Tree Based Broadcast in Ad Hoc Networks

Alpár Jüttner[1,2]**, Ádám Magi**[1]

[1]Ericsson Research,

Traffic Analysis and Network Performance Laboratory

Budapest, Hungary

[2]Department of Operations Research,

Eötvös University,

Pázmány Péter sétány 1/C, Budapest, Hungary, H-1117

e-mail: {`Alpar.Juttner, Adam.Magi`}`@eth.ericsson.se`

September 9, 2004

Abstract

Although broadcasting using tree structure established in a network is a well known and widely used technique, it is typically claimed to be inappropriate for ad hoc networks, being the maintained tree very sensitive to network changes. On the contrary this paper presents an efficient tree based broadcasting scheme, which is reliable and stable even in case of the ever changing network structure of the ad hoc networks.

To achieve this, first, a novel method is presented to maintain a spanning tree in an ad hoc network in a fully distributed, on-line and asynchronous way. Once the tree is established the broadcast itself is performed based on this tree. Some further improvements on the basic algorithm are also presented that reduce the resource requirements even more, increase the stability of the tree, enable the mobility of the nodes to be taken into account and make the method more configurable.

As it is shown by simulation, the obtained broadcast scheme is stable, reliable and it uses small amount of resources: the acyclic structure of the broadcast tree ensures that the nodes get the broadcast messages only once, so the broadcast needs little bandwidth and the nodes need not store the recent broadcast messages, reducing the computational and memory requirements.

As a byproduct a technique is proposed to measure the mobility of the nodes. This technique needs no additional GPS device or any geographical information but it is based on the stability of the links of the node.

Keywords: *Ad-hoc networks, Broadcast, Multicast, Routing algorithms, Wireless LANs.*

1 Introduction

Mobile ad hoc networks (*MANET*s) are communication networks formed by mobile radio-equipped terminals (e.g. laptop computers, PDAs) without a fixed infrastructure in such a way, that each communicating device (hereafter *node*) can serve as a router for the others. MANETs present a number of specific challenges for the communication protocols used to operate them. First and foremost, the protocols have to take into account the constant changes in the network topology due to node mobility, and in most applications the protocols should use fully distributed algorithms since no central entities are available due to the lack of fixed infrastructure. On the other hand communication should be kept to a minimum over the air interface which is often a scarce resource in these networks, and energy expenditure has to be minimized since most nodes are hand-held devices operating on limited battery.

In the present work the focus is on network-wide broadcast, that is a procedure in which some information is passed on to all participating nodes.

This paper proposes a new solution to the problem of broadcasting in ad hoc networks. The idea is to maintain a spanning tree in the network, and broadcast using this tree by forwarding a broadcast message not to all neighbors but only to those who are neighbors in this tree as well. Since a tree is acyclic, each message is received only once by each node, giving two advantages over the existing methods. Firstly, it is needless to store the previous broadcasts in order to avoid endless multiplications of the broadcast messages along a cycle of links. Only the originator node of a broadcast message need to store it and pay attention to whether its broadcast was probably successful or not if it is of great importance. Secondly, it is very economical considering how many times a broadcast message should be forwarded. Supposing that the broadcast tree is in stable state, a broadcast message should be forwarded only once per a node in case of one-to-one broadcast forwarding. If local broadcast is available, a broadcast message should be forwarded as many times as many non-leaf nodes there are in the broadcast tree.

Broadcasting or multicasting in a network using a tree is not a novel idea. Several algorithms can also be found in the literature that aim at automated and distributed building of spanning tree in the network (see Section 2.4). These methods are designed to be applied in networks having fixed architecture. However, as it was mentioned above, application to ad hoc networks raise special demands about such an algorithm. Namely, no node knows the whole network structure, but only their direct neighbors. So, the nodes have to build the spanning tree by communicating with each other. Of course, we want to minimize this communication. The algorithm should treat each node in the same way, in order to avoid that the whole network relies on the reachability of one ore more dedicated nodes. We also want to minimize the data structure that each node has to store during and after the tree computation. Most importantly, *this algorithm should be able to cope with the continuous change of the network topology*. Links disappear and new links appear from time to time, damaging the tree or connecting two previously disconnected networks. The tree algorithm should react immediately to these changes.

In fact, due to these strong requirements, the tree based broadcast is widely considered to be inappropriate for ad hoc networks, being too difficult and resource consuming to maintain (neither connectivity nor the acyclic property are local characteristics) and being too sensitive to the link failures.

On the contrary, as the main contribution, this paper presents a novel distributed spanning tree algorithm that meets all the above requirements. The proposed *TreeCast* method is *fully distributed* and *fully decentralized*. It does not need dedicated nodes or any kind of "leader election" process. Each node executes exactly the same (very simple) process. It is also *asynchronous*, that is the nodes do not need a common timer. Each node has to know only about its direct neighbors. No other information is needed about the actual network.

The TreeCast algorithm is *on-line* in the sense that whenever a new node appears or two separated local networks become connected because a new link appears, the algorithm automatically extends the tree and whenever the tree breaks up because a link ceases, it will repair the tree automatically. The algorithm handles the simultaneous link setups and ceases, and gets quickly into a stable state even after significant changes in the network structure. It works with only a small amount of communication and computing resources.

The TreeCast algorithm is also able to take the *mobility of the nodes* into account. A method is given to measure this property based on the stability of the node's links. Moreover the algorithm allows one to explicitly prescribe some nodes to be a leaf in the tree, if it is necessary to prevent a node from spending lot of resources on broadcasting.

Once the spanning tree is built, the broadcast mechanism is quite simple, as it is seen in details in Section 3.2.

In the case when the network changes really frequently, the proposed spanning tree algorithm may need some fine-tuning in order to increase stability of the tree (i. e. to decrease the time required to reach a stable state even after fundamental change in the network) and to decrease required communication even more. Section 4 proposes several improvements for this purpose. The proposed mobility measure is described in Section 4.3. After some discussion in Section 5, numerical results of the performance of the algorithm under real-life conditions are shown in Section 6. Finally the conclusions are drawn in Section 7.

