Node Presence Detection with Reduced Overhead

Thi Minh Chau Tran Björn Scheuermann Martin Mauve
Heinrich Heine University, Düsseldorf, Germany

***Abstract*—In this paper we propose a distributed algorithm to detect whether a given node is present or absent in an ad-hoc network. This information is valuable since many proposed protocols and applications for MANETs exhibit worst-case behavior when an intended communication partner is currently not present. Our distributed algorithm improves upon existing approaches to presence detection by significantly reducing the overhead that is required to decide whether a node is present: a reduction by 70–80 % is well possible. We describe the proposed algorithm and assess its properties both analytically and through simulation.**

I. Introduction

Node mobility is a key challenge in mobile ad-hoc networks. On the one hand, mobility implies that nodes can move, that communication links between nodes may break, while new links become available, and so on—but mobility also includes the possibility that nodes may enter and leave the network. Be it due to physical absence or because of switched-off devices: nodes in a mobile network will not be present at all times.

In [1], we pointed out that for many applications it is vital to know whether some given destination node is currently part of the network or not. One specific example—and therefore one possible application of the mechanisms discussed in this paper—is reactive MANET routing: protocols like DSR or AODV make the implicit assumption that all desired communication partners are actually present in and reachable over the network. If this is not the case, and a connection to a non-present node is requested, repeated flooding for route establishment attempts causes significant, but entirely wasted overhead. There are many other examples in various contexts, where information on the set of currently present nodes is highly useful.

A first mechanism that tackles this problem was proposed in [1]. It allows to check whether other nodes are present. However, it 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. As an effect of this particular data representation, it also yields distance estimates.

Although such a hop distance estimate 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 if applying a presence detection mechanism, we pick one specific, well-known application scenario—reactive MANET routing—and apply the presented mechanisms there. This underlines and concretizes the results. In both the general evaluation results and the application scenario, we compare the algorithm presented here with the presence detection mechanism previously proposed in [1].

In the following section, we survey related work in the area of obtaining information about the set of nodes that are active in the network. We then introduce the proposed approach in Section III. We evaluate it analytically and using simulations in Section IV. Section V contains the application study with reactive MANET routing. We finally conclude this paper with a summary in Section VI.

II. 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.

Close relatives to presence detection are location services for geographic routing. Examples are GLS [2] and HLS [3]. However, a location service provides significantly more information about a node than just deciding whether it is present or not, namely its current position. Therefore, the cost to keep the information up-to-date and to perform a lookup in such a service is much higher. Furthermore, 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.

Some systems for presence detection in wireless environments have been successful applications on their own, for example the Lovegety [4]. A number of research projects deal with the exchange of presence information via single-hop wireless communication, with or without infrastructure. Examples are [5]–[7]. But none of these systems considers presence detection over multiple wireless hops.

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 [8] 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 [9] can directly determine the presence of a node. However, they do so at the cost of continuously keeping track of all nodes and all routes. This is only viable if a proactive routing protocol can be efficiently employed in a given environment.

More closely related to the problem of presence detection, [10] 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”. 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.

Finally, [1] describes a solution to the problem of presence detection using a soft state variant of Bloom filters, which is further discussed in the next section. In the paper at hand we present an alternative way to distribute presence information, causing significantly less load on the network.

III. 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 [1] 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 [11] 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 aggregate. The solution proposed in [1], called soft state Bloom filter, achieves this goal. However, it 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 in the paper at hand arises: is it possible to perform presence detection with *unmodified* Bloom filters, and still remove old information? In this section, we will first briefly recapitulate Bloom filters and the soft state extension introduced in [1], before we then show how this can, in fact, be accomplished.

A. The Bloom Filter

A Bloom filter [11] 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 $\{1, \ldots, m\}$. 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. 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, including their own ID. Upon receiving such a beacon, they can merge the received information with what they already know. But this does not provide a means to delete a value once it has been added.

B. Soft State Bloom Filter

Addressing the problem of removing the presence of nodes from an aggregate, the soft state presence detection proposed in [1] replaced the single bits in the Bloom filters by l-bit counters. They are interpreted as the “age” of the respective Bloom filter entry. A node initializes all counters with a maximum value. This maximum value indicates that the position of the Bloom filter is not set (it is thus equivalent to setting a bit position to 0 in the standard Bloom filter). Furthermore, each node keeps the bit positions corresponding to its own ID at a value of zero. All other counters that are not already at the maximum value are incremented whenever a beacon is sent.

