Aus dem Institut für Informationssysteme
der Universität zu Lübeck
Direktor:
Prof. Dr. rer. nat. Volker Linnemann

Adaptive Service Migration and Transaction Processing in Wireless Sensor Networks

Inauguraldissertation
zur
Erlangung der Doktorwürde
der Universität zu Lübeck
- Aus der Sektion für Informatik/Technik und Naturwissenschaften -

vorgelegt von
Christoph Reinke
aus Lüneburg

Lübeck, April 2011

1. Berichterstatter: Prof. Dr. rer. nat. Volker Linnemann
2. Berichterstatter: Prof. Dr.-Ing. Thilo Pionteck

Tag der mündlichen Prüfung: 17. Juni 2011

Zum Druck genehmigt. Lübeck, den 21. Juni 2011

Kurzfassung

Das Paradigma der Serviceorientierten Architektur (SOA) kommt in Geschäftsanwendungen bereits seit Jahren zum Einsatz. Es ermöglicht ein hohes Maß an Flexibilität und Abstraktion, da komplexe Anwendungen aus einfacheren Services zusammengestellt werden können. In den letzten Jahren wurde SOA auch im Bereich der drahtlosen Sensornetze ein Thema. Vor allem die Konzepte der Replikation und Migration von Services sind eine wesentliche Voraussetzung für den erfolgreichen Einsatz von SOA in drahtlosen Sensornetzen.

In dieser Arbeit werden Anwendungsfälle für die Migration von Services in drahtlosen Sensornetzen vorgestellt und es wird erläutert, warum Protokolle zur Transaktionsverarbeitung eine wesentliche Voraussetzung für die Gewährleistung von konsistenter Servicemigration sind.

Wir vergleichen das traditionelle Zwei-Phasen-Commit-Protokoll mit dem Cross-Layer-Commit-Protokoll hinsichtlich ihrer Verwendbarkeit in drahtlosen Sensornetzen und stellen unser eigenes Commit-Protokoll *Zwei-Phasen-Commit mit Caching* vor. Wir zeigen dessen höhere Energieeffizienz sowohl in Experimenten mit dem Netzwerksimulator Shawn als auch in einer Versuchsanordnung mit Sensorknoten vom Typ Pacemate.

Darüber hinaus werden traditionelle Protokolle zur Nebenläufigkeitskontrolle analysiert, an drahtlose Sensornetze angepasst und implementiert. Wir zeigen in Simulationen, dass das Zwei-Phasen-Sperrprotokoll am geeignetsten für den Einsatz in drahtlosen Sensornetzen ist und beschreiben unsere Implementierung für Pacemate-Sensorknoten.

Außerdem wurde ein umfassendes Szenario zur Migration von Services implementiert und in Shawn sowie auf mit dem serviceorientierten Betriebssystem Surfer OS programmierten Sensorknoten evaluiert, um die Funktionsweise der implementierten Protokolle zur Transaktionsverarbeitung in der Praxis zu verifizieren.

Abschließend beschreiben wir die adaptive Auswahl von Commit-Protokoll und Routing-Protokoll zur Laufzeit in Abhängigkeit von den vorherrschenden Bedingungen in einem Sensornetz, wie beispielsweise Nachrichtenverlust oder Dichte. Wir zeigen, dass die adaptive Protokollauswahl das Übertragungsvolumen signifikant reduzieren und damit die Lebensdauer eines Sensornetzes erhöhen kann.

Insgesamt unterstützen die Beiträge dieser Arbeit den Einsatz von fortschrittlichen Anwendungen mit erhöhten Anforderungen an Konsistenz und Koordination unter Berücksichtigung der knappen Ressourcen eines drahtlosen Sensornetzes.

Abstract

In the last years service oriented architectures have made their way from business applications to wireless sensor networks. For the successful usage of service oriented architectures, techniques like replication and migration of services are a prerequisite. We outline use cases for service migration and show that transaction processing capabilities are necessary for enabling stateful and consistent service migration in wireless sensor networks. Additionally, transaction processing capabilities are needed for the extension of sensor network databases like TinyDB and StonesDB and for wireless sensor and actor networks.
In this thesis we compare the Two Phase Commit protocol and the Cross Layer Commit Protocol with regard to their usability in wireless sensor networks and infer our own atomic commit protocol *Two Phase Commit with Caching*. We show in simulations and experiments with the sensor node platform Pacemate, that our Two Phase Commit with Caching protocol is the most efficient protocol for use in wireless sensor networks with regard to energy consumption.
Further we implement and evaluate the traditional database concurrency control protocols locking, validation and timestamp ordering for wireless sensor networks and show in simulations that locking outperforms validation and timestamp ordering in terms of efficiency. We also report our implementation of locking for the Pacemate sensor nodes.
We describe a comprehensive service migration scenario, which enables service discovery and consistent migration of stateful services and we outline our implementation for the network simulator Shawn and Pacemates running Surfer OS.
Finally, we describe an adaptive selection of commit protocols and routing protocols in dependency of the required consistency and the given network context to provide the most efficient transaction processing for a given sensor network deployment.
This thesis supports the usage of sophisticated applications for wireless sensor networks demanding increased coordination capabilities while taking the severe resource constraints of wireless sensor networks into account.

Contents

Chapter 1

Introduction

Wireless sensor networks can consist of a large number of tiny microcontrollers equipped with radio interfaces, sensors and batteries. Sensor networks are mainly used to monitor the environment and to provide the observations to the user so that they can be displayed, analyzed and archived in a convenient way. While the vision of tiny wireless devices had already emerged in the early nineties (see, for example, [200]), together with the paradigms Ubiquitous Computing, Pervasive Computing and also Ambient Intelligence, wireless sensor networks have become a reality in recent years. While wireless sensor networks were firstly targeted for use in military environments, nowadays sensor networks have many application domains, for example habitat monitoring, building automation or smart homes [52].
The features of wireless sensor nodes have greatly increased in the last few years, but the programming of sensor nodes is still very difficult for a user like a biologist or a meteorologist. Instead, a computer scientist or engineer is needed to adapt a wireless sensor network to the needs of a natural scientist so that he does not have to deal with low-level issues like event handling or routing algorithms.
We claim that service oriented architectures, which are already widely used in the area of business applications, are also a convenient way of developing applications for wireless sensor networks.

1.1 Service Migration

Indeed, service oriented architectures have gained interest in the broader sensor network community lately [130, 188]. At our university, a prototype of a service oriented sensor network [124] has also been implemented. It is called Surfer OS and enables the user to migrate code into an already deployed sensor network which runs Surfer OS. A scheme of this possibility is shown in Figure 1.1.
This gives the user maximum flexibility because even low level services like routing protocols can be exchanged at runtime.

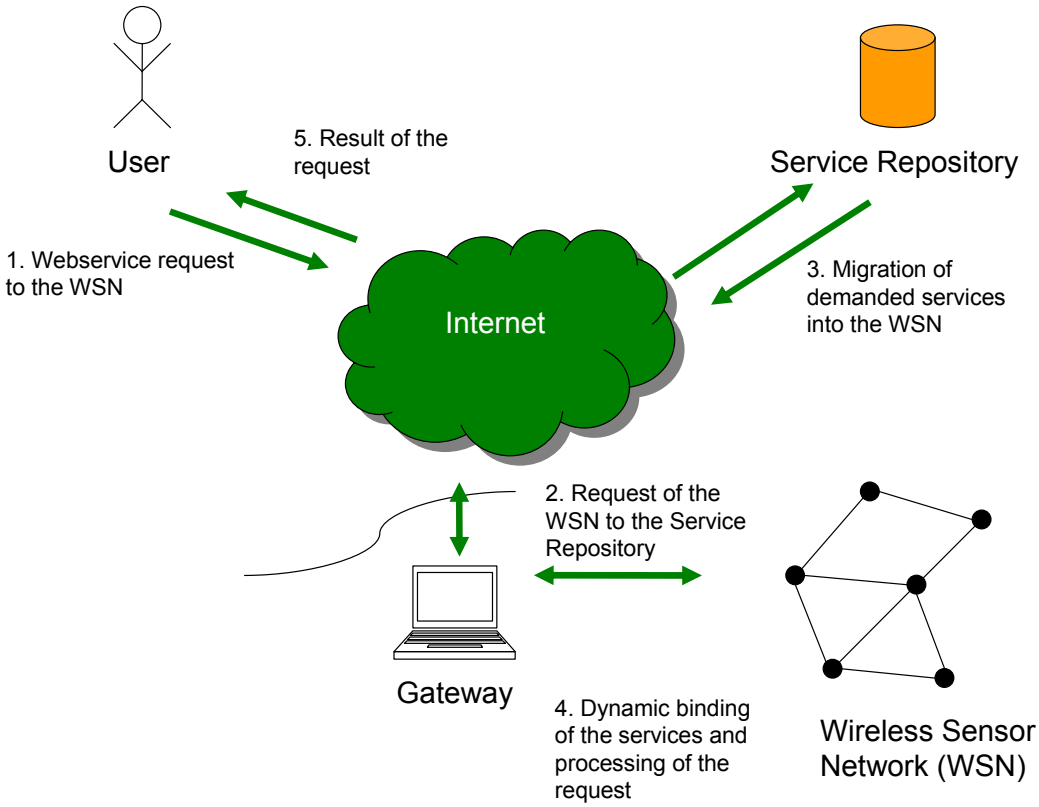


Figure 1.1: Our Service Oriented Wireless Sensor Network. We assume that a user, for example, a biologist, queries the wireless sensor network via the internet (Step 1 and 2). The services that are needed to answer his query are dynamically migrated into the wireless sensor network and assigned to nodes (Step 3 and 4), which send the results back to the user (Step 5).

To take even more advantage of the service oriented concept, we want the sensor network to autonomously migrate services of nodes running out of energy to other sensor nodes in the proximity. Service migration is especially useful when nodes are deployed iteratively, i.e. the wireless sensor network starts to work with an initial subset of deployed nodes and is later extended by additional nodes. Hence, service migration is vital to increase the lifetime of a wireless sensor network. In recent years, many publications in the area of wireless sensor networks have focused on the enhancement of the lifetime. An overview is given by Dietrich et al. [46]. There exist various definitions of lifetime, for example, the time until the first node fails, the time until all nodes fail, definitions that take coverage into account and many more. Dietrich et al. [46] compare different definitions of sensor network lifetime. Since we consider service oriented sensor networks, we are interested in the time during which the network can perform its assigned task successfully. In the worst case, the failure of one node performing a vital role can render the whole network useless, depending on which services the failing node is running. Our approach to circumvent this problem is the consistent and stateful migration of services from nodes about to fail due to

depleted batteries to nodes with more energy.
While the idea of iterative deployments is not new, consistency demands have not yet been considered comprehensively in this context. Different scenarios for service migration are also described by Sommer et al. [188]. A need for consistent updates of data paths in a transactional way is also stated. The demonstrator which is used by Sommer et al. is a wired network of various devices building a smart home [32], but neglecting the challenging constraints of wireless networks. We consider as an example the migration of a service S_3 from node 1 to node 2 as shown in Figure 1.2.

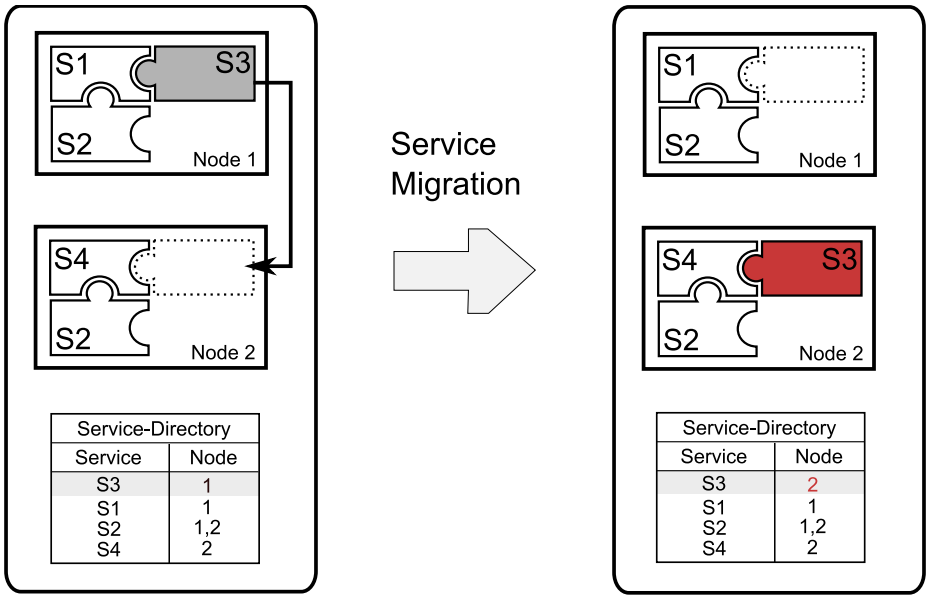


Figure 1.2: Transactional migration of a service

We assume the existence of a service directory which contains a mapping of registered services to the nodes which run the services. The transaction $T_{migrate\_Service}$ is comprised of the following three steps:

1. Replicate S_3 including its state from node 1 to 2 (i.e. copy it physically).
2. Delete S_3 on node 1.
3. Update the mapping on the service directory, so that nodes using this service can find its new location.

All three operations of this transaction have to be performed in an atomic manner, i.e. either all three steps are processed successfully or none of them. It is not always sufficient to demand eventual consistency in this case since the migrated service might be used for time critical tasks like alerts. Hence, a commit protocol for wireless sensor networks is needed to ensure the atomicity of transactional service migrations. In the next section we outline and motivate transactional issues of wireless sensor networks.

1.2 Transaction Processing

Transaction processing not only means guaranteeing atomicity. Instead, a database system is supposed to guarantee the following so called ACID-Properties [64]:

Atomicity. Either all operations of a transaction are performed or none of them. This can become a complex task in wireless sensor networks due to the sheer number of possibly failing nodes and message loss. We describe more use cases for atomic commitment in wireless sensor networks in Section 1.3.

Consistency. If the database has been consistent before the execution of the transaction, it also must be consistent after the execution. Real time wireless sensor networks may also have additional timely constraints like temporal consistency, which we do not particularly consider in this thesis.

Isolation. The executed transactions are isolated, i.e. they do not influence each other. For wireless sensor networks, this means that management queries (updates), for example, can be issued concurrently to continuous queries like described by Guergen et al. [69]. It also means that a (possibly expensive) wireless sensor network deployment can be queried simultaneously by several research teams. We elaborate on these demands in Section 1.4.

Durability. Changes made to the database are persistent after the transaction has committed. A logging system for wireless sensor networks, if at all feasible, has to be light weighted and distributed. Since the sensor nodes used in our experiments only shutdown but do not restart in case of failures, we cannot benefit from logs anyway and hence do not consider durability in this thesis.

1.3 Atomic Commitment

In this section we outline several use cases for atomic commit protocols in wireless sensor networks apart from service migration.

Rescue Scenarios. Obermeier et al. [149] state the need for atomic transactions in rescue scenarios. They consider firetrucks that get instructions, act in different rescue teams and have to perform operations according to plans. Atomicity and isolation have to be guaranteed because recipients cannot perform different actions at a time and also all units are needed to behave according to the same rescue plan.

Home Care. A system which is designed to assist nurses with care of patients at home could be used, for example, to control the amount of drugs the patient needs to take [149]. Transactions could be used if new drugs need to be ordered or the medication has to be changed. An example transaction could look like this:

```
Begin of transaction
Increase dose of drug A from 1 pill to 2 pills
Decrease dose of drug B from 5 pills to 0 pills
End of transaction
```

Sensor Actor Networks. Every time nodes not only measure their environment but also perform actions based on their measurements, a transaction concept might be needed, especially in a military environment but also in civil application domains. As an example for distributed agreement, consider four unmanned vehicles crossing an intersection [12]. Somehow the cars must agree on an order for crossing the intersection to prevent accidents. In general, the coordination of multiple actors can be required in the following situations [4]:

- One actor may not be sufficient to perform the requested operation.
- It might be required that an action is performed by exactly one actor.
- If multiple actors perform an operation, this has possibly to be done simultaneous so that synchronization is required.
- If a region is covered by multiple actors, they might all have to cover their own region, so that no overlaps occur. Hence, mutual exclusion must be guaranteed.
- Ordered execution of tasks might be relevant.

According to Akyildiz et al. [4], a task performed by actors is defined as an atomic unit of computation and control. Also, it might be required that a certain task is by no means executed by all possible actors at the same time, like in the case of disposers of a tranquilizing gas which could lead to catastrophic events.

Several other use cases for atomic commit in wireless sensor networks exist, for example, management queries in sensor database systems [69] and updates in caching systems [84]. We adapt approaches from the area of Mobile Ad hoc Networks (MANets) in this thesis and we describe our results.

1.4 Concurrency Control

Wireless sensor networks also need concurrency control. If a deployment is used by several scientists who pose different queries to the sensor network at the same time, it must be defined which of the possibly conflicting queries are executed. If, for instance, the last arriving query always stops all others, it is possible that none of the queries are successfully executed. On the other hand, if no concurrency is allowed, for example, if a query has to wait until a previous one has finished, the value of a deployment gets limited considering that some deployments are extremely expensive (for instance mars robots).

Hence, to achieve a well defined behavior, concurrency control for wireless sensor networks is needed. The well-known serializability concept from the database area [16] is able to fill this gap in order to enable more sophisticated sensor network applications. However, the usage of well known algorithms like two-phase locking, timestamp ordering or validation (formally introduced in Chapter 2) is not straight forward since wireless sensor networks pose new challenges in terms of message loss and severe resource constraints. We adapt and extend approaches from the area of Real Time Database Systems (RTDBS) in this thesis and we describe our results.

1.5 Adaptive Protocol Selection

We claim that consistent service migration is needed for the realization of service oriented wireless sensor networks. Transaction processing capabilities are a prerequisite for consistent service migration and also for the coordination demands of wireless sensor actor networks. However, there is great diversity of possible wireless sensor network deployments in terms of network size, connectivity, lifetime, heterogeneity and various other parameters described by Römer et al. as sensor network design space [172, 173].
Our initial studies with different commit protocols have shown that the efficiency of these protocols depends on the efficiency of the used routing protocol. In turn, the performance of a routing protocol depends heavily on parameters of the wireless sensor network design space, for example, the number of nodes and the presence or absence of mobility. So it is very unlikely that one single algorithm is superior to others in every possible deployment.
Consequently, our approach is to determine influencing parameters at runtime to be able to adaptively select the appropriate combination of commit protocol and routing protocol that satisfies the consistency needs of a particular deployment. We demonstrate the feasibility and the advantages of our approach in this thesis.

1.6 Contributions and Organization

The remainder of this work is structured as follows:

- In Chapter 2 we introduce the fundamentals of distributed transaction processing, wireless sensor networks and service oriented architectures.
- Chapter 3 contains our analysis and comparison of different atomic commit protocols and presents our own commit protocol Two Phase Commit with Caching. We show in simulations and also with experiments with real nodes that our protocol is more efficient for the use in wireless sensor networks than the compared protocols.
- Different traditional concurrency control protocols are compared with respect to their usability in wireless sensor networks in Chapter 4. We report our experiences with

adapting and implementing them and give evaluation results.

- The consistent replication and migration of services is described in depth in Chapter 5. We report simulation results with a moderate number of nodes and also verify our approach on real sensor nodes running a service oriented operating system.
- We describe our adaptive selection of commit protocol and routing protocol in Chapter 6. The considered transaction parameters and properties of sensor nodes and the network context are explained and various implemented protocol combinations are introduced. We show that our adaptive approach is more efficient for transaction processing in wireless sensor networks than using one single protocol.
- Chapter 7 concludes this thesis and gives an outlook on future work.

The major results of this thesis have also been published in the following reviewed articles: [165, 166, 167, 168, 169].

Chapter 2

Fundamentals

In this chapter we first give an overview of distributed transaction processing in Section 2.1. Then we describe the fundamentals of wireless sensor networks in Section 2.2 and finally we introduce the paradigm of service oriented architectures in Section 2.3.

2.1 Transaction Processing

A database (DB) is a collection of data, usually stored in a computer. The access to a database is provided by a mostly modular software system called Database Management System (DBMS). The unit of both, data and management system, build a database system (DBS) [162]. The access to the database takes place inside of transactions, which consist of one or more operations expressed in a query language. Typical database operations are insertions, deletions, updates and queries. Among other things, it is the responsibility of a DBS that successfully executed transactions leave the DBS in a consistent state and are also persistent. Furthermore, the DBS makes sure aborted transactions have no effects and undoes partial changes to guarantee the atomicity of a transaction.
These guarantees are part of the transaction concept introduced in the next subsection. We describe distributed atomic commitment in Subsection 2.1.2 and distributed concurrency control in Subsection 2.1.3. We also introduce replicated databases in Subsection 2.1.4.

2.1.1 Transaction Concept

A DBS is supposed to guarantee the following four ACID-Properties [64]:

Atomicity. Either all operations of a transaction are performed or none of them.

Consistency. If the database has been consistent before the execution of the transaction, it must also be consistent after the execution.

Isolation. The transactions executed in parallel are isolated, i.e. they do not influence each other.

Durability. Changes made to the database are persistent after the transaction has committed.

The properties atomicity and durability are implemented with the help of logging and recovery techniques. To grant the isolation property in a multi user environment, an algorithm for concurrency control must be implemented in the DBS to synchronize concurrent accesses [17]. The correctness criterion of a concurrency protocol is serializability, that is the outcome of the concurrently executed transactions is the same as if all transactions would have been executed in a serial order. While a wide range of concurrency protocols can be found in the literature, commercial database systems use mostly the Strong Strict Two Phase Locking (SS2PL) protocol also described by Bernstein et al. [17]. Data accessed by a transaction is locked in the DBS to prevent concurrent access by another transaction, the locks are released at the end of the transaction. We give more details on SS2PL in Section 2.1.3.
When databases are not central but distributed, the data is either partitioned between several database nodes in some way, or some portion of the data is replicated (see Subsection 2.1.4). In both cases, the maintenance of the ACID properties gets more complex.

2.1.2 Distributed Atomic Commitment

Atomic commit protocols are needed when a set of distributed operations has to be performed in an atomic manner. A common motivating example used in fundamental database courses is the withdrawal of money. A cash dispenser must open its drawer and a probably distant computer at a bank somewhere must debit the account of money withdrawn. The performance of only one of these operations leaves either the bank or its client unsatisfied. In distributed systems, a very related problem is distributed agreement, which is often introduced by the problem of the two generals. The generals have a common objective like a hill, which they will only conquer if they proceed simultaneously. If only one of them proceeds, he is going to fail. Since the two generals are some distance apart, their communication is through messengers, which are unfortunately unreliable.
The goal is to find some protocol which enables the generals to march together although some messengers get lost, or in database terms, to commit a distributed transaction. Unfortunately, no protocol of fixed length exists. The proof is as follows: Let us assume the existence of such protocols and let us consider the shortest of these called p. Now we assume that the last messenger in p gets lost. Then the last message is either useless or one of the generals does not get a crucial message. Since p is minimal, the message cannot be useless, so one of the generals does not get the message and therefore does not march. So it is proven by contradiction that no such protocol exists.
Fortunately, a solution is possible if the restriction of some finite fixed length protocol is relaxed. In the following, we introduce the standard solution for distributed databases.

Two Phase Commit (2PC)

The Two Phase Commit (2PC) protocol is presented by Gray in "Notes On Database Operating Systems" [63] and is also described by Mohan et al. [136]. Gray's original pseudo code is shown in Listing 2.1 respective Listing 2.2. A graph of the protocol is also shown in Figure 2.1.

```
1  COORDINATOR: PROCEDURE;
2           VOTE='COMMIT'; /* collect votes */
3           DO FOR EACH PARTICIPANT WHILE(VOTE='COMMIT') ;
4                   DO;
5                   SEND HIM REQUEST COMMIT;
6                   IF REPLY != 'AGREE' THEN VOTE = 'ABORT';
7                   END;
8           IF VOTE= 'COMMIT' THEN
9                   DO; /* if all agree then commit */
10                  WRITE_LOG(PHASEI2_COMMIT) FORCE;
11                  FOR EACH PARTICIPANT;
12                          DO UNTIL (+ACK) ;
13                          SEND HIM COMMIT;
14                          WAIT +ACKNOWLEDGE;
15                          IF TIME LIMIT THEN RETRANSMIT;
16                          END;
17                  END;
18          ELSE
19                  DO; /* if any abort, then abort */
20                  FOR EACH PARTICIPANT
21                          DO UNTIL (+ACK) ;
22                          SEND MESSAGE ABORT;
23                          WAIT +ACKNOWLEDGE;
24                          IF TIMELIMIT THEN RETRANSMIT;
25                          END;
26                  END;
27          WRITE_LOG(COORDINATOR_COMPLETE); /* common exits */
28          RETURN;
29          END COORDINATOR;
```

Listing 2.1: Two Phase Commit protocol run by the coordinator [63]

The protocol works as follows:

1. To initiate the voting process, the coordinator sends a *Prepare* (or *BeginVote*) message to all participants (see lines 2 to 5 in Listing 2.1).

2. Upon the reception of a *Prepare* message, a participant executes the local subtransaction. If this could be done successfully, it logs the results and replies with a *Ready*

(or *Prepared* or *VoteCommit*) message. If the subtransaction fails, the node replies with a *Failed* (or *VoteAbort*) message and releases its locked data (see lines 2 to 7 in Listing 2.2).

3. If the coordinator receives a *Prepared* message from every participant, it writes a commit data set to the log and sends a *Commit* message to all participants (see lines 8 to 17 in Listing 2.1).

   Note that we do not consider logging aspects in this thesis since the sensor nodes used in our experiments only shutdown but do not restart in case of failures, which makes recovery impossible.

   If one of the participants has replied with *Failed*, the coordinator sends an *Abort* message (see lines 18 to 26 in Listing 2.1).

4. Upon the reception of a *Commit* message, a participant writes the commit set to the log and releases its data locks (see lines 9 to 13 in Listing 2.2). Upon the reception of an *Abort* message, the transaction is undone. Finally, the participants send an *Acknowledge* message to the coordinator.

```
1  PARTICIPANT: PROCEDURE;
2           WAIT_FOR REQUEST COMMIT; /* phase 1 */
3           FORCE UNDO REDO LOG TO NONVOLATILE STORE;
4           IF SUCCESS THEN /* writes AGREE in log */
5                   REPLY 'AGREE';
6           ELSE
7                   REPLY 'ABORT';
8           WAIT FOR VERDICT;           /* phase 2 */
9           IF VERDICT = 'COMMIT' THEN
10                  DO;
11                  RELEASE RESOURCES & LOCKS;
12                  REPLY +ACKNOWLEDGE;
13                  END;
14          ELSE
15                  DO;
16                  UNDO PARTICIPANT;
17                  REPLY +ACKNOWLEDGE;
18                  END;
19          END PARTICIPANT;
```

Listing 2.2: Two Phase Commit protocol run by a participant [63]

The communication complexity of the protocol is $4 * (N - 1)$, while N is the number of nodes.

The protocol has been widely used in database management systems, but suffers from blocking, since nodes block their locked resources while they are waiting for a decision.

Other nodes willing to use these resources have to wait for the locks to be released. This is especially serious if nodes fail or even the coordinator fails permanently, since some participants can never resolve their transactions, causing their resources to be tied up forever. To prevent this case, non-blocking commit protocols have been investigated.

Three Phase Commit (3PC) Protocol

The Three Phase Commit (3PC) protocol has been proposed by Skeen at al. [186, 187]. It uses an additional *Pre-Commit* state to be able to recover from a coordinator failure, leading to an additional message round. The comparison of 2PC and 3PC is shown in Figure 2.1.

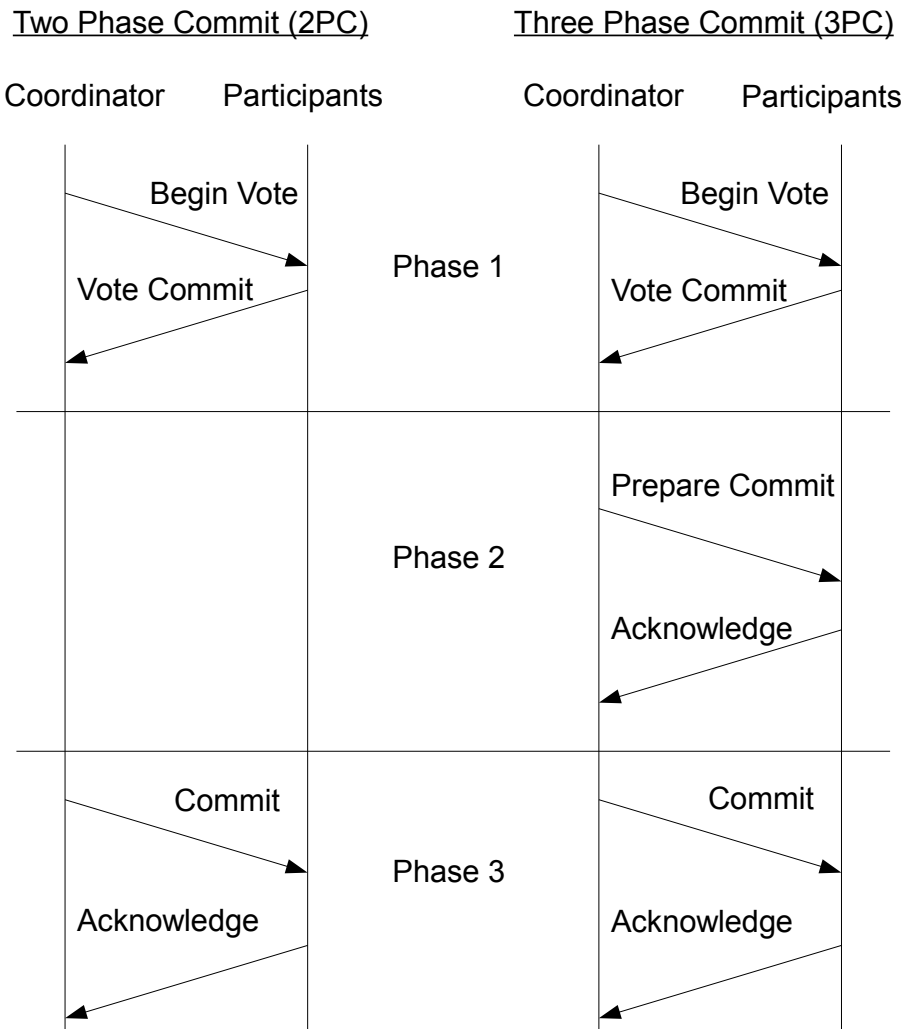


Figure 2.1: Comparison of 2PC and 3PC, failure-free case

The first two steps of 3PC are analog to 2PC: the coordinator sends a *BeginVote* message to the participants, the participants willing to commit reply with *VoteCommit*. Afterwards, instead of already sending the *Commit* message, the coordinator sends a *PrepareCommit*

message, which must be acknowledged by the participants. As soon as all participants have acknowledged the *PrepareCommit* message, the coordinator sends the *Commit* message.
If a coordinator fails in 3PC, another coordinator can take over and complete the commit protocol. Nevertheless, 3PC is still vulnerable to network partitioning. If participants get isolated, their resources may be blocked. However, the protocol guarantees the non-blocking execution of the commit process if no network partitioning occurs and the number of failing nodes is limited.

Paxos

An even more sophisticated approach is the usage of the Paxos Consensus algorithm [107, 108] for the commit decision [66]. Paxos differentiates between the three roles proposer, acceptor and learner. The three different roles do not necessarily need to be performed by distinct nodes. The algorithm works in two phases:

Phase 1. (a) A proposer chooses a proposal number n and sends a *Prepare* message to the acceptors.

(b) If the proposal number n is larger than any proposal received previously, then each acceptor sends a *Promise* message not to accept proposals less than n, and sends the value it last accepted for this instance to the proposer.

Phase 2. (a) If the proposer receives replies from a majority of acceptors, it chooses the highest value accepted and sends it to the acceptors with an *Accept* message. The proposer may choose any value if none has been accepted.

(b) Upon the reception of an *Accept* message, an acceptor accepts the proposal unless it has already responded to a *Prepare* message with a number greater than n. Finally, the learners can be informed by the acceptors.

The idea to use Paxos as a commit protocol was published in "Consensus on Transaction Commit" by Gray and Lamport [66]. The Paxos Commit protocol uses 2F+1 coordinators running Paxos Consensus and comes to a decision if at least F+1 coordinators are working for a sufficient long time. The algorithm is shown in Figure 2.2.
In contrast to 3PC, Paxos Consensus / Paxos Commit works correctly even if network partitioning occurs, as long as a majority of participants can be reached. Several other atomic commit protocols exist, for instance One Phase commit [65] and nested (linear) Two Phase Commit [63].

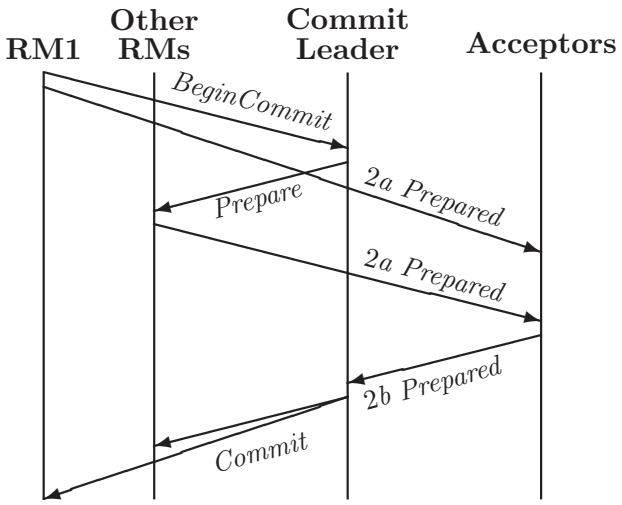


Figure 2.2: Failure-free case of the Paxos commit algorithm, from [66]

2.1.3 Distributed Concurrency Control

To make sure that the isolation property is also guaranteed for distributed systems, extended concurrency control protocols are necessary. A classification of the most common protocols for concurrency control is shown in Figure 2.3.
We explain the most important concurrency control protocols in the next subsections, beginning with locking.

Locking

Traditionally, data items accessed by a transaction are locked during the access [53]. Read and write locks exist. A transaction first acquires a read lock on a data item if it wants to read the item, respective a write lock if the item is to be updated. If another transaction wants to access an item which is already locked, the transaction has to wait or can be aborted. While read locks on data items can be held by several transactions concurrently, write locks are exclusive. When two or more transactions aiming at accessing the same data items acquire their needed locks in different orders, deadlocks can occur, i.e. each transaction is waiting for the other one to release its locks.
The standard Two Phase Locking (2PL) protocol was introduced by Eswaran et al. [53]. The idea is to first acquire locks needed for the execution of the transaction (phase 1, growing phase) and releasing them afterwards without acquiring new locks (phase 2, shrinking phase). It guarantees serializability. The commercially widely used Strong Strict Two Phase locking (SS2PL) described by Bernstein et al. [17] and also called Rigorousness by Breitbart et al. [29] goes a step further. Here, all locks are held until end of transaction.
When using locking in a distributed environment, the management of the locks can be done centralized or distributed. Although the centralized approach works analog to a centralized

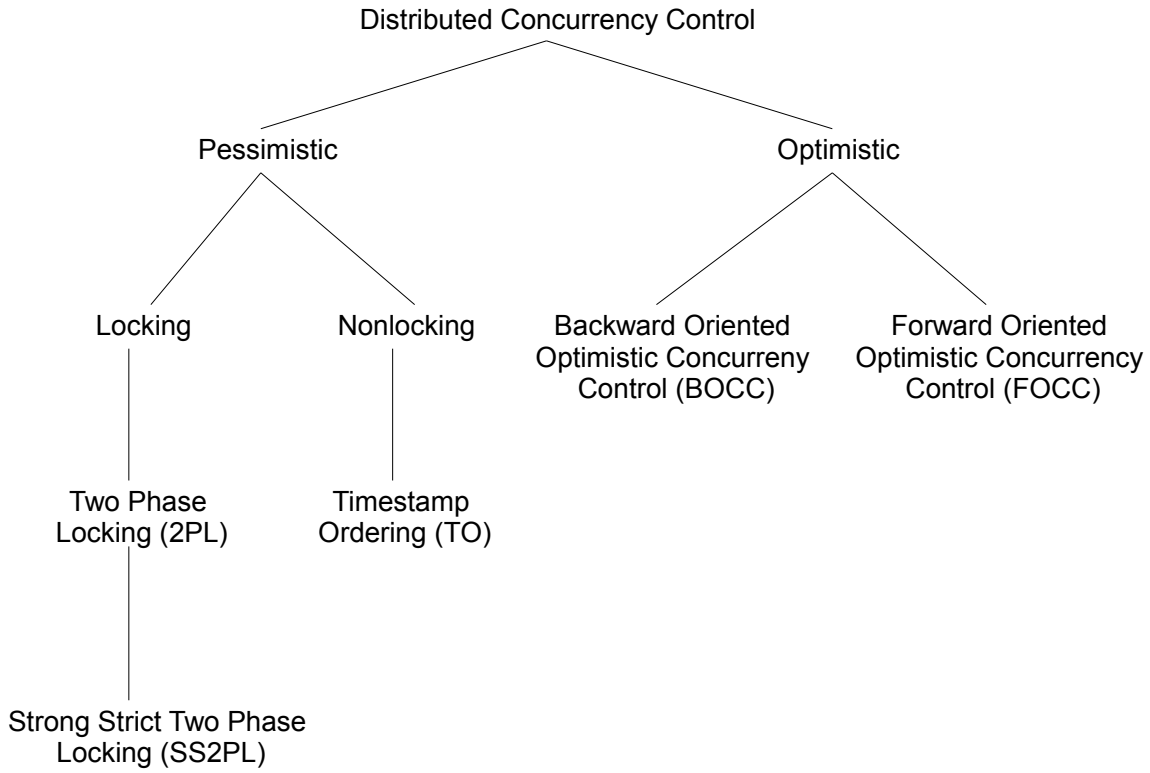


Figure 2.3: Classification of common concurrency control protocols in pessimistic and optimistic approaches, adapted from [162]

DBS and is straight forward to implement, its severe disadvantages are the communication overhead and the single point of failure. Therefore, we consider only distributed management of locks in the following. When SS2PL is implemented in a distributed environment, global deadlocks due to locking generate automatically voting-deadlocks in 2PC, and are thus resolved automatically by 2PC. This behavior has been generalized for other concurrency protocols by the principle of commitment ordering by Raz et al. [164].
In this thesis, all implemented concurrency control approaches are used in combination with 2PC. Hence, we do not give details about deadlock detection or prevention and related approaches like Wait / Die and Wound / Wait [174].