2 State-of-the-Art

Network-wide broadcast has two major uses, namely the dissemination of control (e.g. routing related) information and that of user data. According to this distinction we can formulate different requirements for the two types of broadcast.

Broadcast of user data often requires reliability to some degree. This is because application-level acknowledgments from all peer nodes would unnecessarily overload the network. These broadcast algorithms can sometimes benefit from the routing information gathered by the underlying (unicast) routing algorithm. Since these broadcast events are rare and they constitute user traffic they can be relatively expensive in terms of resource usage.

Control data broadcast on the other hand typically needs less reliability. The most widespread use of network wide broadcast in MANET routing protocols is the propagation of "route query" type packets. In most cases it is sufficient that the broadcast reaches a single node that is aware of the location of the node ID in the query. Thus the requirement is that the broadcast should not *systematically* leave out some nodes or parts of the network. Usually these broadcast algorithms have to rely on very little available routing and topology information. Also they can be quite frequent (e.g. "route query" each time a priorly unknown node is contacted) and they constitute control traffic. Thus resource usage has to be kept to a minimum.

Network-wide broadcast mechanisms often rely on *local broadcast* if it is available. This is a mechanism by which the message is sent to all of the node's neighbors for further distribution. Such a mechanism is sometimes built into the air interface e.g. in IEEE 802.11b the most popular air interface for MANETs at the moment. If no local broadcast is available then a node has to send the message to all of its neighbors (or a subset of those) one by one. However one must always keep in mind that local broadcast is always unacknowledged and suffers from the loss of messages because of their collisions, thus much less reliable than point-to-point message distribution.

It was shown in [1] that achieving network wide broadcast in a fully distributed, low energy, mobility-aware way is not a trivial task.

2.1 Flooding

The current method of achieving network-wide broadcast in a MANET is called *flooding* (or classical flooding) [2]. In this procedure each node receiving a copy of a broadcast message first checks whether it has already received it. If so, the message is silently discarded. If the message was received by the node for the first time then it is sent to all of its neighbors.

In [1] it is shown that the classical flooding algorithm has several drawbacks. First of all it is rather costly in terms of air interface usage and energy expenditure. This is because each node receives the broadcast message (in an ideal case) from each of its neighbors while theoretically one reception per node would be sufficient.

Secondly, flooding is not reliable. This is because neighboring nodes try to use the unreliable local broadcast all at the same time. Thus collisions are likely to occur and remain mostly unnoticed. So, the multiple transmissions may have a reverse effect leading to the total loss of the broadcast message.

2.2 Extended Topological Knowledge Based Solutions

A number of network-wide broadcast algorithms [3, 4, 5] are based on each node having some non-local knowledge of network topology. Typically the connectivity of the node's k hop neighborhood for some small integer k is assumed to be known. The algorithms use this partial topology knowledge to reduce the number of nodes which relay the broadcast message.

The authors of [3] propose two related solutions *Self Pruning* and *Dominant Pruning*. In the first one each node decides whether it should refrain from forwarding a broadcast message based on the knowledge of its own neighborhood and the neighbors of the sending node (which is included in the broadcast message in this solution). The second algorithm builds on each node n knowing its two hop neighborhood, and computing a forwarding set of its neighbors. Nodes are selected into this set in such a way that they cover n's entire two hop neighborhood. When forwarding the broadcast message n communicates the forwarding set as well, and only the nodes included in it will forward the message. In [4] the *LENWB* algorithm is presented, in which each node computes whether it should forward the message based on the knowledge of its two hop neighborhood and the degree of its one and two hop neighbors. The algorithm is shown to be reliable in the case that local broadcast is reliable.

The Internet draft [5] presents the *Bordercast Resolution Protocol* as part of the Zone Routing Protocol framework. The algorithm builds on the concept of routing zones ($k \geq 2$ hop neighborhood of each node n), arbitrary broadcast inside each zone, forwarding border nodes (a subset of n's k hop neighbors), and full knowledge of n's $2k$ hop neighborhood topology.

These solutions expect the nodes to acquire and store large amounts of information about the topological or geographical structure of the network. In methods that are based on non-local topology knowledge the node has to store and keep up-to-date its 2, 3 or 4 hop neighborhoods. This requires substantial communications between neighboring nodes.

2.3 Geographical Location Based Solutions

Another set of algorithms [1, 6] assumes each node in a network to be equipped with a GPS device. The nodes communicate their location to their neighbors and this geographical knowledge is exploited to reduce the number of forwarding nodes.

In [1] the authors propose, among several others, two schemes which make use of geographical data. In the first one (*Distance-Based Scheme*) only the nodes' distance (relative to the radius of their radio coverage area) is used, while the *Location-Based Scheme* assumes that all nodes know the geographical location of their neighbors. In both cases each node decides whether to forward a broadcast packet based on the size of the expected extra coverage area.

The *Internal Node Based Broadcasting* algorithm of [6] supposes that each node has knowledge of all of its neighbors' geographical coordinates as well as their degrees. With this knowledge each node decides whether or not it is an internal node with regards to broadcasting. Only internal nodes relay the broadcast messages.

The methods that are based on the knowledge of the exact location of the nodes need an additional GPS hardware component, which is expensive and quite unrealistic in case of many devices such as wireless mice, keyboards or headsets. Why should anyone carry 4 or 5 GPSs with oneself all the time? (One in his notebook, other ones in his mobile phone, headset, palm-top or camera.) Moreover these methods suppose that visibility can be estimated merely from the position of the nodes, i.e. it mainly depends on the distance of the nodes. This is realistic only if there are no shading objects, that is, the users are in a plain field.

2.4 Algorithms for Tree Creation and Maintenance

One more common drawback of all the methods mentioned above is that the nodes have to store the broadcast messages to be able to respond only to the first reception of each message. Otherwise the messages could arbitrarily multiply in the network. The most evident way to avoid the multiple reception of a message is to forward the broadcast messages only on an acyclic subset of the links, i.e. on a tree.

