Cross-layer Overlay Synchronization in Sparse MANETs

Thomas Plagemann, Katrine S. Skjelsvik, Matija Puzar,
Aslak Johannessen, Ovidiu Drugan, Vera Goebel, Ellen Munthe-Kaas
Department of Informatics, University of Oslo
{plageman, katrins, matija, aslakjo, ovidiu, goebel, ellenmk}@ifi.uio.no

ABSTRACT

Mobile Ad-Hoc Networks maintain information in the routing table about reachable nodes. In emergency and rescue operations, human groups play an important role. This is visible at the network level as independent network partitions which are for some time stable before their members change through merging or partitioning. We use the information from stable routing tables to optimize the synchronization of Mediators in a Distributed Event Notification System. In a stable partition each node has the same information, thus a single Mediator can efficiently coordinate the synchronization, while all other Mediators just receive updates. We show in our experiments that just a few seconds are needed until routing tables stabilize and all nodes have a common view of the partition. We present a heuristic to determine the proper time to synchronize. Furthermore, we show how exceptions, like disappearing coordinating Mediators and unexpected messages, can be efficiently handled.

Keywords

Synchronization, sparse MANETs, cross-layer optimizations, overlays in MANETs, publish-subscribe.

INTRODUCTION

In order to efficiently handle crises and emergencies, emergency and rescue (ER) teams benefit from well working communication infrastructures for command, control and coordination. However, first responders are typically confronted with an environment where no communication infrastructure is available, either because it was nonexisting, or the earlier existing ones have been destroyed. Therefore, wireless Mobile Ad-Hoc Networks (MANETs) formed by devices carried by ER personnel are often the only means to establish a communication infrastructure. However, the mobility of the ER personnel combined with the size of the emergency area (which is typically multiple times larger than the coverage of individual IEEE 802.11 radios) and obstacles in the area reflecting radio waves, leads to the situation that there is often not one single MANET connecting all ER personnel. Instead, multiple partitions might exist and change over time through merging and partitioning. Typically, these partitions correspond to groups of ER personnel that have a common task to fulfill. Due to the dynamics of ER operations, groups might need to change their locations, and group memberships might change. This is reflected at the network level through changes in the routing table. Evidence for such group mobility is not only given by our study of ER operations, but also confirmed by recent studies of social mobility [2] and community detection [6] in opportunistic networks.

We have developed a Distributed Event Notification Service (DENS) [10] to support remote patient monitoring. DENS uses Mediators to replicate subscriptions, to enable graceful degradation in case of network partitions, and to convey subscriptions and notifications from source to destination. If there is connectivity to the destination the Mediator uses the OLSR MANET routing protocol [4] and IP to transport the packets to their destination. However, if a destination node is turned off or in a different partition, OLSR and any other MANET routing protocol fails. Therefore, the Mediators form an overlay on top of the MANET to perform delay tolerant transport through so-called “store-carry-forward” operations [10]. The replication of undelivered subscriptions and notifications increases the probability that one of the Mediators at a later point in time can join a partition with formerly unreachable destinations. The number of Mediators can be dynamically adapted to the particular ER operation and by this trade availability of DENS against resource consumption, i.e., the more Mediators the higher the availability and the higher the resource consumption.

The message ferry approach [12] is determining the mobility, including speed and trajectory of special nodes called ferries, to make sure that a previously unreachable destination and the ferry are coming into communication range. This is not in general possible in ER operations. Therefore, we replicate undelivered subscriptions and notifications

to all Mediators, similar to the approach in epidemic routing [11] where packets are forwarded to neighbors and then spreads through the network, hence the name. However, we make use of a proactive routing protocol and do not depend on the event that two nodes meet, to enable exchange of undelivered messages. Instead, we can synchronize Mediators immediately after they join a common network partition, which are typically formed by different ER teams.

There exist various works related to routing in intermittently connected networks, or sparse MANETs. In [13] Zhang gives an overview of different approaches for the store-carry-forward routing. They differ in how they select the next-hop-node, e.g. random selection, using knowledge about the whereabouts of nodes, or predict future location. Other approaches assume some knowledge such as last known location of the destination, and therefore only forward packets in the same area. There exist some cross-layer approaches such as EMMA [9] where synchronous communication is used if possible; in case of no end-to-end connection, epidemic routing is used instead.