Timestamp Based Concurrency Control

Timestamp based concurrency control is also called Timestamp Ordering (TO) and has been described by Bernstein et al. [17]. Using this technique, serializability is guaranteed by timestamps assigned to data objects and transactions. The validation of timestamps is performed at the location of the manipulated data, leading to an inherently distributed,

deadlock free protocol [162].
A globally unique timestamp is assigned to every transaction at begin of transaction (BOT). The timestamp can, for example, be determined by using the node's id and its local time [106]. Without synchronization, no monotonic increase of the timestamps is guaranteed, which can lead to the unnecessary abort of transactions.
Using TO, the global execution order of transactions is predefined by their timestamps. Conflicting operations must occur in the order of the transaction timestamps.
To be able to validate the transaction timestamps, a read timestamp (RTS) and a write timestamp (WTS) is assigned to every data object. These timestamps are always updated if the data object is accessed by a transaction. A read transaction T on an object x is aborted, if $ts(T) < WTS(x)$ holds. A write transaction T on an object x is aborted, if $ts(T) < \max\{WTS(x), RTS(x)\}$ holds.
Although TO is deadlock free, blocking can occur. Assume T1 has successfully updated data object x, but has not committed yet. Assume also, that T2 wants to read x. If T2's read access is allowed and T1 is aborted later, then T2 must also be aborted. This can lead to cascades. So T2 must wait until T1 has committed, which is analogous to locking.

Optimistic Concurrency Control

Optimistic concurrency control (OCC) by validation is introduced by Kung et al. [103]. OCC protocols are based on the assumption that conflicts occur rarely, which makes locking an unnecessary overhead. We differentiate between the three phases read, validate and write.

- Read phase: In this phase, a transaction reads its needed data objects and performs its writes to an own private workspace.
- Validation phase: The validation phase is started at end of transaction (EOT). It is checked, if conflicts between concurrent transactions have occurred. If this is the case, the transaction is aborted. Neither blocking nor deadlocks can occur, but frequent aborts can lead to the starvation of a transaction.
- Write phase: The updates stored in the private workspace are written to the log and also to the actual database, becoming visible for other transactions.

To perform the validation, the objects to be accessed by transaction T_i are assigned to its write set $WS(T_i)$ respective read set $RS(T_i)$. According to [71], two classes of OCC protocols can be distinguished: Backward Oriented Optimistic Concurrency Control (BOCC) and Forward Oriented Optimistic Concurrency Control (FOCC).
BOCC validates transactions against already committed transactions (see Listing 2.3) while FOCC validates transactions against concurrently running transactions (see Listing 2.4). Both techniques guarantee serializability by making sure that a transaction has read all changes written by all transactions validated successfully before.

```
1 VALID: = TRUE;
2 FOR Ti = Tstart+1 TO Tfinish DO
3           IF RS(Tj) ∩ WS(Ti) ≠ ∅ THEN
4                   VALID : = FALSE;
5 IF VALID THEN COMMIT
6           ELSE ABORT;
```

Listing 2.3: Backward Oriented Optimistic Concurrency Control (BOCC) [71]

```
1 VALID: = TRUE;
2 FOR Ti = Tact1 TO Tactn DO
3           IF WS(Tj) ∩ RS(Ti) ≠ ∅ THEN
4                   VALID: = FALSE;
5 IF VALID THEN COMMIT
6           ELSE RESOLVE CONFLICT;
```

Listing 2.4: Forward Oriented Optimistic Concurrency Control (FOCC) [71]

The advantage of FOCC is that only real conflicts lead to aborts. Also, FOCC is more flexible. While BOCC always aborts the validating transactions in case of conflicts, FOCC also allows to abort one of the conflicting transactions, possibly by taking priorities into account.

When using distributed validation, every subtransaction is validated on the node it is executed on. Global transactions can be synchronized by means of the commit protocol as follows: The *Prepare* message is also used as a request for validation. After a successful validation, every subtransaction saves its changes in a local workspace and sends a *VoteCommit* message. If the validation fails, the subtransaction sends *VoteAbort* and deletes the workspace. If all local validations have been successful, the coordinator sends *Commit* to the participants. These write the changes stored in their workspaces to the actual data.

This procedure is only sufficient for guaranteeing local serializability, because the validation order of global transactions can be different on different nodes. The problems of distributed validation are described in detail by Schlageter [178]. One method for guaranteeing global serializability is to use timestamps and to make sure that the validation of all global transactions is performed in the same order on all participating nodes. When the local execution order on all nodes is identical, it is also the same as the global execution order.

Globally unique timestamps are assigned to every transaction at end of transaction (EOT), determining its position in the global execution order. Transactions arriving late (with a smaller timestamp than that of the transaction already validated) are aborted.

An additional problem arises due to the time lag of local validation phase and write phase. Since it is not guaranteed that transactions validated successfully locally are also committed (unsure updates), the outcome of transactions reading the unsure updates is undetermined. One possibility to prevent this is the locking of unsure updates.

Several optimizations of the fundamental algorithms exist, many in relation to real time databases where validation is more commonly used than locking. Haritsa et al. [74], for example, describe a real-time optimistic concurrency control algorithm which monitors transaction conflict states and gives precedence to urgent transactions, requiring that transactions are annotated with deadlines.

2.1.4 Replicated Databases

When the data in distributed databases is not partitioned (either vertically or horizontally) but duplicated data items exist, one speaks of replicated databases. Several approaches exist to guarantee several kinds of consistency for replicated databases.

Write All Read Any

One copy serializability (1-SR) means that the replicated database system behaves like a traditional database system consisting of one copy as far as the perception of the user is concerned [17].
The implementation of this behavior is denoted as Write-All-Read-Any or Read-One-Write-All (ROWA) strategy. It demands for a synchronous update of every replica before an update transaction can be committed. Since it is guaranteed that a replica is up to date, any replica can be chosen for a read operation. The advantage of this approach is that only one replica has to be available for a read request.
The disadvantage is that the availability of every replica is needed for a successful write operation. The overall update availability is even lower than without data replication because every replica needs to be online and reachable for a successful update transaction. It also leads to a significant amount of time to acquire all needed write locks and also a significant communication overhead is caused, since all nodes have to be participants of the commit protocol.
When using a two phase commit protocol, in the first phase, all updates have to be sent to all replicas, before the replicas can be updated and the locks can be released in the second phase.

Write All Available

To mitigate the problems of the ROWA technique, the Write-All-Available technique is described by Bernstein et al. [17]. Using this technique, only available replicas are updated and a log contains the updates missed by the unavailable replicas, which are used for reconciliation if the replica is online again. However, this approach is only feasible if no network partition occurs and if it can be guaranteed that no read access occurs before all missed updates are executed.
The overhead can be reduced by integrating optimistic concurrency control protocols into the commit protocol. The Principle of Commitment Ordering, which is a generalization

of SS2PL, is introduced by Raz [164]. Several other approaches exist for the consistent maintenance of replicated data, for instance weighted voting [61] and snapshot replication [1].

2.2 Wireless Sensor Networks

In this section we describe the fundamentals of wireless sensor networks. Wireless sensor networks are large networks of tiny, battery powered sensor nodes operated by a microcontroller, connected by a wireless interface in an ad hoc manner, mostly deployed to sense their environment and to transfer their observations to the user. They are closely related to Mobile Ad hoc Networks (MANets), which may consist of mobile phones, notebooks or palms.
The differences of wireless sensor networks and MANets are, among others, outlined by Karl et al. [95] and Römer [173]. Additional properties of wireless sensor networks include

- Greater diversity of application areas
- Larger networks
- Scarcer energy supply, mostly without opportunity to reload the battery
- Needed self configuration because of impossible human interaction
- Data-centric view of the network
- Much severer resource limitations in terms of memory capacities and processing power

First and foremost, saving energy by reducing transmission costs is extremely important, since sending a bit over the shared medium consumes as much energy as thousands of operations, which is known as the R^4 signal energy drop-off [62].
We survey common application scenarios in Subsection 2.2.1, sensor nodes in Subsection 2.2.2 and routing in wireless sensor networks in Subsection 2.2.3. We also introduce general properties of wireless sensor networks (Subsection 2.2.4) and give an overview of Sensor Network Database Systems (SDBS) in Subsection 2.2.5.

2.2.1 Application Scenarios

One of the first deployments was the Sound Surveillance System(SOSUS) [142]. Among other things, it was used to monitor submarines, whales and earthquakes at the Atlantic coast of the United States since 1950. While the sensors deployed in the ocean where connected by undersea communication cables to the base station, wireless connection via satellite was used for the connection with several base stations.
In the remainder of this subsection we outline some application scenarios for wireless sensor networks, namely environmental monitoring, rescue applications, the military, health care, vehicular networks and smart homes.

Environmental Monitoring

One of the most popular and most cited deployment of wireless sensor nodes is the deployment on Great Duck Island (GDI) [192]. Szewczyk et al. describe their experiments with 150 Mica2Dot motes which were used for the monitoring of distribution and abundance of sea birds. The architecture of the deployment is shown in Figure 2.4.

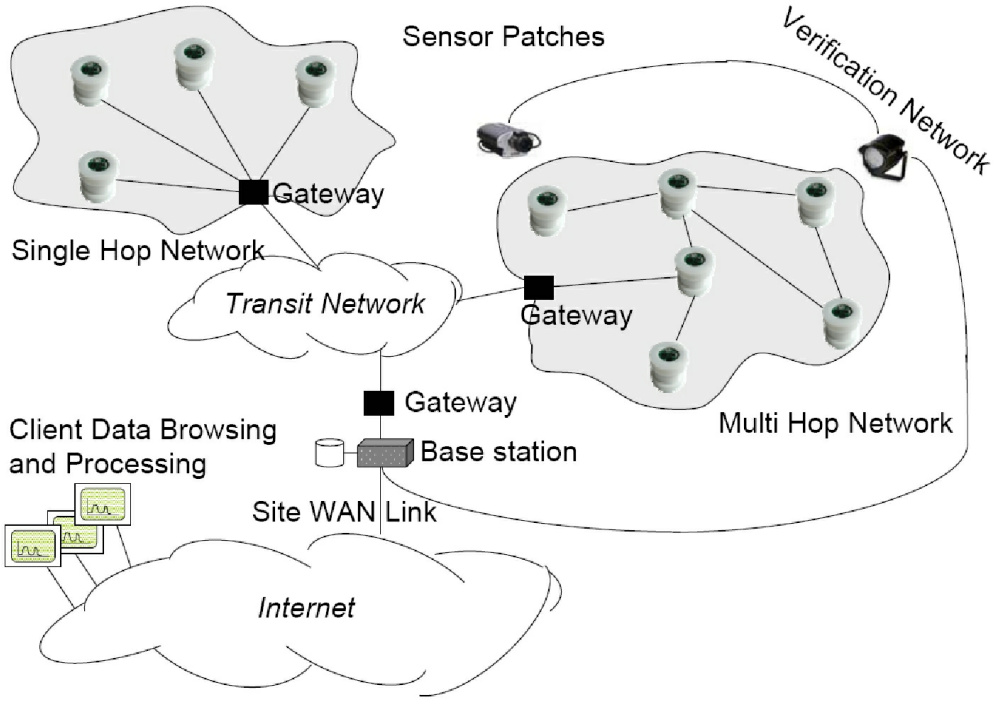


Figure 2.4: Architecture of the deployment on Great Duck Island [192]

In addition to the 150 sensor nodes, Figure 2.4 shows several gateways connected with a transit network and also a base station for the connection with the internet. Two types of nodes were deployed, burrow motes and weather motes. While the weather motes monitored temperature, humidity and barometric pressure, the burrow motes monitored temperature, humidity and occupancy of nesting burrows for 115 days, leading to over 650 000 data records.

The GDI deployment is a typical sense-and-send application, which is also called an external storage or dumb data collection sensor network [153], since the nodes only sample their sensors and transmit their readings to the base station where the readings are stored.

Another deployment is ZebraNet, which is a mobile deployment since the nodes were directly attached to the animals [93]. Wireless sensor networks are also used for the monitoring of vineyards for providing proactive agriculture [33], for glaciers [131] and might even be used on mars (by now in the Antarctica, but the authors Delin et al. are aiming at similar environmental conditions [43]).

A large worldwide deployment of wireless sensor networks is ARGO [7]. Over 3000 nodes

have been deployed in the sea for allowing its continuous monitoring of the temperature, salinity, and velocity of the upper ocean like shown in Figure 2.5.

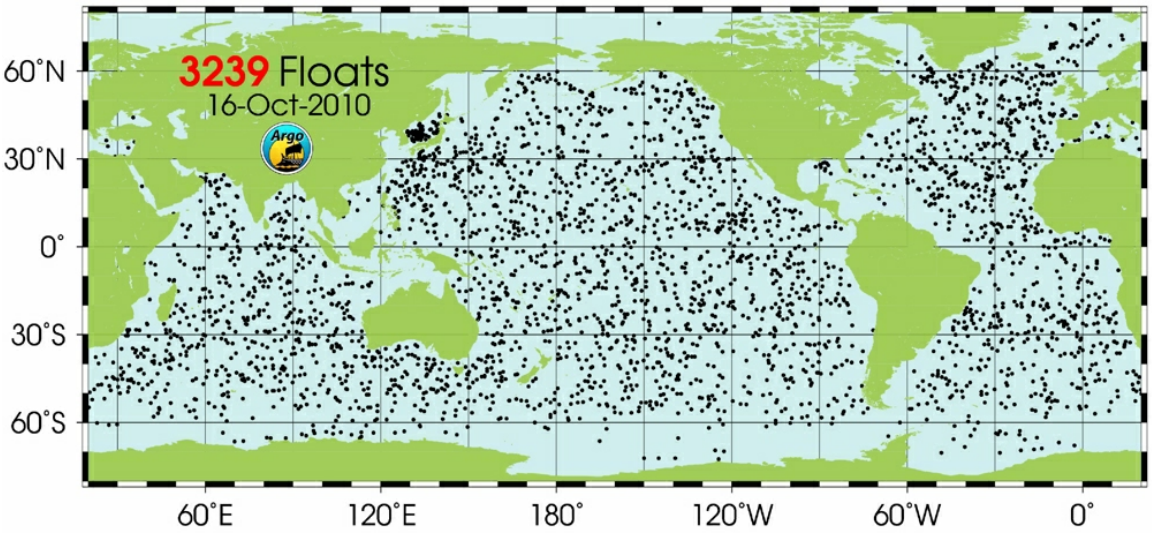


Figure 2.5: Worldwide deployment of ARGO [7]

Rescue Applications

Obermeier et al. describe a rescue application for firefighters [150]. Such an application can be used for the development of rescue plans, for the building of teams and for the location of fire trucks. Sha et al. propose a special architecture for the support of firefighters called FireNet [184].

Military

Since wireless sensor networks emerged from military research, there are a lot of application domains in that area, including intrusion detection and tracking [75], counter sniper systems [185], vehicle tracking with Unmanned Aerial Vehicles (UAVs) [15] and the detection of seismic and acoustic signatures [191].

Health Care

Bauer et al. describe a three layered architecture for a telemedicine environment [14]. Patients are monitored with several wireless sensors, these communicate with one middle tier per patient, which in turn transmits the patient data to a base station. Baldus et al. focus on the reliable setup of a body sensor network for medical patient monitoring [13].

Vehicular Networks

The European project CarTalk 2000 is an advanced driver assistance system based on vehicle-to-vehicle communication [54]. Vehicles send warning messages when they detect breakdowns, high traffic densities and such. These messages are forwarded by other cars. The project FleetNet also applies ad hoc principles to vehicular networks to achieve similar goals [57].

Smart Homes

While the automatic control of heat, doors and lights at home is already a reality [163], the existing systems are mostly wired and could benefit from wireless sensor node technology in terms of costs and flexibility.

2.2.2 Sensor Nodes

Wireless sensor networks consist of several tiny, autonomous devices which are battery powered and equipped with a microcontroller, a transceiver, memory, sensors and possibly actors. A schematic representation of a sensor node is shown in Figure 2.6. Sensor nodes sense a broad range of physical conditions, for example temperature, vibration, sound, humidity, pressure, motion and pollution.

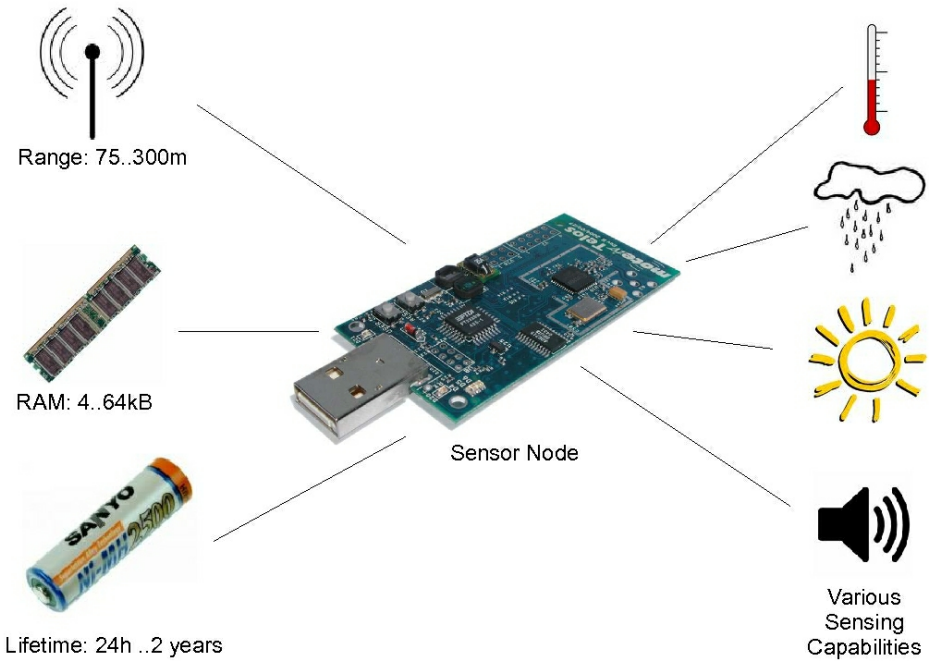


Figure 2.6: Schematic representation of a sensor node (Telos [160])

Wireless sensor networks are an interdisciplinary research area drawing from technologies out of the areas distributed systems, ubiquitous computing, embedded systems, peer to peer systems, signal processing and especially mobile ad hoc networks.
With the ongoing miniaturization of electronic components, the Smart Dust project was envisioned by Kahn et al. [94]. They describe a mote containing micro fabricated sensors, an optical receiver, passive and active optical transmitters, signal processing and control circuitry, and a power source, on a millimeter scale.
The sensor nodes used in today's deployments and research prototypes are not yet at millimeter scale. Since mostly AA batteries are used, the limiting factor is the size of the power source. A comparison of different sensor node platforms is shown in Table 2.1. It can be seen that the amount of RAM varies from 4 kilobytes to 64 kilobytes, while the amount of PROM varies from 48 kilobytes to 512 kilobytes.

Name	Manufacturer	RAM [kB]	PROM [kB]
MICA2dot [192]	Crossbow	4	128
MICAz Mote [40]	Crossbow	4	128
Tmote Sky (Telos B) [160]	Moteiv	10	48
Pacemate [122]	University of Luebeck	32	256
IMote [138]	Intel	64	512
BTnode [18]	Art of Technology	64+180	128

Table 2.1: A comparison of common sensor nodes, sorted by RAM size

Different operating systems for sensor nodes exist. While the Mica motes are running TinyOS [79] and are programmed using nesC [59], in this thesis we mostly used the Pacemate sensor node platform shown in Figure 2.7, which can be programmed using the iSense middleware [35].
The main reasons for the selection of the Pacemate platform are as follows:

- The Pacemates are equipped with displays and buttons, significantly simplifying the debugging of applications.
- The use of the iSense middleware allows to build the same code for the network simulator Shawn [55], allowing the verification of algorithms prior to their real deployment.
- High availability of nodes in our laboratory (over 100 nodes).

The usage of Shawn, compared to the widely used network simulator ns2 [90], has the additional advantage that simulations are magnitudes faster. This is because Shawn simulates only effects of phenomenons instead of the phenomenons itself. However, the used models, for example for communication, can be extended to be as realistic as necessary.

Figure 2.7: A Pacemate sensor node

2.2.3 Routing

Important for the intelligent and appropriate behavior of nodes is the idea of context. Dey [44] defines context as any information that can be used to characterize the situation of an entity.

Network Context

In RFC 2501 [38] various parameters making up the network context are named. We briefly survey these parameters since they have a great influence on the effectiveness and efficiency of a routing protocol.

- **Network Size.** The number of nodes varies greatly between different deployments. While only a small number of nodes might be sufficient for specialized purposes, like 6 deployed nodes for the monitoring of a glacier [131], a large number of nodes (over 3000 deployed) is needed for worldwide monitoring of the ocean [7].

  Also, the number of nodes simulated in published research papers varies greatly. Kurkowski et al. [104] report a range from 10 nodes to 30,000 nodes in the MobiHoc papers from 2000-2005.

- **Connectivity and covered area.** The area covered by real deployments varies between a few meters and worldwide. Kurkowski et al. report simulations using areas from 25 m x 25 m to 5000 m x 5000 m, with a transmission range varying from 3 m to 1061 m [104]. Network size, area and transmission range can be used to calculate the average neighborhood size as

$$Average\ neighborhood\ size = \frac{\pi \times r^2}{\{\frac{w \times h}{n}\}}$$

while w=width, h=height, r=range and n=# of nodes. Hellbrueck and Fischer [78] conclude that for a MANet with connecting probability of $\sim 95\%$ between two nodes, an average number of neighbors per node between 5 and 15 is necessary. Xue et al. [204] show that the needed density to prevent network partitioning grows with the number of deployed nodes.

- **Topology Change.** Many circumstances can lead to topology changes. Examples are failing nodes, additionally deployed nodes, sleeping nodes, mobile nodes, moving nodes, moving obstacles and weather conditions. A routing protocol should be able to take these changes into account in a timely and efficient manner.
- **Link capacity.** The maximum data rate of todays wireless sensor nodes is limited: the Mica mots reach about 40 kilobytes per second while the Telos B reaches 250 kilobytes per second [160]. The effective link speed however is much less, after accounting for losses due to multiple access, coding and framing.
- **Fraction of unidirectional links.** Kotz et al. [98] demonstrate that realistic deployments contain a significant amount of unidirectional links, that is node A can hear node B but not vice versa. A routing protocol needs to be robust to this phenomenon.
- **Traffic patterns.** While simple sense-and-send applications exhibit a uniform behavior, event detection networks may show bursty traffic patterns, which a routing protocol needs to be able to handle.
- **Sleeping nodes.** Since saving energy is very important in wireless sensor networks, a routing protocol needs to be involved in the coordination of the duty cycling of devices.

Routing protocols for wireless sensor networks should work under various contexts.

Classification of Routing Protocols

A classification of routing protocols can be done in, for instance, proactive, reactive and hybrid protocols [5]. Proactive protocols compute routes before they are really needed, reactive protocols compute routes on demand and hybrid protocols use a combination. The efficiency of a protocol depends on the dynamic of a network, While in more static networks proactive approaches are more efficient, reactive approaches have benefits in more dynamic networks.
The well known distance-vector routing protocols can be implemented in a proactive way like in Destination-Sequenced Distance-Vector Routing (DSDV) [155] or in a reactive

way like in Ad hoc On-Demand Distance Vector Routing (AODV) [156]. Distance-vector routing protocol make use of the Bellman-Ford algorithm for finding a shortest path in a weighted graph. In DSDV nodes maintain routing tables containing distances to destination nodes and also the neighbors used for reaching the destination. AODV uses the same principle but creates routing tables only if and when they are really needed.
Since there are routing protocols which do not fall in one of these categories, like simple flooding where no route is computed at all, we use one of the classifications of routing protocols described by Al-Karaki et al. [5] and differentiate between the following types:

- Flat networks, where each node performs the same role,
- Hierarchical structured networks, where nodes perform different roles (for instance cluster head vs. member), and
- Location based approaches, where the nodes positions are used to route data.

Flat Networks

In wireless sensor networks, communication is usually broadcast. All devices share a single common channel with Carrier Sense Multiple Access (CSMA), but no Collision Detection (CD) ability. Instead, Collision Avoidance (CA) is used. It works by listening for a random period and starting to send if the medium is available. But with absent synchronization and unavailable global network topology information, often simple flooding is used to disseminate information across the network.
This can lead to the broadcast storm problem, which contains the three subproblems redundancy, contention and collision according to [194]. First, since the transmission ranges of several nodes overlap, rebroadcasts can be redundant, since a node receives the same messages from several nodes. Second, the contention of the medium can be high since only one node can successfully broadcast at a time. Third, collisions are likely because unoptimized rebroadcasts occur usually at roughly the same time.
Data centric approaches have been developed to prevent these problems. Early data centric approaches include the Sensor Protocols for Information via Negotiation (SPIN) [101] and directed diffusion [89]. SPIN bases the decision of sending data or not on negotiations with other nodes considering metadata. Directed Diffusion exploits knowledge about the application, caches and processes data inside the network and uses evaluated paths to save energy. Braginsky et al. describe Rumor Routing [26], which delivers queries to events in the network. It allows tradeoffs between the setup overhead and the delivery reliability. Protocols using random walks are introduced by Servetto et al. [183]. Lipphardt [121] claims that the usage of path information is generally problematic in wireless sensor networks since links can get unusable anytime and introduces a gradient based routing protocol called Gradient based Routing for All Purpose (GRAPE). GRAPE overhears traffic

and tries to forward packets only in the direction of the sink. Despite promising simulation results, the performance of GRAPE on real sensor node deployments was poor [121].
Several other techniques to alleviate the broadcast storm problem are introduced by Tseng et al. [194], some of them lead to hierarchical networks or location based approaches:

- Probabilistic technique: a host rebroadcasts only with a certain probability (also called gossiping [70])
- Counter-based techniques: a node counts how often it already heard the message it wants to send. This technique has also been proven to be efficient in the Trickle algorithm by Levis et al. [113, 114]. If a certain threshold is met, the message is not sent. This technique is sometimes called thinning.
- Distance based technique: uses the distance to compute the expected additional coverage of an additional broadcast.
- Location based technique: works similar and more efficient than the distance based technique but needs the not always available location information. If the information is available, Tseng et al. [194] claim it to be more efficient than the counter based technique.
- Cluster based technique: a clustering approach is used where only the cluster heads rebroadcast, possibly together with one of the other techniques. This technique is shown to have problems with reachability and also with the hidden terminal problem [194].

Gossiping. We give more details on the probabilistic gossiping technique by Haas [70] since we incorporated it in some of our own protocols. The basic idea is, contrary to flooding, that messages are forwarded with a defined probability. A more sophisticated gossiping approach uses two thresholds to decide if a message is forwarded or not: the second threshold is considered if a node has less than a minimum number of neighbors, the first threshold is used if the node has equal or more neighbors than a predefined constant. Another parameter is the number of sure broadcasts in the beginning of a message dissemination. To prevent a dying out of the message, it can be defined that a message is really broadcasted for the first hops.
The scheme could look like the following:
$GOSSIP(p_1, k, p_2, n)$, where p_1 is the first forwarding probability, k is the number of sure broadcasts in the beginning of a message dissemination, p_2 is the second forwarding probability and n is the minimum number of neighbors a node must have to use the first broadcasting probability.

Hierarchical Networks (Clustering)

Heinzelmann et al. describe Low-Energy Adaptive Clustering Hierarchy (LEACH) [76]. Their approach is used for data collecting wireless sensor networks with the goal to save energy by electing nodes to cluster heads and cluster members, respectively. Several improvements have been introduced, like adaptive cluster head selection by Nam et al. [139]. Al-Karaki et al. apply a novel clustering scheme that results in fixed clusters and regular virtual topology, called Virtual Grid Architecture (VGA), in combination with a scheme for monitoring residual energy distributions at different parts of the network to be able to perform Energy-Centric Routing (ECR) [6].
The COUGAR approach is explained in more detail in Subsection 2.2.5 [205].

Location Based Approaches (Geographical Routing / Georouting)

Location based approaches are usually called geographical routing or georouting. The first algorithm which works in the presence of obstacles in the network is Geographical Routing with Face Routing by Bose et al. [24]. Several optimizations were introduced in the last years.
The experimental performance comparison of different protocols by Broch et al. [30] shows that the selection of the appropriate protocol depends strongly on the application scenario, for example the presence or absence of mobility.

2.2.4 General Properties

To describe some general properties of deployments of wireless sensor networks, we consider the design space of wireless sensor networks and name helpful principles for the design of protocols for wireless sensor networks.

Design Space

According to Römer et al. [172, 173], the design space for sensor networks comprises the following issues:

- Deployment (random vs. manual, one-time vs. iterative)
- Mobility (immobile vs. partly vs. all, occasional vs. continuous, active vs. passive)
- Size (brick vs. matchbox vs. grain vs. dust)
- Heterogeneity (homogeneous vs. heterogeneous)
- Communication modality (radio vs. light vs. inductive vs. capacitive vs. sound)
- Infrastructure (infrastructure vs. ad hoc)
- Network topology (single-hop vs. star vs. networked star vs. graph)
- Coverage (sparse vs. dense vs. redundant)
- Connectivity (connected vs. intermittent vs. sporadic)

- Network size (few vs. thousands)
- Lifetime (some hours vs. several years)
- Quality of service requirements: real time, robustness, etc.

Römer lists several application examples for a variety of instances described by this design space [173]. The goal is to show that there are not only applications for the so called traditional wireless sensor network which is one-time deployed in a relative randomly way, stays unpartitioned and uses multi hop communication, but also applications for new deployments not fitting into this scheme. Some of the parameters considered in this design space can be used for the adaptive selection of protocol combinations described in Chapter 6.

Design Principles

Also several design principles for the development of protocols for wireless sensor networks are inferred by Römer [173]:

- Adaptive tradeoffs (for instance response time vs. lifetime)
- Multi-Modality (apply two techniques for solving one problem)
- Data centricity (single nodes become replaceable, only their data is important)
- In network data processing (explained in detail in Subsection 2.2.5)
- Cross layer interaction (deviating from the OSI model)

We especially highlight the design principle cross layer interaction, because it is important for the integration of commit protocol and routing protocol in this thesis. While traditional networks are layered according to the Open System Interconnection (OSI) Reference Model [91], this is in many cases too costly for wireless sensor networks, since every layer (data link, network, transport, session, presentation) introduces additional overhead for allowing the communication of applications over the physical layer. Cross layer design can make protocols more efficient, since acknowledgement messages normally considered to be sent on the transport layer can be used to piggyback useful information for the application layer and thus save transmissions costs.

2.2.5 Sensor Database Systems

Traditionally, the programming of sensor nodes has been performed in a procedural way, i.e. the tasks of a node were specified on a low abstraction layer, which is error prone and tedious. To alleviate the user from the peculiarities of coding a distributed application, a number of data management systems have been presented for querying data in wireless sensor networks.

TinyDB and Cougar

The two prominent approaches TinyDB [127] and Cougar [205] were introduced to allow the user to pose queries in a convenient SQL-like manner. Descriptions of other SDBS approaches are described by Guergen et al. [69] and Bonnet et al. [22]. Figure 2.8 shows the architecture of a distributed in-network query processor connected to a base station.

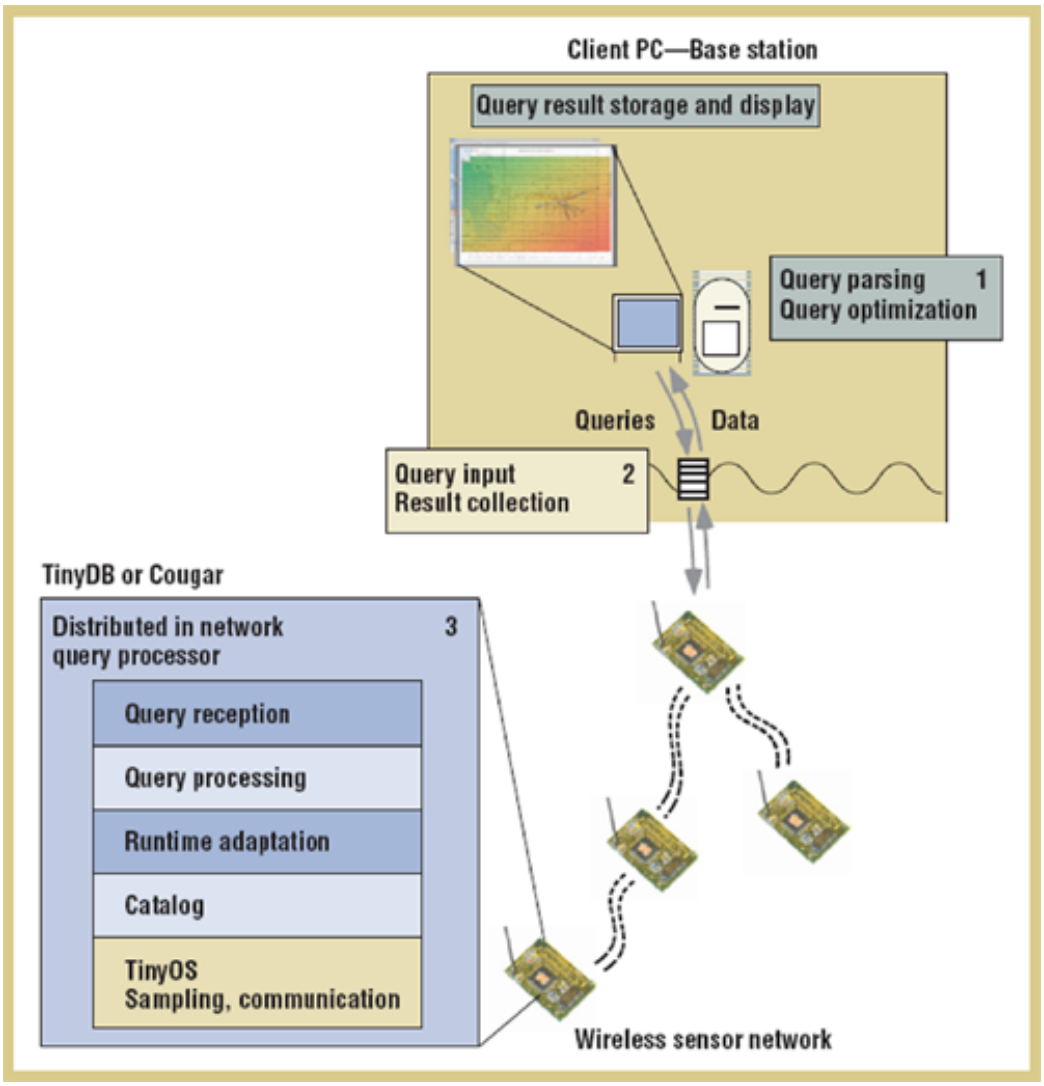


Figure 2.8: Scheme of the Sensor Database Systems (SDBS) TinyDB and Cougar [60]

The source code in Listing 2.5 shows a simple example of a TinySQL query which can be issued in TinyDB. The query specifies that each node should report its own id, light, and temperature readings contained in the virtual table *sensors* once per second for 10 seconds.

```
1 SELECT nodeid, light, temp
2 FROM sensors
3 SAMPLE PERIOD 1s FOR 10s
```

Listing 2.5: Example query of TinyDB

Much effort in the development of TinyDB was spent to expand the lifetime of the sensor network by reducing the energy consumption. This was done by aggregating results and synchronizing sleeping periods, among other techniques. To save as much energy as possible, in approaches like TinyDB data is only generated to answer particular queries, which is often called live data querying.
While there were many efforts in the area of live data querying, there were only a few publications in the area of querying historical sensor data. The term historical data often implies that every measured sensor value has to be saved for later analysis. There are different paradigms where this data should be saved that we explain in the following.

External Storage

The first sensor networks were mainly deployed as simple sense-and-send applications, a paradigm which is also called external storage or dumb data collection sensor network [153]. Each measured value is simply transmitted to the base station and stored there. Although this approach is easy to implement and to deploy, its main disadvantage is the potentially high energy consumption when data is sampled with a high frequency or resolution. Computing complex aggregate functions at the base station over a set of N sensor values will result into at least N messages. In reality, the amount of messages will be much higher since it depends on the number of hops between the sensor node and the gateway and also the used routing protocol.

Local Storage

The contrary approach is called local storage, which means that measured data is stored directly on the sensor nodes. Using this approach for answering continuous queries within sensor networks has the potential to save a high amount of energy. The idea is to push evaluation strategies deep into the sensor network, say to compute complex aggregate functions over a set of sensor values directly on the sensor nodes, so that only the final result will be delivered to the gateway. A typical scenario is a query that requests the mean of all sensor values over a determined period.
While local storage has often been considered impractical because of the limited sensor node resources in terms of computation power and storage capacity, Diao et al. [45] argue that due to technology trends in the area of flash memories, the local storage approach has the potential to save a high amount of energy compared with the external storage approach, because storing data locally on lately developed flash memory is much cheaper than transmitting it via radio.
Gaurav et al. [58] have shown for the MicaZ platform that storage on NAND flash memory is two orders of magnitude cheaper than communication and comparable to cost in computation. So especially for querying historical data, local storage in sensor networks becomes attractive. Evaluating aggregate functions like mean, for instance, over a set of N sensor

values directly on the corresponding sensor node will then result only in a constant number of messages.
Since measured data is stored locally, it becomes necessary to push the processing logic to the sensor nodes. There are several possibilities to do so. While Diao et al. [45] propose the design of a novel sensor database architecture emphasizing mainly local data archiving and query processing at the sensor nodes, this thesis emerged from a project which applies the paradigm service orientation to wireless sensor networks. We describe the details in the next section.

