Optimistic Fair Transaction Processing in Mobile Ad-Hoc Networks[1]

Joos-Hendrik Böse, Katharina Hahn, Lars-Christian Pelz, Manuel Scholz

Department of Computer Science and Mathematics, Freie Universität Berlin
{boese, khahn, pelz, mscholz}@mi.fu-berlin.de

Abstract: Mobile ad-hoc networks (MANETs) are unstable. Link errors, which are considered as an exception in fixed-wired networks must be assumed to be the default case in MANETs. Hence designing fault tolerant systems efficiently offering transactional guarantees in these unstable environments is considerably more complex. The efficient support for such guarantees is essential for business applications, e.g. for the exchange of electronic goods. This class of applications demands for transactional properties such as money and goods atomicity.
Within this technical report we present an architecture, which allows for fair and atomic transaction processing in MANETs, together with an associated application that enables exchange of electronic tokens.

1 Introduction

Mobile devices in form of mobile phones, PDAs and laptops ingress everyday life and form small mobile ad-hoc networks (MANETs) providing services to their users and environment. Depending on the application semantics these services may be processed in a transactional manner e.g. eBusiness applications demand for a certain degree of guarantees like money and goods atomicity [TY98]. Providing transaction processing with guaranteed transactional behavior in volatile environments is a difficult task. Constant change in available connections and unpredictable behavior of the individual users leave little ground to build guarantees on. However, there are applications demanding for guarantees, for instance business applications exchanging electronic tokens and other electronic goods (e.g. ring tones for micro payment) in a MANET. The main requirement of such applications is fair transaction processing. The notion "*fairness*" in this work is understood as described in [AW97]: An exchange of goods is called *fair exchange*, if either each party has received what it expects to receive or no party has received anything valuable. We will show that guaranteeing fair exchange is not possible at hundred percent but only an alleviated notion called *weak fairness*, meaning that fair exchange cannot be achieved but every

[1] We would like to acknowledge and thank Frank Bregulla, Sascha Gottwald, Susanne Richter and Andrea Schuhmann for contributing on the design and implementation of this project.

party can prove its fair behavior and incorrect behavior of its counterpart in front of a third trusted party.

In this report we present an application for the fair exchange of electronic tokens deployed in a MANET. Weak fairness is achieved by using a specifically adapted protocol combined with an underlying middleware architecture called the Shared Log Space (SLS). This work was accomplished with a special application scenario in mind, which is described in the next subsection.

1.1 Application Scenario

The application scenario of this work is the exchange of electronic tokens on a campus site within a MANET. A campus cafeteria hands out electronic tokens called Coffee Points (CP) for each coffee sold. If the buyer has collected a certain number of CPs, he can return these points to get a coffee for free. A token is transmitted to the mobile host (MH) of the buyer, e.g. a PDA or mobile phone. Issuing tokens and swapping the tokens for coffee are transactions processed between a fixed host (FH) (the cafeteria) and a MH (the customer).
Collectors of CPs can trade their CPs for any other files such as lecture notes, minutes of examination or any other document. Such exchange of electronic goods happens in a peer to peer manner directly between MHs without any FHs involved. Since fixed infrastructure e.g. managed WLAN is not assumed to be available everywhere on the campus, MHs must form a MANET to communicate in these areas.
Although a CP has very little value we suppose that the user is only willing to use the system if some degree of money and goods atomicity can be guaranteed and fair behavior is to be expected. Hence a user expects the system to ensure a certain degree of fairness. The following section describes further basic assumptions of the system model.

1.2 System Model

As already mentioned we assume a system model where MHs communicate directly with each other in a peer-to-peer manner. The cafeteria host is the only FH within the system. It is only used for issuing CPs and validating returned CPs. The cafeteria host is the only instance allowed to create CPs. It holds information about the currently circulating CPs. When a CP is returned, the validity of this CP is checked. The cafeteria host is the only authority, that can decide whether a CP is valid or not.

MHs are supposed to be unstable and may join and disappear from the MANET any time. It is possible that a MH will never reconnect to the MANET, because the device is physically lost or destroyed. In this case the CPs stored on the MH are also lost. This is equivalent to losing a purse containing money in real life. A CP is permanently stored on the mobile device so it will outlive energy shortages and user initiated shutdowns. But it is the user's responsibility not to manually destroy CPs, e.g. by formatting disk space.

