

# Organizing a Distributed Application in a Mobile Ad Hoc Network

Cristian Tuduce      Thomas Gross  
Departement Informatik  
ETH Zürich  
8092 Zürich, Switzerland

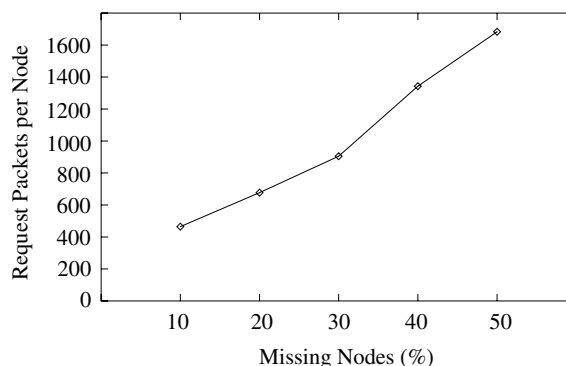
## Abstract

*A distributed application that operates in an ad hoc network formed by mobile nodes must limit its use of all-to-all communication since the overall capacity of such a network is severely constrained. To address this problem, we describe an algorithm that allows an application to impose a hierarchical structure on the participating nodes. A simple tree is maintained as hosts join and leave the ad hoc network. We evaluate this algorithm in the context of a collaboration tool for a network of Linux laptops using IEEE 802.11b network cards. Preliminary performance results from our mobile test-bed indicate that the application responds well to connectivity changes and is sufficiently agile to run in a highly mobile environment.*

## 1 Introduction

Ad hoc networks promise to provide a networking platform in the absence of a fixed, managed infrastructure. However, such networks make only a fraction of the peak link bandwidth available to applications, as has been pointed out both by theoretical analysis [1] as well as by experimental studies[2]. Since ad hoc networks are an active topic of research, no doubt improvements to protocols will be developed over time. Applications in an ad hoc environment (or any mobile environment) should limit their use of all-to-all or broadcast communication operations. The shared medium entails that any such communication step takes away bandwidth from any host (node) that is in the same environment as a node that engages in such a communication step. Simulation results validate these statements. In Figure 1 we plot the overhead induced by generating route requests for nodes that are not available at that moment. The results are obtained by simulating the behavior of 60 nodes over 900 seconds. The simulated routing protocol is Ad hoc On Demand Vector routing protocol (AODV)[3]. The nodes are moving in an 1500x300m area. As more and

Research supported, in part, by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant number 5005-67322.



**Figure 1. Overhead per node, due to route request packets for unavailable nodes.**

more nodes of the network become unavailable, the remaining nodes must deal with more and more management packets. It is therefore important that applications in such an environment should use directed communication operations wherever possible.

Consider as an example a distributed collaboration tool that we have developed jointly with an industrial partner. The target domain is the management of a large-scale construction site, e.g., when building a power plant. As the building goes up, engineers at the site want to download documents or leave messages and notes. These data may be stored at a system in the main field office (or in a system that is reachable via this office). However, given the nature of the site (lots of steel, shielded rooms, construction machines with electrical motors, ongoing construction), providing a network by laying down cabling or installing a few APs (access points) proved cumbersome. An ad hoc network minimizes the need to set up a separate infrastructure during construction. The interesting aspect of an ad hoc network is that no or little on-site management is needed. The notebooks, PDAs, or computers of the users form the ad hoc network, but other hosts can be installed to improve network coverage. E.g., if some area of the site has difficulty connecting to the field office (because of distance or shielding), then one or more computers (“dedicated nodes”) can be installed until network connectivity improves or work in the

area has finished. These nodes, which have the sole purpose of forwarding and routing packets, are part of the ad hoc infrastructure and also use wireless communication; supplying power is usually not a problem but there is resistance to installing a temporary wire-based network.

Such applications that execute in mobile ad hoc networks face a new set of challenges. On one hand, the applications must adapt to connectivity changes as mobile or dedicated nodes move relatively to each other. On the other hand, the application must limit its use of the bandwidth for management so that enough bandwidth remains available to the end-users. In this paper we present a solution to these conflicting requirements in the context of a distributed collaboration tool. Network management aspects (where should dedicated nodes be placed? how can we characterize the connections to other nodes?) are beyond the scope of this paper.