2.3 Service Oriented Architectures

In the mid nineties, service oriented architectures emerged in the area of business applications to allow companies a more flexible way of programming.
In this section we first outline the service oriented paradigm in Subsection 2.3.1. Then we explain the usage of service oriented architectures in wireless sensor networks (Subsection 2.3.2) and give an overview of the project AESOP'S TALE (Subsection 2.3.3). Finally, we introduce the basics of the service oriented operating system Surfer OS in Subsection 2.3.4.

2.3.1 The Service Oriented Paradigm

After the Remote Procedure Call (RPC) has been introduced 1988 by Sun Microsystems [189], in the mid nineties, the paradigm of service orientation emerged. The idea of service oriented architectures is to provide functionalities as services that can be published, found and used by other entities. The general service discovery triangle is shown in Figure 2.9.

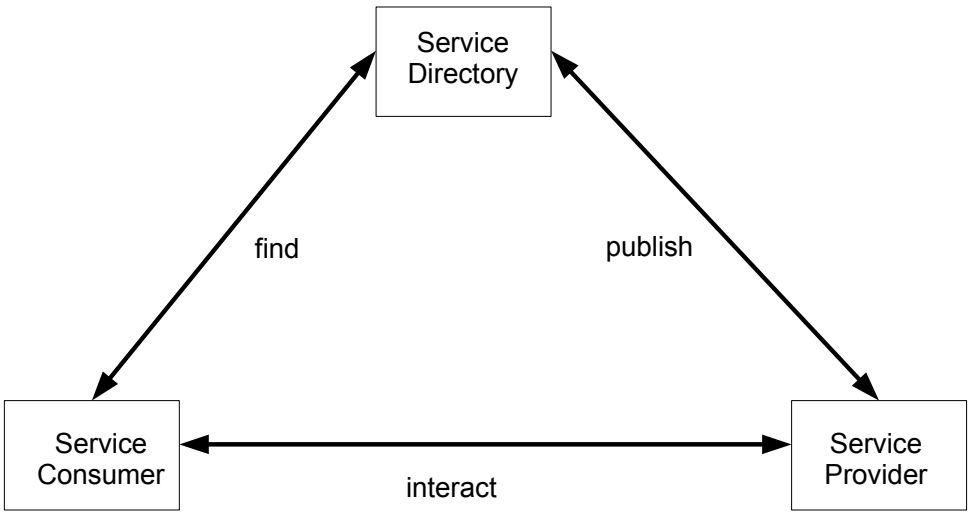


Figure 2.9: Different roles in a service oriented architecture

The three considered entities are a service provider, a service consumer and a service repository. The service provider publishes its service to the service directory, which contains a mapping of services to nodes. A service consumer can perform service discovery, which means it can ask the service directory where to find the service of interest. The service directory then transmits the location of the service (the location of the provider) to the consumer. Finally, the consumer can access the provided service. A reference model was published by the Organization for the Advancement of Structured Information Standards (OASIS) in 2006 [146].

2.3.2 Service Orientation in Wireless Sensor Networks

While service oriented architectures have been a well known and commonly used concept in business applications for years, they have also gained interest in the broader sensor network community [41, 42, 130]. Blumenthal et al. propose the service oriented coupling of device drivers to the operating system [20]. Delicato et al. describe the usage of a service oriented, reflective middleware for wireless sensor networks [41, 42]. The use of service oriented architectures in the project Cerberus is outlined by Prinsloo et al. [161]. Hof et al. introduce a secure overlay for service centric wireless sensor networks [87]. Marin-Perianu et al. describe the prototyping of service discovery and usage in wireless sensor networks [130]. Kushwaha et al. describe OASIS, a programming framework for service oriented sensor networks [105].

2.3.3 AESOP'S TALE

This thesis emerged in the context of the project "AESOP'S TALE: Applying and Extending the Service Oriented Paradigm to Sensor Network Application Engineering" [123]. The components of the system are shown in Figure 2.10. The transaction engine is in particular used to migrate services but can also be used by any other service requiring transactional guarantees.

The usage of a service oriented architecture in a wireless sensor network has two main advantages. The first advantage of a service oriented architecture is a convenient programming style, since simple services can be composed to more complex ones to fulfill individual tasks.

The second advantage is that sophisticated techniques like replication and migration of services can be used to make the sensor network self-organizing. These techniques require a transaction model to guarantee consistency.

While there is already a Web Services Atomic Transaction standard defined by the Organization for the Advancement of Structured Information Standards [147], this specification merely uses 2PC and does not take the unique properties of wireless sensor networks into account.

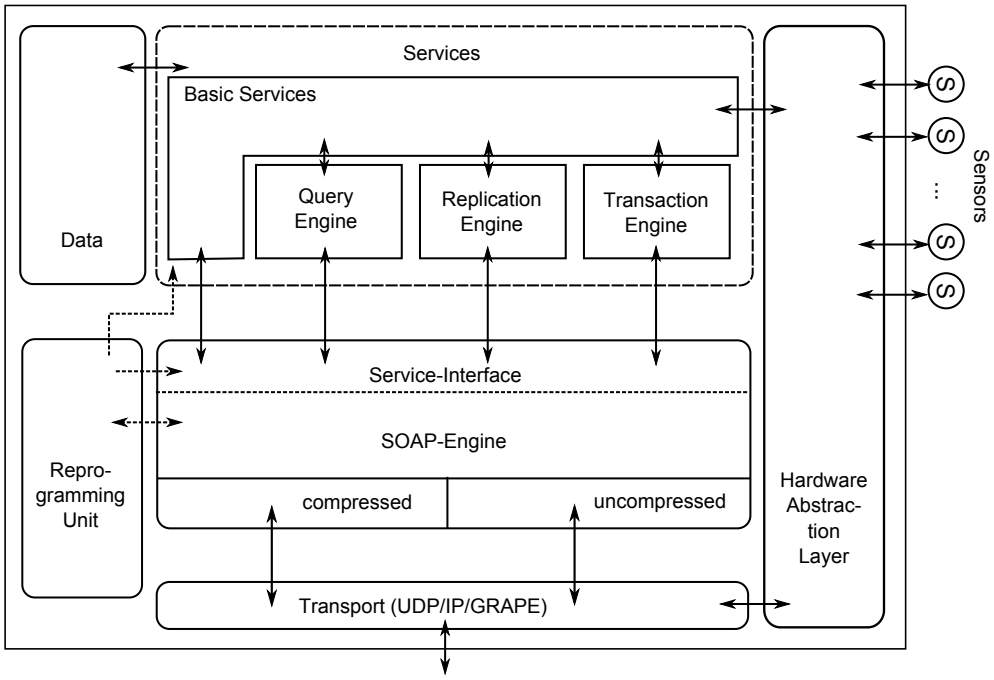


Figure 2.10: AESOP'S Tale: Components on one node

The transaction processing protocols developed in this thesis have also been partly implemented for the service oriented operating system Surfer OS which is described in more detail in the next subsection.

2.3.4 Surfer OS

The operating system Surfer OS realizes the paradigm of service orientation for embedded systems [124]. Surfer OS is reduced to a minimum of code and functionality and provides the following functionalities:

- Hardware Abstraction Layer (HAL)
- Task management
- Memory management
- Dynamic distribution, binding and invocation of services

Since all components of Surfer OS are considered as services, even the radio protocol stack including the routing protocol can be exchanged at runtime, providing a maximum of functionality. Hence, it can be determined at run time which nodes provide which services. Surfer OS provides also a stateful service migration, that means a service can keep the values of its local variables while migrating to another node. A demonstration of how

services can be migrated to sensor nodes at runtime can be seen in Figure 2.11. The user can load services from a repository, transfer loaded services to nodes or delete services on nodes. Surfer OS has also been ported to the well known Telos B sensor node platform [160].

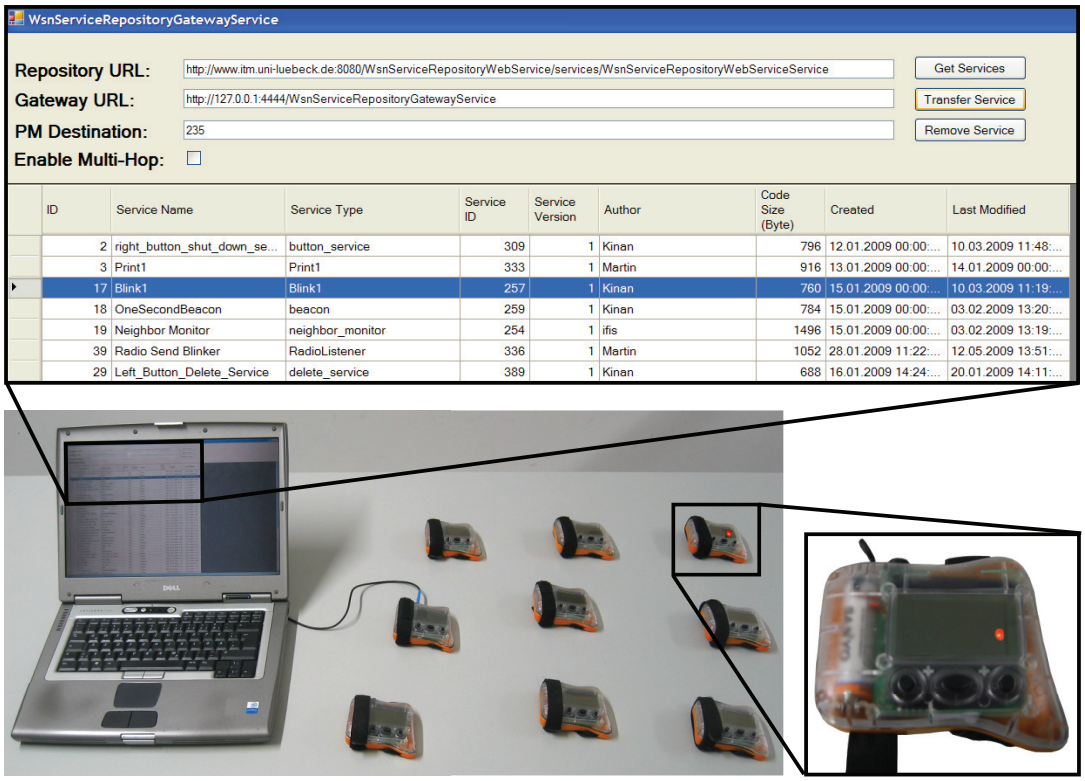


Figure 2.11: Demonstration of Surfer OS at SenSys 2009 [124]

Chapter 3

Atomic Commitment

The previously introduced database systems TinyDB and Cougar for wireless sensor networks do not support transaction processing. We are also not aware of any other research project aiming at transaction processing in wireless sensor networks. Only Guergen et al. [69] state a need for transaction processing concepts in wireless sensor networks and discusses the ACID properties with respect to wireless sensor networks. As already outlined in the introduction, there are many use cases for atomic commit protocols in wireless sensor networks and especially for use cases in the newer area of Wireless Sensor and Actor Networks (WSANs).

Organization

- In this chapter, we first present initial studies of the applicability of the Two Phase Commit (2PC) protocol in wireless sensor networks in Section 3.1.
- Then we review related work in the areas of fixed infrastructure wireless networks and ad hoc wireless networks, particularly analyzing the protocols Partition-Tolerant Atomic Commit (ParTAC) and Cross Layer Commit Protocol (CLCP) in Section 3.2.
- We present our own atomic commit protocol for wireless sensor networks called *Two Phase Commit with Caching (2PCwC)* in Section 3.3.
- Section 3.4 contains a brief description of our implementation and Section 3.5 presents the evaluation of the compared protocols.

3.1 Two Phase Commit in Wireless Sensor Networks

The Two Phase Commit (2PC) protocol is commonly used to achieve agreement in commercial distributed databases. So it is an obvious approach to use it also in wireless sensor networks. We briefly outline the changes needed to adapt the protocol for a wireless sensor

network. Special attention is paid to the required memory needed to store status information of running and finished transactions since memory is a scarce resource on wireless sensor nodes.
The actors in the 2PC are a coordinator c and a set of P participants $p_1, .., p_i$ with $i < N$, while N is the number of sensor nodes. In the context of our currently implemented service oriented sensor network, we allow that every node can play the role of the coordinator c and we also allow concurrent transactions. Running concurrent transactions in general requires measures to guarantee the prevention of read/write oder write/write conflicts. However, we do not consider the isolation aspect here but instead focus on one particular transaction for the description of the 2PC protocol. Concurrency control is discussed in Chapter 4.

3.1.1 Implementation

Our implementation of the 2PC protocol for the sensor node platform Pacemate is based on the original protocol described in Section 2.1.2. However, the protocol has been adapted to the wireless environment, there is, for example, only one *Prepare* message sent from the coordinator to all participants. This is sufficient since this message is flooded and contains the ids of all participants. We distinguish between the six message types:

BeginVote contains the transaction_id, coordinator_id and all participant_ids,

VoteCommit contains transaction_id, coordinator_id and participant_id,

VoteAbort contains transaction_id, coordinator_id and participant_id,

Commit contains transaction_id and coordinator_id,

Abort contains transaction_id and coordinator_id and

HelpMe contains transaction_id and coordinator_id.

The pseudo code of the implemented algorithm is shown in Listing 3.1. If the coordinator c wants to initiate a transaction like the migration of a service, c starts a preparation phase by sending *BeginVote* to the predefined participants of the transaction (lines 1 to 3). The coordinator registers the transaction as running and also registers a timeout to check for the outcome of the votes.
If a node p_i receives a *BeginVote* message which includes the node as participant (see lines 5 to 13), p_i decides whether it wants to vote for abort (1) or for commit (2) for this transaction:

1. If p_i decides for abort, which could be the case if, for instance, a node has not enough free memory to accept a replica of a service, it sends the corresponding *VoteAbort* message and aborts.

2. If p_i decides for commit, it registers the transaction as pending, sends a *VoteCommit* message and also registers a timeout to check for incoming decisions.

If the coordinator c receives a *VoteCommit* which is addressed to it (see lines 15 to 19), c checks if all participants have voted for commit yet. If so, c sends a *Commit* message to all participants and removes the transaction from its record of running transactions. Analogously, on receiving any *VoteAbort* message, it sends *Abort* and removes the transaction. If a participant receives a *Commit* respective *Abort* message, it decides accordingly, if it has registered the given transaction as pending (see lines 26 to 28 respective lines 30 to 32).
On receiving a *HelpMe* message (see lines 34 to 39), a node checks whether it has some information about the state of the transaction. If it knows that the transaction has been committed, it can send a *Commit* message. If it has heard a *VoteAbort* or *Abort* message concerning this transaction, it can send an *Abort* message. Otherwise it can just forward the *HelpMe* message.
On the firing of a timeout the coordinator checks if there are running transactions for which it has not received all required *VoteCommit* messages to send *Commit*. These transactions are aborted. Analogously, a participant checks if there are pending transactions for which it has not received a *Commit* or *Abort* message. If this is the case the node sends a *HelpMe* message.

```
1 ON SEND_BEGIN_VOTE:
2   register_running_transaction;
3   register_check_votes_timeout;

5 ON RECEIVE_BEGIN_VOTE:
6   if (i_am_receiver){
7     if (decide == abort) {
8       send_vote_abort;
9       abort;}
10    else {
11      register_pending_transaction;
12      send_vote_commit;
13      register_check_decisions_timeout; } }

15 ON RECEIVE_VOTE_COMMIT:
16  if (i_am_receiver){
17    if (count_commit){
18      remove_running_transaction;
19      send_commit; } }

21 ON RECEIVE_VOTE_ABORT:
22  if (i_am_receiver){
23    remove_running_transaction;
```

```
24      send_abort; }

26 ON RECEIVE_COMMIT:
27    if (remove_pending_transaction)
28    commit;

30 ON RECEIVE_ABORT:
31    if (remove_pending_transaction)
32    abort;

34 ON RECEIVE_HELP_ME:
35    if (received_commit) {
36     send_commit; }
37    if (received_vote_abort || abort){
38      send_abort; }
39    forward_help_me;

41 ON CHECK_VOTES_TIMEOUT:
42    if (timed_out_transactions) send_abort;

44 ON CHECK_DECISIONS_TIMEOUT:
45    if (timed_out_transactions) send_help_me;
```

Listing 3.1: Pseudo code of the Two Phase Commit protocol implemented for Pacemate sensor nodes

3.1.2 Correctness

The 2PC guarantees correctness, that means a transaction which is committed on one participant or coordinator can never be aborted on another participant and vice versa. However, transactions can have an undecided state, if a *Commit* or *Abort* message sent by the coordinator does not reach its destination. In this case, an uninformed participant sends a *HelpMe* message after a timeout.

3.1.3 Blocking

The problematic case is the blocking of participants if a coordinator of a running transaction fails before it has received any *Abort* message or all *Commit* messages. In this case, the transaction is blocked on all participants until the coordinator recovers. In this thesis we do not try to solve this case which can, for example, be done by using Three Phase Commit (3PC) protocols [187] or Paxos [107, 108], which increase the number of transmitted messages. When a participant does not receive the decision sent by the coordinator due to repeated message loss, the resources on this node remain also blocked.

3.1.4 Memory Consumption

To be able to manage a set of running and finished transactions, these have to be stored in the memory of the nodes. Although the needed memory is minimal, since only transaction_id, coordinator_id, participant_ids, a timestamp and the decision itself have to be stored, we limited the list of records of finished transactions to the number of nodes in the network to be scalable.
Since sensor nodes are equipped with only small RAM (about 32 kilobytes to 128 kilobytes), a node cannot keep track of all finished transactions and messages it overhears. Older entries are overwritten. The draw-back is that *HelpMe* messages concerning older transactions might remain unanswered. Apart from a ring buffer to manage message forwarding and the records of running and finished transactions, no significant amount of memory for status information is needed.

3.1.5 Message Complexity

In a wired network the corresponding complexity of one transaction is $4(P-1)$ while P is the number of participants. Since we do not assume a certain topology, each message has to be flooded, so in the worst case, for every *BeginVote*, $N-1$ messages have to be send to reach any of the P participants while N is the number of nodes. So in a wireless network without a certain presumed topology, the corresponding complexity can be estimated as follows:
Every node has to send the *BeginVote* message (N), the decision message (N), the vote of each participant (PN) and the acknowledge of each participant (PN). We implemented the protocol with *HelpMe* messages instead of acknowledgments to save transmission costs, but this is only efficient as long as a majority of decisions messages is received successfully. Hence, if we neglect this difference, the corresponding complexity of 2PC is because of message forwarding in the worst case:

$$N + PN + N + PN = (2 + 2P)N$$

One way to decrease the corresponding complexity is the generation of a tree, which reduces the number of messages to flood an information from N to $log(N)$. The draw-back is the amount of messages caused by topology maintenance to ensure a stable tree in case of node and link failures. It could also be considered if and how transactional guarantees could be integrated into routing protocols. We describe our deployment and our experiments in this section.

3.1.6 Deployment

We implemented 2PC on the Pacemate [122] sensor node platform shown in Figure 2.7. A pacemate has a Phillips LPC 2136 processor with 32 kilobytes RAM, 256 kilobytes

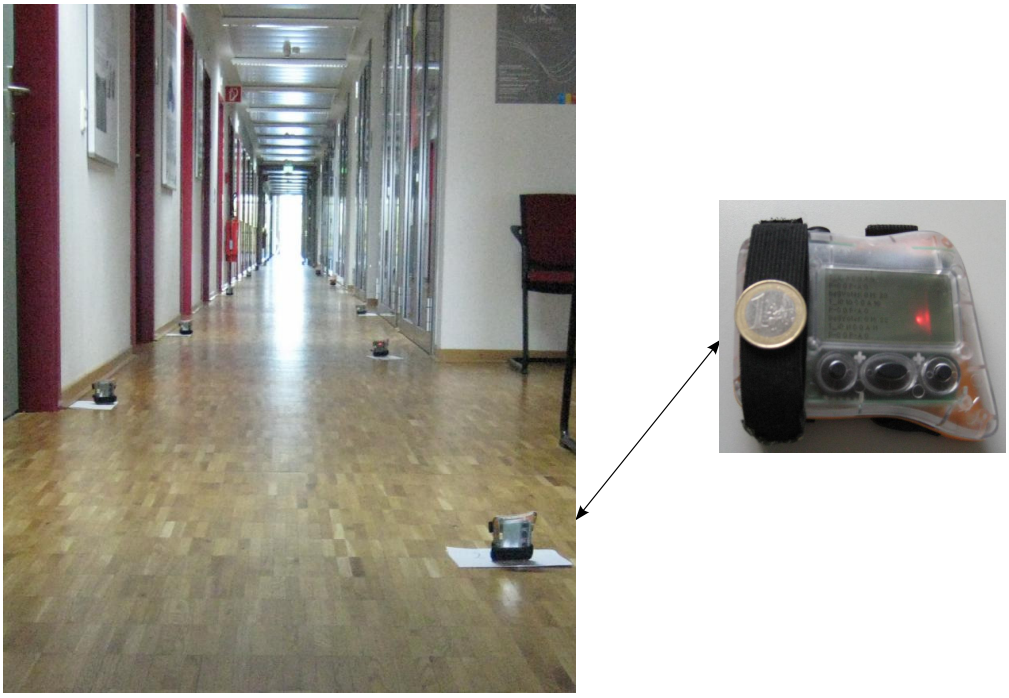


Figure 3.1: The deployment of 20 Pacemates on our corridor

Programmable Read Only Memory (PROM) and a 868 MHz radio transceiver. It is also equipped with a 128 x 64 dots display and three buttons allowing human interaction during runtime. We placed 20 Pacemates on the corridor of our institute as shown in Figure 3.1. Out of our 20 nodes we chose seven coordinators which initiated a transaction by sending a *BeginVote* message to two participants every two seconds for 20 times, so 140 transactions were started during our experiments in total. All messages were flooded. To be able to verify the correctness of the results, the selection of the coordinators and the participants was constant. A node was either coordinator, participant of one or more transactions, or both. The experiments were started by pressing a button on one of the Pacemates. The other Pacemates were activated by the first message received. During the 40 seconds of our experiment, where at any time about seven transactions were running concurrently, we measured the following parameters:

- number of started transactions (*BeginVote* messages sent),
- number of *BeginVote* messages received,
- number of transactions committed,
- number of transactions aborted,
- number of messages transmitted (including message forwarding)

We repeated the experiment several times with different intervals (1 second, 2 seconds, 5 seconds) between the starts of two consecutive transactions and obtained similar results.

The decisions for sending a *VoteCommit* or a *VoteAbort* message were done by a pseudo-random-function, which uses the system id of the particular node as seed, so the decisions were the same in each run of the experiment. The probability to commit was set to 90%. Aborts are more likely than in traditional DBMS, where 3-5% is considered as an average abort rate (see [68] and [64]). We did not use message retransmission.

3.1.7 Results

In this subsection we discuss the results of a typical run of the experiment with a transaction sending interval of 2 seconds for the 2PC protocol.
Since the qualitative results of the experiments were quite similar and differed only slightly in the number of received *BeginVote* messages (167 to 198 out of 280 possible *BeginVote* messages were received), we chose a particular run to discuss the results of our experiments. Of course, for 140 transactions only 140 *BeginVote* messages were sent by the coordinating nodes, but since we had two participants in each transaction, we had also a maximum of two reception events per *BeginVote* message sent, one at each participant.
In this particular run 191 out of 280 possible *BeginVote* messages were received, which equals about 68%. Madden [126] has shown for the ChipCon radio used on Mica2 motes a message loss of 20% to 30% is reported for the distance of 10 meters, which is quite similar to our results.
Detailed analysis have shown that these 191 messages belonged to 96 transactions, while 95 transactions were complete, that means both participants received a message, while in the 96th transaction, only one participant received a *BeginVote* message. That means that 45 transactions were aborted due to lost *BeginVote* messages.
As shown in Figure 3.2, out of the successfully started 95 transactions, 90 reached agreement on the coordinator and all participants (43 committed and 47 aborted), 5 transactions had the state undecided at the end of the 40 seconds on the side of the participants. That means they were waiting for a commit (4 transactions) or an abort (1 transaction).
The average number of transmitted messages per node was 380. This is a significant lower number of messages than estimated for the worst case after the estimation given.
Calculating the corresponding complexity after $(2 + 2P)N$ with $N = 20$ nodes and $P = 2$ participants, we get $6 * 20 = 120$ messages per transaction and

$$\frac{120 * 140\ transactions}{20\ nodes} = 840\ messages\ per\ node$$

The difference is caused by message loss, in particular by loss of 45 *BeginVote* messages.
To sum up, the results of the experiments show that the applicability of 2PC to enable transaction processing in wireless sensor networks is limited, since although agreement was achieved for 135 out of 140 transactions in the first attempt, the commit rate was low ($\sim$ 30%). Alone 45 transactions were aborted because of lost *BeginVote* messages, leaving much room for improvement of the protocol for wireless sensor networks.

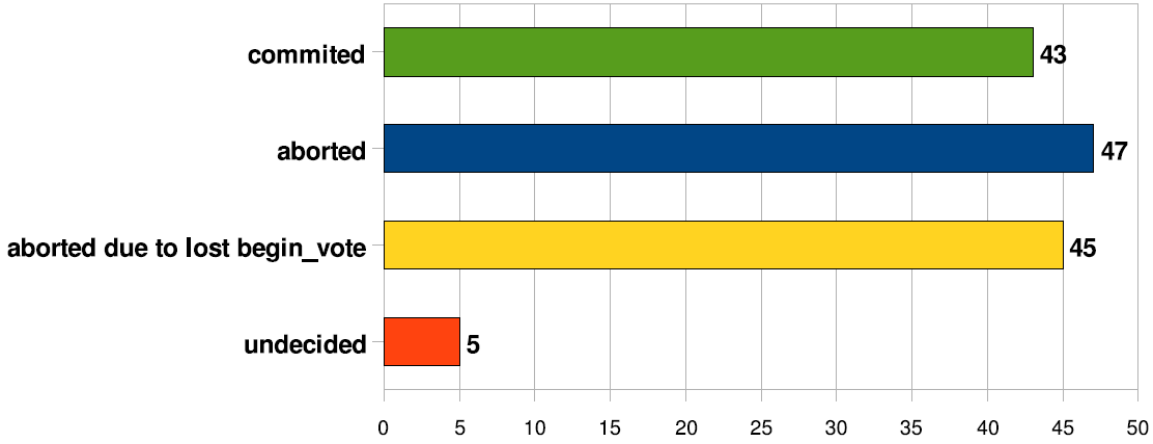


Figure 3.2: Outcome of 140 started transactions with the 2PC protocol

In the next section we discuss related work about atomic commitment in the area of wireless networks with fixed and also with ad hoc infrastructure.

3.2 Related Work

To the best of our knowledge, the only work dealing with transactions in sensor networks apart from our own work is [69], which focuses mainly on the isolation property of transactions and does not propose a specific commit protocol. Instead, an algorithm for the concurrent processing of continuous queries and one time queries is proposed. Also, the authors focus on wired networks and do not pay special attention to the properties of wireless networks. Consequently, we focus on related work in fixed infrastructure networks and also Mobile Ad hoc Networks (MANets) in this section.

3.2.1 Relaxing ACID Properties

Since wireless networks and mobile nodes pose challenges for traditional transaction processing protocols, several researchers considered the relaxation of the ACID properties. One possibility of relaxation is the extension of the traditional flat transaction approach by nested transactions [137]. A nested transaction is a transaction that contains subtransactions, which, in turn, may also contain subtransactions. Subtransactions can be classified according to the following properties [195]:

Open vs. Closed The results of an open subtransaction are visible to other transactions while the results of closed subtransactions are only visible to the parent transaction.

Vital vs. Non-Vital If a non-vital subtransaction is aborted, the parent transaction may still be allowed to commit.

Dependent vs. Independent A depended subtransaction must be aborted if the parent transaction is aborted.

Compensatable vs. Non-Compensatable For a compensatable transaction [2], always another transaction called compensating transaction exists which semantically undoes its changes. This is not the case for every transaction in general, for example the withdrawal of money from a cash dispenser cannot be undone easily.

Substitutable vs. Non-substitutable A transaction is substitutable if there exists another transaction which can be executed on its behalf.

We assume closed, vital, dependent, non-compensatable and non-substitutable subtransaction in our work to make it widely applicable.

3.2.2 Fixed Infrastructure Networks

Early research projects in the area of mobile databases emerged about 10 years ago. The Moflex Transaction Model for Mobile Heterogeneous Multidatabase Systems is introduced by Ku et al. [99]. The characteristics of this model have been adopted by several researchers. The considered mobile computing environment is shown in Figure 3.3. It consists of Mobile Host (MHs), which are in the focus of this model, that are connected to traditional Fixed Hosts (FHs) by the means of Mobile Support Stations (MSS), also called base stations (BSs).
An overview of mobile transaction processing is given by Serrano-Alvarado et al. [181] and Tuerker and Zini [195]. A plethora of contributions exist, we name a few in the following.

Clustering outlined by Pitoura and Bhargava [158, 159] is sometimes also called weak-strict transactions. A fully distributed, clustered environment is assumed, where a disconnected MH becomes a cluster by itself. Every object exists in two versions: a strict version which is globally consistent, and a weak version, which can tolerate some degree of global inconsistency but is at least locally consistent. Transactions are also divided in strict and weak, while strict transactions access strict versions and weak transactions access weak versions of objects. If disconnected MHs connect again to the database, the weak versions are synchronized. Distributed transactions are only processed as strict transactions, and MHs can only participate while connected.

Two-tier Replication proposed by Gray et al. [67] is a lazy replication mechanism where participants can get occasionally disconnected. For each data object, one master version and several replicas exist. Transactions are divided into base transactions and tentative transactions, while base transactions access master objects and tentative transactions access the replicas, even when the MHs are disconnected. If an MH

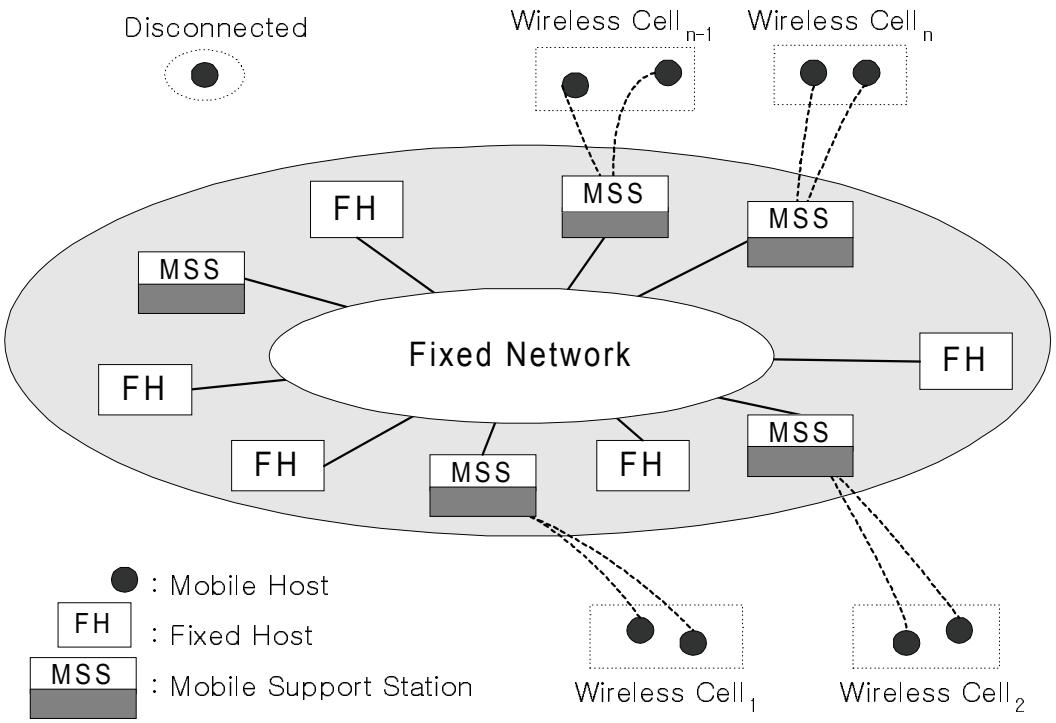


Figure 3.3: Mobile computing environment considered for MOFLEX [99]

reconnects, its tentative transactions are re-executed as base transactions to achieve global consistency.

High Commit Mobile (HiCoMo) is described by Lee et al. [110]. The protocol also differentiates between two transactions classes like the approaches described previously: base transactions and mobile HiCoMo transactions. HiCoMo transactions are executed on disconnected MHs and are transformed into base transactions at reconnection. However, semantic properties of the transactions are exploited, thus restricting the allowed operations only to additions and subtractions, severely limiting the applicability of this approach.

Isolation Only Transactions (IOT) by Lu and Satyanaranyanan [125] also considers two transaction classes to allow disconnections: first class transactions which are executed if all accessed resources are reachable and second class transactions which are executed under disconnections. Second class transactions are not committed immediately after execution but go into a pending state and are validated on reconnection, making resolution strategies or even user interaction necessary.

Pro-motion introduced by Walborn and Chrysanthis [197] considers the complete mobile system as one very large long-lived transaction executed on the FH.

Prewrite is described by Madria and Bhargava [128]. Here, the transactions execution is

also divided between sever (FH) and mobile client. While the transaction manager at the MH may execute the transaction, permanent writes are exclusively done by the server.

Reporting and Co-Transactions in a system of open and nested transactions are introduced by Chrysanthis [36]. While reporting transactions delegate their state to another transaction during execution, co-transactions are reporting transactions where the control is passed to the transaction receiving that report. The considered co-transaction is then suspended until receiving the report.

Semantics-based transaction processing is proposed by Walborn and Chrysanthis [196]. It makes use of semantic information about objects to improve the autonomy of MHs while being disconnected. Exclusive master copies of objects can be given entirely to MHs and then transactions can be directly executed on them. However, a reconciliation process is initiated by the server on reconnection.

Unilateral Commit Protocol for Mobile and Disconnected computing (UCM) is a One Phase Commit protocol introduced by Bobineau et al. [21]. To be able to come to a commit decision in just one phase, the authors make strong assumptions, for example in terms of reliable message delivery.

Transaction Commit on Timeout (TCOT) described by Kumar et al. [102] is based on 2PC but uses timeouts to save message transmission. Instead of sending *VoteCommit* messages, a coordinator implicitly commits a transaction if no *VoteAbort* message is received within a timeout. It provides only semantic atomicity and relies on compensating transactions.

Kangaroo Transactions outlined by Dunham et al. [50] is a mobile transaction model that focuses on the movement of MHs. However, transactions are coordinated by the base stations to which the MHs are assigned during transaction execution.

The Multi Database System Transaction Processing Manager (MDSTPB) is proposed by Zaslavsky et al. [208]. It runs on a FH and works as a proxy for MHs, allowing the MHs to submit a transaction and then disconnect without having to wait for the transaction to commit.

Pre-serialization (or Toggle Transactions) are described by Dirckze and Gruenwald [47, 48]. They use a set of global transaction coordinators located at each base station to verify the global serializability in advance.

Mobile 2PC is introduced by Nouali-Taboudjemat et al. [143] and compared with several other commit protocols [144]. Although the special requirements of mobile participants are considered, it is mandatory that the transaction coordinator resides on a fixed host.

CO2PC proposed by Serrano-Alvarado [180] provides semantic atomicity as it allows unilateral optimistic local commit for compensatable subtransactions. The coordinator must also be a FH that is chosen at the initiation of the transaction.

self-Adaptive Component-based cOmmit Management (ACOM) also described by Serrano-Alvarado [182] chooses either the presumed-commit 2PC or the presumed-abort 2PC [135] depending on the actual commit rate of the application.

Partial Validation with Timestamp Ordering (PVTO) is a protocol proposed by Lee et al. working on the application layer and relying on guaranteed message delivery [112], requiring also a fixed host for transaction execution.

Mobile Agents for Distributed Transactions of a Distributed Heterogeneous Database System are described by Ye et al. [206]. This approach also relies on a fixed infrastructure and bridges to traditional DBMS.

Argos Transaction Layer. Arntsen et al. introduce the Argos Transaction Layer in [9]. It allows the simultaneous deployment of several concurrently running transaction services providing different transactional guarantees. However, Arntsen et al. target their approach at standard transactions and long running business transaction in the web service environment and do not consider resource constraints.

CATran. An adaptive and context aware transaction model for ubiquitous environments is described by Tang et al. [193]. It considers mobile phones and mobile notebooks as devices, but the phones are only allowed to initiate transactions, not to execute subtransactions.

The previously described approaches rely on fixed networks including servers. Therefore, they are not directly applicable to wireless sensor networks.

3.2.3 Mobile Ad Hoc Networks (MANets)

Transactions have been an area of significant research in the area of Mobile and Ad hoc Networks (MANets) in recent years. Contrary to fixed networks, MANets communicate in an ad hoc manner, without using servers or any kind of infrastructure, and thus are more related to wireless sensor networks.
But there are also several differences between MANets and wireless sensor networks. Wireless sensor networks normally consist of a higher number of nodes, these are distributed with a higher density and in varying topologies. Wireless sensor networks suffer from higher failure rates and also from more severe restrictions in terms of storage capacity and processing power. Another major difference is that sensor nodes are seldom recharged, in contrast to MANet devices like mobile phones or palms. So most of the applications for wireless sensor networks have to take a limited lifetime into account, which is restricted by

the used battery. Therefore, in wireless sensor networks it is much more important to save energy. Since transmitting and receiving consumes the most power in sensor networks, most energy can be saved by reducing the number of messages and the transferred volume of data. We describe related approaches in the following.