Synchronization of Mediators seems to be simple, but several complicating factors have to be considered: First, each node has its own view of "its" network partition which does not necessarily correspond to the view of the other nodes in this partition. Second, merging of partitions is not an atomic action, the routing daemon in each node discovers new nodes iteratively over several time steps. Third, nodes are mobile and there might never be a globally correct definition whether partition merging has finished or not. Fourth, propagation of messages from source to destination through epidemic routing might take quite a long time depending on the network topology and the movement of the nodes; however, certain messages in ER applications have stringent timing requirements, like an alarm from a patient monitoring application. Finally, bandwidth is a scarce resource and synchronization among multiple Mediators should be as efficient as possible.

We propose in this paper to leverage the existing information about the network topology from the OLSR routing protocol. By this each Mediator can establish its own view of the network without putting any additional load onto the network. Furthermore, we use the fact that ER operations are performed in groups, resulting in partitions that are stable for some time (with respect to the nodes that form the partition) before new mergings or partitionings occur. We show through simulation studies that the time it takes to merge network partitions and the time it takes until all nodes in the new partition have updated routing tables, is rather short. By assuming a common view among the Mediators in each partition, we can for each partition immediately identify a coordinator that acts on behalf of all Mediators in its partition. This in turn enables us to minimize the number of exchanged messages in the synchronization process.

In the remainder of this paper, we briefly describe DENS in Section 2. In Section 3, we analyze how useful information from the IP layer, i.e., the OLSR routing table is. In Section 4, we outline the basic idea of the synchronization protocol and describe how exceptions are handled, followed by some conclusions in Section 5.

DENS

In publish-subscribe systems, subscribers express their interest in events through subscriptions, and publishers publish events of interest. DENS is designed to support information exchange even in the presence of network partitions. Subscription information is therefore replicated in the network. There is a trade-off between offering a reliable service, but at the same time not saturate the network by replicating too much information. Filtering of events is therefore performed as close to the source as possible. DENS supports different subscription languages, both simple subject-based languages, and content-based languages where the subscriber can specify the content of the notification it wants to receive information on, e.g., a value range of health sensor data.

DENS has the following components: (1) Subscriber, (2) Publisher and (3) Mediator. Subscriber and Publisher run in every node, and the Mediator runs on some of the nodes. The number of Mediators is dependent on the network size, density and dynamicity. The Subscriber Component sends subscriptions to a Mediator running on the same or another node. The Mediator parses received subscriptions to find keywords that are used to locate potential publisher nodes. The subscriptions are then sent to these publisher nodes. The Publisher filters events according to the subscriptions. When an event of interest occurs, it sends a notification to a Mediator. The notifications are matched with the subscriptions, and then delivered to interested subscribers. Thus, Mediators decouple Subscribers and Publishers.

DENS keeps track of Mediators by listening to beacons sent by such nodes in its partition. The Mediators form a DENS overlay. In addition to forwarding subscriptions and notifications, Mediators replicate subscriptions and possibly notifications to obtain a higher degree of reliability in the presence of partitionings caused by

disconnections or that a node is turned off. The task of the overlay formed by the Mediators is to enhance reliability and support delay-tolerant routing of subscriptions and notifications in case of partitions.

Subscribers and publishers send their subscriptions and notifications to a Mediator in their own partition using end-to-end paths set by the network routing protocol. This means that the Mediator is an indirection. Delivering subscriptions and notifications, and replicating subscriptions and undelivered notifications, are done by using the underlying routing protocol and the synchronization protocol. The synchronization protocol is initiated when there are new nodes in the partition. The presence of new nodes provides the means for delivering undelivered stored subscriptions and notifications to the newly connected nodes, and replicating subscriptions and undelivered notifications to newly arrived Mediators. Because of the network partitionings, the Mediators can have an inconsistent view of subscriptions. In the next section, we describe how we can use information from the routing table to detect topology changes and initiate the synchronization process among Mediators.

ROUTING TABLE INFORMATION

One important key element to enable efficient design of middleware protocols over Sparse MANETs, is information about nodes that can be reached through a multi-hop route at a given point in time and the prediction of future connectivity. Parts of this information might be gathered from external sources, like GPS satellites or base stations, but we aim to design our solutions such that they work also when no external information is available. The other possibility to gather this information is that the middleware components periodically broadcast messages, like in Hypergossiping [7], to detect partition mergings, etc. Since bandwidth is a scarce resource, we aim to minimize the number of broadcast messages.