There are several algorithms in telecommunication networks for creating and maintaining tree structures such as the spanning tree algorithm of bridged Ethernet networks [7]. Unfortunately most of these algorithms are designed to work in fairly stable networks and not the constantly changing topology of an ad hoc environment.

There is an important body of work dealing with multicast trees [8, 9, 10, 11, 12] and more specifically multicast trees in ad hoc networks (e.g. [13]). These algorithms however usually maintain several rather sparse trees over a pre-existing routing architecture. In the current work we focus on broadcast of primarily control information which implies one tree spanning all of the network built without prior unicast routing data.

The protocols for the widely examined leader election problem (e.g. [14]) typically generate a spanning tree as a byproduct. Moreover, even the minimal cost spanning tree can be found by an efficient distributed algorithm[16]. These algorithms however deal only with the construction of the tree, they are not suitable for handling the topology changes. The algorithm presented in [15] implicitly also involves in constructing a spanning tree. Although the algorithm is designed for wireless ad hoc context, it is not appropriate for the maintenance of the tree.

There are indeed some ad hoc routing protocols that rely on some underlying tree or forest topology. The DST protocol [18] maintains a spanning tree and uses it to forward data packets, however some aspects of tree maintenance (such as tree merging) involves centralized decision and extensive signaling. DDR [17] on the other hand uses a dynamically maintained forest. Although the authors prove that their algorithm always yields a forest in one step, they do not even aim at constructing a single tree for the network. Moreover the performance of the protocol in an unreliable environment (e.g. possible loss of beacons) is also questionable.

3 The TreeCast Algorithm

This section presents a communication-efficient algorithm to maintain a spanning tree and a broadcasting method based on the spanning tree.

As we mentioned above, the full broadcast mechanism includes two separate tasks: the maintenance of the Broadcast Tree and the broadcast process itself using this tree.

3.1 Maintenance of the Broadcast Tree

Each node is supposed to have an individual ID referred as *NodeID*. NodeID's are supposed to be ordered, that is NodeID's can be compared. For example, the IP address is suitable for this purpose.

Each node belongs to a single Broadcast Tree. These trees also have an ID called *TreeID*. A TreeID is a pair of an integer *serial number* and a NodeID and it is generated by the algorithm. The NodeID of a node and the NodeID in its TreeID will typically be different.

TreeID's are lexicographically ordered, i.e. they also can be compared and the comparison is defined as follows. If their serial numbers are different, then the TreeID having the greater one is defined to be greater. If their serial numbers are the same, then the TreeID having the greater NodeID is defined to be greater.

In addition to its TreeID, each nodes stores that from among its links, which ones belong to the Broadcast Tree. For the sake of simplicity, these links are referred as *Broadcast Link*.

Each link can be added to or deleted from the Broadcast Tree by its end-nodes. Both end-nodes have up-to-date information about whether the link is a Broadcast Link or not. This can be achieved e.g. by an acknowledged message, that has precedence over all other messages related to the broadcasting.

Now we are ready to present the algorithm establishing and maintaining the Broadcast Tree. It is going to be done by describing how a node reacts to different events such as the setup or ceasing of a link or getting a message from one of its neighbor.

- Whenever *a new link appears* and the two newly adjacent nodes recognize that their TreeID's are different (i.e. they belongs to different Broadcast Trees), then this link becomes a Broadcast Link and the two nodes *merge* their Broadcast Trees. The TreeID of the merged tree will be the greater one of their TreeID's. Let G denote the node having greater TreeID and let S be the other node. Thus, S updates its TreeID to the TreeID of G and starts a "NewID" process described later. This process will update the TreeID's of the remaining nodes.

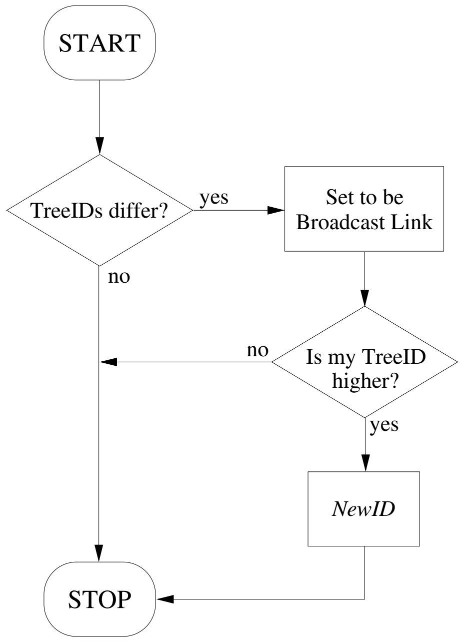

Figure 1: New link process

- If *a Broadcast Link ceases*, that is the Broadcast Tree breaks up into two parts, one of the parts will get a new TreeID. First, the two endpoints of the ceased link make decisions about whose tree will get a new TreeID. Obviously, this decision must be made without communication. However, it is worth mentioning that if both of them will decide to generate new TreeID, it causes no problem, only the process will be a bit more resource consuming. For example a simple way to do this is that the end-node having greater NodeID executes this process. Namely, this node A generates a new TreeID (this process is described later), sets its TreeID to this value and starts a "NewID" process.

  The other end-node B of the ceased link waits for getting a new TreeID for a certain time. If no new TreeID arrives, then it means that node A has totally disconnected from B. Then node B also generates a new TreeID in the same way as A did, sets its TreeID to this value and starts a "NewID" process.

- The *generation of a new TreeID* is quite simple. If the serial number of the previous TreeID was s, then the new TreeID will be $(s + 1, N)$, where N is the NodeID of the node that executes this process. If a new node comes into being, its first TreeID is $(0, N)$, where N is its NodeID. This TreeID generation ensures, that a newly generated TreeID is certainly unique and it is greater then the previous TreeID of the node was.