Group Based Transaction Commit Protocol. Xie [202] describes distributed transaction processing over mobile and heterogeneous platforms. Especially partitioned networks are considered. The author assumes that every mobile node knows all the members of the partition it belongs to. Inside every partition, 2PC is used to come to a tentative decision and the decision is broadcasted inside the partition. If participants join other partitions, they communicate the tentative decision there. The work relies on stable storage and does not consider the severe resource restrictions of wireless sensor networks. Also neither an implementation nor a comparison with other protocols is given.

Bi-State Termination is described by Obermeier et al. [149]. The idea is to tentatively terminate a transaction as committed and also as aborted to reduce the time of resource blocking. When the data written by a tentatively terminated transaction is accessed by another transaction T, several possibilities exist:

- T is aborted completely.
- T is committed and deals with multiple possible results. This might not be possible in all application scenarios.
- Only certain parts of T are committed or aborted.
- T waits.

For this protocol, it is necessary to store all possible commit/abort combinations of dependent transactions, posing difficulties for severely resource constraint wireless sensor nodes.

Integrated / Failure Tolerating Commit Protocol. An integrated commit protocol is described by Bose et al. [23] and [31]. A failure tolerating commit protocol is described by Böttcher et al. [25]. Both protocols use several coordinators to avoid blocking in case one coordinator fails. First, a main coordinator is selected to run the 3PC protocol. If this fails due to coordinator failure or due to a partitioning of the network, Paxos Consensus [107] is used to select a new coordinator. The authors assume that the participants of a transaction are situated in a one-hop environment during transaction execution, limiting the applicability of their approaches.

ParTAC. Ayari et al. describe the Partition-Tolerant Atomic Commit (ParTAC) protocol [10, 11, 12] for the use in mobile ad hoc networks. They demonstrate their approach on four Lego Mindstorm cars equipped with HTC PDAs running their ParTAC algorithm implemented in Java. The application scenario of the four cars is to agree on

the order of crossing an intersection [12]. Although very interesting, the work has some limitations:

- No comparison with other protocols has been done in the evaluation.
- The papers give the approximate number of messages exchanged per participant, but not the exact number of bytes sent.
- The authors do not state which routing protocol was used for the execution of the experiments, they only state that flooding or ad hoc on demand distance vector routing could be used depending on the relation of participating nodes to nodes in the network.
- In ParTAC, coordinators send beacons periodically, but it is neither stated how long a period is nor if and how the beacons are taken into account in the evaluation.

But the most important limitation considering our application scenario is that the severe resource constraints of wireless sensor networks have not been considered.

Cross Layer Commit Protocol (CLCP). The Cross Layer Commit Protocol proposed by Obermeier et al. [148, 151] uses all participants as coordinators and uses consensus to ensure fault tolerance. It is described in more detail in Subsection 3.2.4.

While all of the previously describe protocols yield interesting ideas, they do not take characteristic properties of wireless sensor networks into account. These are in particular:

- A much higher number of nodes. The protocols described previously have only been evaluated with small node numbers (7 nodes (CLCP) to 10 nodes (ParTAC)).
- Low bandwidth wireless radio and message sizes.
- Small ROM and RAM sizes.
- Distant transaction participants.

3.2.4 Cross Layer Commit Protocol

The Cross Layer Commit Protocol (CLCP) [148, 151] has been developed for the use in Mobile and Ad hoc Networks (MANets). The protocol is an advantage over Paxos Commit [66] in terms of message transmissions. Paxos is an advancement of the Three Phase Commit Protocol (3PC) [186] because it can also come to a decision if network partitioning occurs.
CLCP has been evaluated in MANet scenarios, but only in quite specific ones since the authors assume that all participants of a transaction are direct neighbors when a transaction is started. This allows them to use flooding restricted to a one hop distance (bounded flooding). For our use case, this is an unrealistic assumption since we want our services to be distributed over the whole network like shown in Figure 3.4. It shows an example of

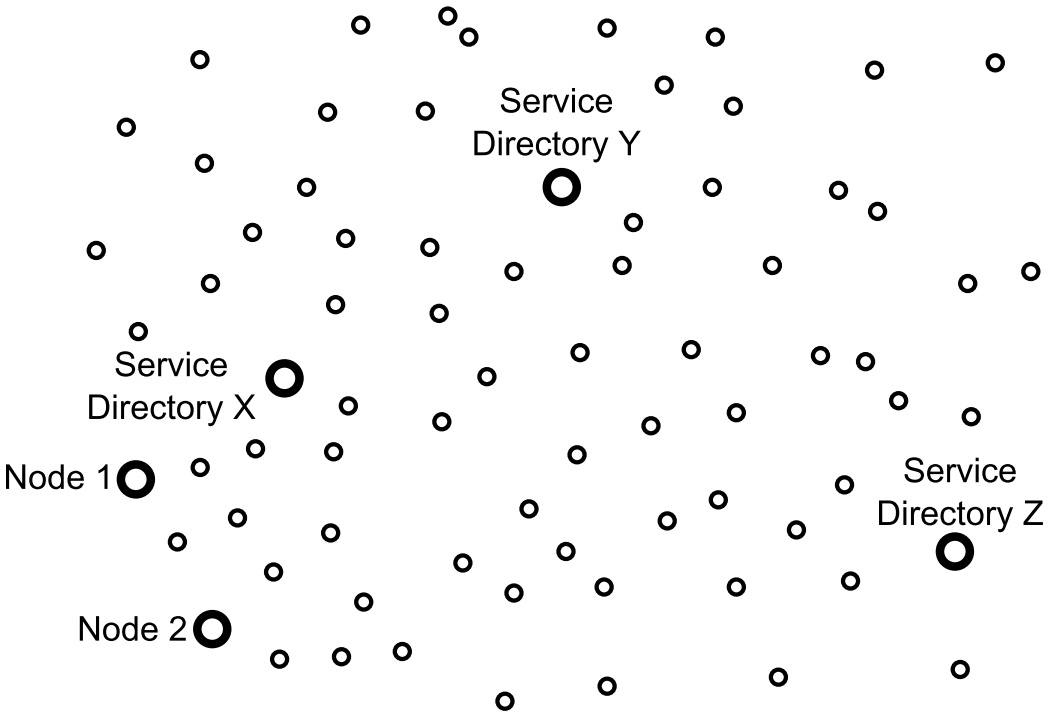


Figure 3.4: Distant participants of a transaction

distantly distributed nodes, namely source node `Node 1`, target node `Node 2` and three service directories which are all participants of one transaction.
We briefly outline the function of CLCP in the following, for a more detailed description see [148, 151]. CLCP consists of two phases. In the first phase, the decentralized commit phase, the participants of a transaction vote concurrently whether a transaction should commit or abort.

Decentralized Commit Phase (DCP)

The key idea of the DCP is to use the knowledge of each participant about the ongoing voting process via a so called commit matrix. After an initial *Prepare* message of a transaction initiator, a commit matrix is broadcasted as a reply and contains the information each participant has about its own vote and about every other participant. We give an example for a commit matrix for three participants in Figure 3.5. The size of the commit matrix increases in square with the number of participants. With more than 10 participants, messages are becoming too large for wireless sensor networks, as these are collision-prone even at message sizes of about 50 Bytes [126]. The commit matrix contains the information that P_1 knows that itself and P_3 voted for commit and that P_3 knows that itself voted for commit. Other possible entries are (ordered by priority): $empty < voteCommit < voteTimeOut < timeOutAck < voteAbort$. Upon the reception of a commit matrix, it is merged with the information a node already has and then broadcasted again as an acknowledgement until a decision is found or the termination phase starts due to a timeout. Three cases which lead

Knowledge of→ / about↓	P_1	P_2	P_3
P_1	$voteCommit$	$empty$	$empty$
P_2	$empty$	$empty$	$empty$
P_3	$voteCommit$	$empty$	$voteCommit$

Figure 3.5: An example of a commit matrix in CLCP

to a decision can occur:

1. All nodes voted for commit and a majority of nodes knows that, i.e. each row in the commit matrix contains a majority of $voteCommit$ entries. Then the decision is commit.

2. At least one node voted for abort. Then the decision is abort.

3. There exists a majority of $timeOutAck$ entries in at least one row. This means one participants was not reachable for a determined time. In this case, the decision is abort.

If neither a $voteAbort$ nor a majority is found the termination phase is started after a timeout.

Termination Phase (TP)

In the case of a timeout, maybe due to a network partitioning, the second phase of CLCP, the so called termination phase begins. The TP works similar to Paxos Commit and marks a participant as a special leader which makes sure that the commit decision is accepted by a majority of nodes. Every time this does not work, a new coordinator is selected based on increasing version numbers.

3.3 Two Phase Commit with Caching

In this section we present our own variant of 2PC called Two Phase Commit protocol with Caching (2PCwC). It was inspired by the Cross Layer Commit Protocol (CLCP) and also takes advantage of broadcast communication, but has a significant lower communication cost than CLCP.
We first describe the key idea behind our protocol and then explain the two optimizations we applied to the Two Phase Commit protocol (2PC) to increase its commit rate and to save message transmissions. Finally, we give some details about the synchronization of our protocol.

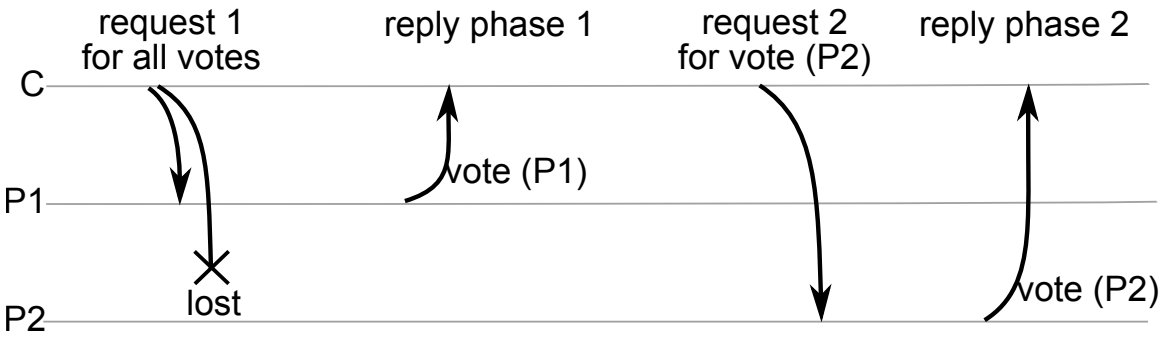


Figure 3.6: 2PC: if a request (or a vote) gets lost, an additional request phase is needed

3.3.1 The Idea

If a node P_1 sends a message M to P_2, P_2 sends an acknowledge message M_{ack} back to P_1 if P_1 needs a confirmation for the reception of M by P_2. Generally, acknowledgement messages are processed on the communication layer of the OSI-7 model, unnoticed by the overlying application.
On the contrary, in CLCP, acknowledgement messages are implicitly used to communicate information about the commit decisions of other participants of a transaction. Information about the votes of other participants are stored (cached) in order to be processed later to speed up the commit decision.
This approach motivates the question to what extent the 2PC protocol could benefit from the caching of information about other participants. Since the used communication in a wireless sensor network is broadcast, nodes can hear each others messages without extra effort if they are in a certain neighborhood. The question is which messages are useful to be cached.
Therefore, we take a closer look at message loss in the 2PC protocol and consider the voting phase which is shown in Figure 3.6. First, the coordinator C sends a request for votes to the participants $P1$ and $P2$. While $P1$ receives the request, $P2$ does not due to message loss. Consequently, only $P1$ votes and after a timeout, C requests the vote of $P2$ again, which replies successfully at this time. The same sequence could occur if the vote message of $P2$ had been lost instead of the request message of C.
So we let all nodes cache the votes of other participants, extend the votes by the ids of the other participants and then distinguish between two cases:

1. The vote of participant $P2$ gets lost before it is received by the coordinator. Then, another participant, let us say $P1$, can forward the cached vote of $P2$ in order to prevent it gets lost again. This case is explained in Subsection 3.3.2.

2. The request message of C is not received by participant $P2$, but $P2$ hears the vote of $P1$. In this case, $P2$ can also send its own vote message. This case is described in Subsection 3.3.3.

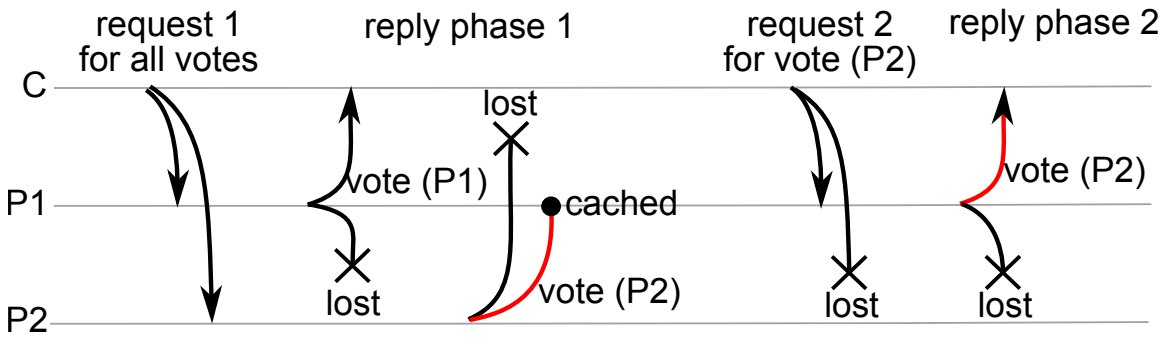


Figure 3.7: 2PCwC: $P1$ replies in place of $P2$ because the vote of $P2$ got lost

3.3.2 Replying in Place of Other Participants

We explain in this subsection how our 2PCwC protocol makes it possible that one participant can reply to the coordinators request in place of another participant.
We assume that both participants $P1$ and $P2$ received the request message of the coordinator C (see Figure 3.7).
While the vote of $P1$ is received by C, the vote of $P2$ is lost on its way to C, but is cached by participant $P1$. $P2$ is not aware of that. After a timeout, the coordinator C misses the vote of $P2$ and sends his request again. The request is lost on its way to $P2$, but is received by $P1$, and since $P1$ cached the vote of $P2$, $P1$ can reply to C in place of $P2$ and the coordinator receives the missing vote.

3.3.3 Replying Without Having Received a Request

In this subsection we describe how a participant can send his vote without having received the request of the coordinator. We assume that the request of the coordinator C is only received by participant $P1$ and not by participant $P2$ (see Figure 3.8). So, only $P1$ sends its vote message and $P2$ does not. However, as soon as $P2$ hears the vote message of $P1$, it infers that the coordinator must have sent a request and can also send its own vote message to C. Consequently, a second request phase is unnecessary and message transmissions are saved.

3.3.4 Synchronization and Other Technical Details

In both variants, 2PC and 2PCwC, the coordinator sends requests only addressed to the nodes which votes are missing. The requests are flooded nevertheless to reach every participant, and since all messages are broadcasted, they can be heard by all other nodes. The important point is that the participants whose votes were already received do not reply again. Additionally, we used flooding with message extinction, so nodes only forward messages which are new to them.

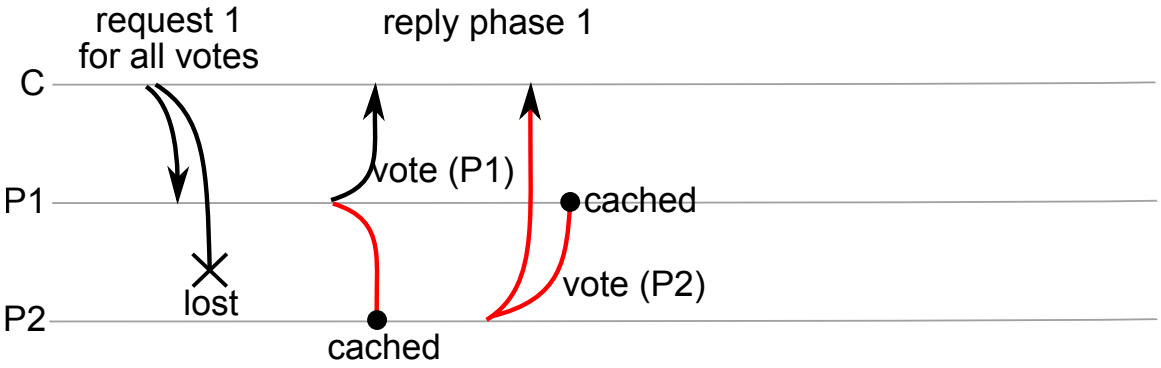


Figure 3.8: 2PCwC: a participant votes without having received the request of the coordinator

Another important implementation detail is that participants in 2PCwC have to wait and listen for other replies for a randomized interval before they reply to requests in place of other nodes. This is necessary in order to prevent the case that too many nodes reply to a repeated request of the coordinator.
Moreover, transactions are deleted from the cache after a determined time since memory is a limited resource on wireless sensor nodes.
Evaluation results obtained from simulations and also from experiments with our sensor node platform Pacemate are given in Section 3.5.

3.4 Implementation

To be able to perform experiments with the three commit protocols 2PC, CLCP and 2PCwC, we implemented the Adaptive Transaction-based Management of Services (ATMoS) for the iSense middleware [37] and built the code for the sensor network simulator Shawn [55]. This allows us to build the same code for the sensor node platform Pacemate [122] to perform more realistic experiments during the course of our study. An overview about ATMoS is shown in Figure 3.9.

3.5 Evaluation

In this section, we present the experiments we performed to compare the three commit protocols Two Phase Commit (2PC), Cross Layer Commit Protocol (CLCP) and Two Phase Commit with Caching (2PCwC). First, we present the simulation environment we used for the sensor network simulator Shawn, then we specify the measured variables and finally describe our results.

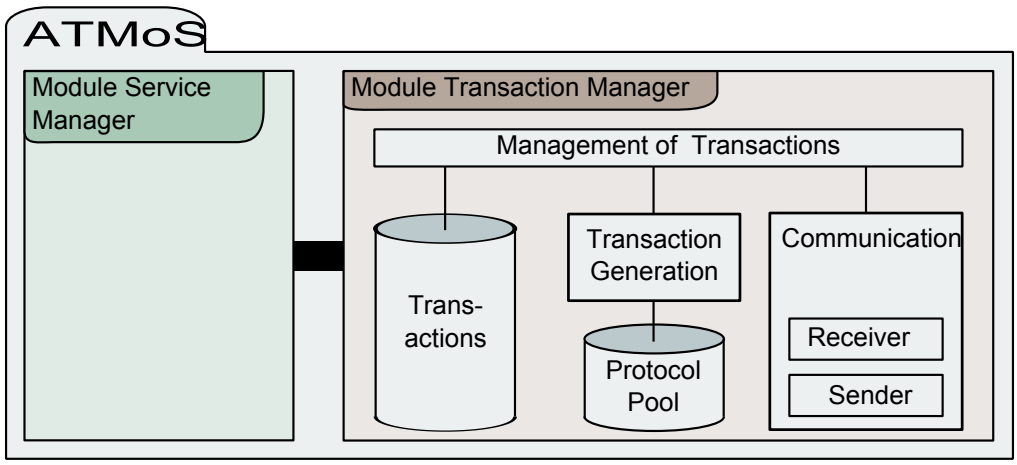


Figure 3.9: Overview of our Adaptive Transaction-based Management of Services (ATMoS) framework

3.5.1 Simulations

In the following, we describe the setup and outcome of our simulations.

Experimentation

We performed experiments with each of the three commit protocols (2PC, 2PCwC and CLCP) and varied the number of participants and the message loss. The constant simulation parameters are shown in Table 3.1. We randomly distributed 100 nodes in an area of 500 x 500 field units (see Figure 3.10) and started 1000 transactions.

Width of area (field units)	500 FU
Height of area (field units)	500 FU
Number of nodes	100
Number of started transactions	1000
Average neighborhood size	9.84
Maximum range (field units)	100 FU

Table 3.1: Constant simulation parameters for the comparison of 2PC, 2PCwC and CLCP

Each node started 10 transactions and the participants were chosen randomly. This means the participants could be located anywhere in the network. We varied the number of participants from 2 participants to 10 participants for each run of our experiments. All messages were flooded in the whole network and participants always vote for commit. This makes an evaluation of the protocols easier, since no empirical values exist for common local commit / abort rates in wireless sensor networks. In each experiment with 2PC and 2PCwC, the coordinator was allowed to ask up to 6 times for missing votes that got lost.

Figure 3.10: Our used topology: 100 randomly distributed nodes where an edge between two nodes means that the distance between them is smaller than the maximum transmission range

While the nodes always had a maximum range r_{max} of 100 field units we varied the guaranteed range r_{min} between 100 field units (no loss) and 1 field unit. Then we determined the actual range by using the Quasi Unit Disc Graph Model (QUDM), which is proposed by Kuhn et al. [100], and also described by Zunig et al. [211]. The probability p that a node at a distance of d receives a message is 1 if $d < r_{min}$, 0 if $d > r_{max}$ and decreases linearly from 1 to 0 if $r_{min} \leq d \leq r_{max}$.
The QUDM was also used by Obermeier et al. [148, 152]. The probability for the reception of a message in a one hop neighborhood and the probability for the distribution of a message in the whole network are shown in Table 3.2.

Max. range	Min. range	p (one hop)	p (network)
100	1	37.76%	63.25%
100	10	40.00%	70.47%
100	100	100.00%	100.00%

Table 3.2: Behavior of the Quasi Unit Disc Graph Model Model (QUDM) when varying the minimal transmission range in the network simulator Shawn

Measured Variables

We were mainly interested in the performance of the protocols in terms of the commit rate and the costs of each transaction measured in number of transmitted bytes. The trade off between these two measures can be expressed as number of transmitted bytes per commit decision. Since memory is a limited resource in wireless sensor networks, we also analyzed the memory consumption of the three commit protocols.

Results

In the following we present the results of our experiments. We begin with a comparison of the commit rates, show the transmitted bytes per commit decision and conclude with a comparison of the memory consumption of the three commit protocols.

Commit rates. The commit rates for the three protocols are shown in Figure 3.11. QUDM-10 means we used a minimum range of 10 field units and QUDM-1 means we used a minimum range of one field unit. It can be seen that with increasing message loss, the commit rates of all protocols decrease. However, this is most severe for 2PC: the commit rate drops here from 40% to 20% and for 2PCwC the commit rate drops from 71% to 53% while CLCP stays nearly the same (89% instead of 95%). But we will see in the following that these commit rates for CLCP come at a high price. Figure 3.11 shows also that an increasing number of participants leads to a better commit rate for CLCP while the commit rates for 2PC and 2PCwC decrease.

Transmitted bytes per commit. Now we look at the price for the achieved commit rates. The transmitted bytes per commit decision are shown for the three protocols in Figure 3.12. It can be seen that CLCP performs worst at QUDM-10 and 2PC performs worst at QUDM-1 while 2PCwC has the best ratio of transmitted bytes to commit rate at both transmission models. In fact, 2PCwC needed only $\sim$50% of the transmitted bytes compared to 2PC or CLCP.

So our results differ decisively from the results of Obermeier et al. [148, 151]. The reasons for this are twofold:

1. We do not assume the usage of IEEE 802.11 (Wi-Fi), where Obermeier et al. assume that point-to-point communication is much more expensive than broadcast.

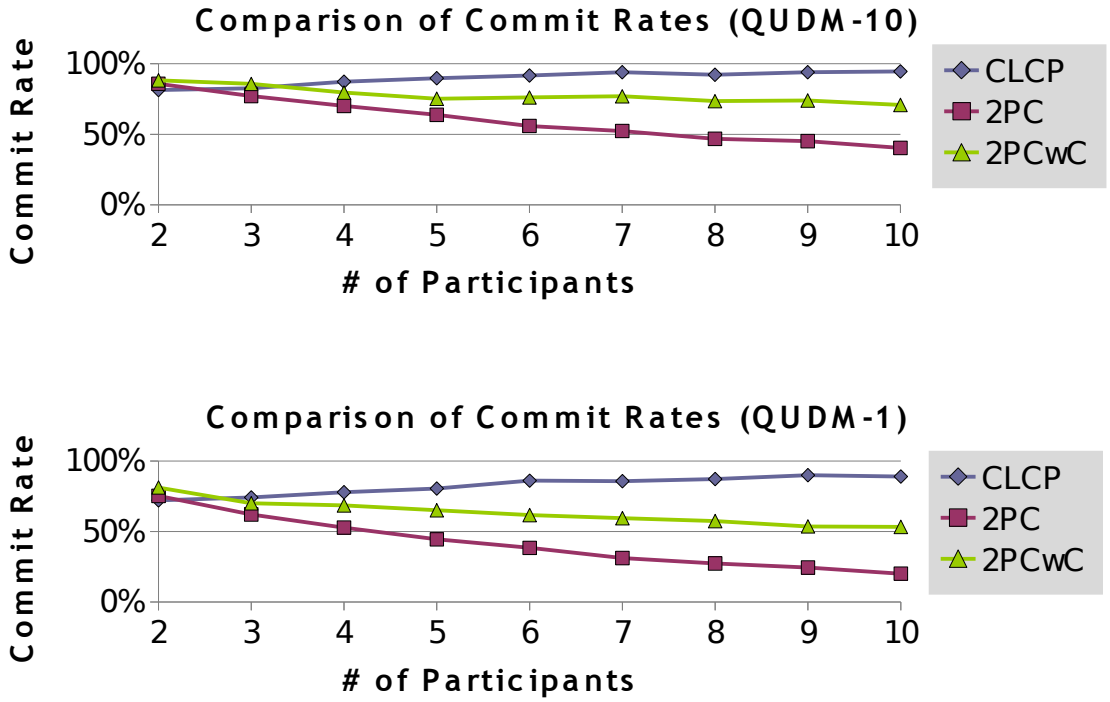


Figure 3.11: With increasing message loss, the commit rates of the 2PC protocols decrease

2. We allow our participants to be distributed over the whole network instead of restricting them to a nearby area. So our experiments are more tailored to wireless sensor networks instead of ad hoc networks and also more general.

Impact on energy consumption and lifetime. Our Two Phase Commit with Caching (2PCwC) protocol needed only $\sim$ 50% of the transmitted bytes compared to 2PC or CLCP, at a guaranteed range of 10 field units with 100 nodes, $\sim$ 50 kilobytes were transmitted in the whole network compared to $\sim$ 100 kilobytes for CLCP and 2PC. So on average, every single node sent 500 bytes respective 1000 bytes per committed transaction. Since we had an average neighborhood size of 9.84 in our simulated example topology, we assume that every message sent was received by $\sim$ 10 nodes.
In the following we calculate the energy needed to execute one transaction and relate the result to the available energy on our sensor node platform pacemate. Therefore, we first determine how long it takes the Pacemates to send 500 bytes respective 1000 bytes. Then we simply divide the energy available by the energy consumed by sending and receiving this particular time.
The Xemics XE1205 transceiver used in our sensor node platform Pacemate has the following power consumption and data transmission rate:

- Power consumption when receiving: 15 mA
- Power consumption when transmitting: 75 mA (15dBm)

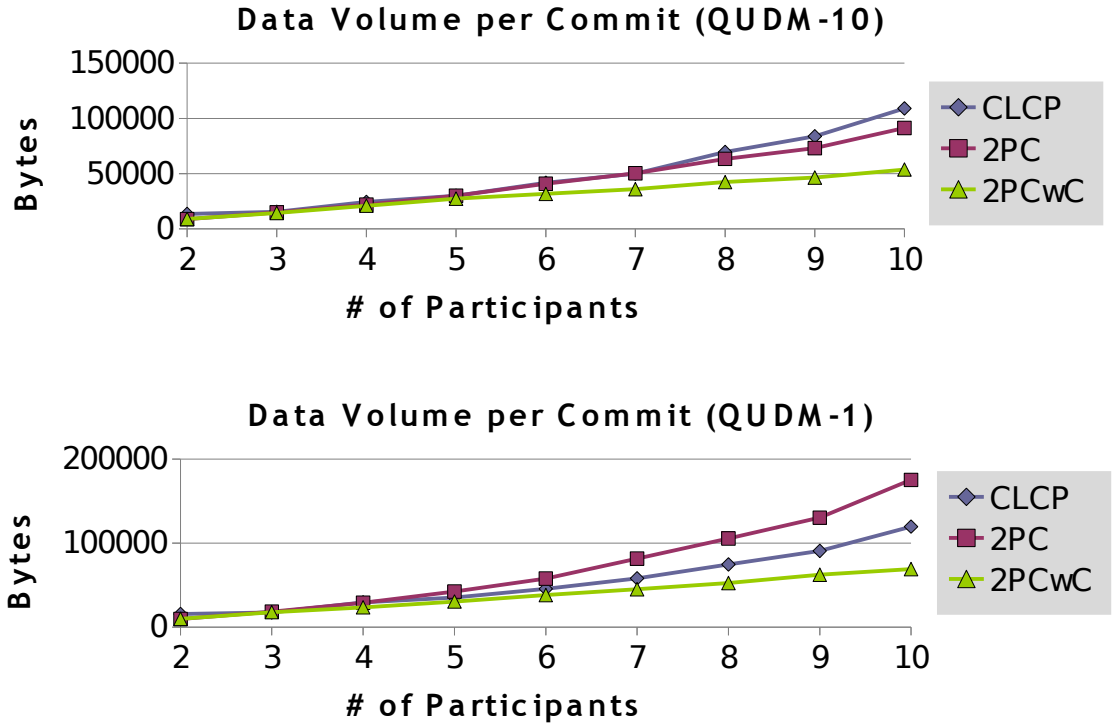


Figure 3.12: 2PC with Caching outperforms 2PC and CLCP at different message losses

- Maximum data rate: 152.3 kilobits per second

The Pacemates are equipped with 2500 mAh rechargeable batteries. If we assume a node would just transmit data until its battery is depleted, then the node could transmit for $2500/75 = 33.\bar{3}$ hours. In the following we calculate how long it takes a Pacemate approximately 500 bytes needed for one transaction with the Two Phase Commit with Caching protocol (respective sending the 1000 bytes needed for one transaction with CLCP or 2PC). The maximum data rate is $152.3\ kilobits/s = 19.0375\ kilobytes/s$, so the cumulative transmission periods take

- 0.025 s for Two Phase Commit with Caching and
- 0.05 s for CLCP or 2PC.

Consequently, we get for the power consumption of one transaction per node

- $0.025s * 75mA + 0.025s * 10\ listeners * 15mA = 5.625mAs$ (2PCwC)
- $0.05s * 75mA + 0.05s * 10\ listeners * 15mA = 11.25mAs$ (2PC or CLCP)

This means if we only perform transactions and do not run an application also, the power of the batteries is sufficient for running approximately 800,000 transactions with 2PC or CLCP or for running approximately 1,600,000 transactions with 2PCwC ($2,500mAh = 150,000mAmin = 9,000,000mAs$ and $9,000,000mAs/5.625mAs = 1,600,000$).

So what do these numbers mean? Of course the chosen protocol does not matter in terms of energy consumption, if only a very low number of transactions are performed during the lifetime of the application. So the need for efficiency strongly depends on the application and the required lifetime of the deployment. But we have to keep in mind that the usage of CLCP for a moderate number of participants (more than ten) is not even feasible when energy consumption can be neglected because (a) CLCP requires a large footprint ($\sim$ 30 kB) and (b) the message sizes grow in square with the number of participants. This is a problem in wireless sensor networks since exceeding a message size of $\sim$ 50 Bytes increases the probability of collisions on the MAC layer [126].

Memory consumption. The memory consumption of a protocol for the processing of a transaction is important because the memory of sensor nodes is limited and so is the maximal number of transactions that can be processed concurrently.

The dynamic memory consumption for the transaction data structures for CLCP, 2PCwC and 2PC is shown in Figure 3.13.

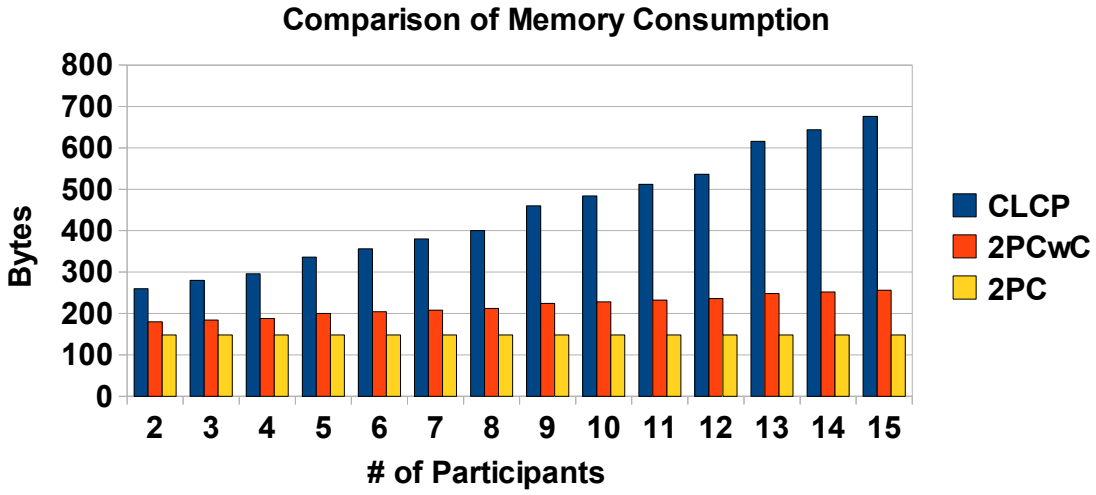


Figure 3.13: Due to the used commit matrices, the memory consumption of CLCP grows with the square of the number of participants while the memory consumption of 2PCwC grows only linearly

While for 5 participants, we have a memory consumption of 336 bytes for CLCP and 200 bytes for 2PCwC, for 10 participants we already have 484 bytes for CLCP and only 228 bytes for 2PCwC. So it is obvious that the memory consumption for CLCP grows in square of the number of participants while the memory consumption of 2PC and 2PCwC only grows linearly with the number of participants.

If we consider the Pacemate sensor node platform with 32 kilobytes memory and assume that 2 kilobytes are reserved for transactions, then we can process 13 transactions concurrently with 2PC, 8 with 2PCwC but only 4 transactions with CLCP when we have 10

participants. The memory available for the transaction management is obviously determined by the memory usage of the general application which is running in the wireless sensor network.
We elaborate on the memory demands of the program code of the three protocols in Subsection 6.2.3.

3.5.2 Real Node Deployment

To confirm the simulation results, we deployed 20 Pacemates on the corridor of our institute like shown in Figure 3.1. We started 25 transaction with 2PC and 25 transactions with 2PCwC. The coordinator could repeat its request for vote six times. We repeated both test runs 25 times.
As shown in Table 3.3, with 2PC, a mean of 53.44% of the started transactions committed, with our 2PCwC, 83.84% transactions committed. The costs were on the average 4988 bytes per commit for 2PC and only 3997 bytes per commit for our protocol 2PCwC. This is 249.4 bytes per node for 2PC and 199.9 bytes per node for 2PCwC. So our 2PCwC protocol performs significantly better in real world environments.

	2PC	2PCwC
Commit Rate	53.44%	83.84%
Transmitted Bytes per Commit per Node	249.4	199.9

Table 3.3: The results on the Pacemates approve the results obtained by simulations

Chapter 4

Concurrency Control

A number of sensor network databases (SDBS) have emerged in recent years. The most prominent ones are TinyDB [127], SwissQM [133, 134] and StonesDB [45]. While TinyDB is mainly an abstraction layer for simplifying the querying of wireless networks measuring live data by providing an SQL-like interface, StonesDB makes use of a node's flash memory to store historic data. In StonesDB the measured data is then processed directly on the node and only relevant items are forwarded to the gateway in order to save transmission costs.

What has been missing in sensor databases compared to traditional databases until now is the capability of transaction processing. This has mainly two reasons: On the one hand, the severe resource constraints of wireless sensor networks make the implementation of complex protocols complicated. On the other hand, until now exclusively read-only queries have been considered in sensor network databases. However, as Guergen et al. [69] point out, the variety of emerging sensor networks applications also demands for update queries, for example, system management queries which allow the alteration of a data model at runtime.

Consider as an example a continuous query that asks for the average temperature in Celsius in each section of a factory. The result is used to trigger a fire alarm if a certain threshold is met. Now consider a concurrent update query which modifies the measuring unit of section A to Fahrenheit. If no concurrency control is present which guarantees the isolation of both transactions, a false alarm might be triggered.

System management queries are not only needed for changing a measuring unit, but also, for example, when nodes are relocated and the position information needs to be changed at runtime. Another application domain for concurrency control protocols and also atomic commit protocols in wireless sensor networks are sensor and actor networks.

Whenever nodes not only measure their environment but also perform actions based on their measurements, a transaction concept might be needed, especially in a military environment, but also in civil application domains. As an example for distributed agreement, consider four unmanned vehicles crossing an intersection [12]. Somehow the cars must

agree on an order for crossing the intersection to prevent accidents. In general, the coordination of multiple actors can be required in the following situations [4]:

- One actor may not be sufficient to perform the requested operation.
- It might be required that an action is performed by exactly one actor.
- If multiple actors perform an operation, this has possibly to be done simultaneously so that synchronization is required.
- If a region is covered by multiple actors, they might all have to cover their own region, so that no overlaps occur. Hence, mutual exclusion must be guaranteed.
- Ordered execution of tasks might be relevant.