## 2 A distributed collaboration tool

**Application issues:** The purpose of the collaboration tool is to allow engineers and contractors at a construction site to exchange messages. The user's communication requirements range from short text messages to large binary files (text, project files, 3D models, recorded voice messages, photos, etc.). E.g., during construction it is sometimes necessary to call in the architects to resolve issues that were not addressed in the plans, and an architect may have to visit the site to assess the situation. If the construction site has access to a network, a field trip may be replaced by transmission of photos. However, the application is not required to support live multimedia streaming.

The size of the construction site is usually about  $1\text{km}^2$ . That area is larger than what can be covered by a single AP. Usually, such a site involves 100+ engineers or contractors (but  $< 1000$ ) that would use the collaboration tool described here. This number of participants (likely to increase once a system is fielded) is large enough that brute force solutions are unattractive.

In addition to message transfers, a shared workspace (or whiteboard) to view or modify drawings on a common canvas is desirable. We use the term *document* to refer to either kind of message or drawing. Documents can be *public* or *private*. The public messages are received or seen by all the participants, whereas the private messages are received only by a selected subset of the participants.

Management of user identities, privacy and security and esp. authentication of the users, as well as the initial configuration of the nodes are outside of the scope of this application (and paper). Since the users belong to a closed community (employees or subcontractors of the construction company), a number of solutions are available to provide unique identifiers (user names, IP addresses) and to address these issues.

**System issues:** The key requirement for the application is to provide a communication tool without the need for global configuration. The application must limit its use of broadcast operations, so if a user wants to inquire about the status of another user's host, it is not acceptable to inquire with a broadcast message. The application must maintain its own structure and consequently needs to adapt in response to connectivity changes caused by node mobility or environmental factors (e.g., a moving crane can temporarily shield some nodes from others and split a network in two clouds).

The ad hoc model of our application is communication and organization with zero-configuration (i.e., the user is not supposed to be involved in the management of the host configuration). We focus on solutions that can be realized at the application layer to obtain portability (and to shelter this application from future changes in the environment). There are still open issues regarding zero-configuration at the network layer.

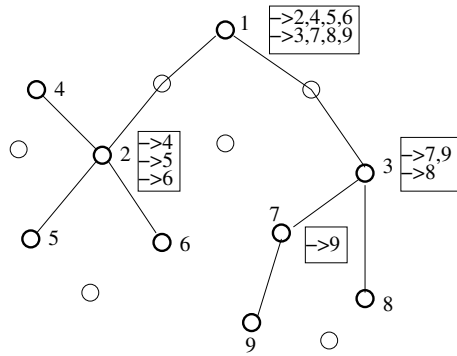
An ad hoc network provides multi-hop communication and there exist a number of protocols. We do not expect that every node in the wireless network participates in the collaboration tool. The nodes that run the application build an overlay network over the common communication fabric. We use at the network layer the AODV routing protocol.

The mobility of the nodes can cause the clients to move too far away to maintain connectivity with the server (the field office), or node mobility can partition the network. The client-server model is therefore not suitable for applications when connectivity between clients must be maintained in case of network splits. To cope with this problem, we moved from the client-server paradigm to a server-less model. The participating nodes (*peers*) find each other automatically using a simple service discovery mechanism. Using this model, network splits form distinct network clouds, but communication inside a cloud is still possible.

To minimize the number of connection between nodes and therefore the connection management overhead, the application organizes the peers in a tree structure. The tree structure is application-specific and is not designed as a replacement of the common communication fabric.

Figure 2 depicts an example. Circles represent the mobile hosts; lines represent the connections between nodes. The bold circles represent hosts on which the distributed application is running. These participating nodes form a tree (how the tree is formed is discussed later); the tree is neither balanced nor is there a constraint on the number of children a node is allowed to have. In the tree structure, every node has knowledge about the nodes in the subtrees rooted at its children. The tables next to each node represent the information a node maintains about its descendants. Here we just show the node identifiers and omit other information like user ids, system status, etc. Different lines in the table summarize different branches.