In order to gather at the middleware layer information about network partitions, mergings, and partition membership information in a non-intrusive manner, we leverage the information that is already available at the network layer in the routing table. In our previous studies [5] we have observed that the routing protocol holds updated information about the neighborhood of a node if the node is actively involved in communication. This claim holds for both proactive and reactive routing protocols. However, proactive routing protocols maintain topology information also if there is no (or not sufficient) communication. The proactive routing protocol OLSR periodically sends beacons (*HELLO messages*) to inform other nodes of its presence. In addition, OLSR tries to maintain at each node a consistent view of the network by exchanging topology information with the other nodes in the network. Whenever there is a change in the topology, the routing table is recalculated. Each entry in the routing table contains information on the destination node, the next hop node, the estimated number of hops to the destination, and the interface used for communication.

In order to optimize the synchronization of Mediators after partition mergings, we need to understand the dynamics of both the merging process and the partitioning process and how it is reflected in the routing tables of the individual nodes. We have performed a number of experiments with the emulation tool NEMAN [8] to analyze how often the local routing table changes over time, whether the change frequency allows us to deduce that a merging or partitioning has finished, and how long it takes until all nodes in one partition have the same view of their partition. We instrumented the code of the OLSR daemon to extract and log all changes of membership information, which enables us to identify how many neighbors each node's routing protocol reported at any point in time. The OLSR's interval for sending HELLO messages is set to 1 second in all experiments.

In order to verify our hypothesis that groups which move in larger areas result in stable membership information for a substantial amount of time, we performed experiments with non-static groups, moving according to the reference point group mobility pattern. The nodes were moving at 2 units/s in an area of 600x600 units. Figure 1 illustrates a representative case in which two groups of 10 nodes came in contact approximately after 11 seconds and remained in contact for approximately 77 seconds. For each node there is one line in the graph that shows how many partition members this node has registered. Overlapping lines indicate a consistent view among multiple nodes. The figure shows on each group subgraph a single line for most of the time, meaning that both groups have a stable view of the network. Occasionally, due to the mobility of nodes a few nodes have a different view.

In order to analyze the time it takes until routing tables are stable (i.e., no changes in membership information for some time) and all nodes in a partition have the same membership information, we performed experiments with three different types of network topologies. The first one is the *chain* topology, where nodes are lined up only to have one or two direct neighbors. We consider this to be the worst case scenario where there is still full connectivity. The second one is the *mesh* topology, where nodes are randomly placed, each having more than one neighbor. And

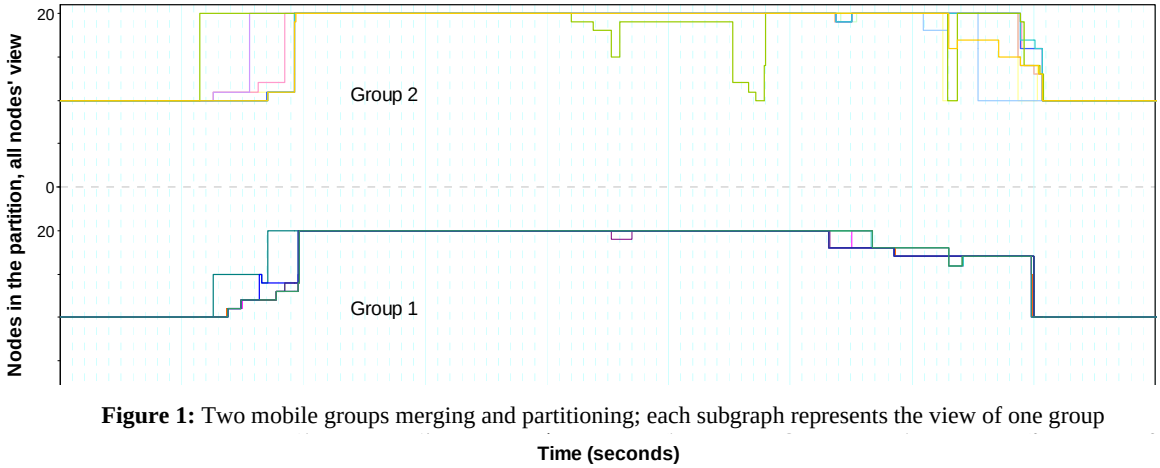

Figure 1: Two mobile groups merging and partitioning; each subgraph represents the view of one group