According to Akyildiz et al. [4], a task performed by actors is defined as an atomic unit of computation and control. Also, it might be required that a certain task is by no means executed by all possible actors at the same time, like in the case of disposers of a tranquilizing gas, which could lead to catastrophic events.
Hence, to achieve a well defined behavior, concurrency control for wireless sensor networks is needed, which is neither considered by Ayari et al. [12] nor by Akyildiz et al. [4]. We believe that the well-known serializability concept from the database area [16] is able to fill this gap in order to enable more sophisticated sensor network applications. However, the usage of well known algorithms like two-phase locking, timestamp ordering or validation is not straight forward since wireless sensor networks pose new challenges in terms of message loss and severe resource constraints.
In this chapter we analyze the mentioned concurrency control protocols with regard to their usage in wireless sensor networks and integrate an adapted version of each protocol in the well known Two Phase Commit (2PC) protocol [16]. Evaluation results of 2PC obtained from simulation and experiments with real sensor nodes can be found in [165, 166, 167, 168, 169].

Organization

The remainder of this chapter is structured as follows:

- In Section 4.1, we outline related work on concurrency control in wireless sensor networks, mainly on sensor network databases and concurrency control protocols in mobile ad hoc networks.

- We describe in Section 4.2 how we adapted the traditional concurrency protocols locking, timestamp ordering and validation for wireless sensor networks with respect to their severe resource constraints.

- We evaluate and compare the implemented protocols by performing experiments with the network simulator Shawn in Section 4.3. We also describe our implementation of locking for the sensor node platform Pacemate. To the best of our knowledge

this is the first work reporting an implementation of a database concurrency control protocol for real sensor nodes.

- Section 4.4 concludes this chapter.

4.1 Related Work

In this section we briefly outline related work in the areas of sensor network databases and concurrency control in mobile and ad hoc networks.

4.1.1 Sensor Network Databases

The existing sensor network databases TinyDB [127] and StonesDB [45] do not provide any transaction processing capabilities. Levent et al. [69] claim that these capabilities are needed and describe an algorithm for the concurrent processing of continuous queries in a wired sensor network. The algorithm is not implemented and the unique properties of wireless networks are not considered in depth by the authors. Therefore, we review related work in the area of mobile and ad hoc networks.

4.1.2 Concurrency Control in Mobile Ad Hoc Networks

There are significant differences between Mobile and Ad hoc Networks (MANets) on one hand and wireless sensor networks on the other hand. Common examples of MANet devices are mobile phones or palms. Wireless sensor networks are considered to be a subset of MANets. They often consist of a higher number of nodes which are distributed with a higher density and in varying topologies. Wireless sensor networks can also suffer from higher failure rates and also from more severe restrictions in terms of storage capacity and processing power. Another important difference is that sensor nodes are seldom recharged, in contrast to devices like mobile phones or palms. Therefore, it is much more important to save energy in wireless sensor networks. Since transmitting and receiving consumes the most power in sensor networks, most energy can be saved by reducing the transferred volume of data.

Lee et al. [111] consider MANets and introduce a protocol for broadcast environments where read only transactions can be validated on clients only. Since we do not differentiate between clients and servers, we need a more general kind of concurrency control.

Brayner et al. define the correctness criterion Mobile Semantic Serializability [28] and describe SESAMO [27], a concurrency control algorithm which relaxes global serializability and provides mobile semantic serializability instead.

Xing et al. [203] propose an optimistic concurrency control algorithm for MANets called Sequential Order with Dynamic Adjustment (SODA) which relies on a clustering algo-

rithm. Although using clustering could also be beneficial for our implementation, the authors do not consider the severe resource constraints of wireless sensor nodes.

4.2 Adapting Traditional Concurrency Control Protocols

As outlined in the introduction, emerging applications for wireless sensor networks demand for concurrency control. Examples are update queries in sensor network databases like system management queries and wireless sensor and actor networks. In this section, we outline our implementation of traditional concurrency control protocols for wireless sensor networks. Every protocol was integrated in the Two Phase Commit (2PC) protocol to provide atomicity and serializability.
One major difference between traditional distributed database environments and wireless sensor network databases are the missing logging capabilities. Traditional database systems provide the possibility to recover from a crash of a participant by writing temporary data to a stable storage which can be accessed after the failed node has been restarted. We did not implement logging in our protocols since contrary to database server, crashed sensor nodes are considered to be either damaged our out of power. Hence, they do not restart and cannot recover.
Another major challenge are the severe resource constraints of sensor nodes. We describe how we have taken them into account in the next subsections.

4.2.1 Locking - Strong Strict Two Phase Locking (2PL)

The widely used Strong Strict Two Phase Locking (SS2PL) is described by Gray et al. [63]. It guarantees serializability by first acquiring locks on all accessed resources (phase 1), performing the requested operations if all locks were granted and releasing the locks afterwards (phase 2). Since our application domain is a distributed sensor network, we also implemented a distributed management of locks. This means each participating sensor node locks the accessed resources (if not already locked) when receiving a begin vote message and unlocks the resources when receiving the commit or abort decision from the coordinator. Global deadlocks due to locking generate automatically voting-deadlocks in 2PC, and are thus resolved automatically by 2PC when the respective coordinator timeout expires. This behavior was generalized by the principle of commitment ordering by Raz [164]. If a node does not receive the commit or abort message from the coordinator, the respective resource remains locked. To prevent this blocking, our participants broadcast a *HelpMe* message if they do not receive the decision message. While this decreases the number of blocked resources, it cannot guarantee that no resources remain locked since coordinating nodes can fail and message loss can occur repeatedly.

4.2.2 Timestamp Ordering (TO)

As outlined in Section 2.1, timestamp based concurrency control is also called Timestamp Ordering (TO) and is described by Bernstein et al. [17]. Using this technique, serializability is guaranteed by timestamps assigned to data objects and transactions. The validation of timestamps is performed at each sensor node participating in the transaction, leading to an inherently distributed, deadlock free protocol. A globally unique timestamp is assigned to every transaction at begin of transaction (BOT). A globally unique timestamp can for instance be determined by using the node ID and its local time [106]. Without synchronization, no monotonic increasing of the timestamps is guaranteed, which can lead to the abort of transactions. Using timestamp ordering, the global execution order of transaction is predefined by their timestamps. Conflicting operations must occur in the order of the transaction timestamps. Although time synchronization in wireless sensor networks is possible and a lot of publications deal with this topic (see, for instance, Römer et al. [171]), it induces a significant overhead and it is hard to guarantee globally unique timestamps in a timely manner under message loss.
To be able to validate the transaction timestamps, a read timestamp (RTS) and a write timestamp (WTS) are assigned to every data object. These timestamps are always updated if the data object is accessed by a transaction. A read transaction T on an object x is aborted, if $ts(T) < WTS(x)$ holds. A write transaction T on an object x is aborted, if $ts(T) < \max\{WTS(x), RTS(x)\}$ holds. Although timestamp ordering is deadlock free, blocking can occur. Assume T1 has successfully updated data object x, but has not committed yet. Assume also, that T2 wants to read x. If T2's read access is allowed and T1 is aborted later, then T2 must also be aborted. This can lead to cascades. So T2 must wait until T1 has committed, which leads to a blocking situation analogous to locking.
We decided to abort transactions in this case, because on the one hand, there may be timely constraints due to the application and on the other hand, because the used 2PC would abort the transaction anyway when the respective coordinator timeout expires.

4.2.3 Forward Oriented Optimistic Concurrency Control (FOCC) by Validation

Optimistic concurrency control (OCC) by validation is introduced by Kung et al. [103]. OCC protocols are based on the assumption that conflicts occur rarely, which makes locking, for instance, an unnecessary overhead. We differentiate between the three phases: read, validate and write.
i) Read phase: In this phase a transaction reads its needed data objects and performs its writes to its own private workspace.
ii) Validation phase: The validation phase is started at EOT. It is checked if conflicts between concurrent transactions have occurred. If this is the case, the transaction is aborted. Neither blocking nor deadlocks can occur, but frequent aborts can lead to starvation of a

transaction.
iii) Write phase: The updates stored in the private workspace are written to the log and update the actual database, becoming visible for other transactions.
To perform the validation, the objects to be accessed by transaction T_i are assigned to its write set $WS(T_i)$ respectively read set $RS(T_i)$. According to [71], two classes of OCC protocols can be distinguished: Backward Oriented Optimistic Concurrency Control (BOCC) and Forward Oriented Optimistic Concurrency Control (FOCC). BOCC validates transactions against already committed transactions, while FOCC validates transactions against concurrently running transactions (see Listing 4.1). Both techniques guarantee serializability by making sure that a transaction has read all changes written by all previous successfully validated transactions.
The advantage of FOCC is that only real conflicts lead to aborts and that in mobile environments, nodes can work autonomously when temporary disconnected. Also, FOCC is more flexible. While BOCC always aborts the validating transactions in case of conflicts, FOCC also allows to abort the other conflicting transaction. When using distributed validation, every sub transaction is validated on the node it is executed on. Global transactions are synchronized by means of 2PC as follows: The prepare message is also used as a request for validation. After a successful local validation, every sub transaction saves its changes in a local workspace and sends a vote commit. If the validation fails, the sub transaction sends vote abort and deletes the workspace. If all local validations have been successful and all vote commit messages have been received by the coordinator, the coordinator sends commit to the participants. These write the changes stored in their private workspaces to the actual data.
The disadvantage of FOCC is normally that read set and write set have to be known in advance. While this is a restriction for some application use cases, it is no problem in our application scenario.

```
VALID := TRUE;
FOR Ti = Tact1 TO Tactn DO
        IF WS(Tj) ∩ RS(Ti) ≠ ∅ THEN
                VALID := FALSE;
IF VALID THEN COMMIT
        ELSE ABORT TRANSACTION WITH LOWER PRIORITY;
```

Listing 4.1: Priority based Forward Oriented Optimistic Concurrency Control (FOCC) adapted from [71]; the currently validated transaction T_j is validated against all other active transactions

This procedure is only sufficient for guaranteeing local serializability because the validation order of global transactions can be different on different nodes. The problems of distributed validation are described in detail by Schlageter [178]. One method for guaranteeing global

serializability is to use timestamps and to make sure that the validation of all global transactions is performed in the same order on all participating nodes. When the local execution order on all nodes is identical, it is also the same as the global execution order.
The problem is that this can easily fail in wireless sensor networks due to message loss and differing hop counts. Therefore, we used a short time interval (10 ms) to guarantee the arrival of the begin vote message at all participating nodes of one transaction. Then we sorted the transactions to be validated by their priorities to make sure that the transaction with the highest priority is executed first.
Transactions arriving late (with a smaller timestamp than that of the transaction already validated) are aborted. The same difficulties concerning time synchronization arise as with timestamp ordering.
An additional problem arises due to the time lag of local validation phase and write phase. Since it is not guaranteed that transactions validated successfully locally are also committed (unsure updates), the outcome of transactions reading the unsure updates is undetermined. One possibility to prevent this is the locking of unsure updates, which again leads to a blocking situation analogous to locking and timestamp ordering. Since sensor nodes are severely resource constrained devices, we again decided to abort depending transactions in this case. Our implemented version of FOCC is called validation in the following sections to improve the readability. In the next section we present our evaluation results.
Several optimizations of the fundamental algorithms exist, many in relation to real time databases where validation is more commonly used than locking. Haritsa et al. [74], for example, describe the real-time optimistic concurrency control algorithm WAIT-50. It monitors transaction conflict states and gives precedence to urgent transactions.

4.3 Evaluation

In this section we compare the concurrency control protocols locking, timestamp ordering and validation in simulations performed with the network simulator Shawn [55] with respect to their usability in wireless sensor networks.

4.3.1 Criteria

We compare the implemented protocols with regard to the criteria commit rate, which is the number of committed transactions divided by the number of started transactions and the costs expressed in transmitted bytes per committed transaction. Our goal is to find the most efficient protocol for concurrent transaction processing in wireless sensor networks to prolong their lifetime.

4.3.2 Simulation Setup

We used the network simulator Shawn to create a simulation scenario. In this scenario, we started 1000 transactions with each protocol for comparison matters. The constant parameters used in our simulations with Shawn are shown in Table 4.1.

Simulation Parameter	Value
Width of area (field units)	500 FU
Height of area (field units)	500 FU
Number of nodes	100
Number of simulation runs	20
Number of iterations per run	1030
Number of transactions per run	1000
Average neighborhood size	9.84
Maximum range (field units)	100 FU

Table 4.1: Constant parameters used in the simulations performed with the network simulator Shawn [55]

To vary the loss rate, we simulated the Quasi Unit Disk Graph Model [100] with a maximum range r_{max} of 100 field units and varied the guaranteed minimal range r_{min}. The probability p that a node at a distance of d receives a message is 1 if $d < r_{min}$, 0 if $d > r_{max}$ and decreases linearly from 1 to 0 if $r_{min} \leq d \leq r_{max}$. We also varied the number of participants of each transaction and the ratio between read-only and writing transactions. We repeated every experiment 20 times and report the averaged values with 95% confidence intervals in the next subsection.

4.3.3 Results

The outcome of our protocol comparison is shown in Figure 4.1 respective Figure 4.2. The diagrams compare the commit rates at a guaranteed range of 100 field units (this means without message loss) in the upper diagram and a guaranteed range of 10 field units (this means with about 63.25% message loss) in the lower diagram. We simulated all three protocols with 10 participants and varied the percentage of write transactions from 0%, which means exclusively read-only transactions were performed, to 100%, which means every transaction performed a write operation.
No message loss and exclusively read-only transactions and hence no conflicts is the ideal scenario, which is shown in the upper diagram. When performed with 0% write transaction, all three protocols committed over 99% of the started transactions. The missing 1% of the transactions have been aborted due to timeouts caused by a partly partitioned network in one out of our 20 simulation repetitions. When the percentage of write transactions is increased also the probability of a conflict between two transactions increases and consequently, the commit rate decreases. It can be seen that validation is superior when simulated

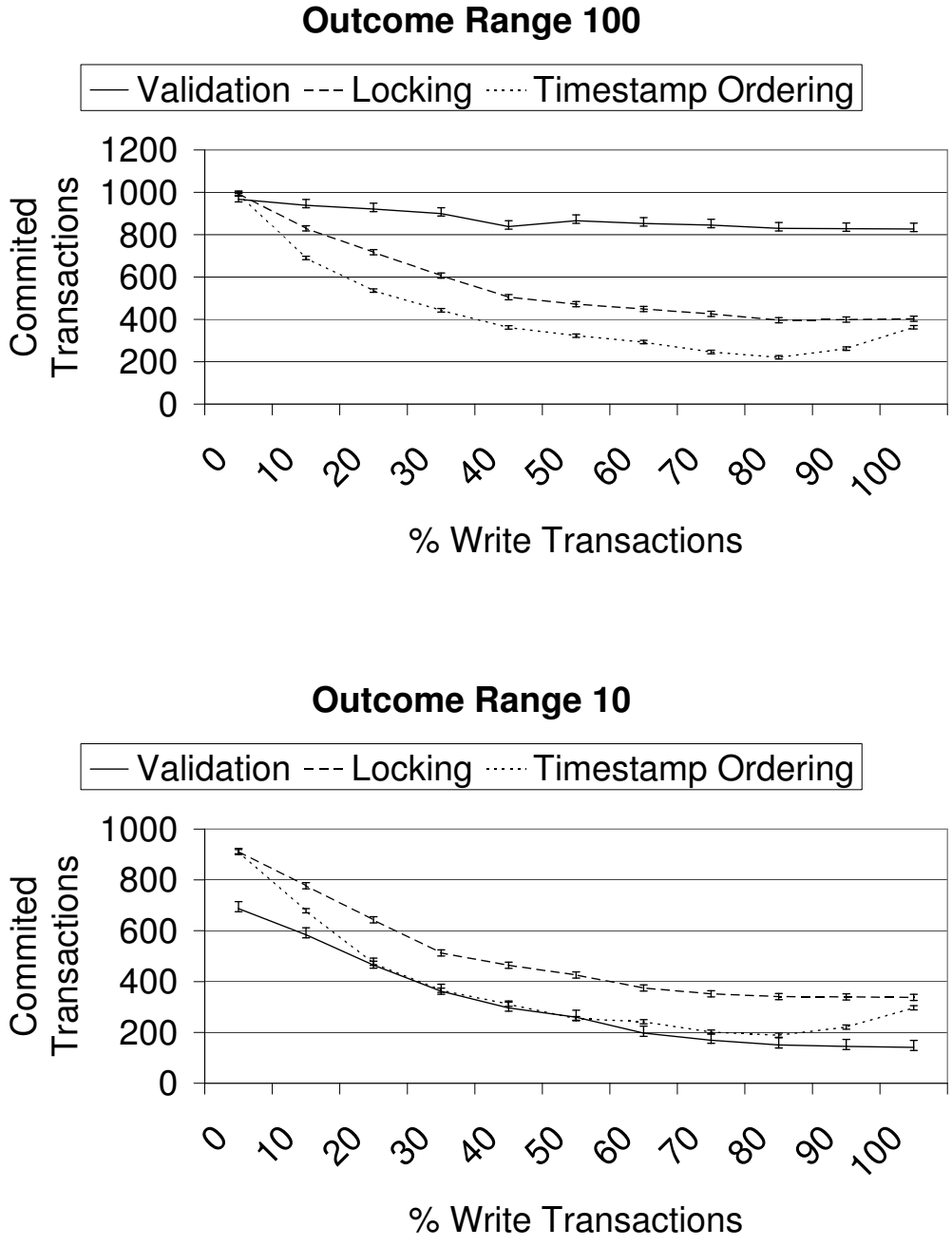


Figure 4.1: Experimental results from comparing the implemented concurrency control protocols with 10 participants

without message loss, while locking has the second highest commit rate and timestamp ordering performs worst. With message loss, there is a tremendous drop of the commit rate and locking becomes superior.
The same can be said about the costs. The results are shown in Figure 4.2. While validation is most efficient without message loss, locking is most efficient with a guaranteed range of 10 field units.
The averaged commit rates and costs are shown in Table 4.2. Figure 4.3 shows the outcome of all three different protocols at different loss rates. Without message loss, the average commit rate of validation was 87%, locking 56% and timestamp ordering 43%. With the guaranteed ranged reduced to 10 field units, the average commit rate of validation was only 31%, locking 50% and timestamp ordering 38%. So while timestamp ordering and locking lose each about 10%, validation loses more than 50% of the commit rate. This is because the collection of begin vote message and sorting by priorities does not perform well if message loss occurs.

Protocol	**Commit Rate**		**Costs in Bytes**	
	No Loss	Lossy	No Loss	Lossy
r_{min}	100	10	100	10
Validation	87%	31%	29907	82449
Locking	56%	50%	52685	61676
Timestamp Ordering	43%	38%	78196	92705

Table 4.2: Commit rates and costs averaged over all write ratios

We used a relatively fast timing and the simulation used perfectly synchronized clocks as well as immediate message delivery, both of which are advantages for the timestamp ordering and validation. Nevertheless, locking outperformed the other protocols when simulated with message loss. Since message loss is typical for wireless sensor networks, we consider locking the best concurrency control protocol for this environment. The added advantage of locking is that no work is done in vain: a significant advantage because it further extends the lifetime of sensor nodes.

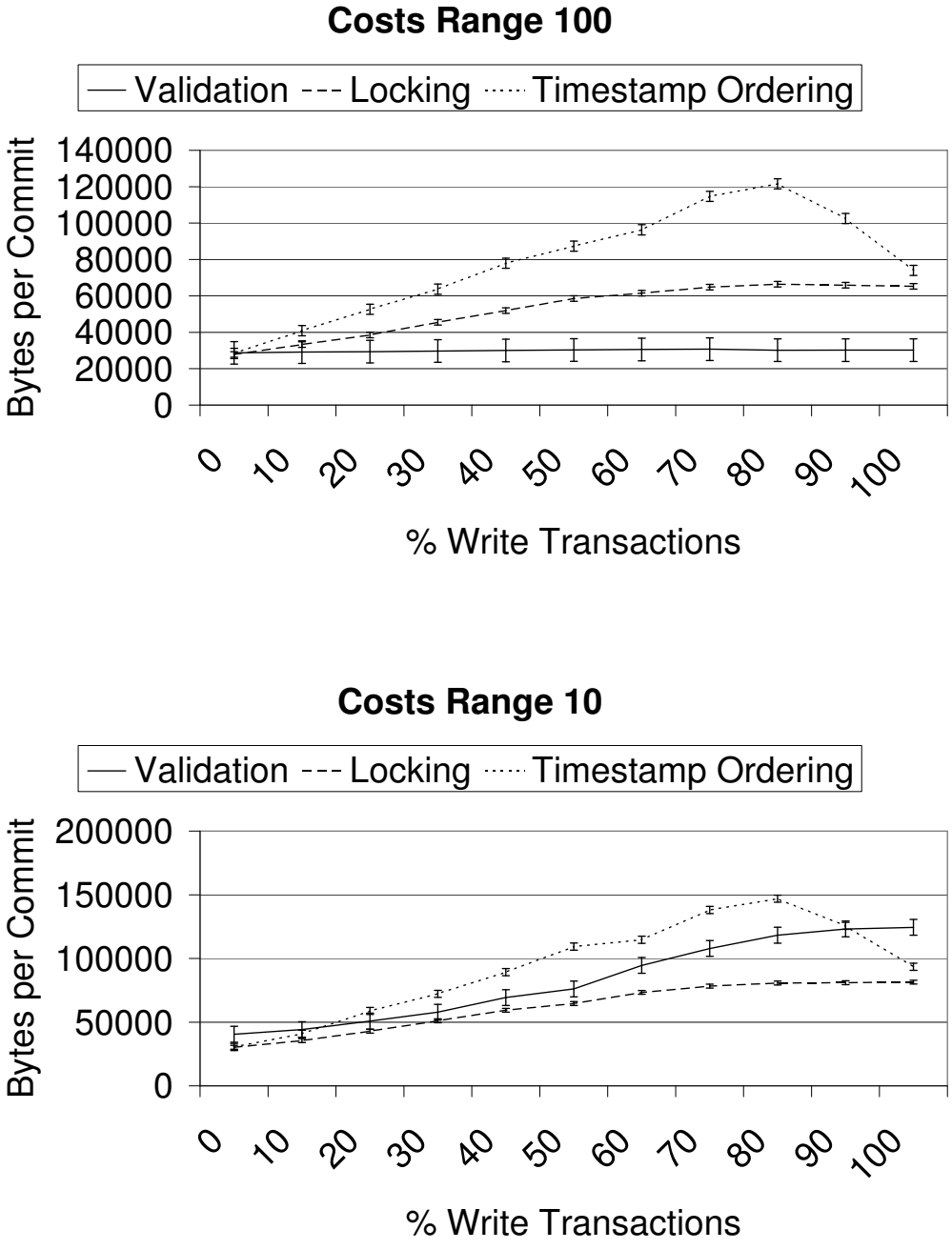


Figure 4.2: Experimental results from comparing the implemented concurrency control protocols with 10 participants

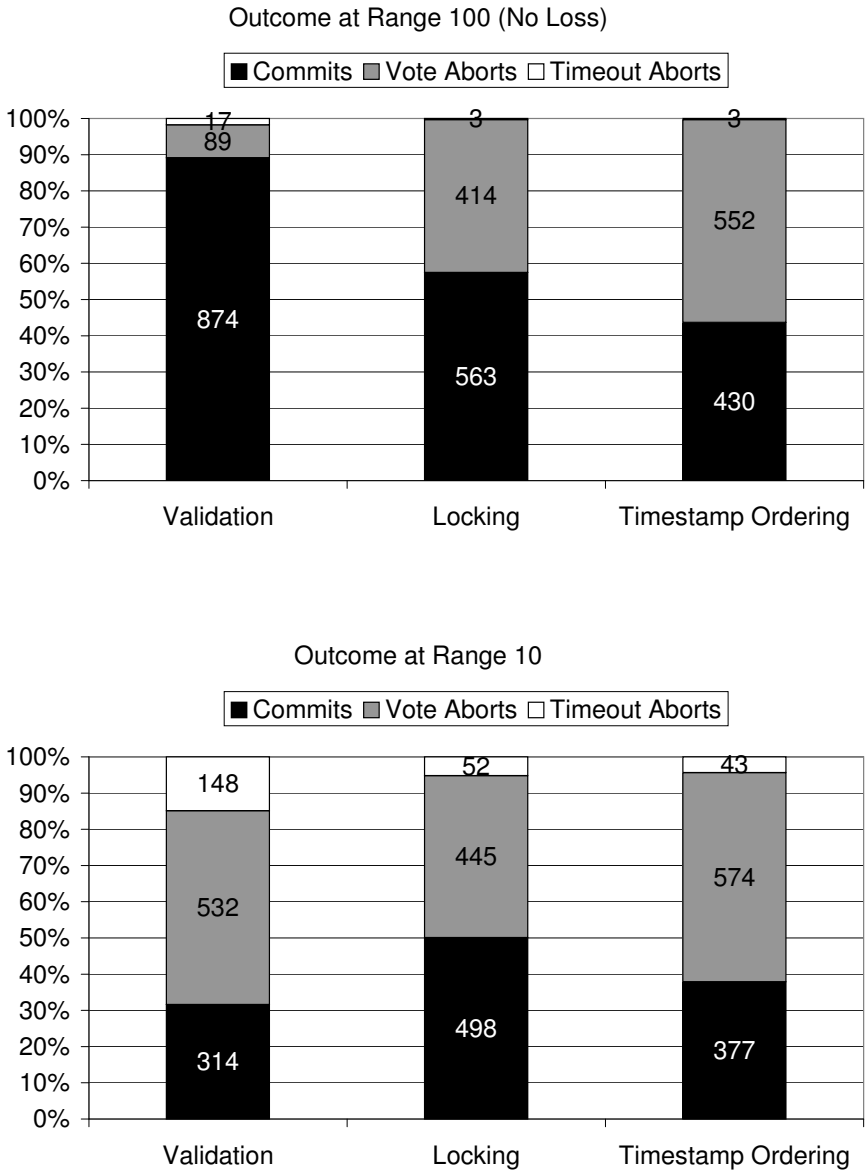


Figure 4.3: Outcome averaged over all write ratios

4.3.4 Memory Demands

The memory demands of the implemented concurrency control protocols for the network simulator Shawn including the simulation scenario are shown in Table 4.3. The whole code is comprised of 3320 statements. Since each of the concurrency control protocols is integrated in 2PC, the code has roughly the same size, which is 513 statements for locking, 566 statements for timestamp ordering and 603 statements for validation. Locking is obviously the protocol which consumes the least resources.

Code	Statements
Resource management	609
Validation	603
Message forwarding	587
Timestamp Ordering	566
Locking	513
Simulation scenario	442

Table 4.3: Code size of the implemented concurrency control protocols, the whole code contains 3320 statements

4.3.5 Experiments with Real Sensor Nodes

Simulations give only limited insight into the real world behavior of algorithms due to their simplifications of reality. Oftentimes a flat world is simulated, resource consumptions are not taken into account or one of many other potential simplifications is calculated. Our simplified simulation environment is an advantage for timestamp ordering and validation because we provided perfect time synchronization, which is not available in real wireless sensor networks. Nevertheless, the results show the advantage of locking and consequently, we did only implement locking for real sensor nodes.

We implemented two phase locking integrated in 2PC for the sensor node platform Pacemate [122], to prove the feasibility of the protocol. With a Philips LPC 2136 processor, 32 kByte RAM and 256 kByte flash ROM it has roughly the same resources as common sensor nodes, like Mica2 nodes, but offers a display and buttons for easier debugging of applications. In the experiments performed with the Pacemate sensor nodes, we gained commit rates similar to the simulation results. There are several ways to improve the commit rate. On the one hand, the acknowledgement of messages can be used and messages can be sent until they are finally received. This is a problem in the presence of node failures, since messages are often sent infinitely because node failures cannot be detected. Another way of improving the commit rate is the usage of more sophisticated commit protocols than 2PC.

We recommend the usage of our variant of 2PC called Two Phase Commit with Caching (2PCwC) which has been introduced in the last chapter. In 2PCwC, participants of a trans-

action cache messages from other participants to be able to reply in place of them if their messages get lost. We have shown in experiments with 20 Pacemate sensor nodes that our protocol can increase the commit rate from 53% to 84% and also significantly lowers the costs. In large sensor networks containing more than about 200 nodes, one should abstain from flooding and use routing protocols with better scalability properties like georouting [169].

4.4 Results

In this chapter we outlined our implementations of traditional concurrency control protocols for resource constrained wireless sensor networks. We have shown that traditional locking is superior to timestamp ordering and validation when considering the commit rate and the efficiency. Beyond that, locking does not require time synchronization like timestamp ordering and validation. The additional advantage of locking is that no work is done in vain in the highly resource constrained sensor networks, which saves energy. We also implemented locking integrated in 2PC for the sensor node platform Pacemate to validate the feasibility of the protocol for resource constrained wireless sensor networks.

Chapter 5

Service Migration

In recent years, service oriented architectures have made their way from business applications to wireless sensor networks (see for example [105, 129, 130]). While the advantage of service oriented architectures is mainly a greater flexibility, they also allow non computer scientists easy development of applications for wireless sensor networks by offering a graphical user interface that can be used to combine simple services to complex services as implemented in the service oriented operating system Surfer OS [124].
One example of the increased flexibility offered by service orientation is the concept of service migration. It allows to deploy a general network running Surfer OS without assigning a task to the network yet and migrating later services into the network to perform their tasks. It is also possible to migrate services including their states from nodes about to run out of energy to other nodes.
Service migration has been implemented and evaluated several times, for example using the Trickle algorithm described by Levis et al. [113, 114], and also in our working group for Surfer OS, where a migration of a service including its state is already possible [124].
What is still missing is the consideration of the consistency of the service migration. If a service is migrated only with eventual consistency like in Trickle, there is a certain period called the inconsistency window during which inconsistencies can arise. These inconsistencies can lead to undesired application behavior, like alarms which are not triggered or results of long running measurements getting lost.
Our approach is to use atomic commit protocols to achieve strict consistency when migrating a service from one node to another. In particular, we implemented service migration with strict consistency using either the Two Phase Commit (2PC) protocol [63] or our own atomic commit protocol Two Phase Commit with Caching (2PCwC) [168], which exploits broadcast communication to increase the commit rate in the presence of message loss.
Special attention must be paid to events happening during the migration or messages sent during the migration of a service. This is because migrating and installing a new service in the operating system takes time, for example for writing the code to the flash. During these operations, the node receiving the new service cannot yet receive messages intended for the

new service. Our approach is to cache these messages and / or events by another node in the proximity, and transfer these to the destination as soon as the migration is finished.

Organization

The remainder of this chapter is structured as follows:

- An overview of related work in the areas of service discovery and service migration in wireless sensor networks is given in Section 5.1.
- We describe our consistent service migration in the context of a complete service discovery scenario for wireless sensor networks in Section 5.2. We take into account the transactional reconfiguration of data paths and buffer messages and events during the service migration. The strict consistency of the service migration is implemented by using atomic commit protocols.
- In Section 5.3, we describe our implementation for the network simulator Shawn [55], where the commit protocols used for service migration also incorporate gossiping [70] to save transmission costs. We also outline our implementation of service migration for the sensor node platform Pacemate [122] running the service oriented operating system Surfer OS [124].
- In Section 5.4, we compare our transactional service migration using different atomic commit protocols, either 2PC or 2PCwC, each implemented using either flooding or gossiping, with service migration guaranteeing only eventual consistency utilizing Trickle. We show that transactional service migration offers a higher degree of consistency since 100% to 30% fewer messages are lost during migration. We also show that our approach is feasible in practice by providing results obtained from experiments with Pacemate sensor nodes.
- We conclude this chapter in Section 5.5.

5.1 Related Work

In this section, we briefly sketch related work in the areas of service discovery and service migration in wireless sensor networks and mobile ad hoc networks.

5.1.1 Service Discovery

Service discovery in wireless sensor networks can be done in many different ways, depending on the characteristics of the particular deployment in terms of size, mobility, heterogeneity and other aspects.

Mian et al. [132] give a survey of service discovery protocols in multi hop mobile ad hoc networks. The authors discuss service description, registration, discovery, routing and also aspects of mobility support. Figure 5.1 shows the design space of service discovery protocols with the parameters network size and mobility. The basic message of Figure 5.1 is that a large number of nodes demands for an infrastructure to enable efficient service discovery, while high mobility makes it hard to use an infrastructure. Considering a moderate network size (100 to 200 nodes) and limited mobility, our scenario corresponds to a directory based one without the support of an overlay network (framed in Figure 5.1).

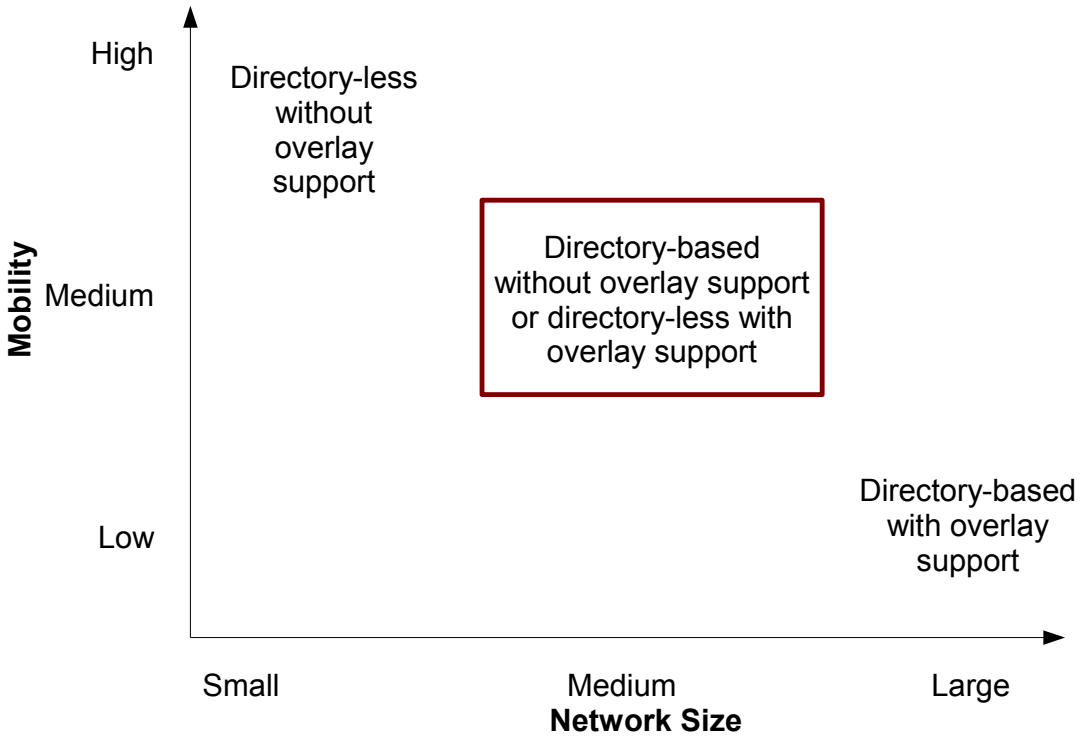


Figure 5.1: Design space of service discovery protocols with the parameters network size and mobility adapted from Mian et al. [132]

Several approaches for service discovery in wireless sensor networks or mobile ad hoc networks exist, for example by Marin-Perianu et al. [130], Li et al. [115] and Sailhan et al. [176], but none of the referenced approaches considers service migration or even consistent service migration.

A trajectory based discovery protocol for convex wireless sensor networks is described by Kaur et al. [97] for TinyOS.

Costa et al. [39] adapt Jini to wireless sensor networks. They develop an own Java Virtual Machine (JVM) and realize Remote Method Invocation (RMI) through Transmission Control Protocol (TCP).

Schiele et al. [177] describe energy efficient service discovery for ubiquitous devices. The idea is to use clustering so that the nodes other than the cluster heads can sleep, which can save 66% of the power. The evaluation parameters are not appropriate for wireless sensor

networks, since a bandwidth from 1 mbps to 10 mbps and message sizes of 1 kilobyte to 20 kilobytes are used in simulations with ns2.
Zhou et al. [210] compare the Service Location Protocol (SLP), Jini, Bluetooth's Service Discovery Protocol (SDP), Salutation, and Universal Plug and Play (UPnP) and state that they are not applicable for wireless sensor networks since they do not consider the specific requirements like large number of nodes.
A taxonomy of self-configuring service discovery systems is introduced by Sundramoorthy et al. [190].

5.1.2 Service Migration

Trickle

Levis et al. [113, 114] propose Trickle, an algorithm for propagating and maintaining code updates in wireless sensor networks. Trickle uses polite gossiping techniques where motes periodically broadcast code summaries to local neighbors but do not broadcast if they have recently heard a broadcast of their own code summary. Table 5.1 shows the parameters and variables used by Trickle, and Trickle's pseudo code is shown in Table 5.2.

τ	Communication interval length
t	Timer value in range $[\frac{\tau}{2}, \tau]$
c	Communication counter
k	Redundancy constant
τ_l	Smallest τ
τ_h	Largest τ

Table 5.1: Trickle parameters and variables (from [114])

τ expires	Double τ, up to τ_h. Reset c, pick a new t.
t expires	If $c < k$, transmit.
Receive consistent data	Increment c.
Receive inconsistent data	Set τ to τ_l. Reset c, pick a new t.

t is picked from the range $[\frac{\tau}{2}, \tau]$

Table 5.2: Trickle pseudo code (from [114])

While Trickle has been proven to be efficient as scale with the density of the network, only eventual consistency is provided. Since the Trickle algorithm evolved to a basic networking primitive in wireless sensor networks, we compare it with our approach in Section 5.4. The description of the determination of Trickle's parameters for our application scenario can be found in Appendix A.