- The "NewID" process simply sends an *"I have new TreeID"* message including the new TreeID to each of its neighbors excluding the node where it received this new TreeID from. If the TreeID was generated by this node, it sends this message to all of its neighbors.
- When a node receives an *"I have new TreeID"* message on the link L, it compares the received TreeID with its own. If the received one is smaller, then it must be obsolete information, so it does nothing. If it is greater, then the node set L to be a Broadcast Link, updates its TreeID and starts a "NewID" process. If these TreeID's are equal, then it means that the node gets this TreeID for the second time, indicating that the "Broadcast Tree" together with L would contain a cycle. To fix this, it sets L not to be a Broadcast Link.

Claim 1 *Whenever the network gets into a stable state, the above algorithm establishes a tree that spans all the nodes of its connected components within $(d + 1)T$ time, where d is the maximum hop-by-hop distance between the node pairs of the component and T is the time required by sending an* "I have new ID" *message and handling it by the receiver node.*

Proof. After the time of sending d messages each nodes receives the highest generated TreeID (s, N). After the time of one more message sending, the nodes do not generate new messages related to the tree maintenance, so the set B of Broadcast Links stabilizes as well. The set B is exactly the set of links on which the TreeID (s, N) was forwarded to a node for the first time, so B forms a connected and acyclic graph. □

3.2 The Broadcast Mechanism

Once the Broadcast Tree is established, the broadcast itself is quite simple. A broadcast message consists of a TreeID and the message itself. When a node receives a broadcast message, it compares the TreeID of the message with its own. If they are different, then it deletes the message since it is obsolete. If they are the same then it forwards the message on each Broadcast Link but the one it got the message from. Only the originator of a broadcast message has to store it. If the originator had to update its TreeID because of an "I have new TreeID" message within a certain time after broadcasting, it draws the conclusion that some nodes possibly did not get the broadcast message because of the inconsistency of the Broadcast Tree. Then the node decides on the rebroadcast of the message depending on its importance.

If local broadcast is available, the above method can be used with the difference that a Broadcast Message sent by the node N also contains the $NodeID$ of the node where the node N got this message from. A received broadcast is forwarded by a node M only if the M is not a leaf, the $NodeID$ in the message differs from M and if the message arrived from one of the tree neighbors of M.

Remarks

- If the Broadcast Tree is in transitional state then a node may occasionally get a broadcast message twice or more. It causes no problem since it rarely occurs and the messages cannot circulate in the network irrespectively of whether or not the Broadcast Tree gets into a stable state.
- On the other hand if the Broadcast Tree is in transitional state then a broadcast message may fail to reach every node. Unfortunately this is an unavoidable phenomenon that occurs with any kind of broadcast mechanisms if links break up. This algorithm ensures however, that the Broadcast Tree gets into stable state soon after the network changes, so the amount of lost messages will not be large. Moreover, as we mentioned above, the originator of the message will recognize the possible occurrence of this phenomenon by getting new TreeID within a certain interval after initiating the broadcast. So, in this case it may decide on re-sending the message.

4 Improvements on the Performance and the Scalability

This section presents some improvements that reduce the amount of resources required by the Broadcast Tree maintenance and update process. They also improve the stability of the tree.

4.1 Reducing the Number of TreeID updates

In the basic algorithm, whenever a Broadcast Link ceases, each network node has to update its TreeID. However the Broadcast Tree can be often repaired locally. The simplest case is when one of the endpoints of the ceased link is a leaf in the Broadcast Tree, i.e. it has no other Broadcast Links. Then, this node can repair the Broadcast Tree by setting one of its other links to be a Broadcast Link. If it has more links, then it can follow different strategies. For example it can use the oldest link (as it seems to be the most stable), the link that connects to a node having the most Broadcast Links (since this may ensure the “best” connectivity and since the Broadcast Trees having more leaves are preferred) or the link that connects to the least “mobile” node (see Section 4.3).

4.2 Further Improvement on the Stability of the Tree

Whenever the Broadcast Tree gets a new TreeID, the updating process almost completely changes the Broadcast Tree. If the stability of the Broadcast Tree is of great importance, the following modification

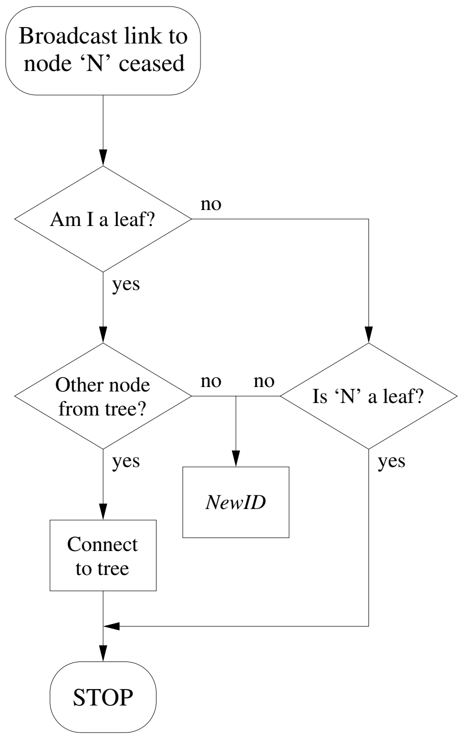

Figure 2: Improved “link ceasing” process

can be used that significantly reduces the difference between the Broadcast Tree before and after a TreeID update.

The modification is in the *“NewID”* process. Namely, first the node sends *“I have new TreeID”* messages only on its Broadcast Links. Then it waits a certain amount of time $NoTreeDelay$, and finally it sends the *“I have new TreeID”* message to its remaining neighbors. With this technique, if two nodes are separated and one of them generates a new TreeID there is a chance that it will reach most of the other nodes on the original tree links before the delayed messages radically reshape the Broadcast Tree.

4.3 Taking Mobility into Account

4.3.1 Measuring the mobility

First, the “mobility” of a node has to be measured. This can be done in several ways, for example based on the physical location of the node or based on how often its links disappear or new links appear. We chose this latter one. We give three alternative ways to measure this quantity.