Thus, each node continuously announces its own presence with age 0 and ages all other presence information by 1 each interval. Upon reception of a beacon from some other node, the position-wise minimum value of the local and the received aggregate are computed; thereby, presence information is disseminated in the network. A node is considered present if all corresponding Bloom filter positions are not at the maximum value. The aging of the information means that entries of leaving nodes will eventually die out, because they are no longer refreshed by a node keeping the respective position at zero.

Note that this is something significantly different than the mechanisms for removing entries from Bloom filters proposed in earlier work on Bloom filter modifications, the most prominent example being counting Bloom filters [12]. These algorithms also often use counters at each Bloom filter entry, but for an entirely different purpose: counting Bloom filters store the number of additions at this position in the counters, which allows for *explicit* (as opposed to soft-state) removals.

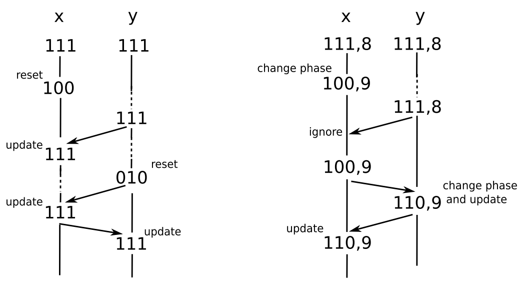
C. Phase Synchronization

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 [1] 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 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 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 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.



(a) Naive resetting. (b) Phase synchronization.

Figure 1. Phase synchronization for the removal of old information.

Figure 1(b) depicts a situation similar to that in Figure 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 1(b), y performs such a transition when receiving a phase 9 beacon from x. Of course, such a neighbor-triggered phase transition 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.

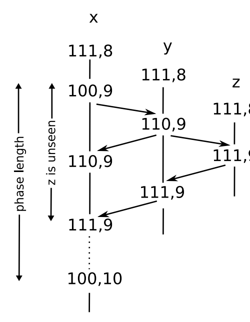


Figure 2. Temporary inconsistency.

D. 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 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 III-B for this purpose. However, in contrast to [1], the soft state Bloom filter is kept only locally. It is not transmitted over the network. Whenever information from a 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.

E. 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:

1) Decay the entries in the local soft state Bloom filter b.
2) If c is at the threshold C, advance the phase p, reset c to zero, and reset a to contain only the presence information of the local node.
3) Broadcast (a, p, c) to the neighbors.
4) Increment c.

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 III.1.

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.

IV. 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

Algorithm III.1 Merge operation

```
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
```

propagation. Where appropriate, comparison with the soft state approach from [1] is also presented.

A. 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

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

B. 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 itseft 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 $0,1,2,\ldots,d$. 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 $0,\ldots,d-1$ must send a beacon. Once d is synchronized to the new phase, each node $d,\ldots,1$ 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. The time D needed to relay beacons from node 0 to d and back is

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

Because beacons are sent at periodic, constant intervals, one may reasonably assume that the offset between the beaconing cycles of two nodes will not change significantly over time spans in the order of some beaconing cycles. Under this assumption, the time span $x_{u,v}$ is always of the same length, regardless which specific pair of subsequent beacons of u and v is considered. Consequently, if the values $x_{u,v}$ do not depend on the specific beacon, it follows that $x_{u,v}+x_{v,u}$ is equal to the time between two successive beacons sent by u—i. e., it is the beaconing interval length, here denoted by B. By rearranging the above sum and using the observation that $x_{u,v}+x_{v,u}=B$, we 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 III-D, 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.

C. Speed of Information Propagation

In the phase-based approach, information is propagated in the same manner as in the soft state approach presented in [1]. 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 $TTL\ldots(C+TTL)$ 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.

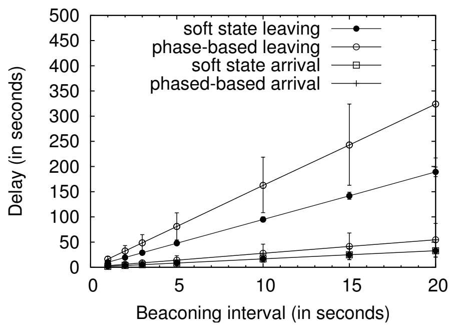


Figure 3. Time until information is received by all nodes.

We conducted simulations to verify this expectation using the ns-2 network simulator [13]. 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 alway starts with phase ID 0, so that it has to synchronize with neighbors before its presence is recognized.

Figure 3 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.

It can be seen that, as expected, the delay increases linearly with the beaconing interval length. The average time until a newly present node is recognized as such (the bottom two lines) is one interval longer with the phased-based approach than it is for the soft state approach. For leaving nodes, the simulation results also confirm the theoretical expectations.