Trading of CPs happens directly between MHs using wireless links establishing a point-to-point connection between two MHs. The wireless communication protocol

assumed here is IEEE 802.11[2]. These links are established directly or via other. If multi-hop routing is available within the MANET we assume that within our scenario no network-partitioning occurs. This assumption is feasible as we imply a dense network[3].

Every node is autonomous and solely decides on its own which actions to perform. There is no way to prevent nodes from malicious behavior. We assume that a repudiation system as described in [OK03] further on motivates fair behavior of all users. In our implementation, fairness is assured by allowing only proprietary software running on each node.

We assume that correctness of coffee point trading is *not* time-dependent. This implies that a CP used in an active transaction cannot be used, as long this transaction is running. Hence a CP is possibly blocked for an arbitrary long time. At the current state no routing or forwarding of datagrams is assumed to span the SLS. It is sufficient to communicate with nodes in single hop distance.

1.3 Requirements

The exchange of electronic tokens as described above requires that sending, receiving and deleting of exchanged items must happen in an atomic manner. Participants expect goods atomicity. This means that a coffee point is not duplicated nor destroyed within the overall, except an MH being physically destroyed or it suffers a total loss of its state.

Assuming arbitrary long disconnection periods, it is critical to assure that two MHs finish their exchange transaction atomically. It is possible that one MH disconnects from the MANET while processing a transaction with another MH. When the MH reconnects to the MANET it must learn about the transaction state of the disrupted transaction. This is normally done by contacting the previous interaction partner but in a MANET the partner MH is most likely also disconnected from the MANET at the point of reconnection. Hence the recovering MH is most likely uncertain about the state of its transaction partner.

We can summarize the requirements to be met by the system as follows:

1. The exchange of a CP must happen in an atomic manner, to assure that eventually a valid CP must reside only on one MH or on the FH of the cafeteria.
2. To meet the first requirement it must be guaranteed, that two MHs processing a transaction eventually come to the same decision: Either aborting the exchange or finishing it.
3. The exchange of a CP for another electronic good must provide for weak fairness. This means that:
    a. If a MH did not receive what it expected, it must eventually be able to prove the faulty behavior of the other party in front of a third trusted party.

[2] The wireless communication protocol has no influence on the methods presented here but on some implementation issues such as neighbor discovery, which is part of the specification in some protocols such as Bluetooth.

[3] Although this is a strong assumption we think it is plausible in our approach, as the campus site is mostly crowded with mobile devices.

b. If a MH acts according to the protocol and is accused by the other party of acting incorrect, it must eventually be able to prove its correct behavior.

To be able to guarantee atomicity as specified in requirements 1 and 2, the system must be either able to prevent link failures or provide mechanisms to recover from

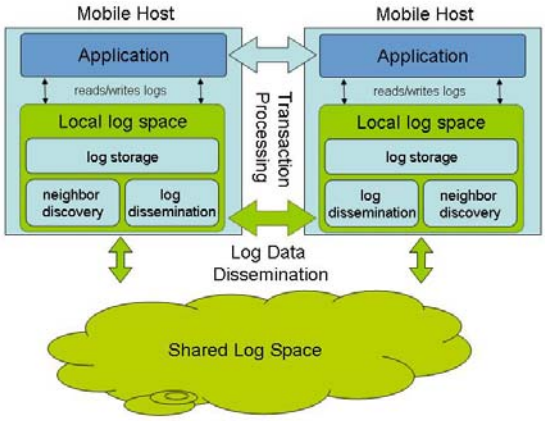

Figure 1: SLS Architecture

link failures. Preventing link failures is not possible, as there is no possibility to influence the behavior of autonomous MHs. Another approach would be to consider the context of MHs and evaluate the probability of link failures during transaction processing. If the risk of a link failure during transaction processing is considered too high, participants could decide not to start the transaction. However, modeling the context of MH is very complex and will be addressed in future work. In this work we focus on providing recovery mechanisms to survive link failures. We preserve the exchange state in case one MH fails and facilitates recovery by providing required recovery information independent of the connection state of the participants. Requirement 3 also demands for such an impartial storage where contracts can be saved to prove faulty and fair behavior independent of the participants' presence. The requirement to preserve information in case of failure and for evidence is addressed by a component called Shared Log Space (SLS). This middleware component is described in Section 2. Section 3 presents the protocol developed to achieve an atomic commit and fairness using the SLS to log evidence information and transactions states for recovery.