- *Exponential Mobility.* Using this method, the mobility of a node decreases exponentially with the time but it increases by 1 whenever a new link sets up. For a precise definition, suppose that new links of a node N set up at time instances $t_1, t_2, \cdots, t_k$. (Some of them may have ceased and set up again. These links appear twice or more in this list.) So, the mobility factor of node N at the time instance T is

$$M(N) := \sum_{i=1}^{k} F^{T-t_i}, \tag{1}$$

where $F < 1$ is a parameter controlling how fast the mobility decreases with the time.

- *Reactive Mobility Measure.* This measure is similar to (1) with the difference that the increment at a link setup is not constant amount, but the less mobile the newly appeared link is, the greater the increment. To give a formal definition, suppose that new links of a node N set up at time instances $t_1, t_2, \cdots, t_k$, and the mobility of the corresponding nodes was $m_1, m_2, \cdots, m_k$ at those times. Then the mobility factor of the node N at the time instance T is

  $$M(N) := \sum_{i=1}^{k} \left(1 - \frac{\min(m_i, M)}{M}\right) F^{T-t_i}, \tag{2}$$

  where M is a normalizing parameter.

- *Average Age Mobility.* Another way is the reciprocal of the mean value of the age of the link. Let suppose that the current links of the node N is established at time instances $t_1, t_2, \cdots, t_l$. Then the mobility factor of the node N at the time instance T is

  $$M(N) := \frac{l}{\sum_{i=1}^{l} T - t_i} \tag{3}$$

- A similar expression for the mobility is

  $$M(N) := \frac{1}{l} \sum_{i=1}^{l} \frac{1}{T - t_i}, \tag{4}$$

  where $t_1, t_2, \cdots, t_l$ are also the time instances of the establishments of the current links of the node N.

Each measure described above expresses the recent behavior of the node N. If some of its links disappear and some new appear, then the value $M(N)$ increases, but if its links stop changing, then $M(N)$ decreases continuously.

This mobility information can be utilized in several ways.

4.3.2 Who starts the TreeID update process?

If the mobility factors of the neighbors are known for the nodes, then a simple alternative strategy is to decide which endpoint node of a broken Broadcast Link should start the TreeID update process: the less mobile one.

This strategy is based on the obvious observation that the node which starts the TreeID update process, will have relatively high number of Broadcast Links and the other one will have fewer.

4.3.3 Mobility Delay

This is a modification of the “NewID” process. The node waits a certain amount of time $MobilityDelay$ before it sends the first “I have new TreeID” message. The value of $MobilityDelay$ depends on the mobility of the node. The higher the mobility factor is the higher the $MobilityDelay$ is. Nodes with small mobility factor will have no $MobilityDelay$.

Using this method, mobile nodes tend to become a leaf in the Broadcast Tree. So, when such a node loses its Broadcast Link, it is able to fix it locally, instead of generating new TreeID (See Section 4.1)

4.3.4 Where to connect a leaf node?

Whenever a Broadcast Link connecting a single node A to the Broadcast Tree ceases, the algorithm can use the idea of Section 4.1 to avoid to the TreeID update process. If A has more than one living links then we have a choice to decide on the link to become a Broadcast Link.

One possible strategy is to connect to the least mobile node. The idea behind this strategy is that the previous techniques result in that the more mobile nodes have less Broadcast Link. So, connecting to the least mobile node helps to keep the most mobile nodes to remain a leaf node in the tree.

5 Discussion

The TreeCast algorithm presented above has a number of attractive features. In this chapter we will discuss these features and compare them with those of some selected algorithms found in the literature.

MANETs have many fields of use from military operations and save-and-rescue missions to office applications. In the present paper we focus on indoor office-like applications of ad-hoc networks where the communicating nodes are laptops, handheld computers, etc. used and carried by humans.

Indoor application means that network connectivity depends much more on the distribution of obstacles as indoor walls, rooms etc. than on geographical distance between communicating devices. Thus, we exclude geographical location based solutions due to their inaptitude in the targeted scenarios.

The number of messages needed to complete a network wide broadcast is rather low in the TreeCast algorithm. If we only use point-to-point messaging it is equal to $N-1$, that is one less than the number of nodes in the network. This is by the way the theoretically achievable minimum. Extended topology knowledge based methods of course try to reach this optimal value, but they usually use more transmissions. Flooding would use $N*(d-1)+1$ messages where d is the average degree of the nodes in the network.

If the network employs local broadcasts the theoretical minimum is less but it depends on the network topology. In this case flooding uses N local broadcasts. TreeCast still uses considerably less messages

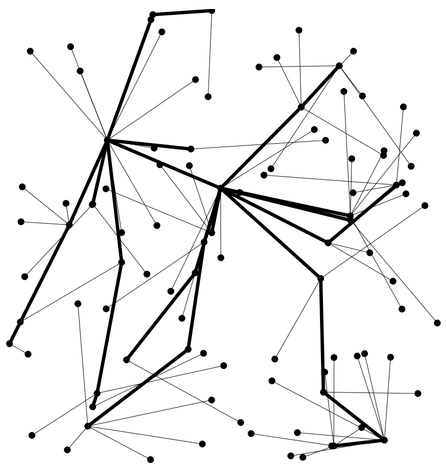

Figure 3: Snapshot of tree in a 100-node network. Tree links connecting non-leaf nodes are indicated by thicker lines.

than flooding, since the leaf nodes don't need to forward the broadcast message. As you can see from Figures 3, 6 and 5, the majority of the nodes are actually leaves in practice.

One advantage of TreeCast over all other presented methods is that nodes do not have to remember previous broadcast messages. Any broadcast message coming from a tree-neighbor is to be relayed (unless the node is a leaf) while all other broadcast messages are to be ignored. The tree topology guarantees that no message can reach a node from two different tree-neighbors.