third, the *full mesh* topology is the case where each node has direct contact with all the other nodes. Table 1 shows the results for a selected set of experiments, including two static groups of 1, 10, or 20 nodes each. Merging and partitioning events were introduced artificially by creating or removing contact between the two groups at a certain number of merging points. The merging time and partitioning time were measured on a global basis, i.e. from the moment the first node noticed the change to the moment when all the nodes in the partition had the same view.

Topology	Merging points	Groups	Merg. time	Part. time
Chain	1	20+20	10,97s	8,73s
Mesh	1	20+20	8,47s	6,79s
Mesh	5	20+20	7,40s	7,80s
Full mesh	full mesh	20+1	0,28s	2,41s
Full mesh	full mesh	10+10	1,17s	1,32s

Table 1: Resulting times for merging and partitioning

In addition to experiments with group mobility models and especially designed topologies, we have also performed experiments with the random waypoint model as a worst case analysis. We have simulated results for 20 nodes in areas of size 500x500, 1000x1000 and 1500x1500, and 250 units radio range. As expected membership changes in routing tables for very dense networks are rare, i.e., membership does not change during the entire run. In larger areas there are also longer periods in which the individual routing tables do not change. In the studied case where the area size was 1000x1000, this was often more than 10 seconds, while in larger areas this stable period was often several times longer. When a merging takes place, the routing daemons recalculate old routes and add new routes towards the new nodes. This takes approximately 5 seconds on each node. This is dependent on the location of the node and the number of new nodes.

SYNCHRONIZATION PROTOCOL

The basic idea for the synchronization protocol is to use information from the routing table to initiate synchronizing of data after a merging of two or more partitions. The protocol is initiated in a node when it detects a routing protocol change that indicates a partition merge, but only after it assumes that the routing table has stabilized. During the synchronization process some of the Mediators resume specialized roles as *partition_representatives*. A *partition_representative* is responsible for synchronizing data in its old partition. The role of being a *partition_representative* for an old partition is taken by the Mediator with the highest node ID. Since we assume that each Mediator keeps a record of all other Mediators in its partition, the *partition_representative*'s identity is implicitly known without any message exchange. Among the *partition_representatives* one takes the role as *coordinator*. The *coordinator* has the responsibility of coordinating the synchronization process among the *partition_representatives*. The *coordinator* is the *partition_representative* having the highest node ID. After the *partition_representatives* have synchronized data among themselves, they send updates to the Mediators of their old partitions.

We first describe the heuristic used to determine when the synchronization should be initiated, before we explain the basic protocol without exception handling and what kind of conditions we assume. Then we describe how exceptions are handled if these conditions do not hold.

Synchronization Initialization

The heuristic to determine when to initiate the synchronization uses two timestamps and three threshold values:

- Timestamp t_{start} records the time a new node is registered in the routing table after a stable period.
- Timestamp t_{last} records the time the last change of membership information in the routing table was detected.
- Threshold value T_s is an estimate whether the routing table is stable, i.e., there are no membership changes during the period $[t_{last}, t_{last} + T_s]$.
- Threshold value T_{GV} estimates the time it takes for all nodes in a partition to have the same membership information after the last membership change in the local table was detected.
- Threshold value T_E is used to ensure that the heuristic is able to start the synchronization process from time to time even if there is never a stable routing table.

The heuristic is started when a new node is registered in the routing table. Both t_{start} and t_{last} are assigned the current time. Each time a change of the membership information occurs, t_{last} is updated with the current time. Normally, the synchronization process is started if the routing table is stable and all nodes have the same membership information. If the exception occurs that the routing table changes continuously for a too long time, synchronization is enforced even if the routing tables are not stable. The pseudo code of the heuristic is given below.

```
t_start, t_last := t_current;
repeat {
 if (membership_change) {t_last:=t_current;}
until (t_last + max(T_S,T_GV) < t_current || t_start + T_E < t_last) }
Start_Synch;
```

Based on our experiments, we are currently using 5 seconds for T_S and T_{GV}. T_E has to be adapted to the application requirements to balance between resource consumption and availability.

