A Mesh-based Robust Topology Discovery Algorithm for Hybrid Wireless Networks

Ranveer CHANDRA[*]
Department of Computer Science
Cornell University
Ithaca, NY 14850
USA
ranveer@cs.cornell.edu

Christof FETZER Karin HÖGSTEDT
AT&T Labs - Research
180 Park Avenue
Florham Park, NJ 07932
USA
{christof, karin}@research.att.com

Abstract

Wireless networks in home, office and sensor applications consist of nodes with low mobility. Most of these networks have at least a few powerful machines additionally connected by a wireline network. Topology information of the wireless network at these powerful nodes can be used to control transmission power, avoid congestion, compute routing tables, discover resources, and to gather data. In this paper we propose an algorithm for topology discovery in wireless networks with slow moving nodes and present various performance characteristics of this algorithm. The proposed algorithm discovers all links and nodes in a stable wireless network and has an excellent message complexity: the algorithm has an optimal message complexity in a stable network and the overhead degrades slowly with increasing mobility of the nodes.

Keywords*: topology discovery, wireless networks, distributed systems, home networks, sensor networks.*

1. Introduction

Wireless networks are becoming increasingly popular. In particular, wireless local area networks are gaining popularity in both office and home settings. Wireless networks are favored over wireline networks for many reasons. For example, they are easier to install in existing buildings and they also give the users complete mobility. As a consequence we expect that in the future, they will be even more widespread.

[*]Work done while at AT&T Labs – Research. Supported in part by DARPA/AFRL-IFGA grant F30602-99-1-0532 and in part by NSF-CISE grant 9703470, with additional support from the AFRL-IFGA Information Assurance Institute, from Microsoft Research and from the Intel Corporation.

One approach to wireless communication is via the use of base stations. In this approach all nodes in the network communicate directly with the base station, which redirects the traffic to the destination node. However, there are several disadvantages to this approach. For example, in certain applications there might be no such preexisting infrastructure available. Also, in many other applications, e.g., home and office or sensor applications, the devices might be battery operated, and the limited power consumption therefore prevents long range communication.

Ad hoc networking, on the other hand, avoids the disadvantages mentioned above by allowing all nodes to communicate directly which each other. Unfortunately, this approach requires partial information of the network to be stored at the wireless nodes, which might not be powerful enough to handle this. In our work we therefore take advantage of the fact that in most wireless networks there is at least one node which is powerful enough to store information when required. We call this kind of network a *hybrid wireless network* and describe it in detail in Section 2.

Many of the applications for these networks rely on the underlying topology information. For example, management of a sensor network or a relief operation requires the knowledge of connectivity of the nodes. The monitoring entity needs to learn of any areas in the network that are too dense or too sparse, or of altogether disconnected nodes. This knowledge can help the monitor make better use of resources in the system. Other applications for topology discovery are power management, routing in ad hoc networks, network visualization, and resource discovery, just to name a few.

In this paper we propose and evaluate a topology discovery protocol. Our design decisions were guided by our target domain: wireless home and office networks. The proposed protocol runs in $\mathcal{O}(N)$ time in a "stable" network (see Section 2.3 for a definition of stable) with N wireless nodes, and worst case in $\mathcal{O}(DN^2)$ when the network is no longer stable, e.g., due to mobility. The term D denotes the average degree of a node. As a part of our protocol we also present a method for building a mesh spanning the network. This mesh can be used in any application to gather data located across the network and is not limited to topology discovery.

In what follows, we first describe our system and failure model in Section 2. In Section 3, we then formally define the topology discovery problem that we solve in this paper. Section 4 goes on to describing the proposed topology discovery protocol, and in Section 5 we discuss some properties of the protocol. Section 6 describes our performance measurements, and in Section 7 we describe the related work. We conclude in Section 8. A list of symbols can be found in Table 3 (at the end of the paper).

2. System Architecture and Model

We believe that future home and small office networks will have the following characteristics. The network will most likely consist of a wireline and a wireless network. The wireline network will at least be needed to connect to a broadband modem (e.g., a cable or DSL modem). In some homes the wireline network might be restricted to connect the broadband modem to a home server which in turn provides services to wireless nodes. These services would most likely include wireless base station functionality, routing, and DHCP service. In new homes the wireline network will most likely connect more devices. It would make sense to connect at least all desktops to the wireline network. Some desktops might have additional access to the wireless network and provide services to the other wireless nodes. Mobile devices, such as laptops and PDAs, will use the wireless network to communicate with each other and the Internet.

We expect that only a few nodes will move at any given point in time and that the speed of movement is very limited. The average speed of a moving node is expected to be limited by walking speed, which is about 1 m/s.

Abstractly, we define the architecture as follows (see Figure 1). The system consists of N_T nodes. Of these N_T nodes, $N_B \geq 0$ nodes are only connected to the wireline network and we denote these *wireline nodes* by B_1, ..., B_{N_B}. There are $N_M \geq 0$ nodes that are only connected to the wireless network and we denote these *mobile nodes* by M_1, ..., M_{N_M}. The remaining $N_G := N_T - N_B - N_M \geq 1$ nodes are connected to the wireline and wireless networks. We denote these *gateway nodes* by G_1, ... G_{N_G}. In what follows, we want to refer to all nodes connected to the wireless network, i.e., all mobile and gateway nodes. We denote these $N := N_M + N_G$ *wireless nodes* nodes by $W_1 := M_1$, ..., $W_{N_M} := M_{N_M}$, $W_{N_M+1} := G_1$, ..., $W_N := G_{N_G}$. The number of wireline, mobile, and gateway nodes might change during the lifetime of a system. Since the execution time of a topology discovery protocol is relatively short, we can assume that N_B, N_M, and N_G are constant during that period.

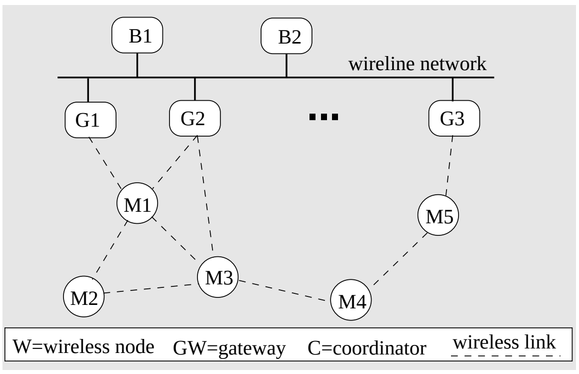

Figure 1. A system consists of wireline nodes (n_i), mobile nodes (M_i), and gateway nodes (G_i). Mobile and gateway nodes are called wireless nodes.

2.1. Communication System

Wireless transmissions have much higher bit error rates than wireline transmissions. Current transport protocols, in particular, TCP/IP, are designed for wireline networks. They do not perform well if the frequency of dropped packets is too high. Hence, wireless technologies like 802.11 use MAC layer acknowledgements to detect and retransmit dropped packets. This permits a sender to detect if the transmission of a message has failed.

The gateway nodes provide routing functionality for all wireless nodes. A gateway node might operate as a wireless base station or it might be a member of a wireless ad hoc network. In either case, it provides routing capability for other wireless nodes in its neighborhood. Logically, the topology discovery protocol has to be independent from the network layer (i.e., routing layer) since the routing tables might be computed using the topology provided by the topology discovery protocol. Hence, a node is restricted by only being able to send messages to its neighbors, i.e., messages sent by the protocol are not routed.

We abstract away the details of the MAC and network layer and assume the following communication model. Each wireless node can send local unicast messages. By *local* we mean that the message is not routed via intermediate nodes. We assume that all messages are unique in the sense that given a message m one can determine the sender $sender(m)$, the destination $dest(m)$, the send time $ST(m)$, the receive time $RT(m)$, and the acknowledgement time $ACK(m)$. The receive time $RT(m)$ is the time at which message m is received. It is undefined, i.e., $RT(m) \uparrow$, if the message is never received. If the message is not delivered, acknowledged, or the acknowledgement is received too late, then $ACK(m)$ is undefined, i.e., $ACK(m) \uparrow$. To simplify matters, we assume that all times are defined with respect to real-time.

Current MAC layers do not provide a robust local broadcast mechanism. Nevertheless, we assume (and show in Section 4.2.3 how to implement) such a mechanism. Any practical implementation will use unreliable broadcasts provided by the MAC layer to implement such a robust broadcast mechanism. We assume that all neighbors of the sender of n will receive the robust broadcast, i.e., if a node does not receive n, it is not a neighbor of $sender(n)$. $ACK(n)$ is the receive time-stamp of the (first) acknowledgement received for n.

We assume that links are likely to be bidirectional, i.e., if W_i can receive messages from W_j then W_j can receive messages from W_i. This is a valid assumption because underlying wireless protocols, e.g., the 802.11 MAC layer, require links to be bidirectional too.

2.2. Failure Model

A node can schedule certain tasks to be executed at a certain point in time. Typically, such a task sends one or more messages. To simplify our model, we assume that a node does not suffer performance failures: a non-crashed node will execute all its tasks at their scheduled times and that the processing times are negligible. Practically, this is just an accounting trick and does not mean that we have a synchronous system model. We achieve this by adding the scheduling and processing delays to the transmission delay of the messages (see Figure 2).