When a node *A* wants to communicate with a node *B*



**Figure 2. Logical organization of nodes participating in distributed application.**

(both participate in a distributed application) we distinguish between the *control path* and the *data path*. The control path includes the nodes that process the meta-messages (e.g., is *B* registered somewhere?). The purpose of the tree is to limit the control path. The control path may include nodes that are not part of the application, but these nodes act only as forwarders and are therefore not mentioned any further. The data path includes the nodes that carry the application-specific message. The data path and the control path can be identical, but that is not a requirement; the data exchange between two nodes can flow along the tree structure or use the common communication fabric.

Whenever the destination of a message is unknown, the message is sent up in the tree for delivery. E.g., if node 8 wants to send a message to node 7, node 8 searches in its table and fails to find the way to node 7. Therefore the message is sent to node 3. Node 3 knows the way to node 7, and delivers the message. This example illustrates the message delivery in the tree structure. If a message reaches the root of the tree and the location of the destination is still unknown, the destination node is not connected to the tree (not on-line or not in the communication range of this wireless cloud). When the destination of a message is not connected the message must be either stored or dropped, depending on the message properties.

In the above scenario, control and data path are identical since the tree nodes carry the data message. However, after a node has received a message from another node, with high probability the sender will be around to receive a response or acknowledgments. This fact can be exploited to separate control and data paths. E.g., node 8 wants to send a message to node 9. The routing protocol has no route for node 9. The message is therefore sent along the tree structure. The message is delivered via nodes 3 and 7 to node 9. Node 9 now initiates a request to find a route using the common communication fabric, and uses this route to return a response.

The destination triggers the route request for the source node. Using this mechanism we eliminate flooding the network with route requests for nodes that do not exist. Using

just the underlying communication fabric, the routing protocol would flood the network with route request packages for a node that does not exist.

Furthermore, flooding of the network can be limited by setting the TTL (Time To Live) of the route request packet to the distance in the tree structure between the source and the destination.

Separating the data path from the control path is beneficial when the underlying routing protocol can provide a better (shorter, faster) route. This may happen when there are nodes relaying network packets but not running the application.

### 3 The service discovery mechanism

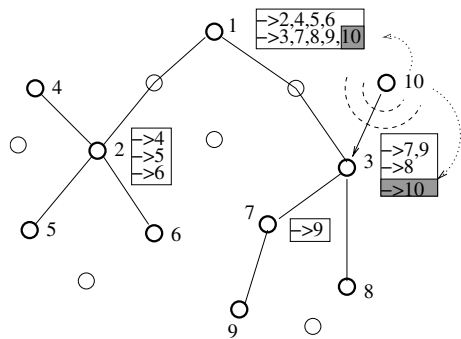
To discover the nodes running the distributed application, we use a very simple discovery mechanism. Service discovery in mobile ad hoc networks is still a topic of active research issue, and this paper is not trying to solve the general problem. Other discovery mechanisms can be used as well.

The service discovery mechanism works as follows. A node searching for a peer broadcasts a UDP packet – the search query. Those nodes that already run the distributed application and that receive the search query reply with a unicast packet. A timer is used at the inquiry side to avoid waiting for an answer for an infinite period of time. The time-out value is halved with every received reply. This action creates a balance between receiving as many replies as possible and having a short delay after the last reply. In the current implementation the node chosen to connect to is the node that replied first. Choosing the first reply is a coarse approximation of choosing the peer with the lowest delay.

### 4 Protocol description

In this section we describe the events that cause a node to react; the node's reaction then adapts the structure as needed. The events of interest are:

- node join – a node moves into the communication range of an existing structure; a special case is that a node starts the distributed application and does not discover any peers;
- network merges – when one wireless cloud moves into the range of another wireless cloud;
- node departure – when a single node moves away from the wireless cloud and/or loses connectivity with the other peers, or a node stops the distributed application;
- network splits – when a set of wireless nodes moves away from the rest of the nodes and thereby creates a new wireless cloud.