V. 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 [1].

We analyze both algorithms when they are applied in the following scenario. In networks with AODV routing [8], 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 [13]. 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.

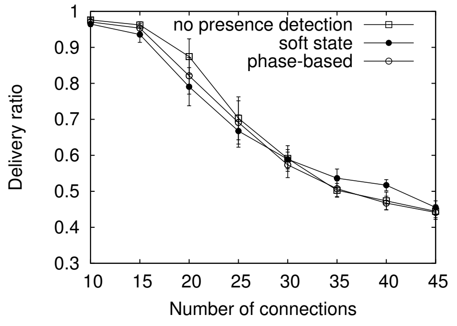
The communication radius in our simulations is 250 meters, with a 550 meter carrier sense range. The nodes move according to the random waypoint mobility model with random speeds in the range from 1 to 10 meters per second and a pause time of 20 seconds. In order to overcome the well-known limitations of random waypoint [14], 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.

Obviously a large number of other parameter settings would have been conceivable. We have selected the parameters outlined above in such a way that the network is in an uncongested state for 20 connections when all nodes are present and no presence detection scheme is used. We then examine the impact of additional load caused by additional connections to present and non-present nodes and by the present detection schemes.

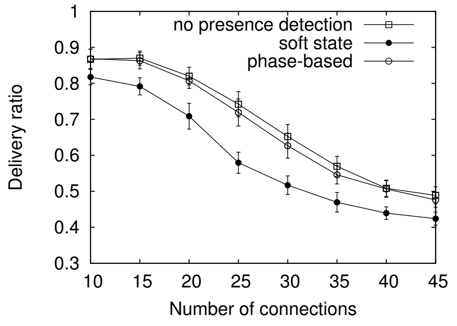
A. Algorithm Parametrization

In simulations with medium-sized networks of 200 nodes, we use Bloom filters with $m = 1024$, $k = 4$, hence the false positive rate is about 0.086. Since networks of these dimensions do typically never exceed a diameter of ten hops, we use four bits per soft state Bloom filter position 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.

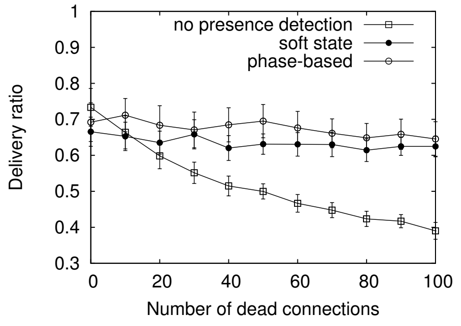


(a) Medium-sized networks.

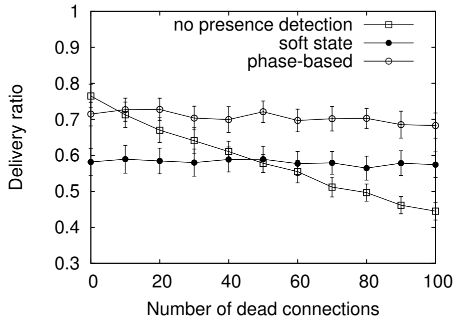


(b) Large-sized networks.

Figure 4. Packet delivery ratio as the number of connections increases, in a worst-case scenario.



(a) Medium-sized networks.



(b) Large-sized networks.

Figure 5. Packet delivery ratio as the number of connection attempts to non-present nodes increases.

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.

B. 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.

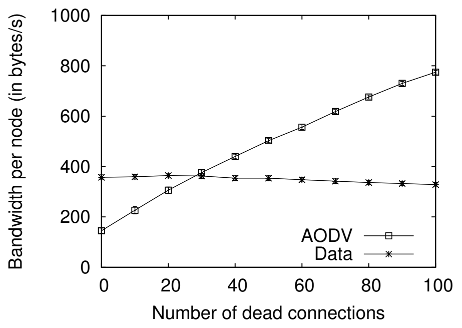
Figure 4 shows the performance of networks of medium and large size, in terms of the packet delivery ratio with 95% confidence intervals, for an increasing number of connections. Because all nodes are always present, we may not expect any benefit from presence detection. But we can assess the impact caused by the additional beaconing traffic in this worst case situation. For medium-sized networks, Figure 4(a) shows little negative impact of both approaches, the network performance in all three cases is roughly the same. The cost of both present 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(b)), the impact of the soft-state approach becomes quite pronounced, due to the large beacon size.