A node has crash failure semantics, i.e., a node can only fail by prematurely stopping the execution of its program.

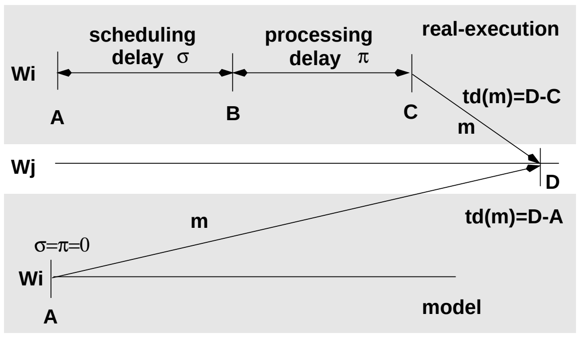

Figure 2. In our model we add the scheduling and processing delays to the transmission delay of messages.

We assume that if a message m from node W_i is delivered to node W_j, then W_i has indeed sent m to W_j. This implies that a corrupted message is never delivered and hence, it is never acknowledged. If an acknowledgement of a message m is not delivered within a predefined time-out delay Δ, we say that the transmission of m has failed. Note that even if m is received by $dest(m)$ but the acknowledgement of m is not received by $sender(m)$ within Δ, we say that m failed.

2.3. Stability Properties

We denote the link from wireless nodes W_i to W_j by $L_{i,j}$: $L_{i,j}$ is the link that W_i uses to send messages to W_j. We say that a link $L_{i,j}$ is *stable* in a time interval I if 1) there exists at least one message that is sent between W_i and W_j in I, and 2) all messages sent between W_i and W_j in I are delivered and acknowledged in time. Predicate *stable* is symmetric in the sense that if $L_{i,j}$ is stable in I, then $L_{j,i}$ is also stable in I. By requiring that at least one message is sent via a stable link, we ensure that a broken link cannot be called stable during periods in which no messages are exchanged. Formally, we can express predicate $stable$ as follows:

$$\begin{aligned}&stable(L_{i,j}, I) :=\\ &\quad \bigwedge \exists m \exists s \in \{W_i, W_j\}, \exists r \in \{W_i, W_j\} - s : ST(m) \in I \wedge sender(m) = s \wedge dest(m) = r\\ &\quad \bigwedge \forall m \exists s \in \{W_i, W_j\}, \exists r \in \{W_i, W_j\} - s : ST(m) \in I \wedge sender(m) = s \wedge dest(m) = r \Rightarrow ACK(m) \downarrow\end{aligned}$$

We say that a link $L_{i,j}$ is *disconnected* in an interval I if 1), there exists at least one message that is sent between W_i and W_j in I and 2), no message sent between W_i and W_j in I is delivered. We define predicate $disconnected$ as follows:

$$\begin{aligned}&disconnected(L_{i,j}, I) :=\\ &\quad \bigwedge \exists m \exists s \in \{W_i, W_j\}, \exists r \in \{W_i, W_j\} - s : ST(m) \in I \wedge sender(m) = s \wedge dest(m) = r\\ &\quad \bigwedge \forall m \exists s \in \{W_i, W_j\}, \exists r \in \{W_i, W_j\} - s : ST(m) \in I \wedge sender(m) = s \wedge dest(m) = r \Rightarrow RT(m) \uparrow\end{aligned}$$

A disconnected link is never stable and vice versa. However, a link might neither be stable nor disconnected. We call such a link *unstable*:

$$unstable(L_{i,j}, I) := \neg stable(L_{i,j}, I) \wedge \neg disconnected(L_{i,j}, I).$$

A node W_i is *reachable* by a node W_k in interval I, if there exists a path of stable links from W_k to W_i:

$$\begin{aligned}&reachable(W_i, W_k, I) :=\\ &\quad \exists l, \exists P = (P_1, ..., P_l), \forall o \in \{1, .., l-1\} : stable(L_{P_o, P_{o+1}}, I) \wedge W_{P_1} = W_k \wedge W_{P_l} = W_i.\end{aligned}$$

We say that the network is *stable* in a time interval I, if each link is either stable or disconnected and any node is reachable by some wireless node C (see Section 3):

$$stable(I) := \forall i, j \in \{1, ..., N\} : (stable(L_{i,j}, I) \vee disconnected(L_{i,j}, I)) \wedge reachable(W_i, C, I).$$

We introduce a weaker property of a *semi-stable* network. In such a network, links are permitted to be unstable as long as each node is reachable via a stable path from C:

$$semi\text{-}stable(I) := \forall i \in \{1, ..., N\} : reachable(W_i, C, I).$$

In what follows, we often do not provide interval I. In all these cases, I is assumed to be the run-time interval of the protocol.

3. Problem Description

In some system, a wireline or gateway node initiates the topology discovery by sending request messages via the wireline network to all gateway nodes. In other systems, the topology discovery might be initiated by any wireless node. To generalize the problem, we omit the optional first step of forwarding a topology discovery request via the wireline network. We assume instead that a wireless node C (*coordinator*) initiates the protocol.

We define the topology discovery problem as follows. The discovery is initiated by node C at some time s. The protocol has to return the discovered topology to C within a bounded time R, i.e., at some time t such that $s \leq t \leq s + R$. We use $I := [s, t]$ to refer to the run-time interval of the program. We assume that the topology is returned in form of a predicate T. Predicate $T(i, j)$ holds iff the protocol discovered the link $L_{i,j}$, i.e., that W_j can receive messages sent by W_i.

We have to specify that the topology returned by a correct topology discovery protocol reflects the topology of the wireless network. Note that the topology might change due to mobility during the run-time of the protocol and hence, we cannot require that the topology T is identical to the topology of the wireless network. Instead we constrain T as follows. First, if the protocol says that a link $L_{i,j}$ exists then it has to have “proof” that this link existed at least at some point in I. Second, if T says that there is no link $L_{i,j}$ between two nodes W_i and W_j, then either the link must not be stable or neither W_i nor W_j are reachable by C. More precisely, we require the following.

R1 Let us consider that the protocol discovers a link $L_{i,j}$, i.e., $T(i, j)$ holds. We require that there exists at least one message m that is sent and received via $L_{i,j}$ during the run-time of the protocol:
$\forall i, j : T(i, j) \Rightarrow \exists m : sender(m) = i \wedge dest(m) = j \wedge ST(m) \in I \wedge RT(m) \in I.$

R2 Let us consider that the protocol says that nodes W_i and W_j are not connected by a link, i.e., $T(i, j)$ does not hold. In this case we require that neither W_i nor W_j are reachable by C in I or that $L_{i,j}$ is not stable in I:
$\forall i, j : \neg T(i, j) \Rightarrow (\neg reachable(W_i, C, I) \wedge \neg reachable(W_j, C, I)) \vee \neg stable(L_{i,j}, I).$

This specification implies that if the network is stable, the topology returned to C consists of all stable links. It also guarantees that in a semi-stable network, the topology returned by the protocol includes all nodes and it contains all stable links but it might also contain unstable links. However, it will never contain disconnected links.

Note that for a link $L_{i,j}$ to be stable in I, at least one of the neighboring nodes W_i or W_j has to send a message to the other. Because of this requirement, a trivial protocol could send no message, and thus ensure that no link is stable. This trivial protocol could therefore immediately return with an empty topology, and still satisfy our specification. To prevent such trivial solutions, we note that other protocols are allowed to send messages in parallel to the messages to the topology discovery protocol. More precisely, the protocol designer has to consider an adversary that can send messages at arbitrary times between arbitrary nodes during the protocol execution. In this way, we exclude trivial solutions and force a correct protocol to send messages to discover the topology of the network.

4. Protocol Description

The topology discovery algorithm consists of two phases. The first phase, which we refer to as the *Diffusion phase*, is initiated by the coordinator by broadcasting a topology request message. As the nodes receive and rebroadcast this message, they construct their local neighborhood information, and update other data structures used to construct a mesh which at the end of the Diffusion phase spans the complete network. This mesh is then used in the second phase, the *Gathering phase*, to forward the local neighborhood information from all the nodes up to the coordinator. Together, these two phases provide the coordinator with the required topology information.

Before we describe the protocol in detail, we give a brief overview illustrated by some examples. The coordinator initiates the protocol by broadcasting a topology request message. The first time a node receives such a message, it records its sender as a parent, records this information in the message, and retransmits it using a robust local broadcast. To increase the reliability of the algorithm, each node also records the senders of the next $(k-1)$ requests it receives as its parents (unless the request was broadcasted by a potential descendant), and rebroadcasts them. The constant k is bounded by a small integer K, and is determined by the coordinator. Each node records its children by checking if it is listed as a parent of the sender in each received request message. In this manner, all nodes reachable by the coordinator will at the end of the Diffusion phase have anywhere from 1 to k parents, together forming a mesh (see Figure 3).

In the Gathering phase, each node sends its and its descendants' neighborhood information to all its parents via unicast messages. These unicasts are initiated either as soon as it has received the neighborhood information from all its children, or when it has waited a pre-determined amount of time inversely proportional to its distance to the coordinator, whichever occurs first. In a failure free run, the coordinator receives the desired topology information after a total of $\mathcal{O}(N)$ messages (see Figure 4). We describe the Gathering phase in more detail and under less ideal conditions in Section 4.3.