Other Approaches

An overview of several reprogramming approaches has been published by Wang et al. [199]. Consistency demands are not considered explicitly.
Yu et al. describe Melete [207], which works like Trickle but uses dissemination restricted to a defined hop count to limit the flood.
Deluge, described by [88], extends Trickle by allowing the dissemination of code with bigger size by splitting messages and using pipelined dissemination.
Lee et al. [109] propose the use of the Electrically Erasable Programmable Read-Only Memory(EEPROM) of a sensor node when disseminating new code, but also do not consider consistency demands.
Riva et al. [170] describe the implementation of mobile services which can migrate from node to node to accomplish their tasks. The application scenario is migration from vehicle to vehicle, but without the use of service discovery or service consumers.
Fok et al. [56] describe Agilla, an adaptive middleware approach for wireless sensor networks. It uses the cloning of services, but without consideration of consistency guarantees.
Sommer et al. [188] describe different scenarios for service migration. The general application scenario is a network of various devices building a smart home. They state a need for consistent updating of data paths in a transactional way, but the authors do not give details about an implementation of consistent service migration.
We describe our approach for consistent and stateful service migration in the next section.

5.2 Consistent Service Migration

In this section, we first describe our service discovery scenario and state our assumptions. Then we compare consistency demands of different services and describe our approach for consistent service migration using message buffering.

5.2.1 Service Discovery Scenario

We select a service discovery approach which uses a non structured and non overlay network with service directories. Figure 5.2 shows the interaction of nodes performing different roles in our service oriented sensor network.
Instead of using only one service directory as in the traditional service discovery triangle, our publish-subscribe approach uses several service directories to achieve better scalability in service discovery and also fault tolerance. The service directories maintain mappings of service to the nodes that run these services. In Figure 5.2, Service X is located on Service Provider A, for example.
If a requester intends to use Service X, it can get its location by querying one of the service directories, which deliver the location. In our example in Figure 5.2, Service X running on Service Provider A is then bound at runtime to the requester.

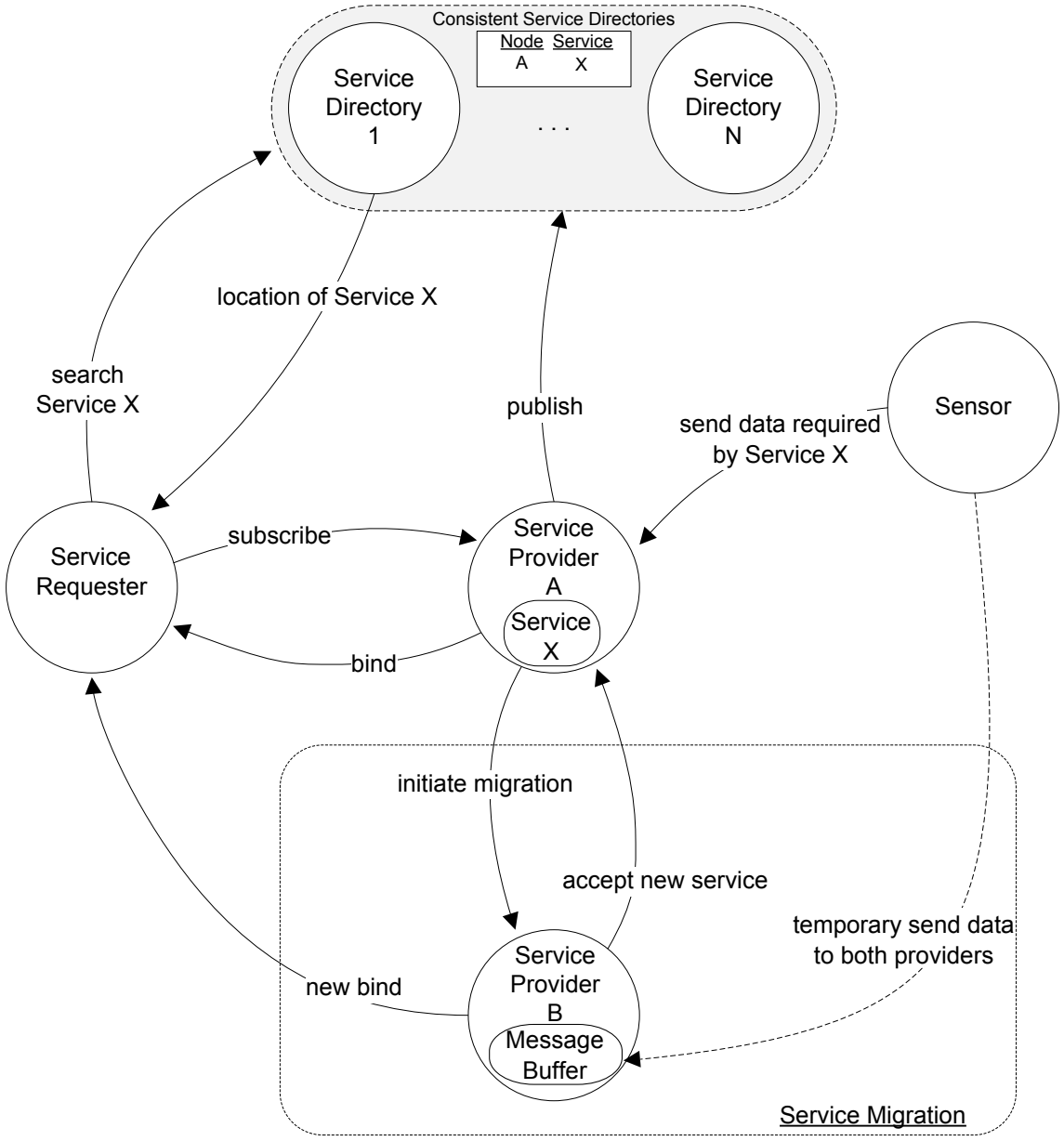


Figure 5.2: We consider different roles in our service oriented sensor network: Service directories, service requesters, service providers and sensors. Directories are used by requesters to locate services running on providers. Providers receive data streams from sensors, process these streams and offer services.

5.2.2 Service Migration

To clarify our service migration scenario, we assume that Service Provider A runs out of energy. To allow the whole sensor network to continue its task, Service X is migrated to Service Provider B. To keep the information maintained on the service directories consistent, this has to be done in the form of an atomic transaction.
This transaction $T_{migrate\_Service}$ contains the following three steps:

1. Replicate Service X including its state from Service Provider A to Service Provider B (i.e. copy it physically).
2. Delete Service X on Service Provider A.
3. Update the mapping on the service directories, so that nodes using this service can find its new location.

In the following, we differentiate between simple service migration, where code is moved or copied from one node to another without any consistency guarantees, and consistent service migration, where we assume that either the service is successfully migrated or no sub steps take place (copy the service to the new node, delete the service on the old node, update the service directory). Our approach is to use atomic commit protocols to achieve strict consistency when migrating a service from one node to another. Atomicity is not only needed for the consistent maintenance of several service directories, but also if the services to be migrated have a status that has to be kept consistent. We clarify this in the next subsection.

5.2.3 Consistency

We consider consistency in two aspects: service discovery and the state of the migrated service.

Consistency of the Service Discovery Process

Since any of the existing service directories can be used by a requester for locating a service provider, it is necessary to maintain a strict consistent mapping of services to their nodes on each service directory. We use the term strict consistency with the meaning of one copy serializability in the database area, which means that the replicated database system behaves like a traditional database system consisting of one copy as far as the perception of the user is concerned [17]. If the service directories can only guarantee eventual consistency, it can happen that service discovery fails repeatedly because false locations are delivered to the requesters. Figure 5.3 shows an example, where messages get lost due to migration with only eventual consistency using Trickle.

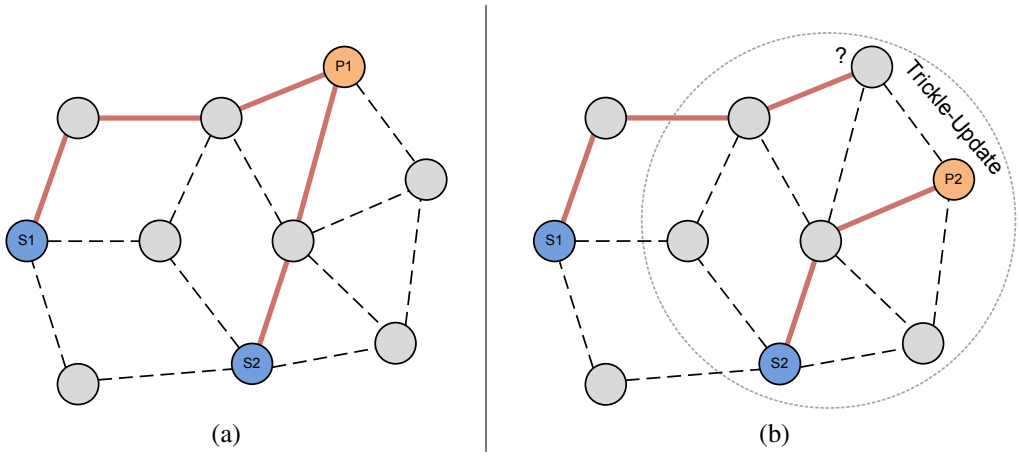


Figure 5.3: At first, S1 and S2 send their data to P1. After the migration, both should send to P2, but since S1 has not received the update, it still sends to P1.

Consistency of the Migrated Service

First we distinguish between stateless and stateful services. Traditionally, simple stateless services are migrated with eventual consistency, so no transaction concept is needed. Examples for simple services are queries or plain functions like data converters or logic operators like described in [188]. The migration of these services can be done in an efficient way via Trickle [113, 114], since these services do not even require any state.
More complex services like an alarm service or a data aggregation service can have a defined state. When the service migrates, the state of the service also has to be transferred in a transactional way, for example with the Two Phase Commit (2PC) protocol. Otherwise, a false alarm could be triggered in case of an alarm service, or no alarm could be triggered when needed, which can have severe consequences.
Consider a long term deployment where a large number of sensor nodes have been dropped over a huge forest and no gateway is present. In this case, consistent service migration could be used to transfer a service, which has already aggregated important data items over a long period of time, from a node with a nearly depleted battery to a fresh node.
Table 5.3 shows a comparison of services needing strict or eventual consistency when being migrated. The table shows that for the consistent migration of stateful services, additionally to the mere migration of the code, also messages sent and events occurring during the migration have to be buffered and data paths have to be reconfigured in a transactional way. We explain the buffering of messages during a service migration in the following subsection.

5.2.4 Message Buffering

A simple example of a service migration considering the data paths is also shown in Figure 5.2. When Service X is migrated from Service Provider A to Service Provider B, the

	Strict Consistency	Eventual Consistency
Actions	-Transactional migration of code -Buffering of messages and events during migration -Transactional reconfiguration of data paths	Replication of Code
Examples	Alarm service Data aggregation ...	Data converter Logic operator ...

Table 5.3: Consistency Demands of Different Services

sensor needs to start sending its data temporarily to both Service Providers, A and B. When the migration is done and the status of Service Provider A also has been transfered to Service Provider B, the data messages buffered during the migration by node B have to be processed and the status of Service X on Service Provider B must be updated before the requester can get a new bind to Service Provider B. A feasible implementation of transaction processing capabilities is the prerequisite for this.

5.2.5 Cost Function for Service Destination

One open question is how to select the appropriate destination when migrating a service to a new node. For the experiments executed for this chapter, we selected a random node in the proximity of the source of the service to be migrated. A more sophisticated approach would be to take the remaining battery capacity, available RAM and other characteristics into account.

5.2.6 Outlook on Adaptivity

Based on the characteristics of a particular deployment of wireless sensor networks, and especially to the consistency demands of the service to be migrated, the best protocol can be selected at runtime. One could start with looking at the consistency demand of the service to be migrated and then select Trickle or transactional migration. If transactional migration was chosen, the next step would be to choose the appropriate commit protocol for example depending on the number of participants or the current message loss. The adaptive selection of commit protocol in combination with routing protocol has been outlined in [167], Chapter 6 contains a comprehensive report.

5.3 Implementation

The implementation of our transactional service migration approach is outlined in this section. On the one hand, we implemented our solution for the network simulator Shawn [55] to be able to evaluate the behavior of our protocol with a large number of nodes (100 to 200). On the other hand, we implemented a part of our solution for the sensor node platform Pacemate [122] running the service oriented operating system Surfer OS [124] to validate the feasibility of our approach.

5.3.1 Shawn Network Simulator

We implemented 2PC and 2PCwC for the network simulator Shawn [55]. Both protocols were implemented using either flooding, gossiping [70] or georouting [96] as shown in Figure 5.4.

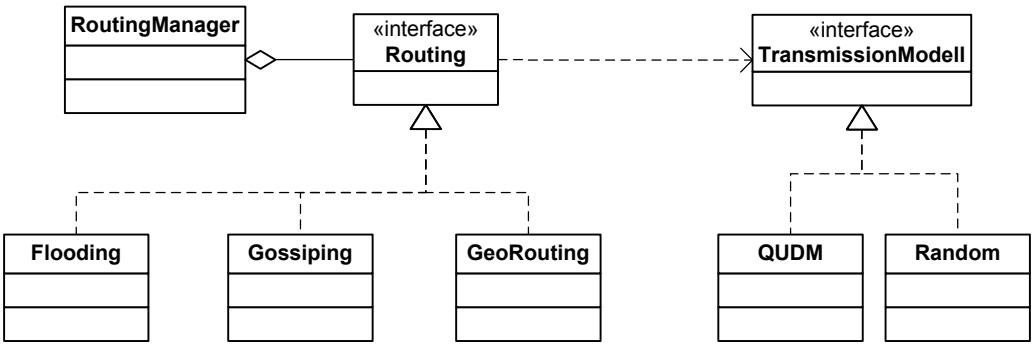


Figure 5.4: Our implemented routing protocols and transmission models in addition to Shawn

We also implemented the Trickle algorithm [113, 114] (see Section 5.1.2). The determination of Trickle's parameter for our application scenario is described in Appendix A.
Gossiping is a technique that can be used in dense networks to alleviate the broadcast storm problem outlined in Section 2.2.3. It means that, in contrast to flooding, each node forwards a message only with some probability, to reduce the overhead of the routing protocols. In summary, we implemented the following protocol combinations:

- 2PC using flooding
- 2PC using gossiping
- 2PCwC using flooding
- 2PCwC using gossiping
- Trickle

Additionally, we implemented the services needed for our service discovery scenario described in Figure 5.2, namely a Service Directory, Service Provider, Sensor, Service Requester and Service Manager.
Listing 5.1 shows a part of the interface of the `Service`-class. The interface offers methods for serializing and de-serializing a service, and also for starting, stopping and destroying the service. The method `fillDependentTo` is used to define other dependent services which need to be participants of a possible transactional migration.

```
1  class Service {
2  public:
3          [...]
4          virtual void serialize(uint8* state, uint8 size) = 0;
5          virtual void deserialize(const uint8* state, uint8 size)
               = 0;
6          virtual uint8 getSerializedSize() = 0;
7          virtual void start() = 0;
8          virtual void stop() = 0;
9          virtual void destroy() = 0;
10         virtual void fillDependentTo(isense::stl::list<uint16>&
               list) = 0;
11         [...]
12 };
```

Listing 5.1: Part of the service interface

Figure 5.5 shows the code size of our implemented system. It can be seen that the commit protocols and the routing algorithms are the lion's share of the implemented scenario.

5.3.2 Pacemate Sensor Nodes

To validate our approach on real sensor nodes, we used the sensor node platform Pacemate [122] running the service oriented operating system Surfer OS [124]. Basically, we implemented the atomic commit protocols 2PC and 2PCwC (in the services Coordinator and Participant), an aggregation service to be migrated and a service for buffering messages during the migration of the aggregation service. We omitted the implementation of service directories for now and focused on the migration in a one hop neighborhood. A summary of the implemented services for a scenario of four nodes is shown in Table 5.4.

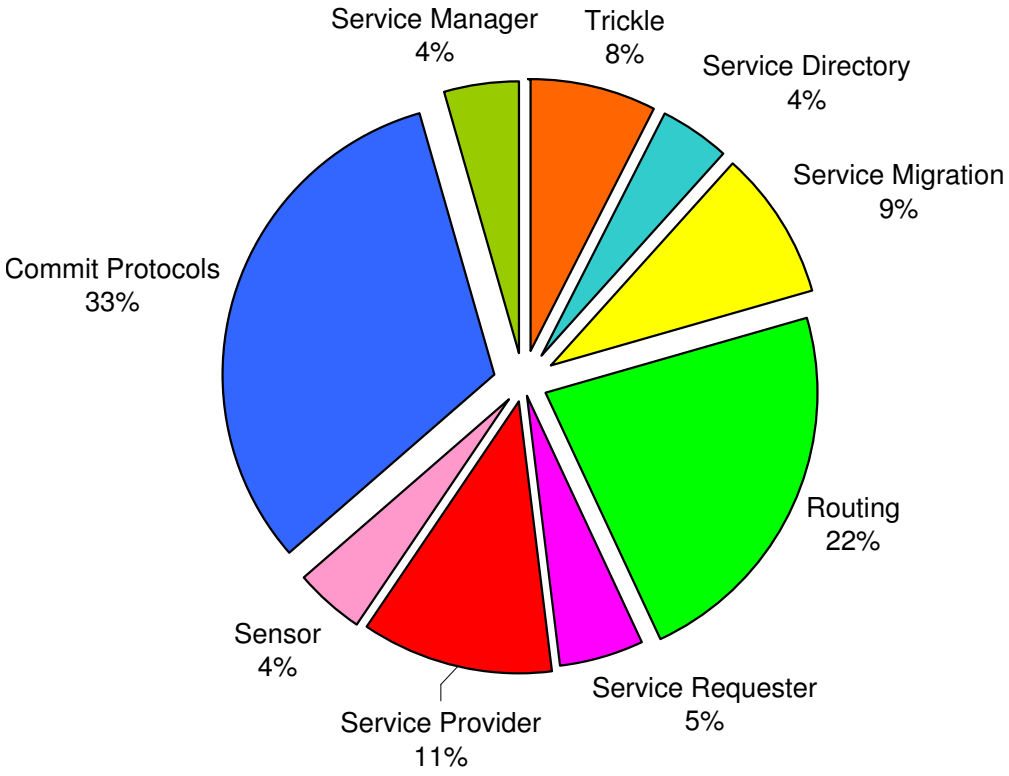


Figure 5.5: The code size of our service discovery and migration scenario implemented for the network simulator Shawn [55]; the entire code contains 2291 statements (semicolons)

Node (Role)	Service	Size in bytes	
		2PCwC	2PC
Node A	Coordinator	6560	5692
(Old Service	Migration	2072	2100
Provider)	Aggregation	1488	1488
		10120	9280
Node B	Participant	7836	5832
(Sensor)	Data Stream	1448	1448
	Vote Handling	1804	1804
		11088	9084
Node C	Participant	7836	5832
(Message Buffer)	Vote Handling	1804	1804
	Message Buffer	2036	2276
		11676	9912
Node D	Participant	7836	5832
(New Service	Aggregation	1488	1488
Provider)		9324	7320

Table 5.4: Implemented services for the operating system Surfer OS [124]

5.4 Evaluation

In this section we first outline the used evaluation criteria. Then we describe the simulations with the network simulator Shawn and the experiments we performed with the Pacemate sensor node platform and report our results.

5.4.1 Criteria

The criteria chosen for comparison of the implemented protocols are the costs in transmitted bytes per service migration and the degree of consistency in number of missed messages. In the simulation scenarios, we basically executed a number of service migrations and compared the costs. The scenarios are explained in more detail in the following subsections.

5.4.2 Simulations

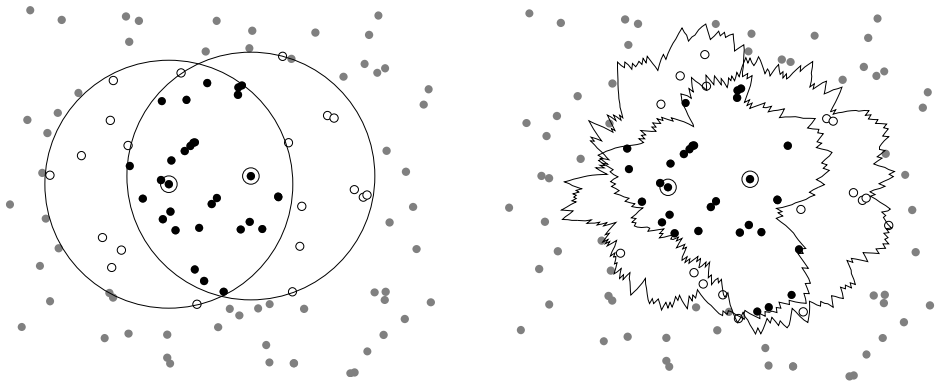
We describe the experimental setup for Shawn and the results of our simulations in the following subsections.

Experimental Setup

To perform the simulations, we used Shawn to create a scenario according to Figure 5.2 and initialized it as follows. Before the start of the simulation, the service providers were attached to nodes, and to sensor services. The service directories were started with consistent views of the deployed services. The requester node only has a certain intention which service it wants to find, but has to do service discovery first.
When the simulation started, the sensors sent their data streams to the service providers every five seconds. The service discovery itself is not embedded in a transaction. That means the service directories are not aware of bindings. The service providers sent the average of the last five received sensor values every five seconds to the requesters. The provider service was then migrated to a new destination chosen in the neighborhood. Between two migrations there is a pause of 2.5 seconds, which leads to a migration of each service every 12.5 seconds. Each service migration had nine to ten transaction participants: five service directories, two sensors, two providers and in some cases a service requester. The constant parameters used in our simulations are shown in Table 5.5.
We also used different communication models (Unit Disk Graph [100] and Radio Irregularity Model [209]) and either used a perfect MAC layer and varied the message loss from 0% to 60%, or simulated a Carrier Sense Multiple Access / Collision Avoidance (CSMA/CA) Medium Access Control (MAC) layer with the standard parameters of Shawn [55]. We repeated the experiments 20 times. A comparison of the disk communication model and the Radio Irregularity Model (RIM) is shown in Figure 5.6. RIM is more realistic and also yields unidirectional links.

General	Width of area (field units)	500 FU
	Height of area (field units)	500 FU
	Number of nodes	100
	Number of simulation runs	20
	Number of iterations per run	1000
	Sum of migrations over all runs	7486 - 7860
	Average neighborhood size	9.84
	Maximum range (field units)	100 FU
Services	Number of service directories	5
	Number of service requesters	3
	Number of service providers	5
	Number of sensor per provider	2 (=10 in total)
Trickle [113, 114]	τ_{low}	100 ms
(see Appendix A)	τ_{high}	60000 ms
	Redundancy constant k	6
Gossiping [70]	Hops for sure broadcast	4
	Broadcast probability 1	0.6
	Broadcast probability 2	1.0
	Neighbor threshold	6

Table 5.5: Constant parameters used in the simulations performed with the network simulator Shawn [55]



(a) Unit Disk Graph ("Disk Model") (b) Radio Irregularity Model(RIM)

Figure 5.6: Comparison of communication models (from [157])

Results

In the following, we compare the implemented service migration protocols considering achieved consistency and costs. The consistency is expressed as number of missed messages and the costs are expressed as the number of transmitted bytes per service migration, respectively per transaction commit.
Figure 5.7 shows the number of missed messages per protocol. It can be seen that the transactional service migration performed with the Two Phase Commit (2PC) or the Two Phase Commit with Caching (2PCwC) protocol leads to a lower number of missed messages, and therefore to a higher degree of data consistency. Depending on the loss rate, the number of missed messages using transactional service migration is only 0% to 77% compared to the message loss using Trickle. The gossiping variants missed a high number of messages at a loss rate of more than 40%. Note that the message loss that occurred within the transactional service migration is only caused by the simulated message loss because of interference and not due to the service migration itself. The transactional service migration also has the additional advantage of ensuring strict consistency of the mapping on the service directories. Thus, the discovery of a service is more likely to be successful.
Figure 5.8 shows the number of transmitted bytes per commit. The transactional service migration via an atomic commit protocol is more expensive than Trickle, especially at message loss higher than 40%. But we believe that this is a reasonable effort for increased consistency in many use cases.
We also evaluated all protocols with Shawn's more realistic CSMA/CA MAC layer. The results are shown in Figure 5.9 and Figure 5.10. They support our thesis that the higher degree of consistency achieved by the use of an atomic commit protocol is affordable in many use cases.
Gossiping improved the consistency and the costs only in parts of the experiments, in particular when simulating CSMA/CA on the Unit Disk Model like shown in Figure 5.9 and Figure 5.10. This is due to the higher average number of neighbors in the experiments with the Unit Disk Model compared to the experiments using the RIM. The behavior of gossiping is analyzed in more detail in the next chapter.

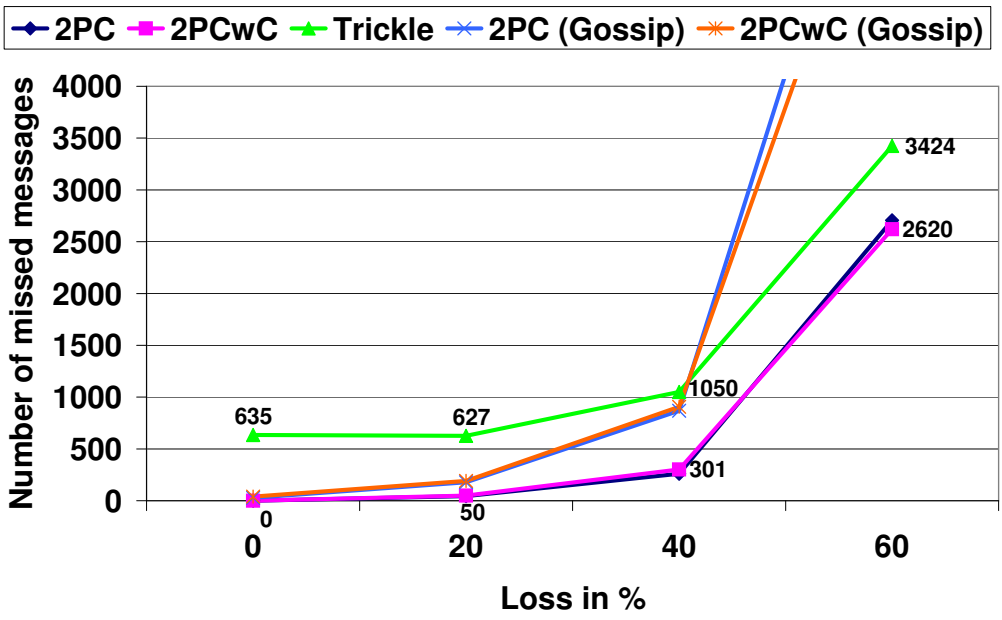


Figure 5.7: The transactional service migration performed with the Two Phase Commit (2PC) or the Two Phase Commit with Caching (2PCwC) protocol leads to a lower number of missed messages and therefore to a higher degree of data consistency (QUDM)

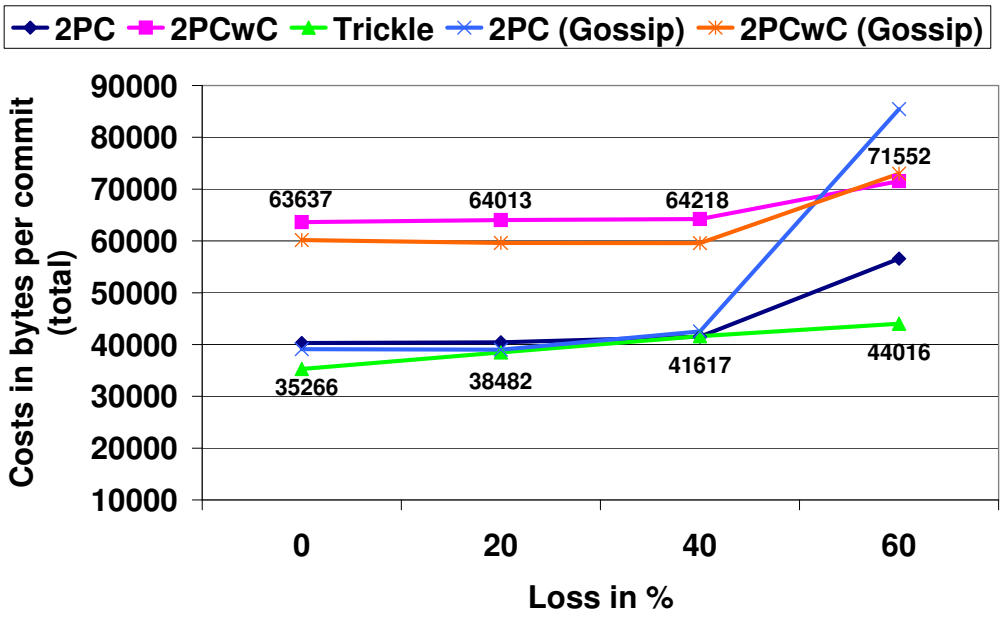


Figure 5.8: The transactional service migration via an atomic commit protocol is more expensive than Trickle, especially at message loss higher than 40% (QUDM)

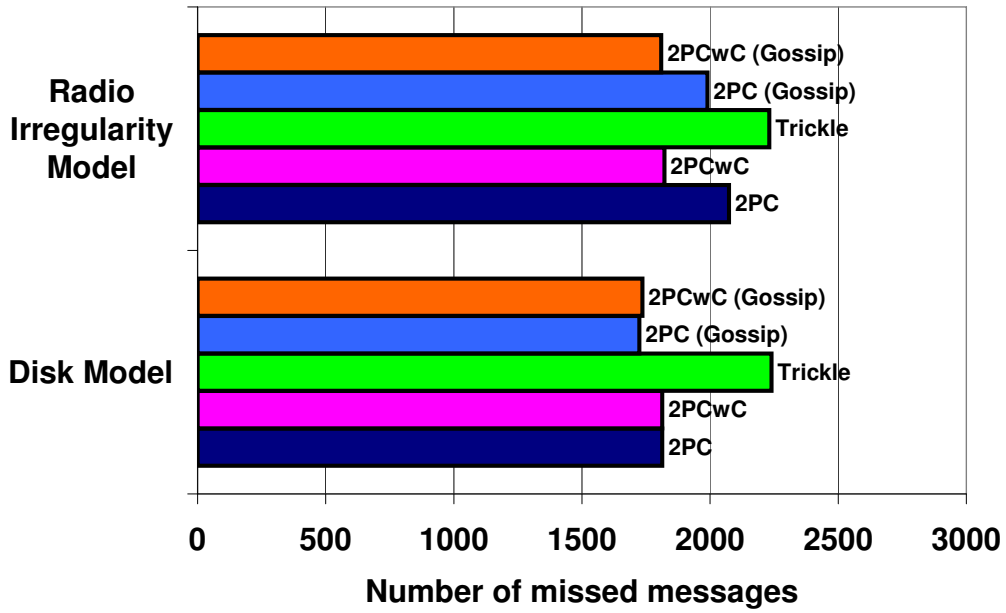


Figure 5.9: The simulation of all protocols over a CSMA/CA layer confirms the results with the random transmission model

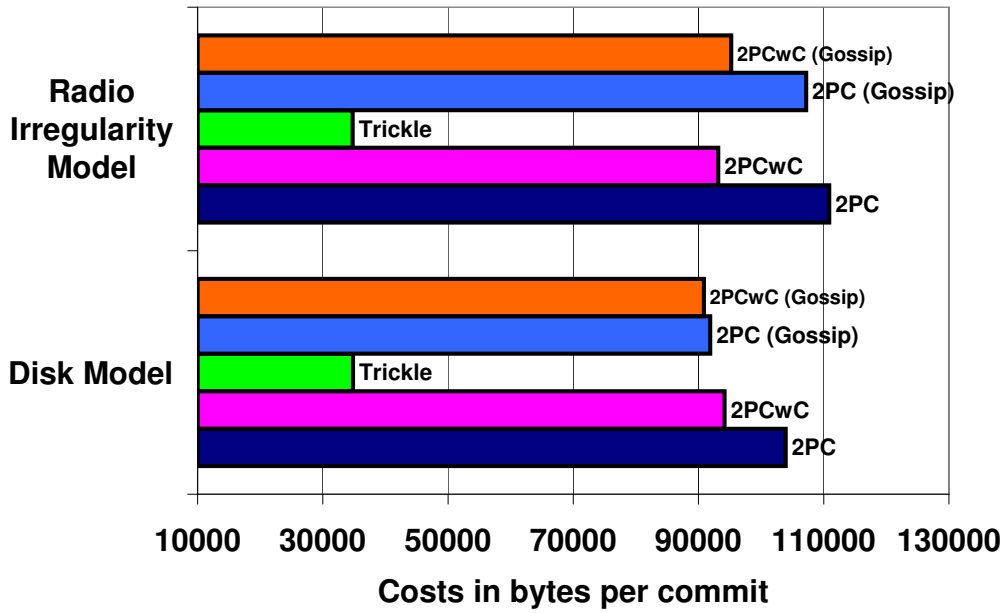


Figure 5.10: The simulation of all protocols over a CSMA/CA layer shows also a higher overhead of transactional migration

5.4.3 Experiments with Pacemate Sensor Nodes

We also performed experiments with the Pacemate sensor nodes [122] to prove the feasibility of our transactional service migration approach on real nodes.

Experimental Setup

We performed a service migration with four sensor nodes like described in Subsection 5.3.2 as a proof of concept. The nodes were deployed in a one hop distance and 20 service migrations were started with the 2PC and also with the 2PCwC protocol. Figure 5.11 shows the simple deployment of the Pacemate sensor nodes used in our experiments. We measured the time needed for the service migration.

Figure 5.11: The deployment of four Pacemate sensor nodes [122] used for prototyping our consistent service migration technique. The nodes performed the roles A) Old Service Provider B) Sensor C) Message Buffer and D) New Service Provider as described in Subsection 5.3.2.

Experimental Results

In our experiments, 19 of the started 20 migrations were committed when using 2PCwC. When using 2PC, 18 transactions could be committed. The total time needed for the trans-

actional migration of a data aggregation service in our experiments was 11.2 seconds on average for 2PCwC and 13 seconds for 2PC. This time is the sum of the time needed for writing the new service to the flash and the timeouts of the used commit protocols. The timeouts used to check the successful update of the flash memory were 8 seconds for both protocols, so the migration time could probably be reduced to about 3 seconds.
The flash memory of the Pacemate sensor node controlled by the operating system Surfer OS is structured into sectors of different sizes (4 kilobytes to 32 kilobytes) [121]. The node needs 1 ms for writing 256 bytes to the flash and 400 ms for erasing a sector, so writing 32 kilobytes takes $\frac{32\ kilobytes \times 1024}{256bytes} = 128ms$. So theoretically, it then takes 1056 ms to change a 32 kilobytes sector A, performing the following operations: Writing A to B (128 ms), erasing A (400 ms), writing B in a modified way to A (128 ms) and erasing B (400 ms).
In our experiments, the time needed for updating the flash was 1334 ms on average. Since 90% of the migrations performed with 2PC and 95% of the migrations performed with 2PCwC were successful, we claim that consistent service migration in a transactional way is not only feasible in simulations but also applicable in real world scenarios.

5.5 Results

In this chapter we have outlined our concept for consistent service migration and have shown in simulations as well as in experiments with real sensor nodes that transactional service migration along with message buffering during the migration increases the availability and consistency of services.
We believe that the ability of consistent service migration broadens the application spectrum of wireless sensor and actor networks.

Chapter 6

Adaptive Protocol Selection

Wireless Sensor Networks are deployed in various application scenarios under a variety of conditions. Roemer [173] gives a comprehensive comparison of several prototypical applications deployed in various contexts.
There are small deployments with only a few nodes and large deployments with more than 1000 nodes. Some networks are deployed very densely. That means a node has many neighbors within the range of its communication interface, while in other networks, a node might have only a few neighbors. The features of sensor nodes can also vary. Some are equipped with a GPS module to provide location awareness, while other simpler nodes are not.
Some of these characteristics also change at runtime. For instance, the node density increases abruptly if additional nodes are added to an existing deployment by being dropped out of a plane. Afterwards, while the deployed nodes perform their tasks and their batteries deplete, the density is reduced as nodes run out of energy. There are many other examples. Instead of providing just one algorithm for a problem or task, our approach is to provide several algorithms for one problem and choose the appropriate algorithm at runtime. Our experiments in the last chapters have shown that different protocols have advantages in different environments. For instance, the Two Phase Commit with Caching (2PCwC) is more efficient than the Two Phase Commit (2PC) protocol at high loss rates, while the opposite is the case at low loss rates (see Figure 3.12). In this chapter we again consider the problem of service migration and propose a schema for the adaptive selection of a commit protocol in combination with a routing protocol to guarantee efficient service migration and a longer system lifetime.

Organization

The remainder of this chapter is structured as follows:

- In Section 6.1, we outline related work on adaptive transaction processing and adaptive routing protocols.

- Section 6.2 surveys candidate parameters for our adaptive protocol selection.
- We explain the flow of our adaptive protocol selection in Section 6.3 and describe, how optimal thresholds are determined for the considered parameters.
- Our implementation is outlined in Section 6.4 and the evaluation is reported in Section 6.5
- We summarize our results in Section 6.6.

6.1 Related Work

We briefly review related work on adaptive protocol selection in this section and divide it into adaptivity in transaction processing and adaptivity in routing.

6.1.1 Adaptive Transaction Processing

The approaches by Arntsen et al. [8, 9], Serrano-Alvarado et al. [182] and Tang et al. [193] considering adaptivity used in atomic commitment have already been mentioned in Section 3.2. Arntsen et al. [8, 9] describe an adaptive transactional middleware framework for Web Service environments. It is explained how different transaction services with different strictnesses can be run concurrently, while the strictness has to be specified manually. Serrano-Alvarado et al. [182] and Rouvoy et al. [175] describe a transaction system which adapts to the current commit rate and uses either the 2PC-Presumed-Commit or the 2PC-Presumed-Abort protocol to reduce messages. Tang et al. [193] describe an adaptive context-aware transaction model for mobile and ubiquitous computing but does not consider the specific properties of wireless sensor nodes.
Other approaches not only or not directly related to atomic commitment are described in the following.
An approach by Payton et al. [154] computes the achieved consistency of a transaction in the domain of query issuing in wireless ad hoc networks and attaches it to the result of a query but does not allow to perform a transaction with a given consistency. Nuno et al. [145] propose the use of transaction policies to support adaptive transactions to be able to take varying properties of mobile networks into account. These policies have to be specified manually by the user. Helal et al. [77] describe how transactions can be scheduled adaptive to parameters like size of the transaction, number of operations, conflicts, aborts etc. These approaches do not take the characteristics of sensor networks into account but provide useful ideas for the application of adaptivity in sensor networks. All of these contributions in the area of mobile wireless networks rely on a scenario where transactions run on a network of mobile hosts where fixed hosts and mobile support stations are mandatory [50]. In these scenarios it is assumed that coordinators of transactions have to be fixed hosts that can communicate with mobile hosts only with the help of mobile support stations. This

enables more resource consuming techniques like logging, etc. Therefore, these approaches are not directly applicable in wireless sensor networks.

