Using Routing Information to Optimize Synchronization of Replicated Event Notification Mediators in Sparse MANETs

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

ABSTRACT
Mobile Ad-Hoc Networks maintain information about reachable nodes in the routing table. In many application scenarios, 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 our 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 which each individual node uses 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

1. 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 in which no communication infrastructure is available, either because it was not existing before, 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, tasks might change and group memberships might change, which is reflected at the network level through changes in the partitions. Evidence for such group mobility is not only given by our study of ER operations, but also confirmed for example by recent studies of social mobility [2] and community detection [6] in opportunistic networks.

For application domains with mobile groups that have for a certain time stable membership, we have developed a highly available Distributed Event Notification Service (DENS) [10]. DENS is based on two design principles: First, we use Mediators to replicate data about subscriptions, because replication enables graceful degradation in case of network partitions. Second, we use Mediators 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 the 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 over the MANET to perform delay tolerant transport through so-called "*store-carry-forward*" operations [12]. The replication of undelivered subscriptions and notifications increases the probability that one of the Mediators at a later point in time joins a partition with a formerly unreachable destination.

There exist various works related to routing in intermittent connected networks, or sparse MANETs. In [14], 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, or using some knowledge about the whereabouts of nodes, or prediction of future location. Epidemic routing is a simple scheme that just forwards packets to neighbors. 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. The message ferry approach [13] 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 the application domains we target. Therefore, we replicate undelivered subscriptions and notifications to all Mediators, similar to the approach in epidemic routing [12]. 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.

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 iteratively over several time steps new nodes. Third, nodes are mobile and there might never be a globally correct definition whether partition merging has finished or not. 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 that is collected by the OLSR routing protocol. By this each Mediator node can establish its own view of its network partition 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 the membership information in their 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 and description of future work in Section 5.

2. DENS

In publish-subscribe systems, subscribers express their interest in events in subscriptions and publishers publish events of interest. The subscribers and publishers are decoupled by the event notificatservice. 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 performed at the source to avoid sending notifications about events that no subscriber has expressed interest in. DENS itself is subscription-language independent, by using subscription language plug-ins. How this works, is described in [11].

DENS has the following components: (1) Subscriber, (2) Publisher and (3) Mediator. The Subscriber and Publisher Component run in every node, and the Mediator Component runs in some of the nodes. The Subscriber Component receives subscriptions from applications and other middleware services. These are then sent to a Mediator Component, either running on the same node or another node. In addition, the component receives notifications and forwards these to the correct application. The Publisher Component receives subscriptions concerning events of interest that may occur on the node, and filters events according to the subscriptions. When an event of interest occurs, it sends a notification to a Mediator Component. The Mediator Component parses received subscriptions to find keywords that are used to locate potential publisher nodes by the help of another Ad-Hoc InfoWare component, the Knowledge Manager [11]. The subscriptions are then sent to these publisher nodes. From publisher nodes, the Mediator receives notifications. The notifications are matched with the subscriptions, and then delivered to interested subscribers.

The nodes running the Mediator Component are called Mediators for short, and decouple subscribers and publishers. The DENS keeps track of Mediators by listening to beacons sent by these nodes in its partition. The Mediators form a DENS overlay. The Mediators in the overlay deliver subscriptions to publisher nodes, and notifications to subscriber nodes. In addition, the Mediators replicate subscriptions and possibly notifications. The reason for storing the subscriptions in the overlay is to obtain a higher degree of reliability in the presence of partitionings, either because of disconnections or that a node is turned off. The task of the overlay is thus to enhance reliability and support delay-tolerant routing of subscriptions and notifications in case of partitions. To summarize, subscriptions are stored at the following nodes:

- At the node where the subscriber application runs, so the Subscriber Component can forward a notification to the correct application.
- At all the Mediators in the overlay.
- At the publisher node explicitly requested by the subscriber or at all publisher nodes where the event of interest described in the subscription may happen, to do source filtering.