In the rest of this section, we first present the message formats together with the data structures stored at each node in Section 4.1. This is followed by a formal description of the Diffusion phase of the algorithm in Section 4.2, and the section is concluded in Section 4.3 with a description of the Gathering phase.

4.1. Message Formats and Data Structures

In this section, we present the formats of the messages used in the protocol, as well as the data structures maintained locally at each node in the network. This section is meant to be used as a reference section for Sections 4.2 and 4.3, where the details of how these messages and data structures are used and updated are described.

The protocol uses three message types: `DiffReq`, `DiffAck` and `GathResp`. `DiffReq` is the topology request message initiated by the coordinator, and later rebroadcasted by the other nodes, `DiffAck` is a special acknowledgement sent as part of the reliable broadcast described in Section 4.2.3, and `GathResp` is the message used to propagate the topology information up to the coordinator during the second phase of the protocol. The fields of the three messages are described in Table 1, the data structures maintained locally at each node are presented in Table 2. The details of how these data structures are used and updated will be described in Section 4.2 and 4.3.

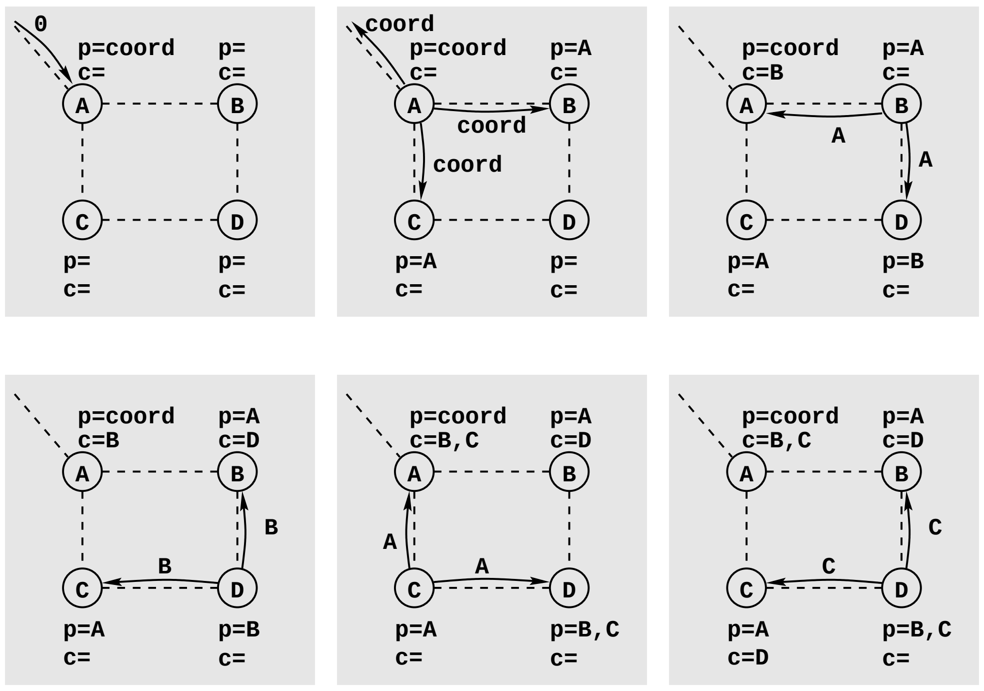

Figure 3. An example of the Diffusion phase in a sample network. Starting at the coordinator, each node broadcasts a request message containing its parent node. p **and** c **denotes the lists of each node's parents and children, respectively.**

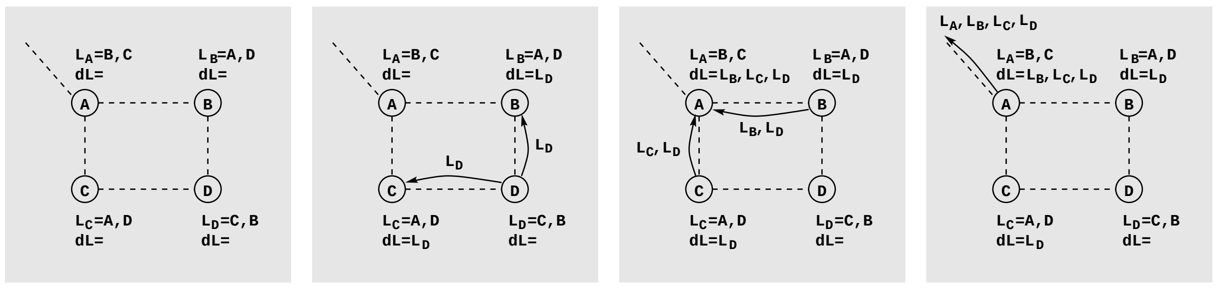

Figure 4. An example of the Gathering phase in the same sample network as in Figure 3. Starting at the leaves, each node i sends its neighbor information (L_i) and its downstream neighbor information (dL_i) to its parent(s).

Field	Description
`coord`	the coordinator of the topology discovery
`sender`	the sender that broadcasted this message
`parent`	this message is a rebroadcast of a message received by this parent
`hopCount`	the distance, in hops, from `coord` to `sender`
`k`	maximum number of parents of a node
`bcastId`	a unique integer that identifies the current protocol run
`maxEccentricity`	the estimated maximum distance from any node to the `coord`
`powerLevel`	transmission power to be used by the transport protocol

(a) The fields of the `DiffReq` message

Field	Description
`sender`	the node sending this `DiffAck` message
`dest`	the `sender` field of the `DiffReq` being acknowledged
`coord`	the `coord` field of the `DiffReq` being acknowledged
`bcastId`	the `bcastId` field of the `DiffReq` being acknowledged

(b) The fields of the `DiffAck` message

Field	Description
`sender`	the sender of this message
`coord`	the coordinator
`bcastId`	a unique integer that identifies the current protocol run
`topoInfo`	the accumulated topology information from the sender and the downstream nodes
`eccentricity`	the maximum distance from any downstream node to the `coord`
`panicmode`	set to true iff sender is in panic mode

(c) The fields of the `GathResp` message

Table 1. Format of the three messages used in the protocol.

Data structure	Description
`L`	list of discovered neighbors.
`Parents`	a list of the parents of the node
`Children`	a list of the children of the node
`dL`	the accumulated neighborhood information of the downstream nodes
`HopCount_th`	the hop count of the first parent in the `Parent` list of the node

Table 2. The data structures kept at each node in the network.

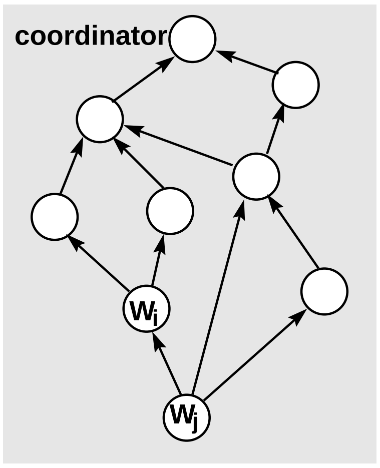

Figure 5. In a 3-resilient mesh each node has at most 3 parents.

We use an adjacency list to represent the information in the `topoInfo` field of the `GathResp` message, and the `L` and `dL` data structures at the nodes, even though a bitfield would reduce the message size in dense networks.

4.2. Diffusion Phase

The Diffusion phase is the first phase of the topology discovery protocol. The coordinator initiates it by broadcasting a `DiffReq` message. When a node receives such a message, it rebroadcasts it (at most $k \geq 1$ times). The Diffusion phase has two purposes. First, it updates the local neighborhood information of the nodes. A node W_i that is reachable by the coordinator will receive a message from all its neighbors and vice versa. Hence, W_i will add all its neighbors to its neighbor list (`L`). The second purpose of the Diffusion phase is to build a coordinator rooted mesh spanning across the entire network. If the network is partitioned the mesh might not contain all nodes. However, in a semi-stable network it contains all nodes reachable from the coordinator. This mesh is used by the Gathering phase to propagate the accumulated neighborhood information up to the coordinator.

It is important to note that the mesh can be used for any kind of data gathering application, and it is not in anyway dependent on the topology discovery application. It can also be used in applications such as determining the power usage, or any kind of data aggregation in a sensor network. We therefore start by describing how to construct this mesh. In Section 4.2.2 we then show the remaining steps of the Diffusion phase, and finally, in Section 4.2.3 we show how we have implemented the broadcasts, to increase their robustness.

4.2.1 Building a Mesh

A *k-resilient mesh* is a connected directed acyclic graph (DAG), rooted at the coordinator, in which any node W_i has at most k parents, where $1 \leq k \leq K$, and K is small, constant integer. A parent of W_i is any node W_j such that there exists a link in the DAG from W_i to W_j. We call k the *resiliency factor* of the mesh. Note that a k-resilient mesh is also a $(k+x)$-resilient mesh, $\forall 1 \leq x \leq K-k$. Figure 5 gives an example of a 3-resilient mesh.