1.4. Atomic Commit Guarantees

When reasoning about atomicity guarantees in MANETs it is of interest, if the Atomic Commitment problem is solvable in such a system model, as Atomic Commitment relates on Consensus which is known to be unsolvable in the

asynchronous system model. The impossibility of solving the Consensus problem in asynchronous system-models was proven in [FLP85], which motivates [CT96] to extend the asynchronous system model with the concept of unreliable failure detectors. The model of failure detectors assumes that every process has access to a failure detector, which can suspect other processes to be crashed; such suspicion can be faulty and possibly later be corrected by the failure detector. It is shown in [CT96] that Consensus is solvable with unreliable failure detectors in asynchronous systems. If the system-model assumed in [CT96] equals our system-model, Consensus is also possible to solve in our system. Further on we must decide if the Atomic Commitment problem is equal to Consensus. The system-model assumed in [CT96] makes the following assumptions:

1. There are no upper time bounds about the relative speed of processes and delivery times of messages. Processes do not have access to synchronized clocks.
2. Processes can crash at any time.
3. Each process has access to a local failure detection module, which maintains a list of processes, which are currently suspected to have crashed. A failure module can make mistakes by adding a process to its list of suspects that is still running.

In our MANET system-model we also assume that mobile devices crash at any time, which is equivalent to assumption two of the system-model assumed in [CT96]. We assume that mobile nodes can recover and reconnect to the MANET at any time, which is equivalent to assumption one of [CT96], because processes that are running slow exhibit the same behavior. They will be able to answer a message or proceed within a protocol as recently as CPU-time is available. Mobile nodes exhibit the same behavior, as they are only able to communicate with other nodes when reconnecting to the MANET. Assumption three is implementation related, as we can easily implement an unreliable failure detector based on time-outs within our system. Hence, our MANET system-model is equal to an asynchronous system-model with unreliable failure detectors as proposed in [CT96] and we can conclude that the Consensus problem is also solvable in our system-model.

It is further shown in [Gu95] that Non-Blocking Weak Atomic Commitment (NB-WAC) is reducible to Consensus and hence solvable in asynchronous systems with unreliable failure detectors. We must show that NB-WAC satisfies our requirements defined in Section 1.3 NB-WAC in [Gu95,GL04] is defined by the following safety and liveness properties:

1. Safety:
    a) Stability: Once a process has entered the committed or aborted state, it remains in that state forever.
    b) Consistency: No process can be in a *committed* state while another process is in the *abort* state.
2. Liveness:
    a) Non-Triviality: Commit must be decided if all processes enter the prepare state and no process is ever suspected to be crashed.
    b) Non-Blocking: If, at any time, a sufficient large network of nodes is non-faulty for long enough, then the algorithm executed on those nodes will eventually reach either the *committed* or *aborted* state.

The requirements one and two, described in Section 1.3 are obviously covered by the two safety properties of NB-WAC[4]. While requirements one and two of Section 1.3 ensure atomicity, weak fairness is not assured by NB-WAC. Hence we are able to provide atomic guarantee on the item exchange transaction, but we cannot provide any guarantees about fair behavior of the nodes. We will propose a protocol in Section 3, which facilitates the SLS architecture described in Section 2 to ensure weak fairness.
To conclude this Section it can be summarized, that it is proven to be possible to provide atomicity guarantees in our MANET system-model. But for efficient support of atomicity guarantees the liveness properties are crucial, such properties ensure that used protocols make any progress and participants will not block forever. As stated in the liveness property b) it is only possible to guarantee liveness if a sufficient large network of nodes is non-faulty for long enough. We cannot make any statements about how long a network must be non-faulty or when a non-faulty network is large enough. Hence we are not able to give any more detailed guarantees than stated above. For example time critical transaction semantics cannot be supported. We will propose an approach in Section 2 to improve liveness of atomic commitment by reducing the probability of blocking.

