag{A.7}$$

Appendix B

Phase-based Approach to Reachability Detection

In this appendix, we briefly describe how to adapt the phase-based algorithm of Chapter 4 to solve the problem of detecting reachability of nodes.

Out of three operations, two need modifications. The query operation is not affected.

In the timeout and refresh operation, the local neighbor aggregate needs to be included in the beacon to be broadcasted; this aggregate is then to be emptied so as to be ready to record neighbors during the next interval. The adapted algorithm for the timeout and refresh operation is given in Algorithm B.0.1, in which a, p, c and nb are the local reachability information aggregate, phase ID, counter and neighbor aggregate, respectively.

```
Decay the entries in the local soft state Bloom filter $b$.
**if** $c \geq C$ **then** {$c$ is at the threshold $C$}
  $p \leftarrow p + 1$
  $c \leftarrow 0$
  Empty $a$ and add own presence information to $a$
**end if**
Broadcast ($a$, $p$, $c$, $nb$) to the neighbors.
$c \leftarrow c + 1$
Empty $nb$.
```

Algorithm B.0.1: Timeout and refresh

The merge operation is performed upon receival of a beacon. This operation requires two additional tasks. First, the sender address of the incoming beacon needs to be registered to the local neighbor aggregate. Second, the node must look up its own address in the received neighbor aggregate to see if the beacon sender did receive its last beacon, then reject the beacon if the look-up gives negative answer. Denoting the data in the incoming beacon as (a', p', c', nb'), Algorithm B.0.2 is the adapted merge algorithm.

```
Add sender address to $nb$
Look up own address in $nb'$
**if** the answer is negative **then** {the sender did not receive my last beacon}
    {do nothing}
**else if** $p > p'$ **then** {the sender is less advanced}
    {do nothing}
**else if** $p = p'$ **then** {both are in the same phase}
    $a \leftarrow$ merge_bloom_filters($a, a'$)
    $b \leftarrow$ merge_bf_into_softstate_bf($b, a'$)
    $c \leftarrow \max\{c, c'\}$
**else** {the sender is more advanced}
    $a \leftarrow a'$
    add own presence information to $a$
    $b \leftarrow$ merge_bf_into_softstate_bf($b, a'$)
    $c \leftarrow c'$
    $p \leftarrow p'$
**end if**
```

Algorithm B.0.2: Merge operation

Bibliography

Own Publications

[TSM07] Thi Minh Chau Tran, Björn Scheuermann, and Martin Mauve. Detecting the presence of nodes in MANETs. In *CHANTS '07: Proceedings of the 3rd ACM MobiCom Workshop on Challenged Networks*, pages 43–50, September 2007.

[TSM09a] Thi Minh Chau Tran, Björn Scheuermann, and Martin Mauve. Lightweight detection of node presence in MANETs. *Elsevier Ad Hoc Networks*, 7(7):1386–1399, 2009.

[TSM09b] Thi Minh Chau Tran, Björn Scheuermann, and Martin Mauve. Node presence detection with reduced overhead. In *WONS '09: Proceedings of the 6th Annual Conference on Wireless On-demand Network Systems and Services*, pages 37–44, February 2009.

Other References

[AK04] Ian F. Akyildiz and Ismail H. Kasimoglu. Wireless sensor and actor networks: research challenges. *Ad Hoc Networks*, 2(4):351–367, 2004.

[aod] AODV-UU, version 0.9.5. Online, `http://core.it.uu.se/core/index.php/AODV-UU`.

[Blo70] Burton H. Bloom. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Communications of the ACM*, 3(7):422–426, July 1970.

[CDD08] T. Clausen, C. Dearlove, and J. Dean. Manet neighborhood discovery protocol (NHDP), 2008. draft-ietf-manet-nhdp-07.

[CDM07] Landon P. Cox, Angela Dalton, and Varun Marupadi. SmokeScreen: Flexible Privacy Controls for Presence-sharing. In *MobiSys '07: Proceedings of the 5th International Conference on Mobile Systems, Applications, and Services*, pages 233–245, June 2007.

[CIXU05] Kai Cheng, Mizuho Iwaihara, Limin Xiang, and Kazuo Ushijima. Efficient Web Profiling by Time-Decaying Bloom Filters. *Database Society of Japan Letters*, 4(1):137–140, June 2005.

[CXI+05] Kai Cheng, Limin Xiang, Mizuho Iwaihara, Haiyan Xu, and Mukesh M. Mohania. Time-Decaying Bloom Filters for Data Streams with Skewed Distributions. In *RIDE-SDMA '05: Proceedings of the 15th International Workshop on Research Issues in Data Engineering: Stream Data Mining and Applications*, pages 63–69, April 2005.

[EP05] Nathan Eagle and Alex Pentland. Social Serendipity: Mobilizing Social Software. *IEEE Pervasive Computing*, 4(2):28–34, April 2005.

[FCAB00] Li Fan, Pei Cao, Jussara Almeida, and Andrei Z. Broder. Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol. *IEEE/ACM Transactions on Networking*, 8(3):281–293, June 2000.

[FZL+03] Zhenghua Fu, Petros Zerfos, Haiyun Luo, Songwu Lu, Lixia Zhang, and Mario Gerla. The impact of multihop wireless channel on TCP throughput and loss. In *INFOCOM '03: Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 3, pages 1744–1753, March 2003.

[GH99] Silvia Giordano and Maher Hamdi. Mobility Management: The Virtual Home Region. Technical Report SSC/1999/037, EPFL-ICA, Lausanne, Switzerland, October 1999.