Another feature that differentiates TreeCast from the other algorithms is that it has been specifically optimized for using one-to-one transmissions. This way the many drawbacks of local broadcasts (e.g. it being unacknowledged and collision-prone) do not affect the algorithm. It means that TreeCast is very well suited to perform reliable broadcasts. And this reliability can be achieved with the same number of transmissions as in case of the widely used, unreliable flooding.

TreeCast has advantages over the extended topology knowledge based methods as well. While all of these methods require some extra messaging to keep up some state in the network, TreeCast has been optimized to minimize this extra signaling traffic. This will be demonstrated in Section 6.

The fact that TreeCast relies on an underlying tree topology raises some questions. A tree is very vulnerable to changes in topology as the loss of any tree-link or node (unless it is a leaf) destroys its connectivity. However TreeCast has both explicit and implicit methods to increase the stability of the tree.

The mobility based extensions (see section 4.3) force nodes of high mobility to the perimeter of broadcast tree. Thus the topology changes caused by the mobility of these nodes do not generate new tree topologies. Furthermore the “NoTreeDelay” extension of section 4.2 can enforce the stability of undamaged parts of the tree even in case of loss of tree connectivity.

6 Testing the Behavior of the Algorithm

Several tests were performed through event driven simulation in order to measure the capability of the algorithm in realistic environment.

The simulation works on message level. The several different delay parameters were chosen so that the behavior will be similar to a real WLAN network.

The simulator handles the collisions of local broadcast messages.

The nodes take place in a rectangular area. For the sake of simplicity, the visibility merely depends on their distance in our test cases.

As our prime targets are humans carrying their computers the mobility of the network is special. Nodes do not move with high speeds, and do not move all the time. Think of employees in an office or customers in a coffee shop. Possible scenarios will thus have only one part of the nodes moving at any given time with relatively low speeds (up to 6 km/h).

To simulate a usual office environment we defined different node-types as follows:

- *Walking User.* This kind of node simulate a user that is walking continuously. It repeats the following: It randomly chooses a walking direction and speed between 3 to 6 km/h and it takes a random distance in that direction with the chosen speed.

- *Mobile User.* This type of node intends to model the behavior of employees' palmtops. It takes walks for 2 to 15 minutes and waits a random interval between 2 and 90 minutes between these walks.

- *Laptop Node.* It chooses a random position, goes there with a randomly chosen speed, stays there a longer time (1 to 5 hours) and repeats this process.

- *Immobile Node.* This type of node does not move at all.

6.1 Effect of mobility

In the following test we compared TreeCast and the extended topology knowledge based algorithms with respect to the resource requirement of the maintenance of their respective data structures.

We modeled the extended topology knowledge methods by simulating a network where nodes notify all of their neighbors whenever there is a change in their connectivity (i.e. a neighbor is lost or a new one appears). This models an algorithm that keeps all nodes' 2 hop neighborhoods up-to-date as described

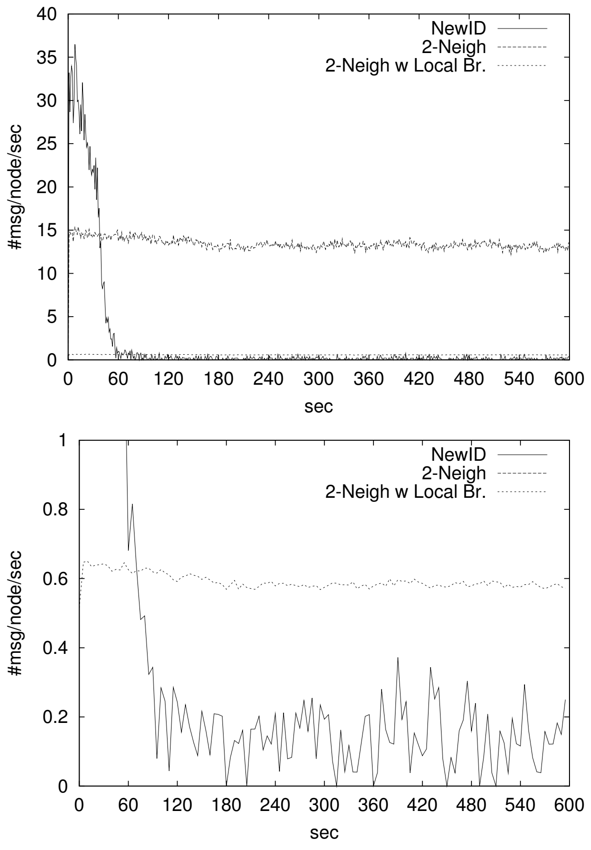

Figure 4: Effect of swapping the set of mobile and non-mobile nodes (the second picture gives a rescaled view)

in [3] or [4]. We termed this algorithm *2-Neigh* in the following comparisons. This algorithm was tested both with and without local broadcast .

In order to perform the comparison we set up two groups of 50 nodes each. At any given time all of the members of one group are moving continuously while the members of the other group are immobile. The role of these groups is changed in every 10 minutes. Figure 4 shows the number of the messages per node per second sent by all three algorithms (the figure shows the average of 100 subsequent iterations).

It can be seen that whenever the roles of the mobile and immobile groups have been changed quite a lot messages have to be sent to maintain the Broadcast Tree. But after a short period the algorithm recognizes the mobile nodes and as they become a leaf in the Broadcast Tree the number of sent messages declines well below that of 2-Neigh even when local broadcast is available. In fact, after 60 second, there are no “I have NewID” messages disregarding some sporadic TreeID updates.

It also has to be taken into account that this comparison only shows the number of transmitted messages. However while control messages in TreeCast only carry a single TreeID the messages of 2-Neigh have to include a node's full neighborhood (*Dominant Pruning*[3]) or even more additional information such as neighbors' degrees (*LENWB*[4]).

6.2 Office Day Scenario

In order to compare the behavior of 2-Neigh and TreeCast under more realistic conditions, we simulated a full virtual day of an office of 300 employees with PDAs (Mobile Users) and 100 laptops (Laptop Users) from 8AM to 6PM. The simulation was done in a rectangular area with a ratio of its sides' lengths $1 : 5$. This elongated form intends to decrease the connectivity, thus modeling the effect of various obstacles in an indoor environment.