C. Absent nodes

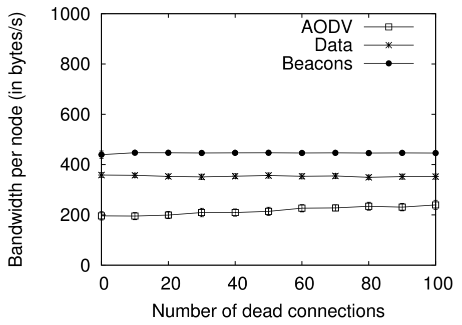
Now we keep the number of "working" connections to present, available nodes fixed at 25. Figure 5 shows the effects on the network performance, when the number of additional connection attempts to non-present nodes increases. It can be seen that, without presence detection, an increasing number of connection 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 6. For medium-sized networks we do not show the results here due to space limitations; the effect is very similar, though.

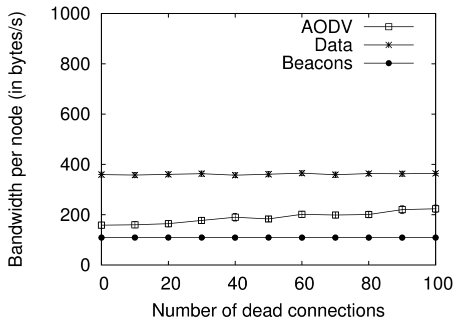
Both presence detection services are able to block the vast majority of connection attempts to non-present nodes. Thus, with presence detection, the bandwidth usage remains almost constant: with an increasing number of connections to non-present nodes there is only a minor increase in bandwidth used 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.



(a) Without presence detection.



(b) Soft state presence detection.



(c) Phase-based presence detection.

Figure 6. Large-sized networks: bandwidth use per node as the number of connection attempts to non-present nodes increases.

In Figure 6(b), it is evident that with soft state Bloom 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 5. 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.

VI. Conclusions

In this paper we proposed a new presence detection scheme for MANETS. In comparison to prior work, 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.

The aggregation of presence information comes at the cost of an adjustable amount of false positives, while it guarantees the absence of false negatives. 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 an exemplary application of presence detection to the route discovery process of AODV.

Acknowledgments

The authors thank the German Research Foundation (DFG) for funding this research.

References

[1] T. M. C. Tran, B. Scheuermann, and M. Mauve, “Detecting the Presence of Nodes in MANETs,” in *CHANTS ’07: Proceedings of the 3rd ACM MobiCom Workshop on Challenged Networks*, Sept. 2007, pp. 43–50.

[2] R. Morris, J. Jannotti, F. Kaashoek, J. Li, and D. S. J. DeCouto, “CarNet: A Scalable Ad Hoc Wireless Network System,” in *Proceedings of the 9th ACM SIGOPS European Workshop*, Sept. 2000, pp. 61–65.

[3] W. Kiess, H. Füßler, J. Widmer, and M. Mauve, “Hierarchical Location Service for Mobile Ad-Hoc Networks,” *ACM Mobile Computing and Communications Review*, vol. 8, no. 4, pp. 47–58, Oct. 2004.

[4] Y. Iwatani, “Love: Japanese Style,” Wired News, online, http://www.wired.com/news/culture/0,1284,12899,00.html, June 1998.

[5] N. Eagle and A. Pentland, “Social Serendipity: Mobilizing Social Software,” *IEEE Pervasive Computing*, vol. 4, no. 2, pp. 28–34, Apr. 2005.

[6] M. Terry, E. D. Mynatt, K. Ryall, and D. 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*, Apr. 2002, pp. 816–817.

[7] L. E. Holmquist, J. Falk, and J. Wigström, “Supporting Group Collaboration with Interpersonal Awareness Devices,” *Springer Personal and Ubiquitous Computing*, vol. 3, no. 1/2, pp. 13–21, Mar. 1999.

[8] C. E. Perkins and E. M. Royer, “Ad-hoc On-Demand Distance Vector Routing,” in *WMCSA ’99: Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, Feb. 1999, pp. 90–100.

[9] P. Jacquet, P. Mühlethaler, T. Clausen, A. Laouiti, A. Qayyum, and L. Viennot, “Optimized Link State Routing Protocol,” in *INMIC ’01: Proceedings of the 5th IEEE International Multi Topic Conference*, Dec. 2001, pp. 62–68.

[10] C. Lindemann and O. 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*, Sept. 2006.

[11] B. H. Bloom, “Space/Time Trade-offs in Hash Coding with Allowable Errors,” *Communications of the ACM*, vol. 3, no. 7, pp. 422–426, July 1970.

[12] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, “Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol,” *IEEE/ACM Transactions on Networking*, vol. 8, no. 3, pp. 281–293, June 2000.

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

[14] 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*, Mar. 2003.