The coordinator initiates the construction of the mesh by broadcasting a `DiffReq` message, which contains the resiliency factor of the mesh in the `k` field of the message. When a node receives this message, it does the following:

- If this is the first `DiffReq` the node has received, then it
  - sets `HopCount_th` to be equal to the `hopCount` field of the message,
- If the node has less than `k` nodes in its `Parent` list, and the field `sender` is not in the `Parent` list, and this `DiffReq` has a `hopCount` field smaller than or equal to `HopCount_th`, then the node
  - adds the `sender` of the message to its `Parent` list,
  - increments the `hopCount` field of the message by one,
  - copies the `sender` field into the `parent` field,
  - sets itself as the `sender`, and
  - rebroadcasts the message.
- If the node is in the `parent` field of the `DiffReq`, then it
  - adds the `sender` to its `Children` list.

4.2.2 Updating Neighborhood Information

To update the local neighborhood information we add the following to the algorithm described in Section 4.2.1.

- The node always adds the `sender` of any message to its neighbor list `L`.

At the completion of the Diffusion phase the complete connectivity information of the network can be constructed from the local neighborhood information available at all the nodes. During the Gathering phase we collect this info and propagate it up to the coordinator through the mesh built during the Diffusion phase.

A node will only add links to its neighbor list across which it has received at least one message during the execution of the protocol. This means that only links that satisfy requirement **R1** are added. Note that the `DiffReq` message will be received by each node W_i that is connected to the coordinator via a path of stable links. In particular, W_i will rebroadcast the `DiffReq` message and add all nodes connected to W_i via a stable link to its neighbor list. We show in Section 4.3 that the neighbor lists of W_i will be forwarded to the coordinator – which is needed and sufficient to satisfy requirement **R2**.

4.2.3 Robust Broadcast

The success of the Diffusion phase depends on the reliability of the broadcasts: if many broadcasts fail, fewer links are discovered. Broadcasts in most MACs for ad hoc networks are not reliable. In 802.11 for example, broadcasts are sent whenever the carrier is sensed free and can therefore result in a number of collisions as shown in Figure 6. In this example we have simulated the Diffusion phase using GloMoSim [15] with three different transmission powers. We see that as we increase the transmission power and therefore the neighbor density, more broadcast messages collide and a smaller fraction of the links are discovered.

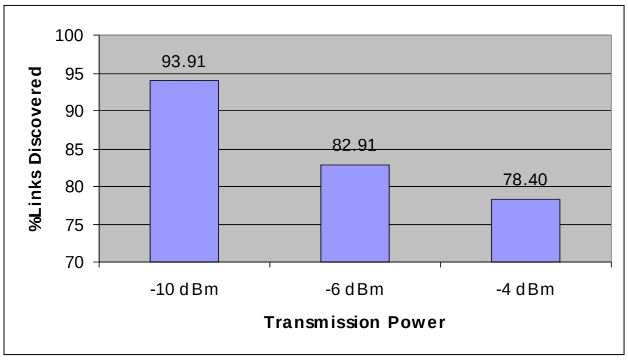

Figure 6. Simulation results for three different transmission powers. These GloMoSim simulations used 50 nodes in a 200mx200m area.

We increase the robustness of broadcasts by using a variation of the RTS/CTS scheme proposed by [14]. The idea is to ensure that the broadcast reaches at least one node in the neighborhood. To use this scheme, we could let each node do a RTS/CTS with the node from which it received a request whenever it wishes to rebroadcast it.

An RTS/CTS mechanism is however mostly helpful for large messages. In the Diffusion phase the `DiffReq` messages that are being broadcast are small in size, and the lengthy message exchange of an RTS, followed by a CTS, the `DiffReq` message, and finally an acknowledgement is unnecessary. Instead, a node simply broadcasts the `DiffReq` and expects a `DiffAck` in response from the node from which it received the request it is now retransmitting. If a node does not receives a `DiffAck` within a certain amount of time, it rebroadcasts the message. This is repeated a small number of times, or until it receives a `DiffAck`, which ever occurs first. Our simulations show that this scheme adds a lot of robustness to the Diffusion phase. For example, in the scenarios simulated in Figure 6, the `DiffReq/DiffAck` scheme discovers 100% of the links.

4.3. Gathering Phase

In this section we describe the Gathering phase, the second of the two phases of our topology discovery protocol. In the Gathering phase the k-resilient mesh built in the first phase is used to send the topology information back to the coordinator. The Gathering phase is initiated by the leaves in the mesh, who send their local neighborhood information to their parents in a `GathResp` message. Intermediate nodes wait for replies from all the children before they in their turn forward a `GathResp` message to their parents. If no failures occur, the information from all nodes reaches the coordinator who at this point learns the entire topology information.

In the rest of this section, we describe the Gathering phase in detail. We first describe it for a stable network, followed by a discussion in Section 4.3.2 on how a few unstable links are handled. In Section 4.3.3 we then describe how the algorithm is designed to cope with a large number of unstable links.

4.3.1 Stable Network

The mesh structure built in the Diffusion phase defines a dependency between `GathResp` messages: a node waits for a response from all its children before initiating a response itself. If a node has no children, i.e., if it has not received any `DiffReq` messages after a certain amount of time with itself listed as a parent of the sender, it initiates the Gathering phase. Each leaf W_i in the mesh uses a unicast to send a `GathResp` to each of its parents, with set $\{(W_i, \texttt{L})\}$ as the `topoInfo` field, where `L` is the list of neighbors discovered by W_i.

When receiving a `GathResp` message, a node W_j combines its `dL` data structure with the `topoInfo` field: `dL` is set to the union of `dL` and `topoInfo` and then all pairs containing neighborhood information for the same node are combined. When a node has received a `GathResp` from all its children, the node builds its own response message and unicasts it to all its parents, with the combination of $\{(W_j, \texttt{L})\}$ and `dL` as the `topoInfo` field of the message.

In a stable network, the `topoInfo` field in a `GathResp` message contains the complete neighborhood information for all downstream nodes. The coordinator therefore has access to the complete topology information upon receipt of the `GathResp` message from the last of its children. The coordinator has thus discovered all the links and nodes reachable at the end of the Gathering phase.

4.3.2 A Few Unstable Links

The protocol described above works as long as the mesh structure consists of stable links. An unstable link can cause a parent to wait for a `GathResp` message from its child, although the child might never be able to send it successfully. This problem is more serious in meshes with a lower resiliency factor. In a 1-resilient mesh one unstable link can prevent the downstream information to reach the coordinator. However, a mesh with a resiliency factor greater than one will be able to tolerate some unstable links if alternate paths exist from the nodes to the coordinator. The scheme to make a mesh tolerate unstable links is described in the rest of this section.

A parent should not wait for an unbounded amount of time for the reply of a child since a reply might never be received, e.g., due to a link that failed or due to the crash of a downstream node. The parent should instead time-out after a bounded amount of time to make sure that it forwards its own neighborhood information and that of its other children that it has received so far. The time-out has to be chosen carefully. If the time-out is too short, some topology information might never be forwarded to the coordinator since any information that arrives after the time-out is ignored.[1] If the time-out is too long, the topology information might become stale. Due to mobility some discovered links might for example become disconnected and some new links might appear before the information is forwarded to the coordinator.

We first introduce some terminology for a better understanding of our approach. Let the i-th parent of node W_j be P_i and its depth from the coordinator perceived at W_j be d_{P_i}. So the distance of node W_j to the coordinator along the path through P_i is $d_{P_i} + 1$. Constant Δ (see Section 2.2) is the time-out delay for unicast messages, i.e., the time after a sender gives up retransmitting a message. Let ecc denote the eccentricity with respect to the coordinator, i.e., the maximum distance from the coordinator to any node in the network. In our current implementation the ecc is an estimate made by the coordinator and corresponds to the `maxEccentricity` field of the `DiffReq` message. The `eccentricity` field is included in the `GathResp` message to the coordinator for a better estimation of ecc in the next Diffusion phase. However, there are other ways to estimate the eccentricity of the network and we hope to explore them in the future.

We cascade the time-outs of the nodes. Ideally, a child W_i time-outs exactly Δ time units before its parent P_i to

[1]If it is not ignored and the time-out is too short, the message complexity would increase unacceptably.

make sure that W_i has sufficient time to send its current topology information to P_i before P_i time-outs on W_i. Hence, we set the time-out such that W_i waits for $timeout = 2 * (ecc - d_{P_i} + 1) * \Delta$ after receiving the first `DiffReq` from P_i. It will initiate a `GathResp` when all its children have replied, as described in 4.3.1, or when its $timeout$ for P_i has expired. In the second case, the node W_i sends all the information it has available to parent P_i.

4.3.3 Many Unstable Links

The protocol described so far gathers all link information as long each node has at least one stable link to a parent during the Gathering phase. Since the response is sent without much delay and the nodes in a home or office network are not expected to move at high speeds, the mesh will usually stay stable enough for the Gathering phase to succeed. In these common scenarios our protocol has an optimal $\mathcal{O}(N)$ message complexity and is able to discover all the nodes and links in the network. However, in a rare situation a node in the mesh might displace itself fast enough to become disconnected from all its parents such that none of the parents are reachable during the Gathering phase. The support provided by our protocol for these uncommon scenarios is detailed in the rest of this section.