2 Architecture of the Shared Log Space (SLS)

Liveness of recovering nodes is usually provided by recovery mechanisms. Recovering nodes must be able to proceed with their protocol and reach a new consistent state. This is usually done by analyzing local log data, which is stored in the local stable storage and by communicating with other participants previously involved in transaction processing to learn about their state. But in the MANET system-model fluctuation of nodes is high and thus querying previously participants or other defined nodes about passed events after its disconnection is not a good choice, because all other nodes involved in transactions may likely have already disconnected from the MANET.

The main idea of the SLS is to establish a virtual shared log device, which every node within the MANET can access. The SLS is collectively formed by individual so-called *Local Log Spaces* (LLS) components residing on every MH, which share local log data among them[5]. The content of the SLS is the union of the content of all LLS. A node accesses the SLS through its local LLS; hence its “view” is limited to the content of its LLS. Log data written to the LLS of a node is distributed among the other LLS of the SLS. Log items written to the LLS are persistently stored on the device to survive shutdowns.

The LLS is designed as a middleware component that can be accessed by any application running on the same device. Figure 1 depicts the basic architecture of the SLS. To form an SLS every participating node must perform the following actions:

[4] From an implementation view we can implement the stability property because we assume that every node has a stable memory to safe states persistently.

[5] The SLS approach was inspired by other tuple-space based middleware approaches such as [MPR03]

1. When connecting to a network, a node discovers neighbors participating to the SLS in transmission range.
2. The content of its LLS must be synchronized with the content of neighbor LLS.
3. If an application writes log data into its LLS, the LLS must disseminate these logs among the other participants of the SLS.
4. Log data received by other LLS must be stored in the local LLS and if associated applications are running on the node, these applications must be informed.
5. Every LLS periodically synchronizes itself with other LLS in its environment.

In the following the components of the SLS and their responsibilities as well as some implementation issues of the SLS, are introduced in detail.

2.1 SLS Components

Every LLS consists of a log storage component, a neighbor discovery component and a log dissemination component, each responsible for one of the described tasks above. Figure 1 depicts the overall architecture of the SLS. In the following sections the mode of operation of these components is described in detail.

2.1.1 Neighbor Discovery Component

Nodes must be able to seek other nodes in their neighborhood in order to update the content of their LLS and contributing to the LLS of their neighborhood. Discovering neighbors and keeping track of them is the main task of the neighbor discovery component. It employs a simple and stable mechanism to detect other nodes contributing to the SLS in the local surrounding. As a simple implementation we decided to broadcast a presence message in single-hop environment. Every node periodically broadcasts a short message to inform other nodes in single-hop distance about their presence. Nodes receiving a presence message can extract the unique identifier and address of the sending node and maintain a presence list. Such a list contains all neighbors that have recently sent a message. A node is removed from this list, if no message was received for a certain time. We found that this mechanism is sufficient for neighbor discovery in our system.

2.1.2 Log Dissemination Component

The log dissemination component is responsible for two different tasks: It synchronizes the LLS with nodes in the direct neighborhood when reconnecting to the MANET and periodically while in connected state. It is also responsible for disseminating log items written to the LLS by a local application among its neighbors.

Synchronization of Log Items

An application accesses the SLS by querying the local LLS. Its view of the SLS is determined by the content of this LLS. Hence it must synchronize its LLS content with other nodes contributing to the SLS. The main goal here is to achieve a high

freshness of available log items. Synchronization is initialized whenever a node is entering or reconnecting to the MANET.
While the simplest way of getting missing log entries is to request the contents of all neighbor LLS' and merge them with the local one, this causes a huge overhead of redundant data being sent. To minimize the message overhead we employ to use *lists of identifiers of log items.*
Such a list contains the IDs of all logs currently stored in the LLS. A synchronization process starts by broadcasting a list of identifiers of local log items. Each receiving node then determines the difference-set, by comparing the received list to its own. The difference-set contains all log IDs the node owns, which are not listed in the received list. The size of the difference-set is sent back to the synchronizing node. This node collects the sizes of all difference-sets and then requests the node with the greatest difference-set to send the according logs. Such synchronization strategies is also used in some dissemination protocols, for example in [BBHS05]. Because log data is never altered after having been submitted to a LLS it is satisfactory that a synchronization of two LLS always results in the union. A conflict in the sense that an item has been changed concurrently by two parties cannot happen.