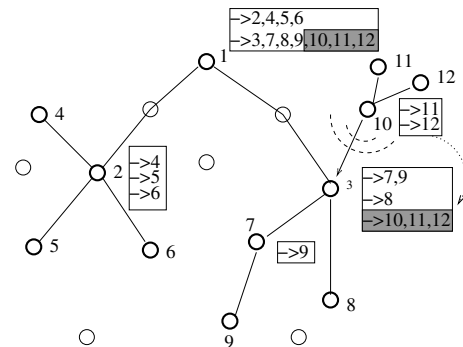
**Figure 3. Information propagation when a new node joins the tree**

Because network merges and node joins use the same adaptation mechanisms, they are discussed together in Section 4.1. The adaptation on node leave and network split events are described together in Section 4.2. During the re-configuration process multiple nodes can connect to each other, degrading the tree structure to a graph. In Section 4.3 we describe the cycle detection and elimination mechanisms.

#### 4.1 Adding nodes to the tree structure

Figure 3 depicts the join of a new node to the tree structure. When a new node joins the network, it uses the discovery mechanism to detect other peers in the network. When a new peer is discovered, the node connects to the tree. Information about the location of the new node in the tree is transmitted to the root of the tree (in the example, node 3 updates its table to include node 10 and propagates the information to node 1). The information change in the tables is highlighted by the dark boxes in Figure 3. If no peer is discovered, the node assumes it is the root of a tree. Since the service discovery returns the first node that responded (if it exists), there is a good chance that a new node joins the tree as a child of a node that is physically close. However, there is no such guarantee, but such a guarantee is not necessary. Note that any subsequent communication between nodes 10 and 3 (in the above example) is overheard by any node within range (possibly node 1), but the current design does not exploit this information for optimization or adaptation.

When one tree structure enters the communication range of another tree (e.g., when two wireless clouds merge), the two trees must organize as one big tree. Therefore it is necessary that the root of a tree probes for new trees (or nodes). The root is the best candidate for this task since the root has knowledge about all the nodes in its subtrees and can avoid connecting the tree to itself. The tree merge is depicted in Figure 4. When node 10 detects that another tree is in the communication range (not necessarily direct neighbor), it connects to the newly found tree (e.g., node 10 connecting



**Figure 4. Information propagation when two trees connect**

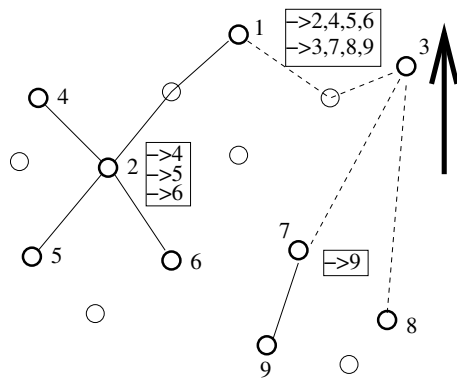
to node 3 in Figure 4). The information table of the *connected root* (node 10) is propagated to the root of the new tree (to node 1). The information propagation is highlighted by the shaded boxes. After the information has propagated through the tree, nodes 3 and 1 know how to reach nodes 10, 11 and 12.

The tree structure may break while an update operation is in progress. In this case the information is up to date in the subtree below the break. Consider the scenario depicted in Figure 4 and subtree 10-11-12 has connected to node 3. Information is up to date at node 3, but not yet at node 1. The link between node 1 and 3 breaks. The subtree with the root at node 3 now either rediscovers the tree with the root node 1 or forms a separate tree (with node 3 as root). If node 3 rediscovers node 1, it connects again. The information about the subtree 3 is now passed to the root, and the system stabilizes with all information up to date. If the subtree remains alone, the information is still up to date.

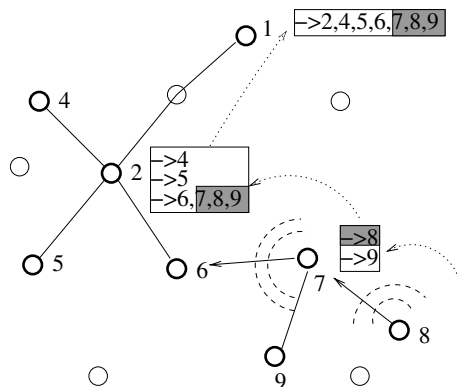
Another scenario may occur when the root of a tree is not in direct connectivity range of a new tree, but one of its children is. In this case, the service discovery packets from the root are carried by the underlying network protocol over one or more children. The two trees detect each other and connect as described earlier in this section. This solution is suboptimal, since some links will be traversed twice. It would be easy to detect such situations, but an implementation has been postponed until larger setups demonstrate the value of such an optimization.