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 un-delivered notifications, are done by using the underlying routing protocol and the synchronization protocol. The synchronization protocol is initiated when there are new node(s) in the partition. The presence of new nodes then provides the means for delivering un-delivered stored subscriptions and notifications to the newly connected nodes, and replicating subscriptions and un-delivered 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 to initiate the synchronization process among Mediators.

3. 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 in the worst case, i.e., 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 involved actively 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 (so-called *HELLO messages*) to inform other nodes of its presence. In addition, OLSR tries to maintain at each node a consistent view of the whole 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 in different partitions that merge, we need to understand the dynamics of both the merging process and the partitioning process and how it is reflected in the routing tables in the individual nodes. We have performed a number of experiments with the emulation tool NEMAN [8] to analyze how often the local routing table is changed 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 routing tables which are stable 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 sub-graph 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 on the very border of the network 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, and which we consider 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. 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, mostly including two static groups with 10 to 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.

Table 1: Resulting times for merging and partitioning

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

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 500x500, 1000x1000 and 1500x1500, and 250 units radio range. As expected are routing tables in very dense networks very stable, i.e., membership does not change during the entire run. In the larger areas there are also longer periods in which the individual routing tables do not change. In the studied worst case, i.e., area size 1000x1000, this is often more than 10 seconds and even in larger areas this stable period is 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.

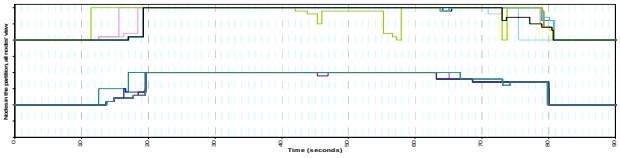

Figure 1: Two mobile groups merging and partitioning; each sub-graph represents the view of one group

4. SYNCHRONIZATION PROTOCOL

The basic idea for the synchronization protocol is to synchronize data after a merging of two or more partitions, using information from the routing table. The protocol is initiated in a node when the local Resource Manager (RM) 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 by the RM 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. One example is that a *partition-representative* or a *coordinator* disappears during the process.

4.1 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 assure 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, the 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.

4.2 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 being its *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. The events triggering the different phases are shown in Figure 2.

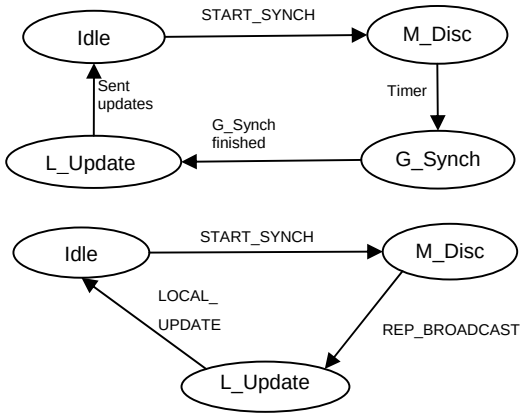

Figure 2: Mediator and *partition_representative* states

The Basic Protocol runs without exception handling 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.

4.2.1 Mediator Discovery

A Mediator enters this phase when it receives a START_SYNCH message from its local RM. The Mediator starts a timer. Each Mediator examines its set of known Mediators and decides whether it is a *partition_representative*. The Mediators that take the role of a *partition_representative*, broadcast a REP_BROADCAST message. This message says that the sender takes the role of a *partition_representative*, and 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.

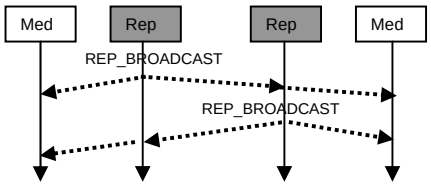

Figure 3: Mediator Discovery (messages)

4.2.2 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 that are used in this phase are: SYNCH_C, SYNCH_REP, and SYNCH_TOTAL. A timer is started when the Mediators enter this phase to ensure that the protocol finishes.