Basic Protocol

For each node the protocol has three phases: the Mediator Discovery phase, where Mediators from merging partitions are discovered and one Mediator from each old partition takes the role of a *partition_representative*; the Global Synchronization phase where the *coordinator* is selected and the *partition_representatives* exchange information; and the Local Update phase where the *partition_representatives* send updates to the Mediators in their old partition. In each phase timers are set to ensure that the phase always completes. The events triggering the different phases are shown in Figure 2.

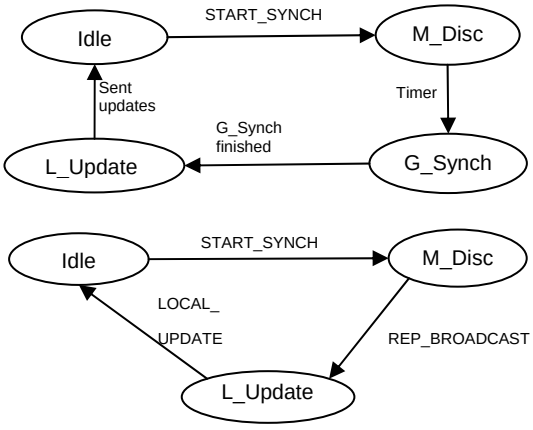

Figure 2: Mediator and *partition_representative* states

The Basic Protocol runs under these assumptions:

- each node knows about every other node in the new (merged) partition,
- each Mediator knows about every other Mediator in its old partition,
- all Mediators in an old partition are synchronized, and
- during the synchronization process no new nodes arrive, no nodes disappear, and no new subscriptions or notifications are sent.

In the following we describe the phases, roles, and messages.

Mediator Discovery

A Mediator enters this phase when it receives a START_SYNCH message. The Mediator starts a timer. Each Mediator examines its set of known Mediators and decides whether it is a *partition_representative*. The Mediators taking the role of a *partition_representative*, floods a REP_BROADCAST message. This message includes a list of the Mediators it represents. When a Mediator receives a REP_BROADCAST from its own *partition_representative*, it enters the Local Update phase and cancels the timer. The *partition_representative* listens for REP_BROADCASTs from the other partitions and waits until there is a timeout. It then enters the Global Synchronization phase.

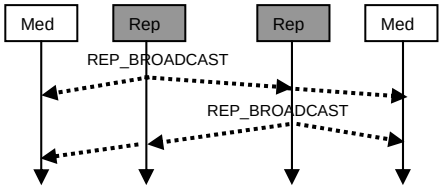

Figure 3: Mediator Discovery (messages)

Global Synchronization

All Mediators that enter this phase are *partition_representatives*, in addition one of them takes the role of being a *coordinator*. Again, this role is taken by the Mediator having the highest node ID. The messages used in this phase are: SYNCH_C, SYNCH_REP, and SYNCH_TOTAL. A timer is started when the Mediators enter the phase.

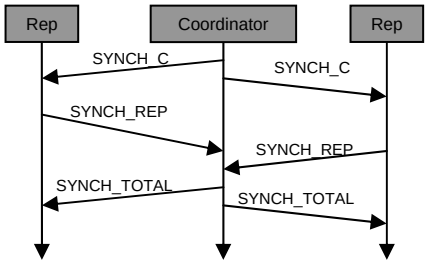

Figure 4: Global Synchronization (messages)

The *coordinator* sends SYNCH_C containing a summary of its subscriptions to the other *partition_representatives*. The other *partition_representatives* compare the summary with their own content and reply with the message SYNCH_REP containing data the *coordinator* is lacking, in addition to a summary of its own data. When the *coordinator* has received replies from all the *partition_representatives*, it sends SYNCH_TOTAL updates to the *partition_representatives*, i.e., its own subscriptions in addition to subscriptions received from the other *partition_representatives*. The *coordinator* cancels its timer, resumes status as an ordinary *partition_representative*, and enters the Local Update phase. When the *partition_representatives* receive the SYNCH_TOTAL message from the *coordinator*, they cancel the timer and enter the Local Update phase.