6.1.2 Adaptive Routing

To the best of our knowledge there are no approaches where different routing protocols are provided by sensor nodes and then chosen adaptively at runtime. Therefore, we briefly review routing protocols that adapt to certain parameters at runtime. Wang et al. [198] describe AdaR, a routing scheme that uses a least squares reinforcement learning technique to adapt to the properties of a wireless sensor network at runtime. Adaption in this case means choosing an optimal route. Dube et al. [49] propose the Signal Stability-Based Adaptive Routing (SSA) approach which finds and maintains stable routes at runtime. Several other algorithms exist [3, 92, 116, 201]. While these approaches are adaptive to parameters of the network context, none of these approaches take the parameters of transactions or properties of sensor nodes into account.

6.2 Considered Parameters

There are several parameters that can be considered for the selection of the appropriate commit protocol and routing protocol. We have already mentioned the parameters message loss, number of nodes, density and location awareness in the introduction to this chapter. In this section we discuss several parameters from the areas transaction properties, wireless network context and properties of sensor nodes to identify the optimal parameters to be considered for the adaptive protocol selection.

6.2.1 Transaction Properties

Consistency. Probably the most obvious parameter to consider when migrating a service is the required consistency. If the service to be migrated is very simple, such as a logic function or a data converter, and does not even require any state, there is no need for migrating the service in a transactional way as outlined in Subsection 5.2.3, and the service can be migrated with eventual consistency instead. This can be done by using Trickle [113, 114] or Deluge [88].

If we do not exclusively consider service migration, but instead have a broader range of applications in mind which need transaction processing capabilities, we can consider different properties of a transaction.

Priorities. In real time systems for instance, transactions with a shorter deadline can be prioritized. Then a more expensive but more reliable commit protocol like 2PCwC can be chosen for these transactions at runtime.

Number of participants. The number of participants influences the efficiency of not only the commit protocol but also of the underlying routing protocol. Our Two Phase Commit

with Caching (2PCwC) protocol caches the votes of other participants. This is more effective if there are more participants. At some loss rates, georouting is more efficient than flooding. As the number of participants approaches the number of nodes in total, flooding becomes more efficient. Consequently, it makes sense to determine the appropriate protocol combination at runtime for each transaction when the number of participants is known.
In the following, we discuss how the parameters of the wireless sensor network as a whole can be considered for adaptive protocol selection.

6.2.2 Wireless Network Context

The parameters belonging to the network context according to RFC 2501 [38] have already been described in the introduction of this thesis. We review them with regard to our adaptive protocol selection.
Network Size. As Roemer [173] points out, the number of sensor nodes in existing sensor network deployments varies strongly. While the small number of nine nodes is sufficient for the monitoring of a glacier [131], the worldwide deployment of wireless sensor nodes ARGO [7] contains more than 3000 nodes. In the publications that have appeared in the 2000-2005 proceedings of the ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc), a similar range from 10 to 30000 nodes has been simulated (surveyed by Kurkowski et al. [104]).
While flooding is feasible in a small network of 10 nodes, flooding does not scale to a large network, and more sophisticated routing techniques like georouting or DSDV have to be used in this case.
Connectivity and covered area. The area covered by real deployments also varies between a few meters and worldwide. Kurkowski et al. report areas used in simulations from 25 m x 25 m to 5000 m x 5000 m, with a transmission range varying from 3 m to 1061 m [104]. Hellbrueck and Fischer [78] conclude, that for a MANet with connecting probability of $\sim$95% between two nodes, an average number of neighbors per node between 5 and 15 is necessary.
In the last chapter we used gossiping to improve the efficiency of our commit protocols when using flooding. Haas et al. [70] claim that gossiping gets more effective when the network contains at least 150 nodes. We consider the number of nodes and also the connectivity for our adaptive protocol selection in this chapter.
Another possibility to cope with an increased density and an increased collision probability is the reduction of the transmission power. However, this is not possible on all sensor nodes.
Topology Change. Examples of topology changes are failing nodes, additionally deployed nodes, sleeping nodes, mobile nodes, moving nodes, moving obstacles and weather conditions. These events can dramatically influence the connectivity inside a wireless sensor network and must be considered by our adaptive protocol selection.

Link capacity. The limited data rate of todays wireless sensor nodes is regarded to be constant, although the effective data rate can vary due to interference and is not directly considered in our adaptive protocol selection. However, it is considered indirectly because the loss caused by interference is indeed detected.
Fraction of unidirectional links. Since Kotz et al. [98] demonstrate that realistic deployments contain a significant amount of unidirectional links, we take this into account for our evaluation of the adaptive protocol selection but do not exploit the fraction of unidirectional links directly.
Traffic patterns. While the variety of sensor network applications exhibit different traffic patterns, from constant in simple sense and send applications to bursty in event detection systems, we do not consider this parameter for our adaptive protocol selection.
Sleeping nodes. Since conserving energy is very important in wireless sensor networks, a routing protocol needs to take the duty cycling of devices into account. We believe that the appropriate duty cycling depends more on the actual application and do not consider this parameter further.
The characteristics of an application are also very important for the appropriate selection of the most efficient routing algorithm. A classification and comparison of routing protocols with respect to different application scenarios and network topologies can be found in [19].

6.2.3 Sensor Node Properties

Since a variety of sensor nodes exist today, varying in size, sensing capabilities and resources, it is also important to take these parameters into account for adaptive protocol selection.
Location awareness. If the sensor nodes in a deployment are equipped with a Global positioning System (GPS), a geographical routing algorithm can be used. It may be enough to have only some of the nodes (anchor nodes) in a network equipped with the GPS since the other nodes can determine their position in relation to the anchor nodes. There are many other techniques for building location awareness. An overview can be found in [173] or [34].
It is also possible that GPS could fail at runtime, for instance, if there is no satellite reception anymore due to obstacles in the line of sight. In this case, nodes can switch from georouting to flooding or other routing algorithms at runtime.
Code size. The amount of Programmable Read Only Memory (PROM) varies from 48 kilobytes in the Tmote Sky (Telos B) [160] to 512 kilobytes in the Intel IMote [138]. Our target sensor node platform Pacemate contains 256 kilobytes PROM.
The PROM available for the deployment of alternative commit and routing protocols is determined by the footprint of the application. In the following, we outline the resource consumption of our implemented protocols. The needed PROM of the atomic commit protocols compared in Chapter 3 is shown in Table 6.1.

Protocol(s)	Code Size (Bytes)
2PC	210438
2PCwC	215638
2PC and 2PCwC	227206
CLCP	258678

Table 6.1: Protocol code sizes compiled for Pacemate sensor nodes

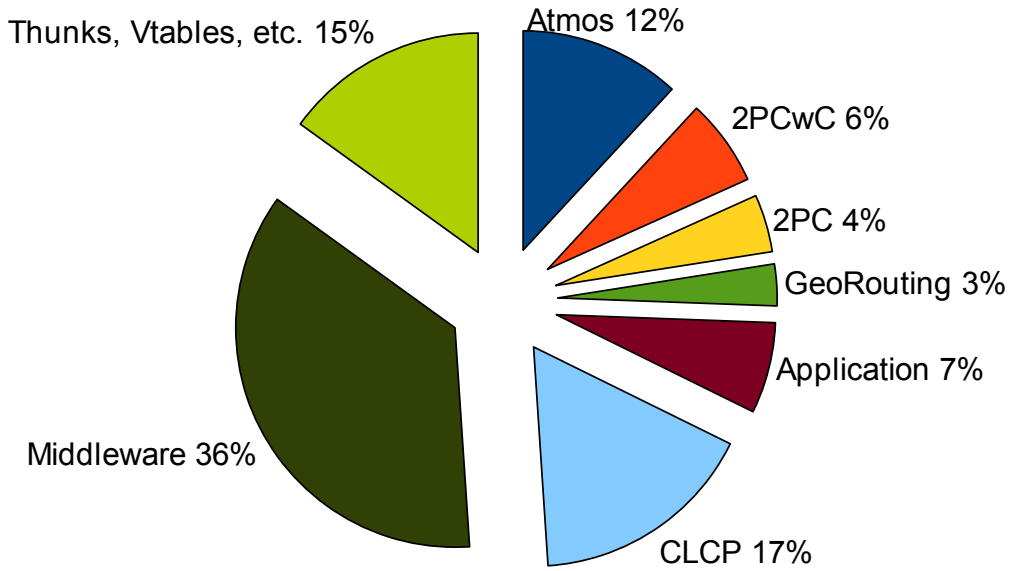


Figure 6.1: Proportionate code size of different modules

Considering the maximum code size for our used sensor node platform Pacemate [122] of 256 kilobytes (262144 bytes), an example application containing a basic service migration scenario compiled with CLCP needs nearly all of the available PROM. The code size of our application compiled with CLCP is 258678 bytes, while 17% of this size are needed for the protocol itself. The sizes of the other modules are shown in Figure 6.1. The protocols 2PC and 2PCwC can be run at the same time on our sensor nodes.
Figure 6.1 shows that the iSense middleware itself is with 36% the lion's share of the code. The second biggest part is CLCP, while non-virtual thunks and virtual tables used for multiple inheritance in C++ are 15% of the binary code. The Atmos Middleware [167], used for the concurrent processing of transactions with different commit protocols (see Section 3.4), needs 12% of the code. The code size of the application itself (7%) comprises the implemented services for our example service discovery and migration scenario described in the last chapter. We use simple defines to set the available and used protocols as shown in Listing 6.1.

```
1 #ifndef SETTINGS_H_
2 #define SETTINGS_H_

4 //#define PROTOCOL_CLCP 0
5 #define PROTOCOL_TWO_PC 1
6 #define PROTOCOL_TWO_PCWC 2

8 #define PROTOCOL PROTOCOL_TWO_PC

10 #define TWOPC_WITH_GEO

12 #endif /* SETTINGS_H_ */
```

Listing 6.1: Static selection of available and used protocols in `settings.h` at compile time

The shown settings define exemplarily that CLCP is not compiled, 2PC and 2PCwC are compiled, and 2PC is used with georouting. The user can choose a setting like this if the geographical positions are known and he needs the remaining PROM for his application. So the number of available transaction processing protocols to be selected at runtime is limited by the available PROM minus the application code.
Since CLCP not only needs considerably more PROM than the other commit protocols and does not scale with the number of participants, we only consider 2PC and 2PCwC for the experiments in the remainder of this chapter.
The important fact is that all of the implemented protocols to be selected at runtime fit into the PROM of our Pacemate sensor node platform at once.

6.2.4 Parameters Chosen to be Considered at Runtime

In summary, we consider the following parameters for our adaptive protocol selection:

1. Consistency demands
2. Message loss
3. Location awareness
4. Number of transaction participants
5. Density (neighborhood size)

We explain in the next section how each of these parameters is interpreted to select the most efficient protocol combination at runtime.

6.3 Adaptive Protocol Selection

In this section, we first explain the flow of our algorithm for adaptive protocol selection. Then we describe how the used thresholds were determined for each considered parameter. We used the parameters given in Section 5.4.2 for the execution of our experiments if not stated otherwise.

6.3.1 Algorithm Flow

We explain the flow of the adaptive protocol selection considering service migration as use case, but apart from the first step, the algorithm is also applicable for any other transaction. The protocol selection process is shown in the flowchart in Figure 6.2. First, the consistency needed for the migration of a specific service is considered. If strict consistency is required for the migration of the considered service, a transactional migration is started.
The first criterion that is determined is the loss rate. If the loss rate is high (above 20%), the next parameter considered is the density. If the loss rate is low (20% or less), the availability of georouting is checked as well as the possibility of a low number of participants.
If georouting is available and the number of participants is below 30 (in a network of 100 total nodes), then georouting is selected as routing protocol. Since the only implemented atomic commit protocol that works with georouting is 2PC, the adaptively selected protocol combination is 2PC with georouting.
If the loss rate is higher than 20% or georouting is not available or the number of participants is higher than 30 in a network of 100 nodes, the next parameter to be considered is the density.
If the density is low, meaning the average number of neighbors is below 10, flooding is used. At this stage, the loss rate, which is higher than 20%, is considered again. If it is higher than 65%, 2PCwC is used, otherwise 2PC is used, both in combination with flooding.
If the density is high, meaning the average number of neighbors per node is 10 or greater, gossiping is used. If the loss rate is higher than 65%, gossiping is used with 2PCwC, otherwise gossiping is used in combination with 2PC.
If the loss rate is lower than 65%, flooding is used with 2PCwC, otherwise flooding is used in combination with 2PC.
The Cross Layer Commit Protocol (CLCP) is not considered in the remainder of this chapter since its resource consumption is too high for resource constrained wireless sensor networks. In the next subsections, we explain how we determined the thresholds given in this subsection.

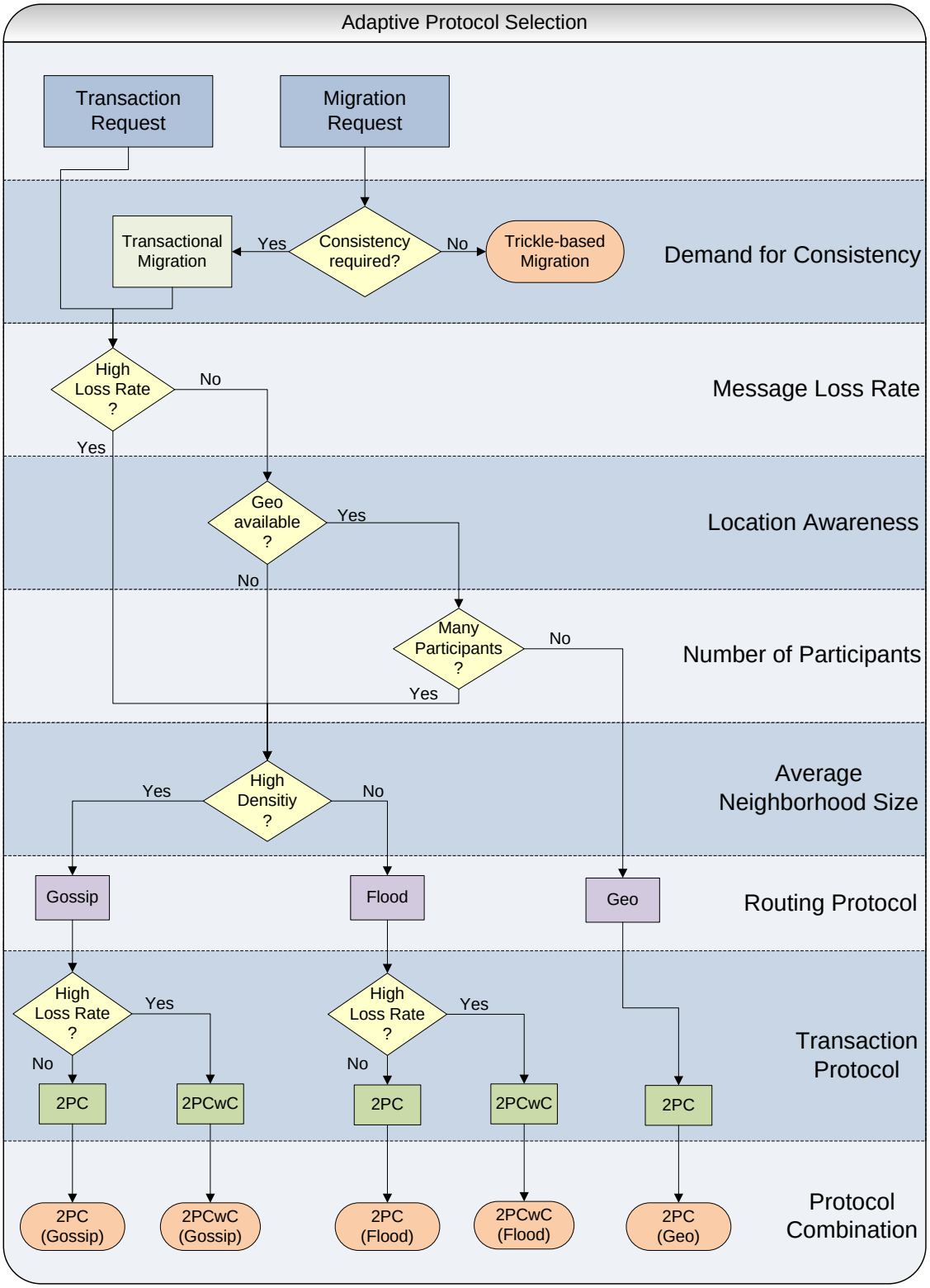


Figure 6.2: Flowchart of our adaptive protocol selection process

6.3.2 Determining the Threshold for Loss

Although various deployments of sensor networks exist today, reliable numbers about message loss in multi-hop networks are hard to find. At least Szewczyk et al. [192] provide a detailed report of 150 nodes deployed over 4 months on Great Duck Island. The authors report medium loss rates of 42% depending on the node type. We assume that loss can also dynamically change in deployments, for instance, due to the application behavior when an increasing communication leads to a higher number of collisions, or due to environmental factors such as changing weather conditions. Consider for instance a sensor network deployed for environmental monitoring. If there are several regular times a day when animals enter the monitored area, maybe in the morning and in the evening to drink from a river, there is an increase of reported events. This can lead to a higher amount of communication in the network and therefore to a higher probability for collisions on the MAC layer. The current loss can be determined by using a simple ping-function as outlined in Section 6.4 Consequently, we believe that it is promising to select the appropriate protocol at runtime. We simulated different thresholds and compare the results in Section 6.5.

6.3.3 Determining the Threshold for the Number of Participants for the Selection of the Commit Protocol

The number of the transaction participants varies depending on the actual application. It is considered for our adaptive protocol selection twice: first for selecting the commit protocol and second for selecting the routing protocol.
Our intuition was that 2PCwC, which caches votes on participants, is more effective when there are more participants. To determine the appropriate participant threshold, we performed experiments with the parameters given in the last Chapter. Figure 6.3 shows our results obtained from simulating 2PC and 2PCwC at a constant loss rate of 65% with a varying number of participants. It can be seen that 2PCwC has a lower cost when the number of transaction participants rises over 14. So the selected threshold for choosing 2PCwC is 14 participants.

Figure 6.3: Impact of number of participants on commit protocol

6.3.4 Determining the Threshold for the Number of Participants for the Selection of the Routing Protocol

The number of participants is also considered for the selection of the routing protocol, namely flooding or georouting. Figure 6.4 shows that for a loss rate of 40%, flooding is more efficient than georouting.
Consequently, we compare flooding and georouting at a loss rate of only 20% and vary the the number of nodes from 50 over 100 to 150 and also vary the number of participants. The comparison is shown in Figure 6.4. It can be seen that flooding becomes less expensive than georouting when the number of participants of a transaction gets greater than 30. So we selected a threshold of 30 participants for the adaptive selection of georouting.

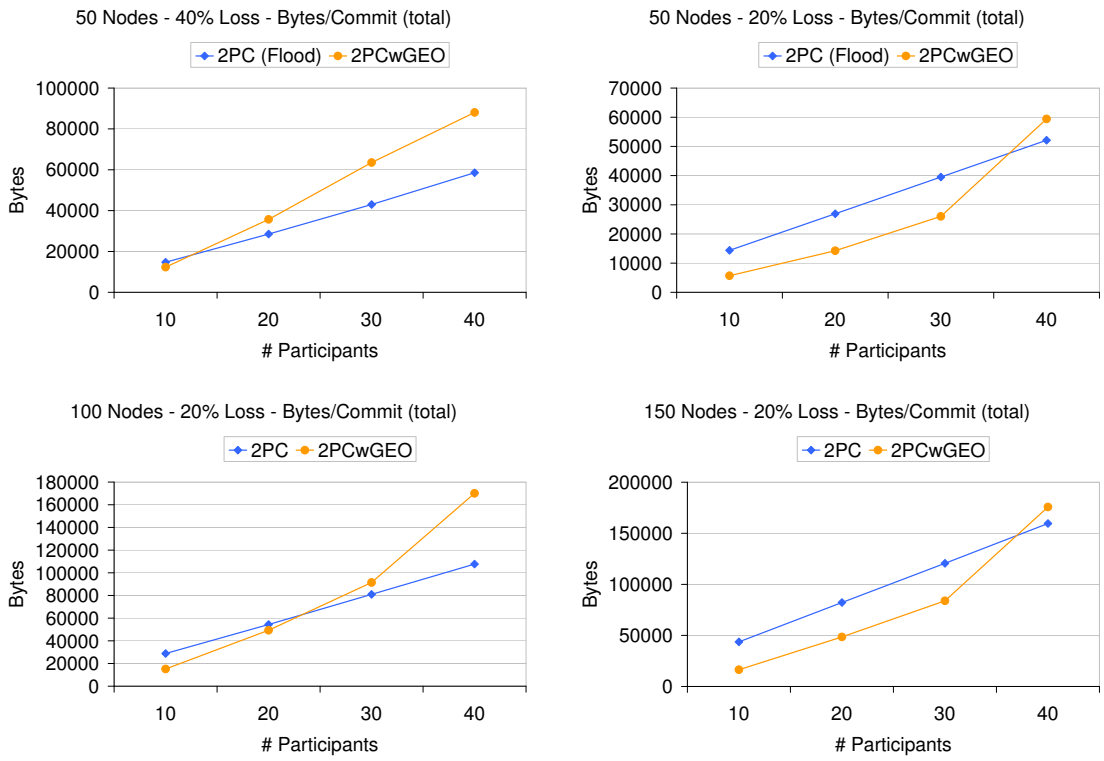


Figure 6.4: Impact of number of participants on routing protocol

6.3.5 Determining the Threshold for the Neighborhood Size

According to Haas et al. [70], gossiping becomes more efficient if the number of nodes in the network exceeds 150. It is obvious that standard flooding leads to many superfluous messages if the network is very dense.

Density

By varying the area of our simulation environment from 200 x 200 field units to 600 x 600 field units with a constant number of 100 nodes, we simulated average neighborhood sizes from 5.52 neighbors to 45.92 neighbors. The comparison of 2PCwC and different gossiping variants is shown in Figure 6.5.
The gossiping variants differ in the number of sure broadcasts in the beginning of a message dissemination. In the last chapter, only 4 hops have been used for sure broadcast (see the diamond graph in Figure 6.5). In this experiment, we can see that gossiping with only 2 sure broadcasts yields even higher cost savings.
At a very high density the advantage of gossiping decreases. This is due to the smaller simulation area. The maximum hop is only 3. In this case, there is no difference between flooding and gossiping, except from the larger header for broadcasting, which makes gossiping even more expensive than flooding at an average neighborhood size of 45.92.

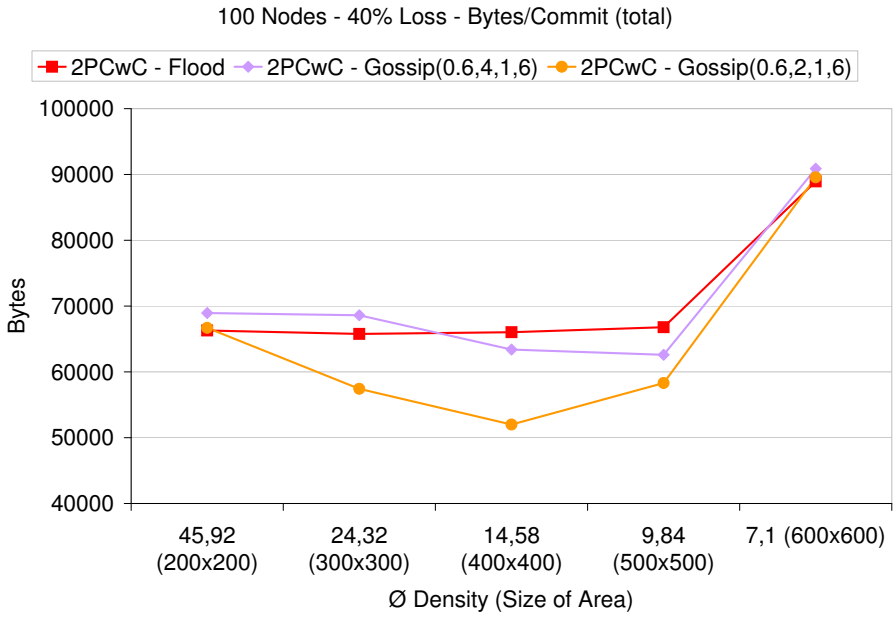


Figure 6.5: Comparison of 2PCwC with and without gossiping when varying the simulation area

Network Size

We also increased the number of nodes in the network along with the simulation area, which lead to a nearly constant number of average neighbors (9.48 to 10.99). Figure 6.6 shows that gossiping saves a significant amount of messages for network sizes greater than 50 nodes. Consequently, the threshold chosen for the adaptive selection of gossiping is 10 neighbors per node.

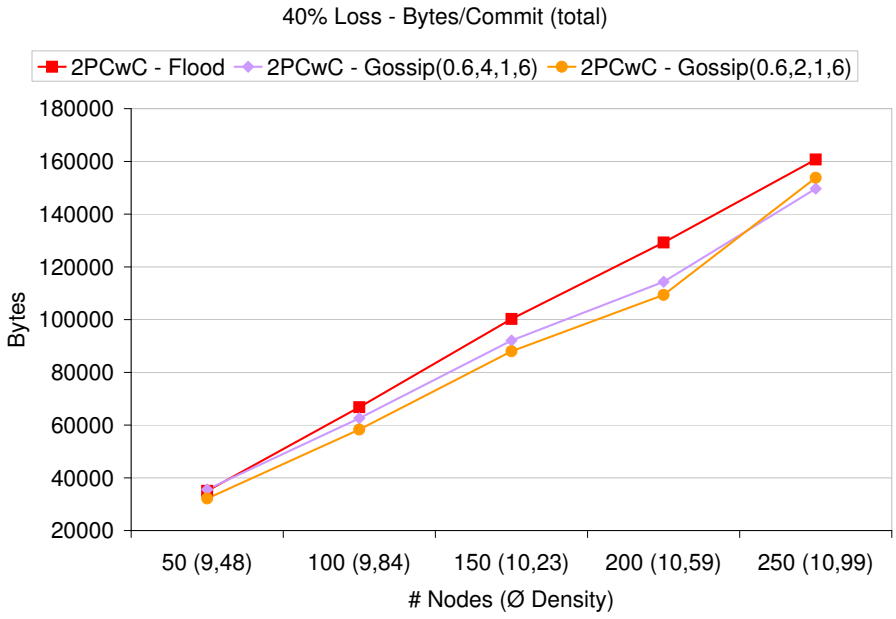


Figure 6.6: Comparison of 2PCwC with and without gossiping when varying the simulation area and the number of nodes

6.4 Implementation

In this section, we briefly outline how the dynamic protocol selection is implemented adapting to the criteria loss rate, number of participants and density (number of neighbors).

6.4.1 Loss Rate

We have shown that different protocols have advantages under different circumstances. For example, 2PC can only be used efficiently if only low message loss occurs. A source code snippet clarifying the adaptive protocol selection is shown in Listing 6.2.

```
1  void create_migrate_transaction
2    (uint16 initiator,uint16 dest,Service* sv){
3    int loss = ping(); //learn current loss in %
4    if (loss < threshold)
5      protocol = PROTOCOL_2PC;
6    else
7      protocol = PROTOCOL_2PCWC;
8    Transaction * t = tm_->create_transaction
9      (initiator, dest, sv, protocol);
10   t.start();
11 }
```

Listing 6.2: Adaptive protocol selection based on current message loss

In line 3 a ping to all neighbors is used to learn the probability that a message is lost. It is calculated by

$$p(loss) = 1 - \frac{|received\_pongs|}{|sent\_pings|}$$

while the number of pings sent to each node was set to 10. If loss is less than 65%, 2PC is selected, otherwise 2PCwC is selected. The loss is detected first for every started transaction. The additional overhead for this is taken into account in the evaluation in Section 6.5.

6.4.2 Number of Participants

Another criterion is the number of participants. Georouting is only efficient as long as the number of participants remains below a certain threshold, otherwise flooding becomes more energy efficient.
An example for a use case where a high number of participants is involved in a transaction is the alteration of metadata. If we assume all nodes in the wireless sensor network measure their environmental temperature in Celsius and we want them to change the unit to Fahrenheit, then we need all nodes to take part in that transaction. In these cases, flooding can be more energy efficient. In the case of service migration, we have only a small number of participants: the source node, the target node and a number of service directories (2,..,n) depending on how much redundancy is required.
Since we consider the transaction participants to be known at the start of a transaction, the appropriate protocol can be chosen when the transaction is started.

6.4.3 Density

Each node counts its neighbors and broadcasts the value periodically. These messages are not flooded to save energy. When such a message is received by a node, it it can calculate the average number of neighbors. This is an estimate for the local density. The additional costs for this are taken into account when comparing the overall costs for the adaptive protocol selection with the static selection of one protocol combination. The question is how accurately the global density can be estimated by considering the local density. This is analyzed in the next section.

6.5 Evaluation

In this section we report the results of our adaptive protocol selection considering the parameters loss, number of participants and density.
To evaluate our adaptive protocol selection we performed again simulations with the network simulator Shawn with the service migration scenario outlined in Figure 5.2. We also compare the costs in terms of transmitted bytes per service migration and the degree of

consistency expressed in number of missed messages. In the simulation scenarios, we basically executed a number of service migrations and compared the costs as described in Section 5.4.2. Contrary to the last chapter, we also varied the number of transaction participants, the area and the number of nodes and the parameters used for gossiping.
We again used different communication models (Unit Disk Graph [100] and Radio Irregularity Model [209]) and either used a perfect MAC layer and varied the message loss or simulated a CSMA/CA MAC layer with the standard parameters of Shawn [55] and repeated every experiment 20 times for statistical soundness.

6.5.1 Loss

We varied the loss rate during our experiments from 40% to 80% as shown in Figure 6.7. The underlying assumption is that there are peak times when more events occur in the network than in other periods, leading to a higher probability of message collisions and consequently more loss.

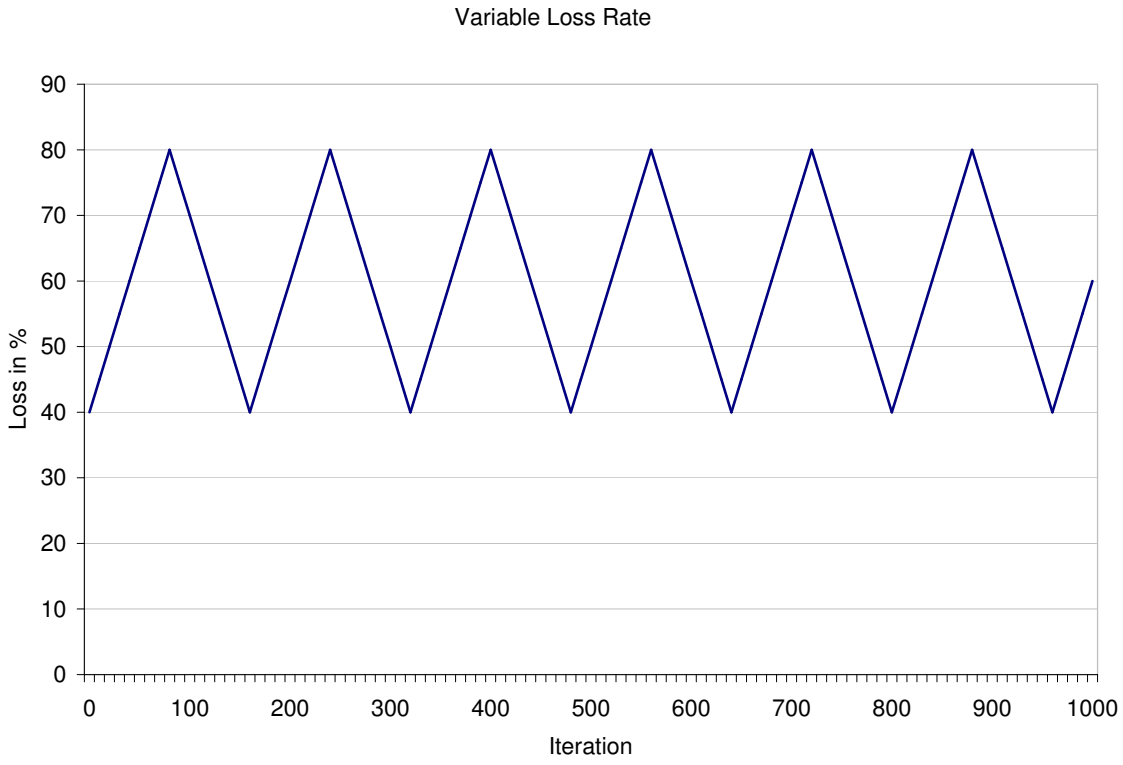


Figure 6.7: Simulation of a variable loss rate

Figure 6.8 shows a comparison of 2PC, 2PCwC and our adaptive protocol selection with different thresholds regarding the commit rate and costs in transmitted bytes per commit. "Adaptive 70" means, for instance, that a threshold of 70% message loss has been used

to select the atomic commit protocol. It can be seen that "Adaptive 85" to "Adaptive 95" significantly save transmission costs and also increase the commit rate compared with 2PC.

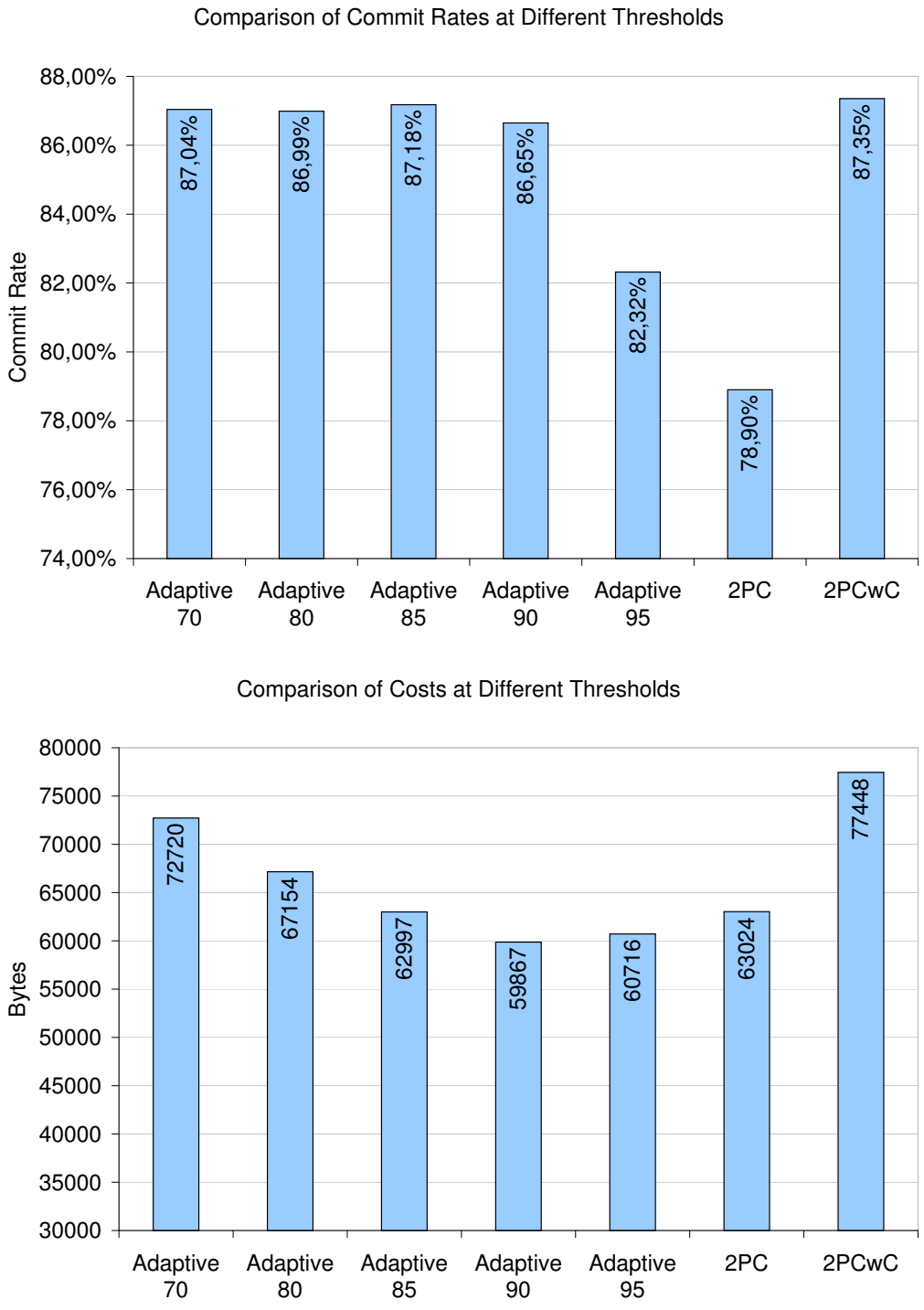


Figure 6.8: Comparison of commit rate and costs with random transmission model at different thresholds

6.5.2 Participants

Selection of Commit Protocol

We randomly varied the number of participants from 9 to 16 and started service migrations a described earlier. The selected threshold was 14. The results are shown in Figure 6.9. We used the following four metrics:

- Transmitted bytes per commit in total, denoted as $\frac{B}{C}$ (total).
- Transmitted bytes per commit and legitimate abort in total, denoted as $\frac{B}{C+A}$ (total). This metric takes into account that some aborts called legitimate aborts are due to vote abort messages caused by the application and not caused by message loss or the atomic commit protocol.
- Transmitted bytes per commit only for the atomic commit protocol, denoted as $\frac{B}{C}$ (protocol). In this metric, the bytes transmitted by the example services are excluded.
- Transmitted bytes per commit and legitimate abort only for the atomic commit protocol, denoted as $\frac{B}{C+A}$ (protocol).