#### 4.2 Deleting nodes from the tree structure

This section describes the steps taken when nodes move out of the communication range of the network (depicted in Figure 5, building on Figure 2). When a node moves away (e.g., node 3 moves in the direction of the arrow), the tree structure is broken. If the departing node can foresee its departure, the departing node sends a disconnect message to the peers in the tree structure. If the departing node has no means to foresee the departure, the remaining nodes must discover the break in the tree structure and reorganize.



**Figure 5. A node moves away and leaves the tree**



**Figure 6. Reorganization of the remaining nodes.**

A reorganization scenario is depicted in Figure 6. The children of the moved node (7 and 8 in Figure 6) detect that they lost connectivity with the parent. This event triggers the discovery mechanism as described in Section 3.

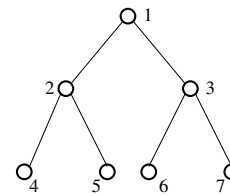
If a node departs or fails, the network may split; this means the remaining nodes form two or more separated network clouds. In such a scenario, the nodes inside each network cloud form a separate tree structure. When the network clouds merge again, the two tree structures detect each other and merge as described in Section 4.1.

### 4.3 Cycle detection and elimination

During the reorganization process two nodes or subtrees can connect to each other. A cycle detection and elimination mechanism maintains the tree structure.

## 5 Preliminary evaluation

We report on a number of experiments to test the agility of the application structure with both announced and unannounced disconnections. The performance metric we use in these experiments is the delay between the occurrence or



**Figure 7. The logical topology of the experimental test-bed.**

detection of the disconnect event and the reorganization of the system.

To evaluate the application and the organization of the nodes we use a wireless ad hoc network with seven nodes. The participating hosts are laptops running the Linux OS. The wireless network is formed using the IEEE 802.11b compliant Cisco Aironet 350 cards. To enable multi-hop operation we use the UCSB (University of California at Santa-Barbara) implementation (version 0.1b) of the AODV protocol. The logical topology the our experiments is depicted in Figure 7. To emulate different node location scenarios, we filter the network packets between different nodes.

When a node foresees a departure from the wireless cloud, it sends a disconnect message to announce its departure from the tree structure. However such a message is not always possible and the remaining nodes must detect the departure of peers. The desired case is when a node announces its disconnection, but it is important unannounced departures are handled gracefully.

### 5.1 Adapting to announced disconnections

To evaluate the adaptation of the system in response to announced disconnection events, we measure the application's performance in different scenarios. The common part of each scenario is that the node that disconnects is node 1 (see Figure 7). The difference between the scenarios is the reachability of different nodes:

- A: Node 2 can communicate only with the nodes in its subtree (4 and 5) and with node 3. Node 3 can communicate only with its subtree (6, 7) and node 2.
- B: Node 2 can communicate only with its subtree and nodes 6 and 7. Similarly, node 3 can communicate only with its subtree and nodes 4 and 5.
- C: The nodes can communicate with each other without any restrictions.

For every scenario we took five measurements. Table 1 shows the shortest and the longest measured time for the three scenarios.

The differences in the reorganization time are determined by the number of connections attempted during the reorganization. Multiple connections are attempted when a cycle is detected and eliminated. Where the cycle is cut, a

Scenario	Time (ms)	
	Min.	Max.
A	739	2838
B	746	1721
C	1028	3181

**Table 1. Time to adapt to announced disconnections.**

child node is separated from its parent. The freshly separated child node searches for other nodes to connect to. If no nodes are found (other than those in its own subtree), the node assumes the root role. In the measurements we consider as reorganization time the time from detection of the disconnect event until a node assumes the role of root after cycle elimination.

## 5.2 Adapting to unannounced disconnections

It is not always possible to announce the departure or failure of a node. In cases in which the departing node does not send a disconnect message, the remaining nodes must detect the break in the tree structure and reorganize.

The easiest way to detect the departure of a node is to sense when the connection to the node is no longer in place. The connection loss sensing can be done using application-level timeouts or enabling the `SO_KEEPALIVE` flag of the TCP connection. In the later case, the operating system's connection timeout mechanism is used.