Local Update

In the last step of the protocol, the *partition_representatives* send LOCAL_UPDATE messages to the Mediators in their old partition. This includes information about subscriptions, but also about new Mediators. Each ordinary Mediator in this phase starts a timer, then it awaits the arrival of a LOCAL_UPDATE message from its

partition_representative. When the LOCAL_UPDATE message arrives, the timer is cancelled and the Mediator resumes ordinary activity.

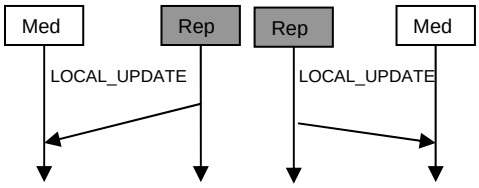

Figure 5: Local Update (messages)

We have implemented the basic protocol. Figure 6 shows the number of known Mediators from one Mediator's perspective when testing it in a scenario where there are two merging events; one at 90 seconds and the second merging at 195 seconds. The increasing number of known Mediators shows that the nodes in the network have detected a partition merging, the routing tables are stabilized, and the Mediator Discovery phase is started.

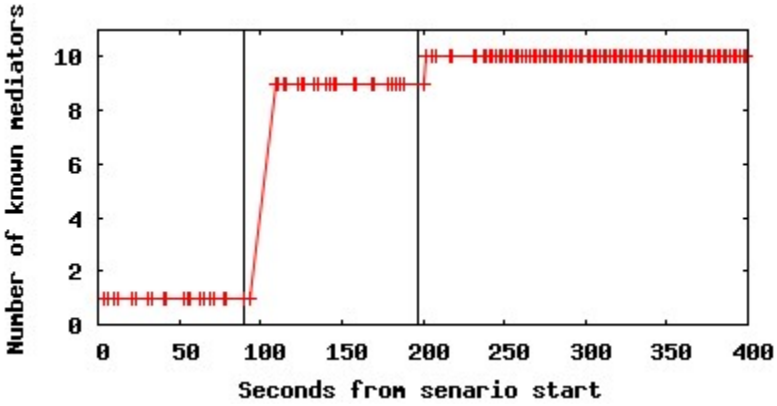

Figure 6: Number of Mediators from one Mediator's perspective during network merging events

Exception Handling

We now discuss how exceptions are handled in the different phases. Examples of exceptions are that the Mediators do not have the same view of the partition membership, that Mediators may appear or disappear during the synchronization process, and that Mediators in the old partition may not be fully synchronized when the synchronization protocol starts. It is important to notice that we cannot assume at any stage that the nodes have the exact same view of where they are in the synchronization process, and what the members of a partition are. The protocol therefore needs to be robust enough to manage these situations. In the following, we discuss the different phases of the protocol from one Mediator's point of view. The notable exceptions are shown in the Tables 2-5. The handling of the exceptions is dependent on the phase and the role of the Mediator.

Role	Exception	Handling
M	REP_BROADCAST	Start synchronization

Table 2: Exception Handling when idle

Role	Exception	Handling

M + R	START_SYNCH	Stack request
	LOCAL_UPDATE	Receive data
R	timeout without any received REP_BROADCAST	Proceed to L_Update
M	REP_BROADCAST from an unexpected R in its old partition	Reconsider the identity of R for its old partition
	timeout, no received REP_BROADCASTs from nodes in its old partition	Reconsider role to R

Table 3: Exception Handling in Mediator Discovery phase

Role	Exception	Handling
C+ R	START_SYNCH	Stack request
	REP_BROADCAST	Stack request
	LOCAL_UPDATE	Receive data
C	timeout without having received any SYNCH_REPs	Proceed to L_Update
	timeout but has only received some of the expected SYNCH_REPs	Proceed with reduced set of recipient Rs
R	timeout without having received SYNCH_C	Reconsider role to C
	timeout without having received SYNCH_TOTAL	Proceed to L_Update
	SYNCH_C from wrong C	Respond with SYNCH_REP but continue to wait for SYNCH_C from true C

Table 4: Exception Handling in the Global Synch phase

Role	Exception	Handling
M +R	START_SYNCH	Stack request
	REP_BROADCAST	Stack request
M	timeout, has not received LOCAL_UPDATE	Proceed