Dissemination of Log Items
New log data from a local application is stored in the LLS' log storage component and is immediately broadcasted to all nodes within the direct neighborhood. Nodes receiving such a broadcast message store the received log item in their local storage component and consider this log item in following synchronization runs. It is shown in [BBHS05] that such, dissemination strategy results in a high up-to-dateness of data within the whole MANET.

From the application view the dissemination component provides some kind of force log write, the log write operation will first return after the log item is distributed and hence secured within the SLS.

2.2 Interaction of Components

A log entry being received from a remote node is immediately written to the persistent storage of the receiving LLS. The write operation used by an application to insert log data into the LLS provides a "forced write" semantic. The operation returns as soon as the log entry is persistently stored by the log storage component and broadcasted in single-hop environment.
An application starting a transaction registers at its local LLS to be informed if new log items belonging to this transaction are received.
LLS receiving the broadcast message of other LLS feed the log items into their log storage component and notify according applications about the new log items. LLS clients may not only be presented with logs on their arrival at the LLS, they can also request subsets of the LLS content concerning e.g. different transactions or other parameters. This enables clients to perform recovery procedures if needed.

2.3 Implementation

We have implemented the LLS agent for three platforms:

- Microsoft C Sharp .NET for PocketPC (.NET Compact Framework)
- Java 1.4 Standard Edition
- Java MIDP 2.0 CLDC

The essential requirement our LLS implementation is built on an IEEE 802.11 compliant WLAN device built into the handheld that the program should work on. In spite of that it should run on any handheld with the PocketPC or Windows CE platform and the Microsoft .NET Compact Framework installed.

2.4 Transactions and Logs

A transaction in the following refers to a set of log entries, which is characterized by having the same transaction identifier. Transaction identifiers can be any string, e.g. a number in base64 encoding. Note that they must be unique in order to be able to distinguish between different transactions. We therefore chose the transaction identifier to be a combination of the node's identifier which is originating the transaction and a local timestamp.
A log or log entry refers to a single data entity in the Log Space. Its uniqueness is guaranteed by the transaction identifier and the according message of the protocol used. Each log entry has a payload field, where the actual log data can be inserted.
All data and all messages are encoded in XML, which ensures portability and expandability.

3 Exchange Protocol

In this section, we describe a protocol using the SLS. We have implemented the scenario described in Section 1.1 and 1.3, where arbitrary electronic items are exchanged in a transactional manner.
In order to initiate such a bilateral trading transaction a participant **O** (originator) uses the SLS to announce his offer. An offer is an xml-document containing information about **O** himself, the description of the item **O** offers $item\_descr_O$ and a description of the electronic item he wants in return. A participant **R** (recipient) of the SLS who is interested in the offer can respond to this offer by informing **O** about himself and a description of its item. **O** then accepts **R**'s offer and starts the protocol.
In the following the transaction protocol is described in detail. It focuses on guaranteeing possibilities to recover from failures. It also assures weak fairness to the participants.
We slightly adapt the Optimistic Protocol for Fair Exchange (OFex) proposed by [AW97] to mobile devices in MANETs. In the OFex protocol fairness is ensured using non-repudiation tokens: The **n**on-**r**epudiation of **o**rigin (NRO) token ensures that the originator of an item cannot later falsely repudiate having originated that item. The **n**on-**r**epudiation of **r**eceipt (NRR) token guarantees that the recipient of an item cannot falsely deny having received the item. OFex therefore guarantees weak fairness to the initiator of the protocol (an affidavit will be issued by a third trusted