In our experiments we use the operating system's keep-alive mechanism. A detailed description of how the TCP keep-alive works can be found in [4]. The TCP keep-alive behavior can be tuned using the *time* keep-alive timer (default 7200sec.), the *probes* parameter (number of times keep-alives are sent before dropping the connection, default 9) and the *interval* parameter, which defaults to 75 seconds.

The negative side of using the TCP keep-alives to detect failures is that tuning the parameters will influence the behavior of all the TCP connections on that operating system. We therefore advocate the implementation of an application layer failure-detection mechanism or the implementation of a operating system service for peer failure detection.

The length of a keep-alive packet, including the lower layer headers, is 66 bytes. The expected consumed bandwidth of an idle connection is the ratio of the packet length over the inter-probes interval. The variation of the bandwidth with the interval time is shown in Table 2.

The consumed bandwidth grows linearly with the number of concurrent connections. In our test environment, we had six concurrent connections, thus *wasting* 3168bps out of a 11Mbps channel. Since this is 0.02% of the bandwidth, the intrusion of this technique can be ignored. Using this technique in networks with 100 nodes, the keep-alive messages consume 52.8kbps, resulting in wasting 0.48% of the bandwidth. In the setup described in Figure 1, 50 percent

Timeout (s)	Bandwidth (bps)
7200	0.07
3600	0.14
60	8.80
30	17.60
5	105.60
1	528.00

**Table 2. The bandwidth consumption of keep-alive messages.**

unavailable nodes results in 10 percent more route request packets.

This reasoning does not take the interference caused by one transmission on the other hosts into consideration. We expect the bandwidth consumption for a large network in real life to be bigger than just 0.48%. But once ad hoc networks of this size are fielded, it will be advantageous to realize that the timeout parameters can be dynamically tuned on each host to cope with the changing environment conditions.

To validate our reasoning on the bandwidth consumption, we report on the influence of the keep-alive packets on other TCP connections. For measurements we use the test-bed described earlier in Section 5. The application running on the seven nodes imposes the topology shown in Figure 7. Node 4 is multi-homed, acting as a gateway between the wireless and the wired worlds.

We measure the bandwidth experienced by a TCP connection from node 5 to a machine on the wired network. We perform five types of measurements. Each type of measurement is repeated 20 times. The measurements are made in the following scenarios:

1. No other TCP connections are established between the wireless hosts. The default values for keep-alive parameters are used.
2. Six TCP connections are established between the wireless hosts, as shown in Figure 7. The default values for keep-alive parameters are used.
3. In this scenario we try to obtain the maximum agility. Six TCP connection are established between the wireless hosts. The keep-alive timer is set to 1s, the interval time is set to 1s, and the number of probes set to 1.
4. Six TCP connection are established between the wireless hosts, the keep-alive timer is set to 1s, the probe interval timer is set to 1s, and the number of probes is set to 3.
5. In this scenario we try to obtain the same behavior as in the previous one, but with different parameter values. The keep-alive timer is set to 3s, interval time 1s, number of probes 1.

Scenario (time/probes/interval)	Avg. bandwidth (KBytes/s)	C.O.V.
1 (7200/9/75)	372.74	6%
2 (7200/9/75)	376.28	5%
3 (1/1/1)	363.68	11%
4 (1/3/1)	361.03	9%
5 (3/1/1)	360.75	8%
Overall	367.06	8%

**Table 3. The bandwidth experienced by a TCP connection in the presence of keep-alive messages.**

Table 3 presents the results measured. Given the low values for the coefficient of variation (C.O.V.), we present the bandwidth figures as an average over 20 measurements.

In the third scenario we tried to tune the TCP keep-alive to have the lowest reaction time. During these experiments, 25% of the TCP connection from node 5 to the host on the wired side were dropped due to keep-alive timeouts. Some of the TCP connections between the wireless hosts timed out as well. The application running on the wireless hosts was forced to reorganize the hosts, although the host were on-line for the entire duration of the tests and no application crash was experienced.

Setting a higher value for the number of keep-alive probes (3) helped the system maintain a stable state and no connection timeouts were experienced.

The TCP keep-alive mechanism can be used as a failure-detection mechanism without much intrusion. The bandwidth consumed by the keep-alive messages is insignificant for a small number of nodes, but in an ad hoc network, we expect-it to grow more than linearly with the number of hosts.