A node first tries to send the `GathResp` message to all its parents in the mesh. If none of the parents are reachable, then it enters a *panic* mode. In panic mode the node sends the response message to all its other neighbors. This set of neighbors is determined during the Diffusion phase as described in Section 4.2.2.

A `GathResp` message from a parent in panic mode causes the parent to be removed from the `Parents` data structure. For example, suppose node W_i is the parent of node W_j. If W_i enters panic mode, then node W_j should not rely on successful communication of its `GathResp` to node W_i, since W_i might still not be able to send the message any further. So, W_j removes W_i from its `Parents` list and enters a panic mode if this list becomes empty.

On receiving a `GathResp` message, the node updates its `dL` variable with the `topoInfo` field of the message, and then does either of the following:

- If the node has not yet sent its `GathResp` to all the nodes in its `Parents` list, then it
  - sends the response as described in Section 4.3.2.
- otherwise if the received message has added new link information to `dL`, then it
  - resends a `GathResp` message with the new link information to all the nodes in `Parents`.[2] If the `Parents` list is empty, it enters the panic mode.
- otherwise it
  - ignores the `GathResp` message.

A worse situation could arise when a node is unable to send its `GathResp` message to any of its neighbors. Although a node in this case does not have any of its original neighbors discovered during the Diffusion phase, it might still get its message across if it broadcasts it. We use a robust broadcast slightly different from the one described in Section 4.2.3; since the `GathResp` messages are big, explicit RTS and CTS messages are used. The RTS/CTS is done with the last neighbor from whom any message was heard or received. To reduce the size of this broadcast we

[2] Note that in panic mode, as opposed to in non-panic mode, a `GathResp` message arriving after an expired time-out is not ignored.

do not send the complete neighborhood information, but rather only the node information. The argument is that if all the neighbors of a node have failed, then its link information is not of any use to the coordinator.

5. Protocol Properties

In this section we discuss some qualitative properties of our protocol. First we explain how our protocol satisfies the requirements **R1** and **R2** introduced in Section 3. Then we analyze the message complexity of the protocol. Due to space constrains we omit proofs of the properties. The quantitative performance is presented in Section 6.

5.1. Correctness

The protocol collects link information locally in a neighbor list and then collates and forwards the neighbor lists to the coordinator during the Gathering phase. The topology information consists of these collated neighbor lists. Only if a node W_j receives a message from W_i with the same protocol identifier, it adds W_i to its neighbor list. Adding W_i to the neighbor list of W_j is necessary for the protocol to set $T(i, j)$. The system model specifies that if W_j receives a message from W_i then indeed the message was sent by W_i. Checking the protocol identifier makes sure that no stale messages will be used – in particular, only senders of messages that were sent after the protocol was started are included in the neighbor list. Hence, our protocol satisfies property **R1**.

Property **R2** is only guaranteed if the panic mode is switched on. However, if the network is stable, the protocol satisfies **R2** even if the panic mode is switched off. If the network is stable, a link is either stable or disconnected and all nodes are reachable by the coordinator. Hence, to violate requirement **R2**, there has to exist a stable link $L_{i,j}$ that is not part of the topology information (i.e., $\neg T(i, j)$). Since all nodes are reachable, nodes W_i and W_j will both send out broadcast messages during the Diffusion phase. Since $L_{i,j}$ is assumed to be stable, W_j will receive the broadcast of W_i and hence, W_j will include W_i in its neighbor list. All links between a parent and a child are stable since each child was able to receive the message of its parent (and the network is assumed to be stable). This implies that the neighbor information of W_j will propagate to the coordinator. Therefore, $T(i, j)$ has to be part of the topology information. Note that in a stable network the discovered topology contains all nodes and all stable links.

If the panic mode is turned on, property **R2** is satisfied, independent of whether the network is stable. The inverse of **R2** states that if W_i or W_j are reachable by the coordinator and $L_{i,j}$ is stable, then $T(i, j)$ has to be part of the topology. If a link $L_{i,j}$ is stable and W_j is reachable, then W_j will receive a message from W_i and add W_i to its neighbor list. Panic mode guarantees that each node W_j can send its neighbor information to the coordinator as long as it is reachable by the coordinator. Hence, the panic mode guarantees that **R2** is always satisfied. Note that in a semi-stable network, the topology returned by the protocol includes all nodes and it contains all stable links but it might also contain unstable links. However, it will never contain disconnected links.

5.2. Message Complexity

The message complexity of the Diffusion phase is $\mathcal{O}(N)$. The protocol sends up to $k * N$ robust broadcast messages during the Diffusion phase, where $k \leq K$ is the maximum number of parents allowed, and K is a small constant integer. In response to each of these broadcasts, a node receives one unicast acknowledgement message. Hence, there are also at most $k * N$ unicast messages. The overhead of a robust broadcast message is comparable to that of a unicast message with respect to bandwidth and transmission power usage. Hence, we count a robust broadcast message as one unicast message in our message complexity analysis. The total message complexity of the Diffusion phase is therefore $\mathcal{O}(kN) = \mathcal{O}(N)$, since k is bounded by a (small) constant K.

The message complexity of the Gathering phase is also $\mathcal{O}(N)$ as long as the mesh is *semi-stable*. We say that a mesh constructed by the Diffusion phase is semi-stable, iff each node is reachable from the coordinator via a path of stable links along the mesh. When the network is stable, the mesh is semi-stable. Note however, that if the network is semi-stable, then the mesh is not necessarily semi-stable. This definition of semi-stable mesh implies that each node has a stable link to at least one of its parents. This prevents the nodes to enter panic mode. Therefore, each node sends at most one unicast message to each of its parents during the Gathering phase. The message complexity is therefore $\mathcal{O}(kN) = \mathcal{O}(N)$ for the Gathering phase as long as the mesh is semi-stable.

When the mesh is not semi-stable, some nodes might enter panic mode. If a node in panic mode receives a neighbor list it has not received previously, it first unicasts a `GathResp` message to all its neighbors. Assuming that a node has on average D neighbors, the average number of unicast messages in panic mode is at most DN per node. If all the unicasts messages the node sent failed, it broadcasts a condensed message (see Section 4.3.3). The number of robust broadcast messages in panic mode is at most N per node. The total number of unicast and broadcast messages in panic mode is therefore at most DN^2 and N^2, respectively. The worst case message complexity for panic mode is thus $\mathcal{O}(DN^2 + N^2) = \mathcal{O}(DN^2)$.

Any protocol has to send at least $N - 1$ messages to satisfy the requirements **R1** and **R2** since each node has to send at least one message to let the coordinator know from which other nodes it can receive messages. Note that we permit links to be unidirectional. Hence, our message complexity of $\mathcal{O}(N)$ for stable networks is optimal.

6. Performance

The topology discovery algorithm was simulated in GloMoSim [15], which uses a parallel, event-driven simulation language called Parsec [3]. 50 nodes were randomly placed in a 200m$\times$200m area. IEEE 802.11 was used as the MAC protocol and the bandwidth was assumed to be 2 Mbps. The Random Waypoint model was used to model the mobility of the nodes in the network. In this model each node moves towards a random destination at a speed chosen randomly between a predefined minimum and maximum value. It then pauses for some duration and continues with this mobility pattern. In our simulations the minimum speed was set to 0 m/s and the pause time to 30 seconds. We expect the speed of nodes in our target application to be walking speed, i.e., approximately 1 m/s.

All nodes were set to have the same transmission power, although simulations were carried out for three different transmission powers: -10 dBm, -6 dBm and -4 dBm. At a transmission power of -10 dBm the average neighborhood size was about 4, at -6 dBm it was close to 11, and about 17 at -4 dBm. When the transmission power was -4 and -6 dBm, the network was semi-stable for all simulated speeds. At -10 dBm however, the network was only semi-stable up to 0.8 m/s.

One node was designated as the coordinator and the percentage of links and nodes discovered at this node is presented in this section. The topology discovery protocol was executed for 12.5 seconds. Link and node information obtained at the coordinator after this interval were not considered.

We evaluate our protocol using three different metrics. First, we measure the percentage of stable links discovered (See Section 2.3 for a definition of a stable link). It is unreasonable to expect that an algorithm would discover links that only existed for a short amount of time during the execution of the algorithm. Furthermore, information about links that do no longer exist is irrelevant, and could also lead to inaccurate information at the coordinator. We therefore compare the number of links that our protocol discovers, to the actual number of stable links, and use this fraction to evaluate our protocol.

Second, we measure the percentage of nodes discovered. In some applications the link information might not be of interest; it is simply enough to learn of the different nodes in the network. We therefore also present the percentage of nodes discovered as a way of evaluating our protocol.

Third, we measure the message overhead of the protocol. The protocol was executed with and without the panic mode and for different values of the resiliency factor of the mesh that is formed during the Diffusion phase. A higher resiliency factor increases the robustness of the protocol but at the expense of an increase in the message overhead. The panic mode further improves the robustness of our protocol but also results in an added increase in the number of messages sent during the Gathering phase. During the Diffusion phase, every node sends a constant number of messages, resulting in $\mathcal{O}(KN) = \mathcal{O}(N)$ messages, where N is the number of nodes, and K is the constant upper bound of the resiliency factor of the mesh constructed during the Diffusion phase. We therefore only present the message overhead incurred during the Gathering phase.