The nodes come to life at random time instances between 8AM and 9AM and they switch off at randomly chosen times between 5PM to 6PM.

There are some special events during the day. A department meeting is organized at 10AM in a relatively small meeting room. Half of the mobile users take part in it, so they start to go to that room at between 9:50 an 9:55. They stay there until 10:45AM and then they start to "walk" again. A similar meeting is held at 3PM. Each employee (mobile node) goes to the canteen to have a lunch between 11:30AM and 13:00PM. They spend form 15 to 30 minutes there.

A snapshot of the spanning tree in the network at 9:10AM and 10:15AM are shown in Figures 5 and 6. The meeting room is located in the lower right corner of the simulation area.

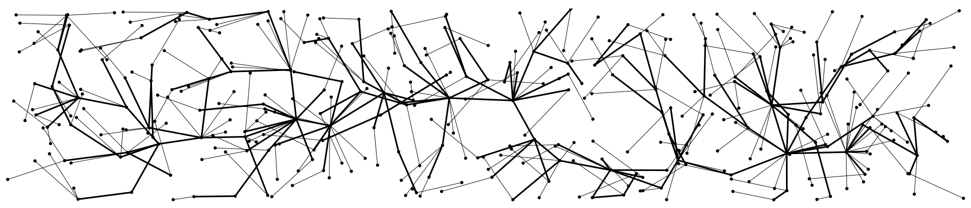

Figure 5: Snapshot of the spanning tree during the full-day simulation at 9:10

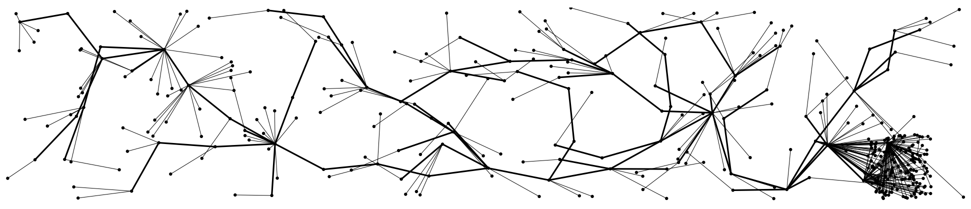

Figure 6: Snapshot of the spanning tree during the full-day simulation at 10:15

Again, three algorithms were tested: TreeCast and 2-Neigh with and without local broadcast. Figure 7 shows the number of control messages for each algorithm. Even at first glance one can conclude that 2-

Neigh with point-to-point transmissions provides the worst performance, far worse than the other two.

It is clearly visible as well that all three algorithms react to the two meetings with definite peaks in the number of messages. The lunch break can also be easily spotted.

We can point out that while TreeCast's peaks are higher during these periods its control traffic stays well below that of 2-Neigh during most of the day. This is remarkable if we take into account that nodes change their behavior from immobile to mobile and vice versa all through the day.

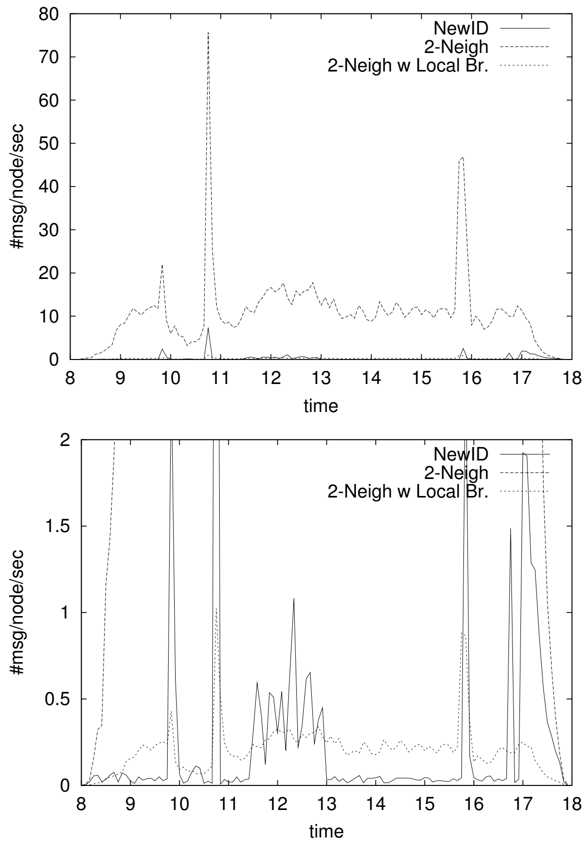

Figure 7: Full-day simulation: Frequency of the "NewID" messages compared to Neighbor distribution (the second picture gives a rescaled view)

6.3 Broadcast Efficiency of TreeCast vs. Flooding

To compare the efficiency of traditional flooding and TreeCast we simulated the office day with broadcast messages emitted by random nodes in every 10 seconds. Of course, in a real application there could be much more broadcast messages, but this amount of messages seems enough to measure the efficiency of the different methods. The broadcast events take place between 9AM and 5PM.

We used two measures in our comparisons: the average of the percentage of uncovered nodes for all broadcasts and the average number of times that a node receives the same broadcast message. The first one indicates how reliable the broadcast is, while the second one measures the number of unnecessary transmissions.

Table 1 shows the above two measures averaged out for the whole day. In case of flooding collisions were taken into account, resulting in the high percentage of lost messages. For comparison we show that without collisions each node would receive broadcasts many times.

Table 1: Performance of flooding vs. TreeCast

	Lost msg	Avg # received msg
TreeCast	0.116%	1.016
Flooding	19.333%	1.608
Flooding w.o. coll.	0%	30.689

7 Further Remarks and Conclusion

In the paper we presented TreeCast, a novel method for broadcasting suitable for MANETs. The method is based on a fully distributed, decentralized and resource-efficient algorithm that maintains a spanning tree. This algorithm can also take into account nodes' mobility to minimize tree maintenance. For this purpose we propose a novel measure of node mobility which is based solely on the changes in the nodes connectivity and does not require any geographical data from GPS devices.