A balance must be found between the agility of the system and the stability. Reacting quickly on idle connection can prematurely cut perfectly good connections. Reacting later on connections losses can diminish the system's agility to react to changes.

To measure the agility of the system on unannounced disconnections, we repeat the measurements from Section 5.1 scenario C. In this scenario, the nodes can communicate without any restriction. For these measurements, the disconnections are unannounced. The TCP timeout is used to detect node departures or node failures. The TCP keep-alive parameters are set for a timeout of 3 seconds: the keep-alive timer set to 3s, interval time 1s and the number of probes 1. We measure the delay from the detection of the connection timeout until the reorganization of the system. Over five conducted measurements, the shortest time for the system reorganization is 1009ms, the longest time is 3285ms. To obtain the reorganization time from the node failure to the system reorganization, we add 3 seconds to the measured delay, thus obtaining 4009ms for the quickest reorganization and 6285ms for the slowest measured reorganization.

The measurements conducted during the preliminary evaluation show that the system is agile enough to react to connectivity changes in a highly mobile environment. The application reorganizes on announced as well as on unannounced disconnections with a delay by the order of seconds. These delays are typically non-intrusive to users.

## 6 Related work

The general idea of imposing a structure over the participating peers is not novel. Previous work regarding peer organization has been done at different levels.

The Astrolabe system[5] organizes the peers participating in large-scale, highly dynamic distributed applications. The participating nodes connect over the Internet. The hierarchy in Astrolabe is created based on the DNS names of the peers. The agility of Astrolabe is reported to be on the order of tens of seconds. This reconfiguration time is however fairly large for mobile ad hoc networks, but it is not clear how Astrolabe would perform in a wireless setup (a fair portion of the time may be due to slow remote responses; details are not available). To the best of our knowledge, the Astrolabe hierarchy must be configured manually, whereas our systems finds the peers automatically.

Several clustering algorithms have been proposed in the recent past [6, 7, 8, 9]. Clustering is used to alleviate the problem of flooded management packets. The broadcast packages are flooded just in the clusterheads network. In some approaches the clusterheads form a separate network (different radio channels or different signal powers); in other approaches additional nodes are taken in the clusterhead network to guarantee connectivity between clusterheads. The first approach is not applicable in our environment, since our nodes have all the same radio characteristics. The second approach gathers a large number of nodes (as the network grows) in the clusterhead network. This way, the number of nodes involved in broadcasts grows fairly large. In contrast, our solution uses unicast messages all along for the detection of missing nodes and not broadcast floods.

The Bluetooth[10] technology groups the participating nodes at the MAC (Medium Access Control) layer. The smallest grouping unit of the Bluetooth network is the piconet. A piconet allows participation of a maximum of seven nodes. One of the nodes in a piconet is the master, and the remaining nodes connect directly to the master. A node can be slave in several piconets, but master only in one. Several piconets can connect forming a scatternet. The structuring of the nodes is formed by connecting several tree-shaped piconets together. The Bluetooth specification is however incomplete when it comes to describing scatternet formation. To the best of our knowledge, at this moment multiple piconets can only be connected together at the application layer.

Related work in the overlay networks area include the CAN (Content-Addressable Network) [11], Chord [12], Pastry [13] and Tapestry [14] systems. In these systems, the network overlay has the function of a distributed hash table. In contrast to the previous approaches, the overlay described in this paper is used to avoid extensive usage of broadcast packets by structuring the communication among a set of nodes.

Plank et al.[15] present in their work a method of predicting and selecting connection timeouts. Such a timeout selection mechanism can be beneficial for systems in environments where network conditions change frequently.

The Aura project [16] is an example of a project that deals with adaptation in mobile (managed) environments. Solutions to the service discovery problem that work in such an environment may be extensible to the setup described here. However, the emphasis of our work is to devise an adaptive structure that allows an application to conserve communication bandwidth for application-level data transfers, whereas the Aura project focuses on general application adaptivity.

## 7 Concluding remarks

This paper describes the design, implementation and preliminary evaluation of an auto-organizing distributed application. The application maintains a tree so that nodes do not have to rely on broadcast operations to find other participating nodes. The overall protocol is fairly simple, but this simplicity results in agility (and a manageable implementation). As the measurements in a controlled testbed indicate, the application is able to react rapidly to node joins and departures. The cost (in terms of bandwidth devoted to tree maintenance) appears acceptable for medium-size ad hoc networks.