In the rest of this section, we present the results for these three different metrics for the three transmission powers of -10, -6, and -4 dBm.

6.1 Transmission Power: -10 dBm

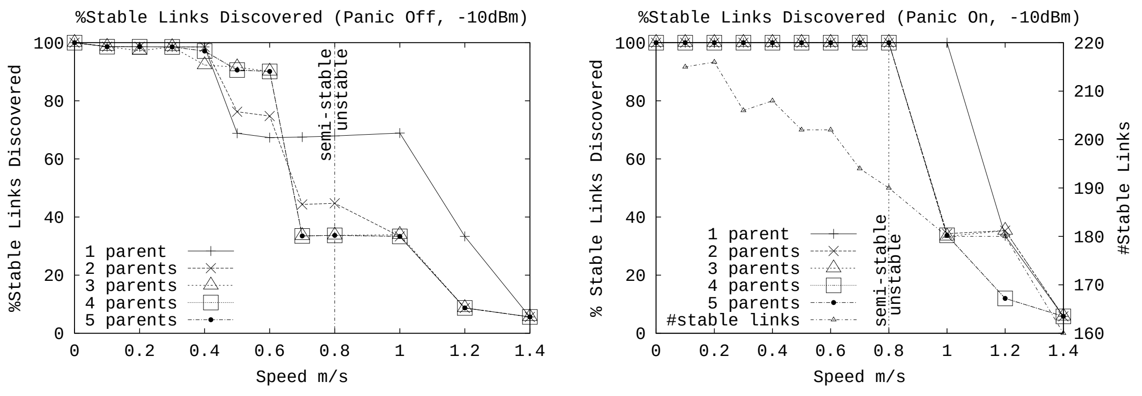

Figure 7. Percentage of stable links discovered for -10 dBm transmission power, and different values of the resiliency factor k. `#stable links` denotes the total number of stable links in the network. The network is semi-stable up to `0.8 m/s`.

The percentage of stable links discovered using a transmission power of -10 dBm is shown in Figure 7. Without the panic mode the algorithm discovers *nearly all* the stable links up to 0.4 m/s. An increase in speed reduces the robustness of the mesh and results in a lower percentage of stable links discovered. The reason for the relatively large decrease in the number of stable links discovered is that when the coordinator learns of a link from a node W_i to another node W_j, it cannot infer that there is a link from W_j to W_i since we do not assume all links to be bidirectional. Had we done that, the number of stable links discovered would have increased, both with and without the panic mode.

It is interesting to note that increasing the resiliency factor, i.e., the maximum number of parents allowed is useful only up to a certain speed: in this case up to 0.6 m/s. When increasing the average number of parents, the average number of children per node grows, and thus also the probability that at least one link between a parent and a child breaks. Especially for higher speeds where link breakages are even more common, an increase in the number of parents thus results in a greater number of nodes that timeout before sending their `GathResp` messages. Due to these time-outs, the propagation of neighborhood information to the coordinator is delayed proportionally to the number of parents. But in a network with sparsely connected nodes moving at a high speed, it is important to forward the information as fast as possible, since a link might disappear before it is used. In a sparsely connected network with high speed, the coordinator might therefore discover a higher percentage of stable links if the nodes are permitted fewer parents, i.e., if the resiliency factor is smaller.

We call this tradeoff between minimizing response time and maximizing mesh resiliency the *inversion problem*. The problem of inversion suggests that the resiliency factor should be a tunable parameter, chosen according to the mobility and density of the network.

With panic mode turned off, the protocol discovers *all* the stable links as long as the network is stable. With panic mode turned on, the protocol discovers *all* the stable links as long as the network is semi-stable. At higher speeds the network is no longer connected using stable links. The coordinator is therefore unable to receive all the response messages and this results in a decrease in the percentage of stable links that are discovered. We can also see the effect of the inversion problem in this case. When the nodes are allowed fewer parents, the `GathResp` messages are sent sooner and the coordinator is therefore able to gather information over more links before they break, resulting in a higher percentage of stable links discovered.

The message overhead of the Gathering phase is shown in Figure 8. When the panic mode is turned off, the `GathResp` messages are only sent along the mesh. No effort is made to repair the mesh and so the number of response messages stays nearly constant across different speeds. The slight variations in each of the curves are due to the differing structure of the mesh at different speeds. As the resiliency factor increases, the average number of parents increases and more links are part of the mesh which results in a higher message overhead.

When the panic mode is turned on, the number of replies sent during the Gathering phase increases more with speed if a larger resiliency factor is used. The reason for this is the inversion problem; when a node has only a few parents its time-out is shorter and it therefore has a higher probability that the links to its parents still exist. This has the effect that a network using a larger resiliency factor enters the panic mode for lower speeds than networks with a smaller value of k: the increase in the number of messages for $k = 3$, 4, and 5, occurs already at 0.7 m/s, whereas the increase for $k = 1$ and $k = 2$ does not occur until 1.2 and 1 m/s, respectively. The number of replies drop for higher speeds since the network is no longer semi-stable. More links are thus broken and fewer nodes enter the panic mode by receiving `GathResp` messages sent by other nodes in panic mode.

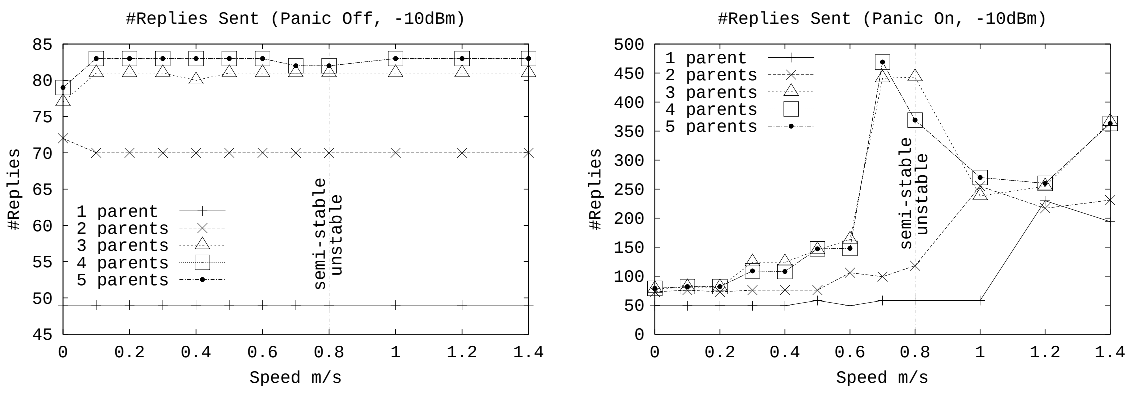

Figure 8. Number of `GathResp` messages sent for -10 dBm transmission power, and different values of the resiliency factor k.

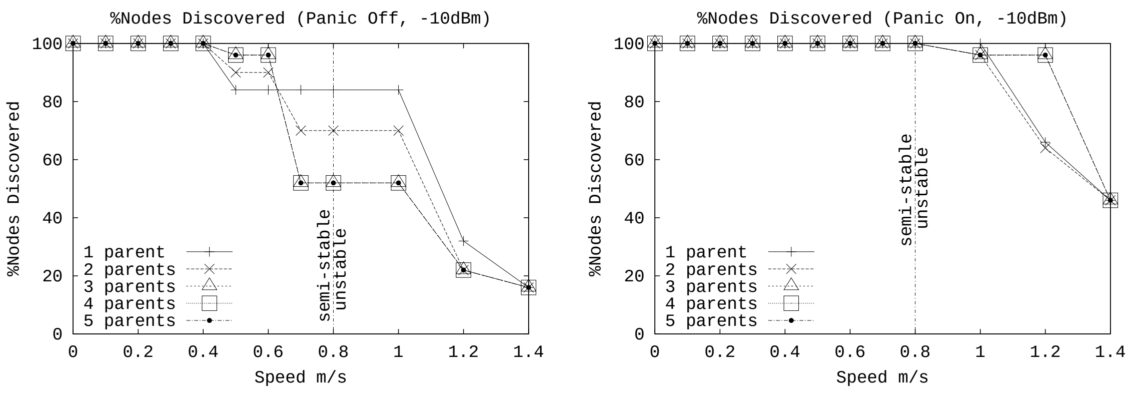

Figure 9. Percentage of nodes discovered for -10 dBm transmission power, and different values of the resiliency factor k.

Figure 9 shows the percentage of nodes discovered by the protocol. Without the panic mode, the percentage of nodes discovered decreases with an increase in speed. This is because the nodes are no longer connected using stable links along the mesh. We can also see that the inversion problem causes better performance for a smaller value of the resiliency factor at speeds greater than 0.6 m/s. Again, the shorter time-out results in more information reaching the coordinator.

When the panic mode is turned on, the percentage of nodes discovered is significantly higher than the percentage of stable links discovered. This is due to the fact that only one link to or from the node has to be discovered for the coordinator to learn about the existence of the node.

Note that in Figure 7, 8, and 9 the protocol has a similar performance for $k = 4$ and $k = 5$. This is because the nodes only have around four neighbors, and they can therefore not have more than four parents even though five parents are permitted by the protocol.