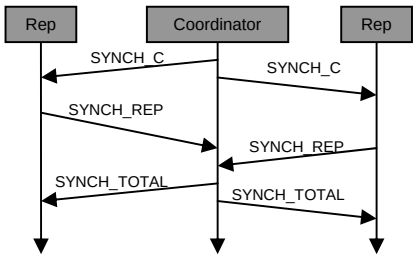

Figure 4: Global Synchronization (messages)

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.

4.2.3 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 to make sure that it will complete the phase, 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.

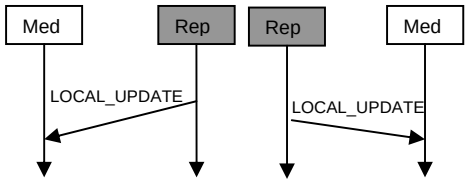

Figure 5: Local Update (messages)

4.3 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, in particular, the other Mediators in their partition, 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.

Table 2: Exception Handling when idle

Role	Exception	Handling
M	REP_BROADCAST	Start synchronization

Table 3: Exception Handling in Mediator Discovery phase

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 4: Exception Handling in the Global Synch 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 5: Exception Handling in the Local Update phase

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

A Mediator enters the Mediator Discovery phase either 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 to be 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 start a new 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 role of being a *coordinator*. 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 and not handled in any specific way. 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 C 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 Mediator receives a timeout, then 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.

5. 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. Besides 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 in each node membership and topology information about their partition. Changes in the membership indicate a partition 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, because for each partition a single Mediator can act on behalf of the others. 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 exchanged to achieve an agreement among the Mediators. Additionally, all non-representative Mediators act as hot-standby in case the representative disappears unexpectedly.

Different optimizations can be done to improve the efficiency of the protocol. These include to prevent Mediators from synchronizing with each other too often based on very frequent changes in the topology; represent the subscriptions in the summaries sent in the Global Synchronization phase in a compact way; and to use information from the Mediator's local Resource Manager about disappeared nodes. To prevent Mediators from 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 the Resource Manager reports about disappeared nodes, it may be the case that some of the timers can be cancelled and the Mediators can resume the protocol quicker. If e.g. a Mediator is waiting for a REP_BROADCAST from its assumed *partition_representative*, and the Resource Manager reports 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 linear 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 Epidemic Routing in terms of bandwidth consumption and delivery delay since we are not depending on the fact that two nodes meet, and we synchronize in most cases more than two nodes. To verify this claim, we are currently implementing the synchronization protocol and compare its performance with Epidemic Routing.

Acknowledgements

This research was funded by the Norwegian Research Council in the IKT-2010 Programme, Project No. 152929/431. It was also supported by the CONTENT Network-of-Excellence. The authors would like to thank the Toilers Group at the Colorado School of Mines, USA, for allowing us to use their mobility models implementation.

6. REFERENCES

[1] Bloom, B., Space/Time Trae-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 2326*, October 2003

[5] Drugan, O., Plagemann, T., Munthe-Kaas, E., Predicting Time Intervals for Resource Availability in MANETs, *IEEE Int. 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] Puzar, 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, *IEEE Distributed Systems Online*, 2004

[11] Skjelsvik, K.S., Lekova, A., Goebel, V., Munthe.-Kaas, E., Plagemann, T. Sanderson, N., Supporting Multiple Subscription Languages by a Single Event Notification Overlay in Sparse MANETs. *ACM MobiDE 2006 Workshop*, June 2006

[12] Vahdat, A., Becker, D., Epidemic Routing for Partially Connected Ad Hoc Networks, *Technical Report CS-2000-06, Department of Computer Science, Duke University*, 2000

[13] 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

[14] Zhang, Z., Routing in Intermittently Connected Mobile Ad Hoc Networks and Delay Tolerant Networks - Overview and Challenges, *IEEE Communication Surveys and Tutorials*, January 2006