Application adaptivity as realized by this protocol is another approach to deal with the changing environments of ad hoc networks. We attempt to address one of the crucial aspects - the need to avoid all-to-all communication when possible. Applications will be able to operate in such environments only if enough bandwidth of the communication fabric is available. The collaboration tool presented here shows a practical path to support applications in ad hoc environments.

## Acknowledgments

We thank Marcos dos Santos Rocha and Thomas Wieland of Siemens AG for their contribution.

## References

[1] P. Gupta and P. R. Kumar, "The Capacity of Wireless Networks," in *IEEE Trans. Inf Theory*, vol. 46, pp. 388–404,

Mar. 2000.

- [2] J. Li, C. Blake, D. S. J. D. Couto, H. I. Lee, and R. Morris, "Capacity of Ad Hoc Wireless Networks," in *Mobile Comp. and Networking*, pp. 61–69, 2001.
- [3] C. E. Perkins, E. M. Belding-Royer, and S. Das, "Ad Hoc On Demand Distance Vector (AODV) Routing." draft-ietf-manet-aodv-10.txt, Internet Draft, IETF, Mar. 2002. (Work in progress).
- [4] W. R. Stevens, *TCP/IP Illustrated, Volume 1: The Protocols*, vol. 1. Addison-Wesley, Dec. 1993.
- [5] R. van Renesse and K. Birman, "Astrolabe: A Robust and Scalable Technology For Distributed System Monitoring, Management, and Data Mining." Submitted to ACM TOCS, Nov. 2001. url: citeseer.nj.nec.com/robbert01astrolabe.html.
- [6] M. Gerla, T. J. Kwon, and G. Pei, "On-demand routing in large ad hoc wireless networks with passive clustering," in *Proc. Wireless Comm. and Networking Conf.*, vol. 1, pp. 23–28, Sept. 2000.
- [7] P. Sinha, R. Sivakumar, and V. Bharghavan, "Enhancing Ad Hoc Routing with Dynamic Virtual Infrastructures," in *Proc. IEEE INFOCOM 2001.*, vol. 3, pp. 1763–1772, Apr. 2001.
- [8] M. Chatterjee, S. K. Das, and D. Turgut, "WCA: A weighted clustering algorithm for mobile ad hoc networks," *Cluster-Computing* 5, pp. 193–204, 2002.
- [9] S. Basagni, D. Turgut, and S. K. Das, "Mobility-Adaptive Protocols for Managing Large Ad Hoc Networks," in *Proc. Intl. Conf. on Comm.*, vol. 5, pp. 1539–43, June 2001.
- [10] B. SIG, "Specification of the Bluetooth System," Feb. 2001. Specification Volume 1.
- [11] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker, "A Scalable Content-Addressable Network," in *Proc. 2001 Conf. App. Tech. Arch. Proto. for Comp. Comm.*, pp. 161–172, ACM Press, 2001.
- [12] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *Proc. 2001 Conf. App. Tech. Arch. Proto. for Comp. Comm.*, pp. 149–160, ACM Press, 2001.
- [13] A. Rowstron and P. Druschel, "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems," *Lec. Notes. Comp. Sci.*, vol. 2218, pp. 329–350, 2001.
- [14] B. Y. Zhao, J. Kubiatowicz, and A. D. Joseph, "Tapestry: An infrastructure for fault-tolerant wide-area location and routing," Tech. Rep. UCB/CSD-01-1141, Comp. Sc. Div., U.C. Berkeley, Apr. 2001.
- [15] J. R. Plank, R. Wolski, and M. Allen, "The effect of timeout prediction and selection on wide area collective operations," in *IEEE Intl. Symp. Network Comp. App. (NCA)*, Oct. 2001.
- [16] D. Garlan, D. Siewiorek, A. Smailagic, and P. Steenkiste, "Project Aura: Towards Distraction-Free Pervasive Computing," in *IEEE Perv. Comp., special issue on "Integrated Pervasive Computing Environments*, vol. 1, pp. 22–31, Apr.-Jun. 2002.