6.2 Transmission Power: -6 dBm

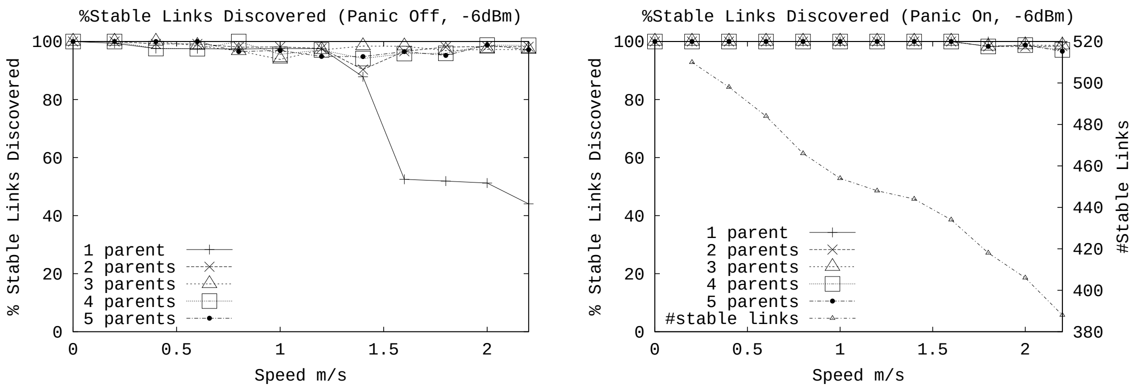

Figure 10. Percentage of stable links discovered for -6 dBm transmission power, and different values of the resiliency factor k.

The percentage of stable links discovered at a transmission power of -6 dBm is shown in Figure 10. Because of a stronger transmission power, the speeds are not large enough to show significant signs of the inversion problem. When the panic mode is turned off and the nodes are allowed to have more than one parent, the algorithm discovers nearly all stable links. This is because most nodes are reachable through stable links along the mesh for all simulated speeds. When there is just one parent, even a single link breakage close to the coordinator significantly decreases the number of links discovered. When nodes have more than one parent alternate paths to the coordinator are ensured, and the mesh is hence more robust.

When the panic mode is turned on, all the stable links are discovered using a resiliency factor of one, even at high speeds. The inversion problem becomes visible for $k \geq 2$ at speeds of 1.8 m/s and above. When $k = 2$, the protocol only discovers 98% of the stable links (instead of 100% when $k = 1$).

The message overhead of the Gathering phase is shown in Figure 11. When the panic mode is turned off, the algorithm sends a message along all the links in the mesh. The number of `GathResp` messages sent is therefore equal to the number of links in the mesh, and thus proportional to the average number of parents. The slight variations in the number of messages are due to different mesh structures at different speeds.

When the panic mode is switched on, the number of `GathResp` messages sent is proportional to the resiliency factor, except in the case when $k = 1$. This behavior can be explained using the graphs in Figure 10. Nearly all the stable links are discovered when more than one parent is allowed and so very few nodes enter the panic mode. Therefore most of the traffic flows along the mesh. However, when $k = 1$, the number of stable links discovered drops significantly at higher speeds. This is because the mesh is much more fragile with only one parent and so it suffers a number of link breakages. To still be able to discover 100% of the stable links, a large number of nodes have to enter the panic mode. When $k = 1$, the number of `GathResp` messages sent therefore increases greatly with an increase in speed.

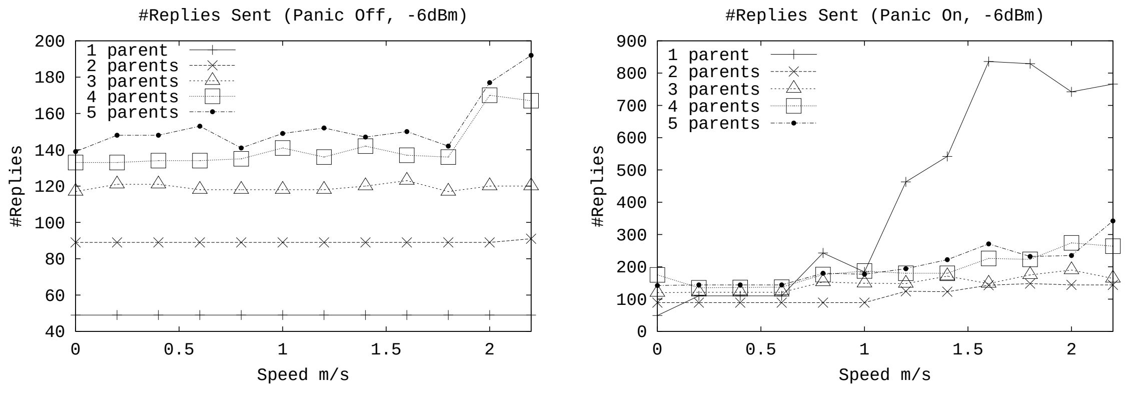

Figure 11. Number of `GathResp` messages sent for -6 dBm transmission power, and different values of the resiliency factor k.

All nodes are discovered in all cases except for at the highest speed, when the panic mode is turned off and only one parent is allowed. In this case 49 out of the 50 nodes were discovered.

6.3 Transmission Power: -4 dBm

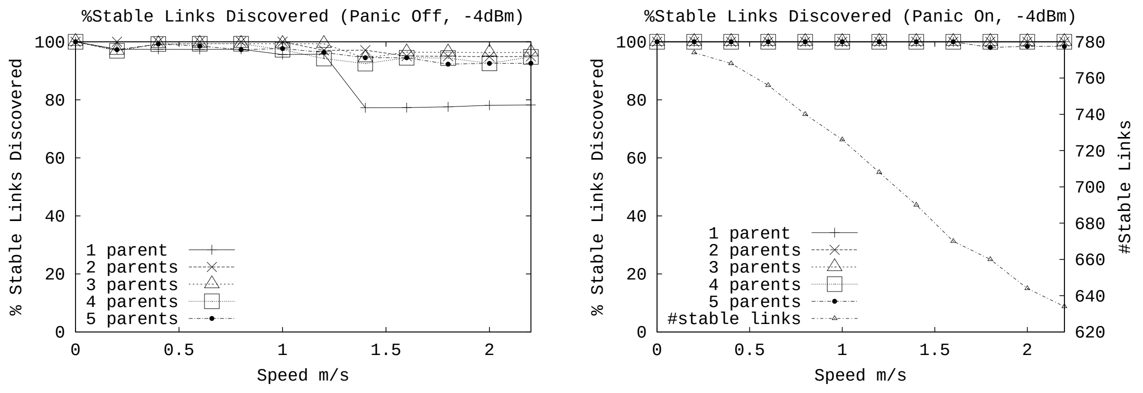

Figure 12. Percentage of links discovered for -4 dBm transmission power, and different values of the resiliency factor k.

Figure 12 shows the percentage of stable links discovered when the transmission power is -4 dBm. In this case the average neighborhood of each node is about 17 and so the algorithm performs quite well even when the panic mode is turned off. Nearly all the stable links are discovered when more than one parent is allowed. When $k = 1$, the mesh is quite fragile and vulnerable to link breakages. The performance of the algorithm is however much better than in the -6 dBm case shown in Figure 10. With the panic mode turned on, the algorithm discovers all the stable links when $k < 5$. Due to the inversion problem, the percentage drops to 98% starting at 1.8 m/s for a resiliency factor of five.

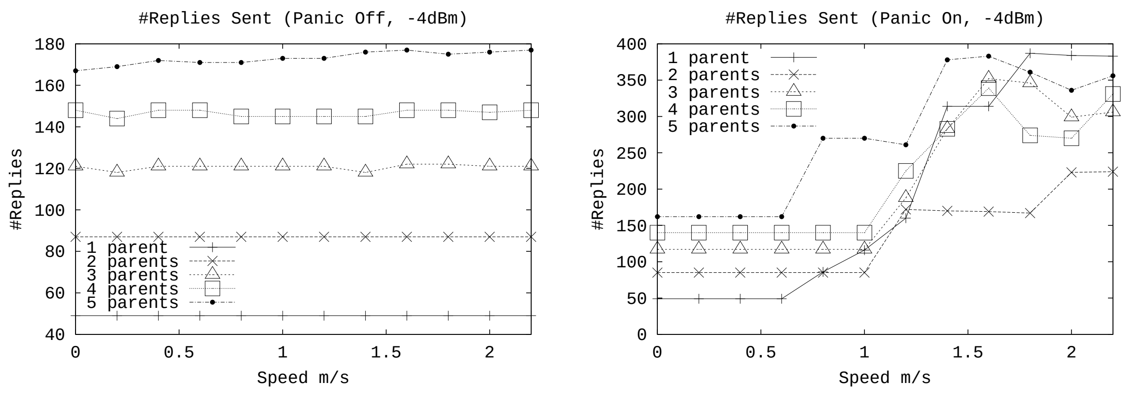

Figure 13. Number of `GathResp` messages sent for -4 dBm transmission power, and different values of the resiliency factor k.