[GJW+06] Robert Gilbert, Kerby Johnson, Shaomei Wu, Ben Y. Zhao, and Haitao Zheng. Location Independent Compact Routing for Wireless Networks. In *MobiShare '06: Proceedings of the 1st International Workshop on Decentralized Resource Sharing in Mobile Computing and Networking*, pages 57–59, July 2006.

[GK00] Piyush Gupta and P. R. Kumar. The Capacity of Wireless Networks. *IEEE Transactions on Information Theory*, 46(2):388–404, March 2000.

[HFW99] Lars Erik Holmquist, Jennica Falk, and Joakim Wigström. Supporting Group Collaboration with Interpersonal Awareness Devices. *Springer Personal and Ubiquitous Computing*, 3(1/2):13–21, March 1999.

[Iwa98] Yukari Iwatani. Love: Japanese Style. Wired News, online, `http://www.wired.com/news/culture/0,1284,12899,00.html`, June 1998.

[JM96] David B. Johnson and David A. Maltz. Dynamic Source Routing in Ad Hoc Wireless Networks. In Tomasz Imielinski and Henry F. Korth, editors, *Mobile Computing*, pages 153–181. Kluwer Academic Publishers, Norwell, MA, USA, January 1996.

[JMC+01] Philippe Jacquet, Paul Mühlethaler, Thomas Clausen, Anis Laouiti, Amir Qayyum, and Laurent Viennot. Optimized Link State Routing Protocol. In *INMIC '01: Proceedings of the 5th IEEE International Multi Topic Conference*, pages 62–68, December 2001.

[KFWM04] Wolfgang Kiess, Holger Füßler, Jörg Widmer, and Martin Mauve. Hierarchical Location Service for Mobile Ad-Hoc Networks. *ACM Mobile Computing and Communications Review*, 8(4):47–58, October 2004.

[KK00] Brad Karp and H. T. Kung. GPSR: Greedy perimeter stateless routing for wireless networks. In *MobiCom '00: Proceedings of the 6th Annual ACM International Conference on Mobile Computing and Networking*, pages 243–254, August 2000.

[KOM08] Wolfgang Kiess, Thomas Ogilvie, and Martin Mauve. The EXC Toolkit for Real-World Experiments with Wireless Multihop Networks. In *EXPONWIRELESS '08: Proceedings of the 3rd Workshop on Advanced Experimental Activities on Wireless Networks Systems*, June 2008.

[LW06] Christoph Lindemann and Oliver P. Waldhorst. Effective Dissemination of Presence Information in Highly Partitioned Mobile Ad Hoc Networks. In *SECON '06: Proceedings of the 3rd IEEE ComSoc Conference on Sensor, Mesh and Ad Hoc Communications and Networks*, September 2006.

[LYKH06] Choonhwa Lee, Sungshick Yoon, Eunsam Kim, and Abdelsalam Helal. An Efficient Service Propagation Scheme for Large-Scale MANETs. In *MPAC '06: Proceedings of the 4th International Workshop on Middleware for Pervasive and Ad-Hoc Computing*, pages 9–12, November 2006.

[MD02] Mahesh K. Marina and Samir R. Das. Routing performance in the presence of unidirectional links in multihop wireless networks. In *MobiHoc '02: Proceedings of the 3rd ACM International Symposium on Mobile Ad Hoc Networking and Computing*, pages 12 – 23, June 2002.

[Mit02] M. Mitzenmacher. Compressed bloom filters. *IEEE ACM Transaction on Networking*, 10:604–612, 2002.

[MJK+00] Robert Morris, John Jannotti, Frans Kaashoek, Jinyang Li, and Douglas S. J. DeCouto. CarNet: A Scalable Ad Hoc Wireless Network System. In *Proceedings of the 9th ACM SIGOPS European Workshop*, pages 61–65, September 2000.

[ns2] The Network Simulator ns-2, version 2.30. Online, `http://www.isi.edu/nsnam/ns/`.

[PR99] Charles E. Perkins and Elizabeth M. Royer. Ad-hoc On-Demand Distance Vector Routing. In *WMCSA '99: Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, pages 90–100, February 1999.

[RK02] Sean C. Rhea and John Kubiatowicz. Probabilistic Location and Routing. In *INFOCOM '02: Proceedings of the 19th Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 3, pages 1248–1257, June 2002.

[Sto99] Ivan Stojmenovic. Home Agent Based Location Update and Destination Search Schemes in Ad Hoc Wireless Networks. Technical Report TR-99-10, University of Ottawa, September 1999.

[TMRL02] Michael Terry, Elizabeth D. Mynatt, Kathy Ryall, and Darren Leigh. Social Net: Using Patterns of Physical Proximity Over Time to Infer Shared Interests. In *CHI '02: Extended Abstracts on Human Factors in Computing Systems*, pages 816–817, April 2002.

[WNC87] Ian H. Witten, Radford M. Neal, and John G. Cleary. Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540, 1987.

[YLN03] J. Yoon, M. Liu, and B. Noble. Random waypoint considered harmful. In *INFOCOM '03: Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies*, March 2003.

[ZCG09] Biao Zhou, Zhen Cao, and Mario Gerla. Cluster-based inter-domain routing (cidr) protocol for manets. In *WONS '09: Proceedings of the 6th Annual Conference on Wireless On-demand Network Systems and Services*, pages 19–26, February 2009.

Index

Die hier vorgelegte Dissertation habe ich eigenständig und ohne unerlaubte Hilfe angefertigt. Die Dissertation wurde in der vorgelegten oder in änlicher Form noch bei keiner anderen Institution eingereicht. Ich habe bisher keine erfolglosen Promotionsversuche unternommen.

Düsseldorf, den 19.06.2009

Tran Thi Minh Chau