Table 5: Exception Handling in the Local Update phase

A Mediator enters the Mediator Discovery phase when it receives a START_SYNCH or a REP_BROADCAST message. It then starts a timer. If a Mediator receives a new START_SYNCH message during this phase, it will just stack the request and enter the Mediator Discovery phase again after it has finished its current synchronization process. If it receives a LOCAL_UPDATE message out of order, it receives data that can be handled locally immediately. It may be the case that the Mediators in the old partition do not have exactly the same view of the partition membership, so a Mediator can receive a REP_BROADCAST from a node that it is aware of but did not consider being the *partition_representative*. In this case, the Mediator reports to the new *partition_representative*. If the timer fires for a node that is assumed not to be a *partition_representative*, it will reconsider which Mediator should be *partition_representative*. If it is the next Mediator having the highest ID, it sends a REP_BROADCAST, if not it will restart the timer. If the timer fires for the *partition_representative*, it sees if it has received any REP_BROADCAST messages, if not, it goes directly to Local Update phase.

In the Global Synchronization phase only *partition_representatives* participate, and one of them takes the *coordinator* role. In this phase both START_SYNCH messages and REP_BROADCAST messages are stacked and handled when the process is finished. LOCAL_UPDATE messages are just received but not handled. If the *coordinator* disappears, the remaining *partition_representatives* will at timeout reconsider their roles, and the one with the next highest id becomes the new *coordinator*. If all *partition_representatives* but the *coordinator* disappear,

the *coordinator* will at timeout enter the Local Update phase. It may happen that none, two or more elect themselves as a *coordinator*. If none starts as a *coordinator*, then there will be a timeout where the *partition_representatives* reconsider their role. If there is a SYNCH_C from a non-assumed *coordinator*, the *partition_representatives* will respond to it but await a SYNCH_C from its true *coordinator before* proceeding to the Local Update phase.

As in the previous phase, both START_SYNCH messages and REP_BROADCAST messages received during the Local Update phase are stacked and handled when the process is finished. If a timeout fires, this indicates that something went wrong, i.e., the *partition_representative* is gone.

If subscriptions or notifications are received by a *partition_representative* at any phase, it will send it as LOCAL_UPDATE messages. If a Mediator is not a *partition_representative* or a *coordinator*, it will replicate it to the other Mediators.

Complexity

In order to compare the complexity of our synchronization protocol with a simple gossiping style synchronization, we use a simple analytic model. We assume two merging network partitions with x respectively y mediators that are synchronized within their partition. We measure the complexity in terms of number of Mediator to Mediator update processes. In our synchronization protocol, first the *partition_representatives* update each other, and afterwards they update the Mediators in their old partition, leading to $1 + (x - 1) + (y - 1)$ updates. If instead all Mediators exchange information with all other Mediators, in the best case each Mediator from partition 1 updates all Mediators from partition 2 or vice versa. This leads us to $x*y$ updates. The complexity is therefore $O(x + y)$ for our synchronization protocol and $O(x*y)$ for gossiping style protocols. The synchronization protocol in addition is deterministic since all Mediators are updated after a merging, and not only when e.g. Mediators become direct neighbors and initiate synchronization.

CONCLUSIONS

One fundamental decision for the design of DENS is to use the proactive MANET routing protocol OLSR at the IP layer and to establish an overlay of Mediators that (1) increase availability of DENS through replication, and (2) perform delay tolerant transport of destinations in different partitions. Beside the advantage that a standardized routing protocol can be used to forward messages to the destinations if there is a route, we use the routing table information to optimize the Mediator synchronization in the overlay. OLSR maintains continuously at each node membership and topology information about its partition. Changes in the membership indicate a network merging or partitioning. If the set of member nodes in a partition is for some time unchanged, all nodes in the partition have the same view. Through simulations we have demonstrated that this assumption is correct, and we have quantified for different scenarios how long it takes until partitions are stable and all nodes have the same membership information. By observing the routing table, we can identify partition mergings and estimate when the merging has finished without exchanging any additional messages. The fact that all nodes have the same membership information, enables us to optimize the synchronization of Mediators since a single Mediator can act on behalf of the others in its partition. Since all nodes in a partition are known, the “election” of a representative is based on the node ID and no messages need to be sent. Additionally, all non-representative Mediators act as hot-standbys in case the representative disappears unexpectedly.