The number of `GathResp` messages sent during the protocol execution is shown in Figure 13. When panic is turned off, the number of `GathResp` messages is equal to the number of links in the mesh since there is exactly one message per link in the Gathering phase. With the panic mode turned on, the message overhead shows a trend similar to Figure 11 (notice the different scale on the y-axis). The message complexity increases with increasing speeds when the number of parents allowed is greater than 1. The reason is that an increase in the number of parents gives rise to alternate paths along the mesh at high speeds. However, when there is just one parent, a number of links in the mesh break, causing a large number of nodes to enter the panic mode. In the case of a transmission power of -4 dBm, the effect is however less severe than in the case of -6 dBm, since each node has more reachable neighbors.

All nodes are discovered in all cases except for at the highest speed, when the panic mode is turned off and only one parent is allowed. In this case 49 out of the 50 nodes were discovered.

7. Related Work

The problem of topology discovery has been extensively studied for the Internet. Commercial systems, such as HP's OpenView [1] and IBM's Tivoli [2], automatically generate the network-layer topology using standard routing information. Furthermore, in [5], the authors present a scheme using SNMP MIB (Management Information Base) information to discover physical topology in multi-vendor IP networks, and an algorithm to determine network-layer topology independent of SNMP is proposed in [13]. In ad hoc networks however, this problem has been largely overlooked.

The problem of topology discovery in ad hoc networks is significantly different than in wired networks. There is no IP subnet hierarchy and nodes might have stale neighborhood information. Additionally, there is no popular network management protocol, such as SNMP, for ad hoc networks. Low overhead, on-demand routing protocols, such as AODV[11] and DSR[10], also do not store enough state to provide complete topology information. On the other hand, link state protocols, such as TBRPF[4] and OLSR[9], require each node to constantly maintain a partial topology of the network. This is an overhead when the link information is required temporarily at a few nodes. Link state routing protocols also provide only network layer topology information and might not be able to discover the complete physical connectivity of the network. The mobility of nodes results in communication paths that are prone to frequent link breakages. These conditions make the problem of topology discovery extremely challenging in ad hoc networks.

There has however been some work related to topology discovery for ad hoc networks. In [6], the authors provide a clustering scheme for ad hoc network management. The protocol, called ANMP, uses a distributed set of nodes or cluster heads to maintain node information. ANMP attempts to incorporate features of SNMP, and uses a hierarchical scheme to gather topology information. The cluster heads are dynamically chosen based on geographic location or network connectivity. ANMP uses the MIBs at cluster heads to gather topology information. However, this scheme has the overhead of constantly maintaining cluster heads in the network. Additionally, the information in the MIBs might be stale due to mobility and could fail to provide a complete link information of the network.

In [8], a Uniform Quorum Scheme for mobility management in ad hoc networks is proposed. The idea is to dynamically maintain the node location databases among the network nodes. These nodes are self organizing and are connected through a virtual backbone. However, this protocol is designed for location and mobility management. It does not provide the complete physical layer connectivity of the network.

Another topology discovery algorithm is presented in [12]. Mobile agents in the nodes periodically gather topology information and disseminate it to all the other nodes in the network. However, this scheme does not provide an instantaneous topology of the network. This algorithm is also extremely intensive in time and messages to discover a complete topology of the network.

It should however be mentioned that none of the the above protocols were designed to solve the particular problem of topology discovery we discuss in this paper. All the above approaches look at different variants of topology discovery as they do not aim to discover all the links in the network. The topology information is moved among nodes in the network depending on the connectivity and traffic in areas of the network. In this paper we look at a different problem where the entire topology information has to be learned at a few pre-specified nodes. Previous approaches described above do not provide an efficient infrastructure to solve this problem.

[7] attempts to solve our problem of topology discovery. It provides a protocol, called TopDisc, to discover topology in sensor networks. A hierarchical tree-based clustering scheme is used to gather neighborhood information from all the sensor nodes. However, this protocol provides only a partial link information of the network. It also assumes a reliable broadcast mechanism, which has not yet been developed for ad hoc networks. The tree structure used by TopDisc is also unstable and susceptible to link breakages because of mobility in the network.

8. Conclusion

In this paper we have defined a precise system model and problem statement for the topology discovery problem. We have also presented a reliable protocol for topology discovery in wireless networks. The protocol consists of two phases; the first phase diffuses the initial topology request message across the whole network and the second propagates the neighborhood information back to the initiating node. The second phase propagates the information using a mesh structure built during the first phase. This mesh can be used for any kind of data gathering application,

Symbol	Description	Sec.
N_T	total number of nodes	2
N_B	total number of wireline nodes	2
N_G	total number of gateway nodes ($N_T - N_B - N_M$)	2
N_M	total number of mobile nodes	2
N	total number of wireless nodes ($N_G + N_M$)	2
B_i	wireline node i	2
M_i	mobile node i	2
G_i	gateway node i	2
W_i	wireless node i	2
$sender(m)$	sender of message m	2.1
$dest(m)$	recipient of message m	2.1
$ST(m)$	time at which message m was sent	2.1
$RT(m)$	time at which message m was received	2.1
$ACK(m)$	time at which the acknowledgement of message m was received	2.1
Δ	one-way message time-out delay	2.2
$L(i,j)$	link between node W_i and W_j	2.3
$stable(L(i,j),I)$	stable link	2.3
$disconnected(L(i,j),I)$	disconnected link	2.3
$unstable(L(i,j),I)$	unstable link	2.3
$reachable(W_i,W_j,I)$	node W_i reachable from W_j	2.3
$stable(I)$	stable network	2.3
$semi-stable(I)$	semi-stable network	2.3
C	coordinator node	3
I	run-time interval of the protocol	3
T	topology predicate	3
R1	topology discovery requirement	3
R2	topology discovery requirement	3
`DiffReq`	message sent in the Diffusion phase	4.1
`DiffAck`	acknowledgement message part of the reliable broadcast in the Diffusion phase	4.1
`GathResp`	message sent in the Gathering phase	4.1
k-resilient mesh	data structure built in the Diffusion phase, and used in the Gathering phase	4.2.1
k	the resiliency factor of a mesh, i.e., the maximum number of parents allowed	4.2.1
K	the upper bound on k	4.2.1
D	the average number of neighbors	5.2

Table 3. Symbols used in this paper.

and is not limited to topology discovery. The message complexity of the protocol is $\mathcal{O}(N)$ in a stable network with N nodes, and it slowly degrades to a worst case of $\mathcal{O}(DN^2)$ when the nodes are more mobile. We show that the protocol discovers close to 100% of the stable links and 100% of the nodes in the targeted applications.

References

[1] HP Openview, http://www.openview.hp.com/.

[2] IBM Tivoli, http://www.tivoli.com/.

[3] R. Bagrodia, R. Meyer, M. Takai, Y. Chen, X. Zeng, J. Martin, B. Park, and H. Song. Parsec: A parallel simulation environment for complex systems. *IEEE Computer*, 31(10):77 – 85, October 1998.

[4] B. Bellur and R. G. Ogier. A reliable, efficient topology broadcast protocol for dynamic networks. In *Proceedings IEEE INFOCOM 1999*, March 1999.

[5] Y. Breithart, M. Garofalakis, C. Martin, R. Rastogi, S. Seshadri, and A. Silberschatz. Topology discovery in heterogeneous IP networks. In *Proceedings IEEE Infocom 2000*, volume 1, pages 265–274, 2000.

[6] W. Chen, N. Jain, and S. Singh. ANMP: Ad hoc network management protocol. *IEEE Journal on Selected Areas in Communications*, 17(8):1506 – 1531, August 1999.

[7] B. Deb, S. Bhatnagar, and B. Nath. A topology discovery algorithm for sensor networks with applications to network management. Technical Report Technical Report DCS-TR-441, Department of Computer Science, Rutgers University, May 2001.

[8] Z. J. Haas and B. Liang. Ad hoc mobility management with uniform quorum systems. *IEEE/ACM Transactions on Networking*, 7(2):228 – 240, April 1999.

[9] P. Jacquet, P. Muhlethaler, A. Qayyum, A. Laouiti, L. Viennot, and T. Clausen. Optimized link state routing protocol. http://www.ietf.org/internet-drafts/draft-ietf-manet-olsr-04.txt, March 2001.

[10] D. B. Johnson and D. A. Maltz. *Dynamic Source Routing in Ad Hoc Wireless Networks*. Kluwer Academic Publishers, 1996. Chapter 5, pp. 153-181.

[11] C. E. Perkins and E. M. Royer. Ad hoc on-demand distance vector routing. In *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, pages 90–100, February 1999.

[12] R. RoyChoudhury, S. Bandyopadhyay, and K. Paul. A distributed mechanism for topology discovery in ad hoc wireless networks using mobile agents. In *IEEE First annual workshop on Mobile and Ad Hoc Networking and Computing (MobiHoc)*, pages 145–146, 2000.

[13] R. Siamwalla, R. Sharma, and S. Keshav. Discovering Internet topology. Cornell University, 1999.

[14] J. Tourrilhes. Robust broadcast: improving the reliability of broadcast transmissions on CSMA/CA. In *Proceedings of the 9th IEEE International Symposium on Personal Indoor and Mobile Radio Communications*, volume 3, pages 1111–1115, September 1998.

[15] X. Zeng, R. Bagrodia, and M. Gerla. GloMoSim: A library for parallel simulation of large scale wireless networks. In *Proceedings of the 12th Workshop on Parallel and Distributed Simulation*, pages 154–161, May 1998.