The algorithm performs especially well in case of point-to-point transmissions, so it is particularly useful for networks with radio technologies that do not support point-to-multipoint transmissions. This also ensures that TreeCast does not suffer from the negative effects of local broadcasts in IEEE 802.11b based networks.

We have shown that for indoor office-type scenarios our broadcast algorithm performs better than those found in the literature. We have demonstrated that TreeCast is both more reliable and more resource-efficient than traditional flooding. It was also shown that it performs efficient and reliable broadcast in such a way that its handling of topology data is better suited to realistic office-type user behaviors than the topology update mechanisms of earlier proposed broadcast methods.

There are several directions to continue the present research. One interesting idea is to use the TreeCast algorithm and its underlying spanning tree to disseminate routing information in the network. Then the overlying routing could benefit from the existence of an implicit backbone. This is due to the fact that TreeCast forces nodes of high mobility to be leaves of the tree, thus currently immobile nodes form a rather reliable backbone. It would be interesting to see the performance gain if the routing information of some routing algorithm (e.g. DST [18]) would be disseminated with TreeCast.

Another possible direction is to explore the applicability of the presented mobility measure for enhancing existing MANET routing protocols. For example routes containing more stable nodes could be favored over those containing high mobility nodes.

References

[1] S.-Y. Ni, Y.-C. Tseng, Y.-S. Chen, J.-P. Sheu, "The Broacast Storm Problem in a Mobile Ad Hoc Network", *Proc. ACM/IEEE MobiCom*, August 1999, pp. 151-162.

[2] C. E. Perkins, E. M. Belding-Royer, S. R. Das, "IP Flooding in Ad hoc Mobile Networks", *IETF Internet-Draft*, draft-ietf-manet-bcast-00.txt, work in progress, 14 November 2001.

[3] H. Lim, C. Kim, "Multicast Tree Construction and Flooding in Wireless Ad Hoc Networks", *Proc. 3rd ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, 2000, pp. 61-68.

[4] J. Sucec, I. Marsic, "An Efficient Distributed Network-Wide Broadcast Algorithm for Mobile Ad Hoc Networks", *CAIP Technical Report*, TR-248, July 2000.

[5] Z. J. Haas, M. R. Pearlman, P. Samar, "The Bordercast Resolution Protocol (BRP) for Ad Hoc Networks" *IETF Internet-Draft*, draft-ietf-manet-zone-brp-01.txt, work in progress, June 2001.

[6] I. Stojmenovic, M. Seddigh, J. Zunic, "Internal node Based broadcasting algorithms in wireless networks" *in Proceedings of the Hawaii Int. Conf. on System Sciences*, Jan. 2001.

[7] *IEEE Specification 820.1d* "MAC Bridges" D9, July 14, 1989

[8] A. Ballardie, P. Francis, J. Crowcroft "Core Based Trees (CBT) an Architecture for Scalable Inter-domain Multicast Routing" *SIGCOMM'93*, San Francisco 1993, pp. 85-95.

[9] D. Estrin et al. "Protocol Independent Multicast – Sparse Mode (PIM-SM): Protocol Specification" *RFC 2362*, June 1998.

[10] A. Adams, J. Nicholas, W. Siadak "Protocol Independent Multicast - Dense Mode (PIM-DM): Protocol Specification (Revised)" *IETF Internet-Draft*, draft-ietf-pim-dm-new-v2-02.txt, work in progress, October 2002.

[11] D. Waitzman, C. Partridge, S. Deering "Distance Vector Multicast Routing Protocol" *RFC 1075*, November 1988.

[12] J. Moy “Multicast Routing Extensions for OSPF” *CACM*, Vol. 37, Aug 1994, pp. 61-66.

[13] C. Perkins, E. Royer, S. Das “Ad hoc On-demand Distance Vector (AODV) Routing” *IETF Internet-Draft*, draft-ietf-manet-aodv-11.txt, work in progress, Aug 2002.

[14] Ch. Lee, M. H. Ammar, J. E. Burns “An Improved Leader Election Protocol in Multi-hop Radio Networks” *International Conference On Computer Communication* Seoul, Korea, 1995.

[15] P.-J. Wan, K. M. Alzoubi, O. Frieder “Distributed Construction of Connected Dominating Set in Wireless Ad Hoc Networks” *IEEE INFOCOM*, New York, June 2002.

[16] R. G. Gallager, P. A. Humblet, P. M. Spira “A distributed algorithm for minimum weight spanning trees” *ACM Trans. on Programming Languages and Systems*, 5(1) Jan. 1983, pp. 66-77.

[17] N. Nikaein, H. Labiod, C. Bonnet “DDR: distributed dynamic routing algorithm for mobile ad hoc networks” *International Conference on Mobile Computing and Networking*, 2000, pp. 19-27.

[18] S. Radhakrishnan, N. S. V. Rao G. Racherla, C. N. Sekharan, and S. G. Batsell “DST - a routing protocol for ad hoc networks using distributed spanning trees” *IEEE Wireless Communications and Networking Conference*, 1999, pp. 100-104.

Biography

Alpár Jüttner received his M. Sc. degree in 1998 at the Eötvös Loránd University of Budapest, where he is currently working on his Ph. D. at Operational Research Departement. He also works as a research fellow at Ericsson Traffic Analysis and Network Performance Laboratory in Budapest, Hungary. His main interests are combinatorial optimization and its applications.

Ádám Magi received his M. Sc. degree in 1996 at the Technical University of Budapest in Electrical Engineering. He is currently working on his Ph. D. there. He also works as a research fellow at Ericsson Traffic Analysis and Network Performance Laboratory in Budapest, Hungary. His main interests are mobile ad hoc networks and routing in telecommunication networks.