party if the originator has sent its item and did not receive the promised item in return) and strong fairness to the recipient.
We slightly adapted this protocol to match the mobile circumstances. The messages that are exchanged slightly differ in order to enable recovery and to prove fair behavior. Additionally to sending messages to the transaction partner, logs are sent to the LLS and then disseminated as described in Section 2.1.2. We assume that sending a message and logging the according log is done atomically. The transaction protocol does not include the exchange of items rather than concluding a contract. This contract is separately used to redeem the item. It includes the complete transaction context consisting of non-repudiation tokens, item descriptions of the involved items, the location where the items can be picked up and commitments of both partners that are used to ensure fairness to both of them.
The protocol can be divided into three phases: In the first phase both participants commit to exchanging their items by sending a description of the according item. Additionally the hash-values of randomly chosen numbers $h(r_O)$ and $h(r_R)$, using an ideally collision-free hash-function, are exchanged which are later needed for the non repudiation tokens. **R** uses the expect-function which involves user-interaction to affirm, that it wants to trade the given items. The messages exchanged in this phase, m_1 and m_2, are signed as a confirmation on which items to exchange.
In the second phase, the participants exchange the description and the location of their items (e.g. a local or remote url). These messages, *m3* and *m4*, can later be used to prove that the given participant of the SLS has committed that the item described can be picked up at the given location.
In the third phase both participants send the actual value of r_O (and r_R respectively) as a confirmation that they have received the item location. Since sending R's actual item location and the confirmation of having received O's item location can be done in one message, phase two and phase three overlap in message m_4 (see
Figure 2: Contract conclusion using the adapted OFex protocol).
The according log messages are sent to the local log space and are then disseminated among the SLS. They serve two purposes: On the one hand, they are used by nodes that suffered from disconnections to get back to the actual transaction status. On the other hand, the log entries can be used to prove one's fair behavior to a third trusted party.

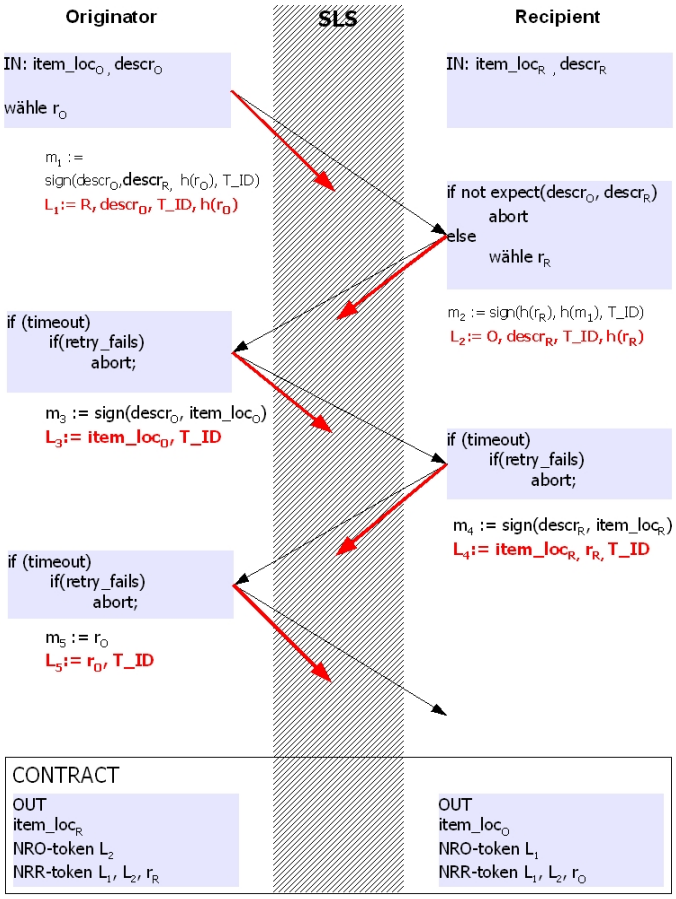

Figure 2: Contract conclusion using the adapted OFex protocol

The output of this procedure is on the originator's side *item_loc*$_R$, at which it can redeem *item*$_R$, the NRO-token (L_2, *key*$_R$, *com*$_R$) which proves, that R has originated *item*$_R$, and the NRR-token (L_1, L_2, r_R) which guarantees that **O** has received *item_loc*$_R$. **R** analogously can now pick up *item*$_O$, according tokens are logged in the SLS. In order to use the SLS as a reliable third party, we assume that logging and sending messages is done atomically. Deliberately unfair behaviour cannot be proven in this protocol. It can only be detected which of the participants did behave as promised.
Separately from the exchange of the item locations, both parties can now pick up the promised items at the given location using the just concluded contract. Proper authentication strategies have to be used to ensure that only the partners of the contract can pick up the item at its location. Therefore the real tenures will converge to the behaviour state of the concluded contracts.