Different optimizations can be done to improve the efficiency of the protocol. These include to prevent too frequent synchronizing among Mediators in case of frequent topology changes; to use a compact representation of the subscriptions in the summaries sent in the Global Synchronization phase; and to use information about disappeared nodes from the Mediator’s local routing table. To prevent Mediators from repeatedly synchronizing with the same Mediators, the Mediators can remember when and with which Mediators they have synchronized. Bloom filters [1] can be used to summarize data. If a node detects disappeared nodes, some of the timers might be cancelled and the Mediators resume the protocol quicker, e.g., if. a Mediator is waiting for a REP_BROADCAST from its assumed *partition_representative* and it detects that this node is gone, it can resume the protocol as if the timer has fired.

However, even without optimizations, the number of messages exchanged increases only linearly with the number of Mediators in the absence of exceptions. We argue that even in the worst case, our synchronization protocol does not perform worse than an approach based on Epidemic Routing in terms of bandwidth consumption and delivery delay since we are not depending on the fact that two nodes meet and we in most cases synchronize more than two nodes. We will continue to work on the implementation of the protocol and evaluate it with real world traces and compare its performance with Epidemic Routing.

ACKNOWLEDGMENTS

This work was funded by the Norwegian Research Council in the IKT-2010 Program, Ad-Hoc InfoWare Project No. 152929/431, and supported by the NoE CONTENT.

REFERENCES

1. Bloom, B., Space/Time Trade-offs in Hash Coding with Allowable Errors, *Communication of ACM, 13(7)*, July 1970
2. Boldrini, C., Conti, M., Passarella, A., Impact of Social Mobility on Routing Protocols for Opportunistic Networks, *1st IEEE WoWMoM Workshop on Autonomic and Opportunistic Communications (AOC 2007)*, Helsinki, Finland, June 2007
3. Camp, T., Boleng, J., Davies, V., A Survey of Mobility Models for Ad Hoc Network Research, *Wireless Communication and Mobile Computing (WCMD)*, 2002
4. Clausen, T., Jacquet, P., Optimized Link State Routing Protocol (OLSR), *RPC 3626*, October 2003
5. Drugan, O. V., Plagemann, T., Munthe-Kaas, E, Predicting Time Intervals for Resource Availability in MANETs, *The IEEE International Workshop on Ad Hoc and Ubiquitous Computing (AHUC2006), Taichung*, Taiwan, June, 2006
6. Hui, P., Yoneki, E., Chan, S., Crowcroft, J., Distributed Community Detection in Delay Tolerant Networks, *ACM SIGCOMM Workshop (MOBIARCH)*, Kyoto, Japan, August 2007
7. Khelil, A., Marrón, P.J., Becker, C., Rothermel, K., Hypergossiping: A General Broadcast Strategy for Mobile Ad Hoc Networks, *Ad hoc Networks Journal*, 2006
8. Pužar, M., Plagemann,T., NEMAN: A Network Emulator for Mobile Ad-Hoc Networks, *8th Int. Conf. on Telecommunications (ConTEL)*, Zagreb, Croatia, June 2005
9. Musolesi, M. Mascolo, C., Hailes, S., EMMA: Epidemic Messaging Middleware for Ad hoc Networks Personal and Ubiquitous Computing, Springer, vol. 10, no. 1, pp. 28-36, February 2006
10. Skjelsvik, K. S., Goebel, V., Plagemann, T. A Highly Available Distributed Event Notification Service for Mobile Ad-hoc Networks, *ACM/IFIP/USENIX 5th International Middleware Conference (Middleware 2004)*, October, 2004
11. Vahdat, A., Becker, D., Epidemic Routing for Partially Connected Ad Hoc Networks, *Technical Report CS-2000-06, Department of Computer Science, Duke University*, 2000
12. Zhao, W., Ammar, M. Zegura, E., A Message Ferrying Approach for Data Delivery in Sparse Mobile Ad Hoc Networks, *5th ACM Symp. on Mobile ad hoc networking and computing (MobiHoc)*, 2004
13. Zhang, Z., Routing in Intermittently Connected Mobile Ad Hoc Networks and Delay Tolerant Networks - Overview and Challenges, *IEEE Communication Surveys and Tutorials*, January 2006