We assume a timeout in each step of the protocol when the participant is waiting for a message to arrive. The participant being connected to the SLS will abort the transaction if it does not receive the requested message in time.
Within the described protocol different failures can occur which have to be distinguished in order to initiate according recovery strategies. The different failure classes are described in the following.

3.1 Failure Classes

In case of connection errors, we distinguish between two failure classes that can occur. If a participant is not able to establish a connection link to his partner, he has to decide (by checking the entries in his neighbor table) whether he is still connected to the MANET. We consider the situation of link failures when nodes still have connection to the MANET (e.g. most of neighbors in the neighbor list are still available) as a link failure of *class one*. A node loosing connection to the MANET including its transaction partner is referred as a link failure of the *second class*.

3.2 Recovery Strategies

If one of the participants suffers from failure of class one and the other one has suffered from failure of class two, recovery works as follows.
The node that is still connected to the network either encounters errors either while sending a message or a timeout runs out while waiting for a message. In both cases he will re-send the according message, which is either the message he had trouble sending with or a request to re-send the message he is waiting for. Additionally he will synchronize with its neighbors to find out, if any logs were written before failure occurred. After a certain timeout the node that is still connected to the network will abort the transaction by writing an abort message to its LLS.
The node that is disconnected from the network will have to block its transaction until it can reconnect to the MANET and the SLS. After reconnection, it searches for the missing log of the blocked transaction. If there is an abort message logged, it will abort the transaction. Else it will do the protocol according to the last known log item. For example if **O** reconnects to the MANET and gathers L_4 as the last known log item, he will search for m_4. **O** proceeds with the protocol by sending m_5.
A node reconnecting to the network after having suffered from failure is not able to abort the transaction unless it has had contact the other participant directly. Otherwise it could lead to non-atomic transaction commit.

We assume that if both nodes suffer from failure at the same time[6] it will be a failure of class two. If both nodes had suffered a failure of class one, network partitioning would have occurred, which we imply not to occur in our scenario.

[6] Note that at the same time does not necessarily refer to the same point in time rather than at the same step of the protocol. E.g. if **R** fails after sending a message and **O** fails before responding this message, it is considered as a failure at the same time.

A node reconnecting to the MANET which is not able to establish a connection with its interaction partner will not be able to abort the transaction and hence will have to keep its transaction blocked. So the node that is reconnecting first can try to send and log a certain message but no timeouts can be used to abort the transaction. Therefore a transaction can still be concluded although both nodes might not be connected to the SLS at the same time by using the needed information stored in the SLS.
The proposed recovery and synchronizations strategies enable atomic contract conclusion for both partners although they might suffer from disconnection. The SLS is used as a shared information space to populate the information that is needed for atomicity. Participants of the SLS mainly benefit from the use of it when disconnection occurs and no direct communication can be established afterwards. In that case, the transaction can still be successfully completed while transaction processing without the SLS leads to aborting or blocking transactions.
Since no item can be picked up unless L_5 is logged the protocol ensures weak fairness to the participants. Both participants can use the concluded contract as a proof of the trade later on. If one party did not expose the item it committed to in the contract, the other one can proof that it did not receive what expected.

4 Performance Predictions

In the following, we discuss the network costs in terms of messages, that need to be transferred in the proposed protocol with the underlying SLS middleware, opposed to the OFex protocol.
When two participants trade items using the proposed protocol and the underlying SLS architecture, the following costs arise. When no failure occurs the two nodes have to exchange the messages which are the same in the original protocol. Additional costs occur by broadcasting the logs that are written in each step of the protocol. So **O**'s LLS has to make three additional broadcasts, **R**'s has to make two.
In case of any failure, both participants will synchronize with their neighborhood, the node inside the MANET in order to avoid missing a possibly written log and the other one when it returns to the MANET. Since we changed the protocol that no electronic goods are exchanged, extra costs occur when picking up the item.
All other participants of the SLS have additional network costs although they might not get involved in a transaction. These consist of the costs of synchronization costs which occur every time a new neighbor is seen. Other synchronization strategies (e.g. a static pull interval) lead to other cost models in this case.

5. Conclusion and further work

In this technical report, we have introduced a system architecture that enables fair and atomic transaction processing in MANETs. The main idea of the architecture is the SLS which is a distributed storage which consists of the union of all LLS of all participating nodes. A log written to the LLS is disseminated among all neighbors in transmission range. A node reconnecting to the network after failure is able to gain

knowledge about the content of the SLS by synchronizing its LLS with those of its neighbors.

We have implemented the SLS along with a dissemination, synchronization and neighbor discovery component on different platforms. The most advanced one is implemented on the Microsoft C# .NET for PocketPC (.NET Compact Framework).

As a sample application for fair transaction processing, we have adapted the OFex protocol to match the mobile environment with the underlying SLS architecture. No electronic goods are exchanged within the protocol. The involved parties conclude a contract exchanging the possession rights of their electronic goods. Using this contract the electronic items can later be picked up. In addition to the messages that are exchanged within the contract, logs about the current transaction status are written to the SLS. These enable the involved parties an atomic commit although one or both might have suffered from disconnection during transaction processing. We have implemented the protocol on the underlying SLS middleware as a sample application for exchanging electronic goods on a campus scenario. Coffee Points are handed out by the cafeteria which can be traded for any other electronic good, e.g. lecture notes or minutes of an examination. The implemented concepts work with different hardware, e.g. Dell Axim X3, Asus MyPal and miscellaneous notebooks.

As part of future work we want to refine the architecture of the SLS in order to be able to use the concept along with different commit protocols among multiple partners. Among others, the refined architecture includes a transaction processor that coordinates a transaction according to the chosen transaction protocol. A pre-processor component could take responsibility for choosing appropriate time-outs according to the protocol, the participants and their local surrounding. Another challenging task is to choose an appropriate coordinator or set of coordinators. This involves evaluation of the nodes' context. Nodes that are more likely to be in the MANET for that duration of the transaction are more likely to be reachable for all participants of the transaction. Possible context-variables that could be taken into account when choosing appropriate coordinators are energy level, signal strength, position and direction.

Using context information, more sophisticated dissemination strategies can be used to disseminate log items. Consider the situation in which relatively stable nodes are available that are able to give high guarantees about their future availability. Having a couple of neighbors that are considered very stable it would be sufficient to disseminate logs among those, rather than disseminating among all LLS. Network and storage costs could be reasonably decreased using context information in the process of data dissemination.

References

[AW97] N. Asokan, M.S., M. Waidner. Optimistic Protocols for Fair Exchange. in Proceedings of the 4th ACM Conference on Computer and Communication Security. 1997. Zurich Switzerland: ACM Press.

[BBHS05] Böse, J., Bregulla, F., Hahn, K., and Scholz, M.: Adaptive data dissemination in mobile ad-hoc networks. In: INFORMATIK 2005 - Informatik LIVE!, Band 2, Beiträge der 35. Jahrestagung der Gesellschaft für Informatik e.V. (GI), Bonn, 19.-22. September 2005. LNI P-68, 2005, pp. 528-532. 2005.

[CT96] Chandra, T. D. and Toueg, S.: Unreliable failure detectors for reliable distributed systems. Journal of the ACM. 43(2):225–267. 1996.

[GL04] Gray, J. and Lamport, L. Consensus on transaction commit. 2004.

[Gu95] Guerraoui, R.: Revisiting the relationship between non-blocking atomic commitment and consensus. In: Helary, J.-M. and Raynal, M. (Eds.), Proceedings of the 9th International Workshop on Distributed Algorithms (WDAG95). Volume 972. pp. 87-100. Le Mont-Saint-Michel, France. 13-15 1995. Springer-Verlag.

[FLP85] Fischer, M. J., Lynch, N. A., and Paterson, M. S.: Impossibility of distributed consensus with one faulty process. J. ACM. 32(2):374–382. 1985.

[MPR03] Murphy, A., Picco, G., and Roman, G.: Lime: A middleware for physical and logical mobility. In: ICDCS '01: Proceedings of the The 21st International Conference on Distributed Computing Systems. p. 524. Washington, DC, USA. 2001. IEEE Computer Society.

[OK03] Obreiter, P., König-Ries, B., Klein, M.: Stimulating cooperative behavior of autonomous devices – an analysis of requirements and existing approaches. Technical Report 2003-1. University of Karlsruhe, Faculty of Computer Science 2003.

[TY98] Tygar, J.D., Atomicity in electronix commerce. netWorker, 1998.