It can be seen that the adaptive selection of the atomic commit protocol leads to a reduction of transmission costs from 5% to 10% when compared with the exclusive usage of 2PC or 2PCwC. This holds regardless of the calculation of the costs.

Selection of Routing Protocol

We randomly varied the number of participants from 9 to 40 and started again the migration of services like described in the last chapter. The results are shown in Figure 6.10. It can be seen that our adaptive selection of 2PC with flooding or 2PC with georouting leads to a reduction of message costs from 11% to 13% compared to the exclusive usage of 2PC with flooding.

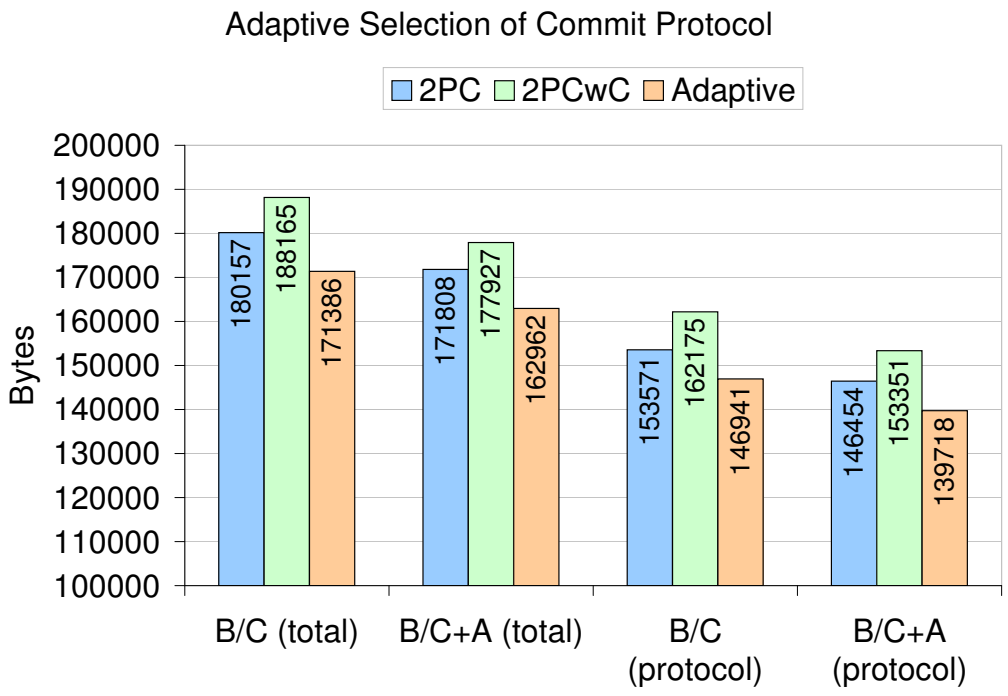


Figure 6.9: Reduction of costs by 5%-10% with 65% loss

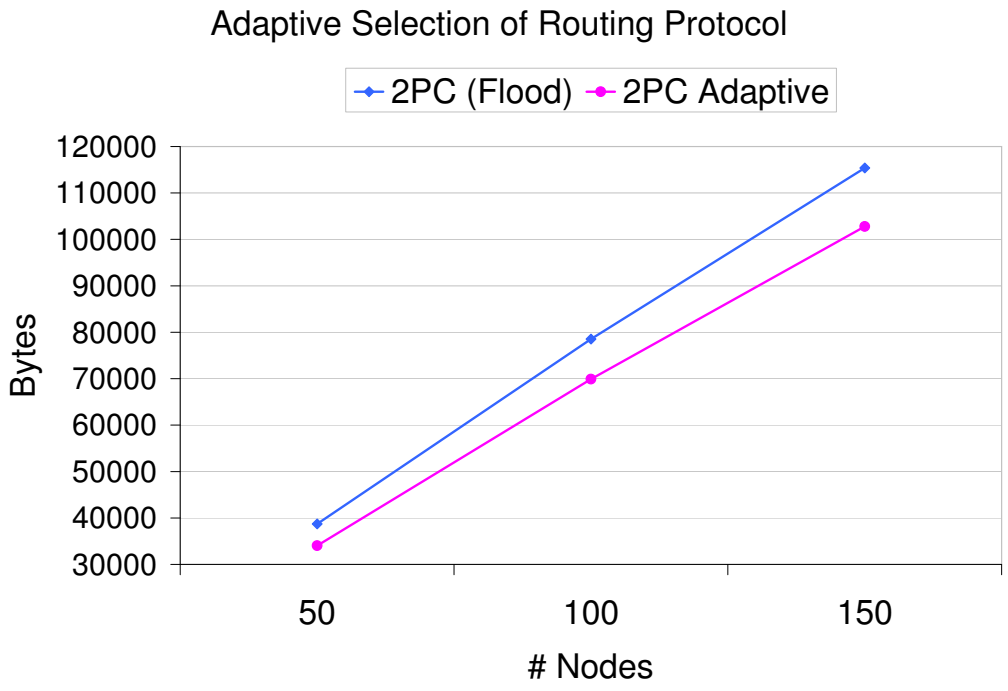


Figure 6.10: Reduction of costs by 11%-13% with 20% loss

6.5.3 Density

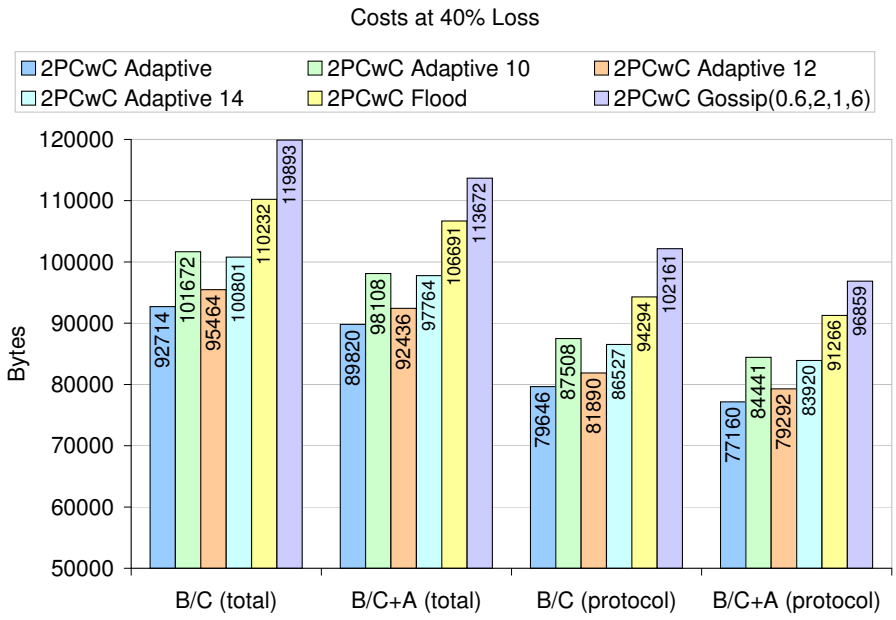
The last parameter we considered for our adaptive protocol selection is the average number of neighbors, which can also be expressed as density. The motivation behind this is that gossiping is only effective if the density is high. We focused on the 2PCwC protocol to analyze the adaptive protocol selection based on the density since the experiments performed with 2PC using either flooding or gossiping were quite similar at a loss rate of 40% or less. To evaluate the benefits of using 2PCwC adaptively with flooding or with gossiping we simulated an iterative deployment. Therefore, an existing deployment was extended at runtime by additional nodes. We started by performing our simulation with 100 nodes as shown in Figure 6.11 (a). After the first half of the simulation, we deployed 100 additional nodes and got the network topology shown in Figure 6.11 (b). In other words, the density is increased from an average of 7.1 neighbors per node to an average of 14.33 neighbors.

(a) 100 nodes (average of 7.1 neighbors per node) (b) 200 nodes (average of 14.33 neighbors per node)

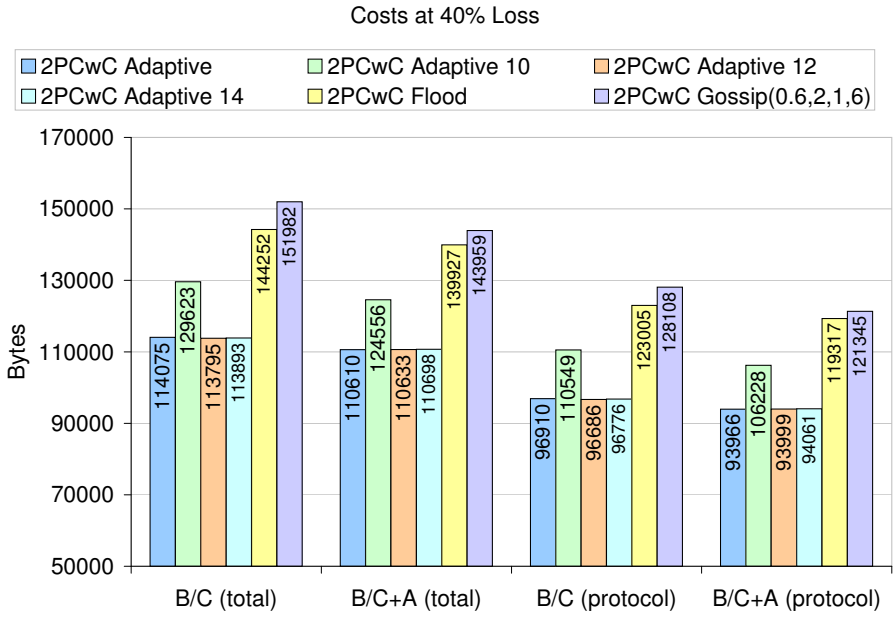
Figure 6.11: Example network before and after increasing the number of nodes

The results are shown in Figure 6.12 (a). We compare 2PCwC using either exclusively flooding or exclusively gossiping with 2PCwC using adaptively flooding or gossiping. "2PCwC Adaptive" denotes an (unrealistic) adaptive protocol selection using global knowledge. This means that it was defined that flooding is used during the first half of the simulation for the scenario with 100 nodes and gossiping is used for the scenario with 200 nodes in the second half. "2PCwC Adaptive 14", for instance, means that 14 neighbors were used as a threshold to select either gossiping (>14 neighbors) or flooding otherwise. The number of neighbors was estimated only locally in the perimeter of the node by broadcasting the node's number of neighbors and averaging the received numbers to save transmission costs. When considering the transmitted bytes per commit regardless of the used metric, it can be

seen that the adaptive protocol selection saves a significant number of transmitted bytes. While for the number of transmitted bytes per commit in total, the unrealistic adaptive protocol selection using global knowledge saved 16% compared to the exclusive use of 2PCwC with flooding, the realistic adaptive protocol selection using 12 neighbors as a threshold could still save 13%.
We also performed the experiment using 300 instead of 200 nodes (see Figure 6.12 (b)). In this case, the savings using the realistic adaptive protocol selection depending on local estimations were 21% of the transmitted bytes per commit using a threshold of 12.
The experiments have been repeated using the CSMA/CA transmission model. The results shown in Figure 6.13 confirm the results obtained from the experiments with the random transmission model. Here, the savings are 10% compared to the exclusive use of 2PCwC with gossiping at a threshold of 10 neighbors in the experiment with 200 nodes and 15% in the experiment with 300 nodes. It is interesting that when using CSMA/CA in this experiment, 2PCwC using exclusively gossiping is more efficient than 2PCwC using exclusively flooding, contrary to the results obtained from experiments with the random transmission model and 40% loss. This is due to collisions on the MAC layer which are significantly reduced by using gossiping.

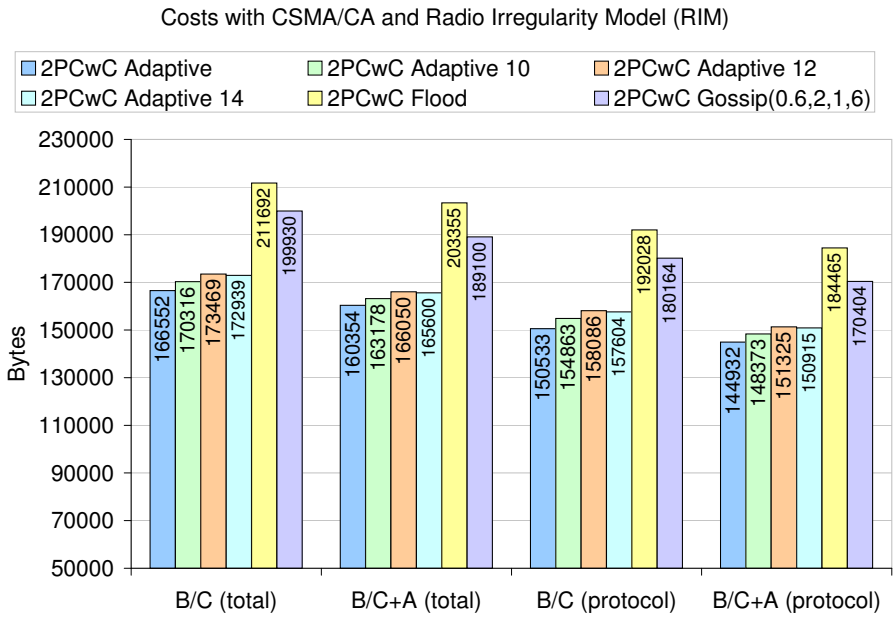


(a) 100 to 200 nodes

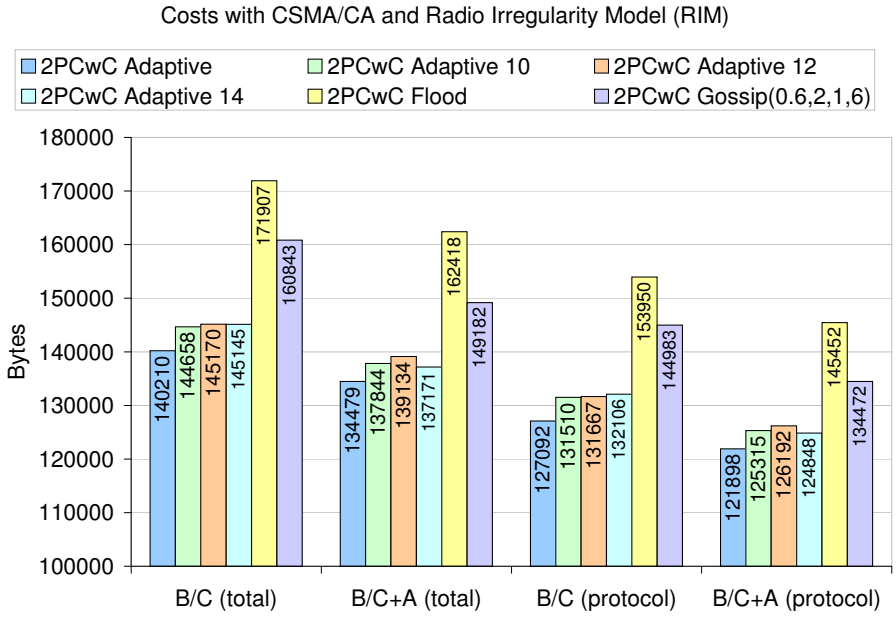


(b) 100 to 300 nodes

Figure 6.12: Comparison of costs of adaptive protocol selection adaptive to density at 40% loss



(a) 100 to 200 nodes



(b) 100 to 300 nodes

Figure 6.13: Comparison of costs of adaptive protocol selection adaptive to density with CSMA/CA

6.6 Results

In the following, we summarize the benefits of our adaptive protocol selection process.
Protocol selection adaptive to the loss rate leads to a reduction of costs by 5-6% with the random transmission model.
Adaptivity sensitive to the number of participants reduces the transmission costs by 5-10% at a 65% loss rate when selecting the commit protocol at runtime and 11-13% when selecting the routing protocol at runtime.
The adaptive protocol selection with regard to the density can save up to 21% of the transmitted bytes per commit.
These savings correspond directly to an increased lifetime of the sensor network deployment since most of the energy consumption in sensor networks is due to message transmission. Consequently, the adaptive selection of commit protocol and routing protocol at runtime can increase the lifetime of wireless sensor network deployments.

Chapter 7

Conclusion

7.1 Summary

This thesis has outlined use cases for service migration and shown that transaction processing capabilities are necessary for enabling stateful and consistent service migration in wireless sensor networks.
We have shown in simulations and experiments with the sensor node platform Pacemate, that our Two Phase Commit with Caching protocol is the most efficient protocol for the use in wireless sensor networks. Compared to Two Phase Commit, we have shown in experiments with 20 Pacemate sensor nodes that the commit rate is increased from 53% to 84% and that the costs are reduced from 249 to 200 transmitted bytes per commit per node. Further, we have shown in simulations with Shawn that traditional Strict Two Phase Locking outperforms validation and timestamp ordering regarding efficiency when simulated with message loss. We have also reported our implementation of locking for the Pacemate sensor nodes.
We have described a comprehensive service migration scenario which enables service discovery and consistent migration of stateful services and have outlined our implementation using our Two Phase Commit with Caching protocol for the network simulator Shawn and Pacemate sensor nodes running Surfer OS. We could prove that consistent service migration is feasible on the resource constrained wireless nodes. These results can be generalized since Surfer OS has also been ported to the well known Telos B sensor node platform [160].
Finally, we have described our adaptive selection of commit protocol and routing protocol in dependency of the required consistency and the given network context to provide the most efficient transaction processing for a given sensor network deployment. We have shown that the parameters loss, density and number of participants can be estimated or detected at runtime to select the appropriate protocol combination. We have shown in simulations with Shawn that the adaptive protocol selection can save 5% to over 20% of the transmitted data volume. Adaptivity could also be easily integrated in our service migration scenario for Surfer OS, but the evaluation is very tedious because a large number of nodes

(100 to 150) would be necessary to see most of the benefits.
In summary, this thesis supports the usage of sophisticated applications for wireless sensor networks demanding increased coordination capabilities while taking the severe resource constraints of wireless sensor networks into account. Application areas that can benefit from this work are not only service oriented sensor sensor networks but also sensor network databases like TinyDB and StonesDB and the emerging wireless sensor and actor networks.

7.2 Future Work

The following topics are part of our future work.

Adaptivity. We are investigating how different concurrency control protocols can be used in an adaptive manner. The usage could be done adaptive to the current message loss, but also to conflicts and type of transactions. For instance, validation could be used if the message loss is really low, otherwise locking could be used.

Cost function. We plan to develop a sophisticated cost function for the selection of the optimal target node for service migration. Possible parameters to be considered are the remaining battery capacity, the available RAM and other properties.

Integration in iSense and Surfer OS Transaction processing capabilities could be directly integrated into iSense and Surfer OS. The usage could also be done via a pre-compiler like in the $XOBE_{SensorNetwork}$ project [80] as shown in the following listing:

```
begin of transaction
      do x
      do y
end of transaction
```

Logging. Sophisticated logging mechanisms for wireless sensor networks are also missing. These could be implemented using our replication strategies for wireless sensor networks [140, 141], where neighbor nodes are used to replicate important data in a consistent way.

Until now, replication in wireless sensor networks has been mostly considered in standard data gathering scenarios [72, 73], and not in service oriented scenarios or in relation to transaction processing.

Persistence in wireless sensor networks has been researched in the context of improving data availability by using special codes when transmitting data from sources to the sink [117, 118, 119, 120], but not in a database context.

This has also been described in [179]. The authors implement a language abstraction for a software transactional memory for a self developed Java middleware and use 3PC to achieve distributed commit.

Mobility. Another important aspect of emerging wireless sensor network deployments is mobility. While this aspect has already been considered in the context of transaction processing in mobile ad hoc networks by Ayari et al. [10, 11, 12], mobile transaction processing has not yet been researched in wireless sensor networks.

Multi-tier deployment. Instead of running the implemented transaction processing protocols directly on the sensor nodes, proxies could be used to run the protocols to release this work from the resource constrained sensor nodes as considered by Guergen et al. [69].

Relaxed consistency. Another open question is, if the consistency could be relaxed in some application domains for wireless sensor networks. This could be done with the help of more flexible transaction models like nested transactions [51].

Transactions on compressed XML data structures. Several approaches for processing transactions on compressed XML datasets inside the sensor network can be investigated (see [80, 81, 82, 83, 85, 86]).

Various service discovery approaches. Another open question is how consistency can be enforced if service discovery is done without designated service directories or overlay approaches [132].

Appendix A

Parameters for Trickle

A.1 Determined Parameters for Trickle

The parameters that have been used for Trickle in the simulations of our service migration scenario with Shawn as outlined in Table 5.5 are explained in this section.
The redundancy constant k is used to determine how many consistent Trickles a node needs to hear before he transmits when counter t expires. If k is large, the probability that the node transmits is higher. Figure A.1 shows a comparison of missed messages and transmitted bytes per migration when varying k. On the one hand, it can be seen that the bytes per migration increases when k is increased. On the other hand, the number of missed messages is minimal when selecting $k = 6$.
The interval t_{low} is used as a lower bound for the transmission frequency. Figure A.2 shows a comparison of missed messages and transmitted bytes per migration when varying t. It can be seen that the costs decrease when t_{low} increases and the number of messages increases.
Consequently, the selected parameters for Trickle are $t_{low} = 100\ ms$ and $k = 6$.

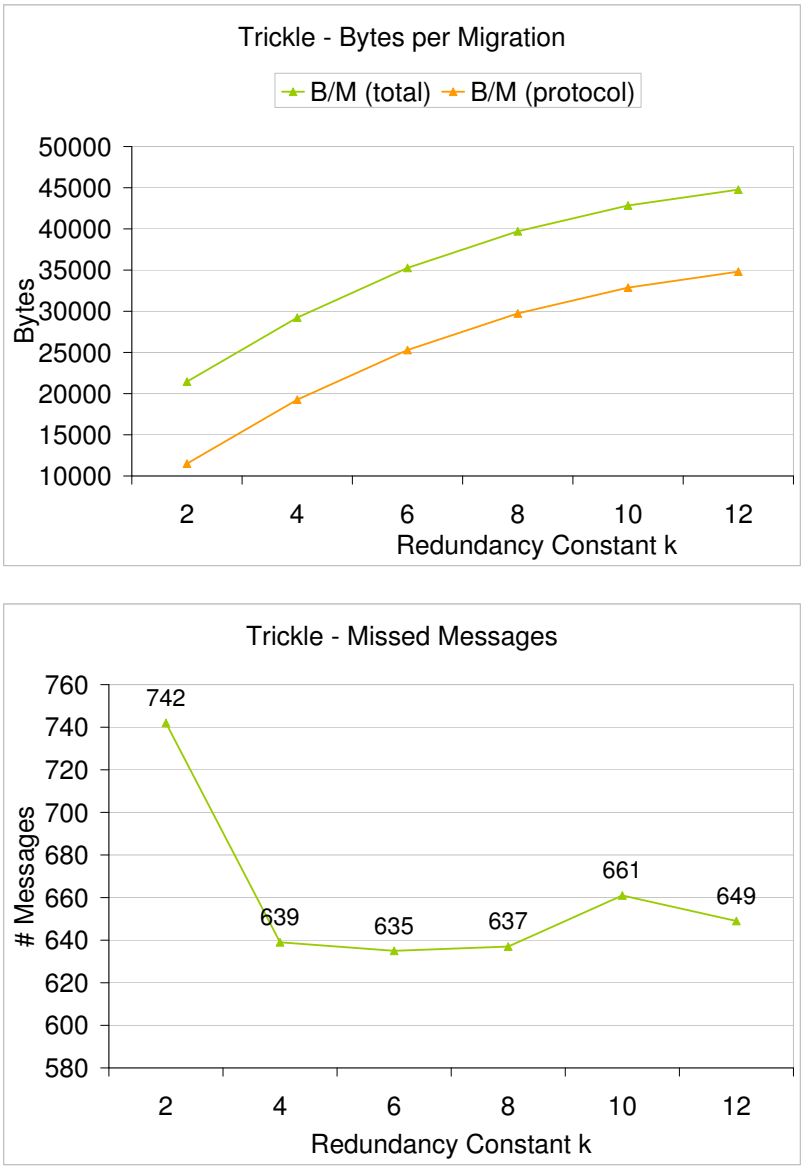
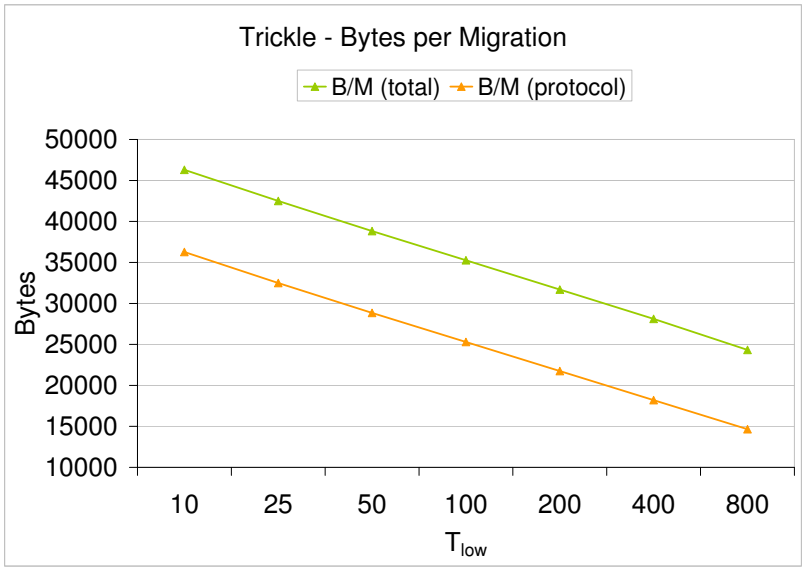


Figure A.1: Determining the optimal k



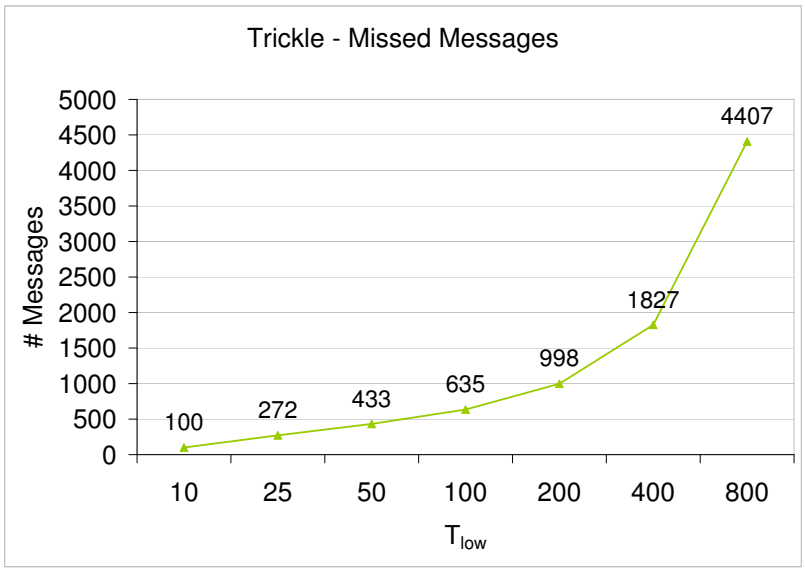


Figure A.2: Determining the optimal lower bound t_{low}

Appendix B

Curriculum Vitae

Privat	
Christoph Reinke	Geboren am 27.12.1980 in Lüneburg, wohnhaft in Lübeck, ledig
Eltern	Berthold und Ingetraut Reinke (geb. Wehner)
Geschwister	Hendrik Reinke

Ausbildung	
10/2002 – 05/2007	Studium der Informatik an der Universität zu Lübeck mit Abschluß als Diplom-Informatiker. Benotung der Diplomarbeit 1,0 und Diplomnote 1,0
08/2001 – 08/2002	Ausbildung zum Fachinformatiker (Fachrichtung Systemintegration) bei der Bundesanstalt für Arbeit (Arbeitsamt Hamburg) in Kooperation mit Siemens Berufsausbildung in Paderborn
08/1993 – 06/2000	Gymnasium Oedeme, Lüneburg (Abitur)

Beruflicher Werdegang	
seit 06/2007	Wissenschaftlicher Mitarbeiter am Institut für Informationssysteme der Universität zu Lübeck
07/2000 – 05/2001	Zivildienst bei der Lebenshilfe Lüneburg-Harburg gGmbH

List of Personal Publications

[1] S. Groppe, J. Groppe, V. Linnemann, D. Kukulenz, N. Hoeller, and C. Reinke. Embedding sparql into xquery / xslt. In R. L. Wainwright and H. Haddad, editors, *Proceedings of the 23rd ACM Symposium on Applied Computing (ACM SAC 2008)*, pages 2271–2278, Fortaleza, Ceara, Brasilien, March 16 - 20 2008. ACM. ISBN 978-1-59593-753-7. doi: http://doi.acm.org/10.1145/1363686.1364228.

[2] S. Groppe, J. Groppe, C. Reinke, N. Hoeller, and V. Linnemann. *Open and novel issues in XML database applications: future directions and advanced technologies*, chapter XSLT: Common Issues with XQuery and Special Issues of XSLT, pages 108–135. IGI Global, Information Science Reference, USA/UK, 2009. ISBN ISBN 978-1-60566-308-1. doi: http://www.igi-global.com/reference/details.asp?ID=33277\&v=tableOfContents.

[3] N. Hoeller, C. Reinke, S. Groppe, and V. Linnemann. Xobe sensor networks: Integrating xml in sensor network programming. In *Proceedings of the 5th International Conference on Networked Sensing Systems (INSS 2008)*, page 229, Kanazawa, Japan, June 17 - 19 2008. IEEE. ISBN 978-4-907764-31-9.

[4] N. Hoeller, C. Reinke, D. Kukulenz, and V. Linnemann. smartcq: Answering and evaluating bounded continuous search queries within the www and sensor networks. In *Fifth International Conference on Innovations in Information Technology (Innovations 2008)*, pages 160–164, Al Ain, United Arab Emirates, December 16 - 18 2008. IEEE. ISBN 978-1-4244-3396-4. doi: http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=4781660.

[5] N. Hoeller, C. Reinke, J. Neumann, S. Groppe, D. Boeckmann, and V. Linnemann. Efficient xml usage within wireless sensor networks. In X. Wang and N. B. Shroff, editors, *Proceedings of the Fourth International Wireless Internet Conference (WICON 2008)*, ACM International Conference Proceeding Series (AICPS), page Article No: 74, Maui, Hawaii, USA, November 17 - 19 2008. ACM. ISBN 978-963-9799-36-3. doi: http://doi.acm.org/10.1145/1554126.1554218.

[6] N. Hoeller, C. Reinke, J. Neumann, S. Groppe, C. Werner, and V. Linnemann. Xml data management and xpath evaluation in wireless sensor networks. In G. Kotsis, D. Taniar,

E. Pardede, and I. Khalil, editors, *Proceedings of the 7th International Conference on Advances in Mobile Computig & Multimedia (MoMM 2009), held in conjunction with iiWAS 2009 conference*, pages 218–227, Kuala Lumpur, Malaysia, December 14 - 16 2009. ACM. ISBN 978-1-60558-659-5. doi: http://doi.acm.org/10.1145/1594187.1594193. This paper received the BEST PAPER AWARD MoMM 2009.

[7] N. Hoeller, C. Reinke, J. Neumann, S. Groppe, C. Werner, and V. Linnemann. Towards energy efficient xpath evaluation in wireless sensor networks. In *ACM International Conference Proceeding Series - Proceedings of the 6th International Workshop on Data Management for Sensor Networks (DMSN 2009)*, pages 1–2, Lyon, Frankreich, August 24 - 28 2009. ACM. ISBN 978-1-60558-777-6. doi: http://doi.acm.org/10.1145/1594187.1594193.

[8] N. Hoeller, C. Reinke, J. Neumann, S. Groppe, F. Frischat, and V. Linnemann. Dacs: A dynamic approximative data caching in wireless sensor networks. In *Proceedings of the 5th International Conference on Digital Information Management (ICDIM 2010)*, pages 339–346, Thunder Bay, Canada, July 5 - 8 2010. IEEE. ISBN 978-1-4244-7571-1.

[9] N. Hoeller, C. Reinke, J. Neumann, S. Groppe, M. Lipphardt, B. Schütt, and V. Linnemann. Stream-based xml template compression for wireless sensor network data management. In *Proceedings of the 4th International Conference on Multimedia and Ubiquitous Engineering (MUE 2010)*, pages 1–9, Cebu, Philippines, August 11 - 13 2010. IEEE. ISBN 978-1-4244-7566-7. doi: http://dx.doi.org/10.1109/MUE.2010.5575094. This paper received the BEST PAPER AWARD MUE 2010.

[10] N. Hoeller, C. Reinke, J. Neumann, S. Groppe, C. Werner, and V. Linnemann. Efficient xml data and query integration in the wireless sensor network engineering process. *International Journal of Web Information Systems*, 6 (4):319–358, 2010. doi: http://dx.doi.org/10.1108/17440081011090248. URL `http://www.emeraldinsight.com/products/journals/journals.htm?id=ijwis`.

[11] D. Kukulenz, C. Reinke, and N. Hoeller. Web contents tracking by learning of page grammars. In *Proceedings of the Third International Conference on Internet and Web Applications and Services (ICIW 2008)*, pages 416–425, Athens, Greece, June 8 - 13 2008. IEEE. doi: http://doi.ieeecomputersociety.org/10.1109/ICIW.2008.58.

[12] M. Lipphardt, J. Neumann, N. Hoeller, C. Reinke, S. Groppe, V. Linnemann, and C. Werner. Xml und soa als wegbereiter für sensornetze in der praxis. *Praxis der Informationsverarbeitung und Kommunikation (PIK)*, 31(3):146–152, Juli - September 2008. URL `http://www.reference-global.com/loi/piko?cookieSet=1`.

[13] J. Neumann, C. Reinke, N. Hoeller, and V. Linnemann. Adaptive quality-aware replication in wireless sensor networks. In *Communication and Networking, Proceedings of the 2009 International Workshop on Wireless Ad Hoc, Mesh and Sensor Networks (WAMSNET09), held in conjunction with the 2009 International Conference on Future Generation Communication and Networking (FGCN 2009)*, volume 56 of *Communications in Computer and Information Science (CCIS)*, pages 413–420, Jeju Island, Korea, Dezember 10 - 12 2009. Springer. ISBN 978-3-642-10843-3 (Print), 978-3-642-10844-0 (Online). doi: http://www.springerlink.com/content/v8n275618q8l3677/.

[14] J. Neumann, N. Hoeller, C. Reinke, and V. Linnemann. Redundancy infrastructure for service-oriented wireless sensor networks. In *Proceedings of the 9th International Symposium on Network Computing and Applications (NCA 2010)*, pages 269–274, Cambridge, Massachusetts, USA, July 15 - 17 2010. IEEE. ISBN 978-0-7695-4118-1.

[15] C. Reinke. Adaptive service migration and transaction processing in wireless sensor networks. In *Proceedings of the 7th Middleware Doctoral Symposium (MDS 2010)*, pages 8–13, Bangalore, India, November 29 - December 03 2010. ACM. ISBN 978-1-4503-0457-3. doi: http://doi.acm.org/10.1145/1891748.1891750.

[16] C. Reinke, N. Hoeller, M. Lipphardt, J. Neumann, S. Groppe, and V. Linnemann. Integrating standardized transaction protocols in service oriented wireless sensor networks. In *Proceedings of the 24th ACM Symposium on Applied Computing*, pages 2202–2203, Honolulu, Hawaii, USA, March 8 - 12 2009. ACM. ISBN 978-1-60558-166-8. doi: http://doi.acm.org/10.1145/1529282.1529768.

[17] C. Reinke, N. Hoeller, and V. Linnemann. Adaptive atomic transaction support for service migration in wireless sensor networks. In *Proceedings of the Seventh IEEE and IFIP International Conference on Wireless and Optical Communications Networks (WOCN2010)*, Colombo, Sri Lanka, September 6 - 8 2010. IEEE. ISBN 978-1-4244-7203-1. doi: http://dx.doi.org/10.1109/WOCN.2010.5587366.

[18] C. Reinke, N. Hoeller, J. Neumann, S. Groppe, S. Werner, and V. Linnemann. Analysis and comparison of atomic commit protocols for adaptive usage in wireless sensor networks. In *Proceedings of the 2010 IEEE International Conference on Sensor Networks, Ubiquitous and Trustworthy Computing (SUTC 2010)*, pages 138–145, Newport Beach, California, USA, June 7 - 9 2010. IEEE. ISBN 978-0-7695-4049-8. doi: http://doi.ieeecomputersociety.org/10.1109/SUTC.2010.12.

[19] C. Reinke, N. Hoeller, S. Werner, S. Groppe, and V. Linnemann. Consistent service migration in wireless sensor networks. In *Proceedings of the 2011 International Conference on Wireless and Optical Communications (ICWOC 2011)*, 2011.

List of Figures

List of Listings

List of Tables

Bibliography

[1] M. E. Adiba and B. G. Lindsay. Database snapshots. In *Proceedings of the sixth international conference on Very Large Data Bases - Volume 6*, pages 86–91. VLDB Endowment, 1980. URL http://portal.acm.org/citation.cfm?id=1286887.1286896.

[2] W. L. M. R. A.K. Elmagarmid, Y. Leu. A multidatabase transaction model for interbase. In *Proceedings of the 16th VLDB Conference, Brisbane, Australia*, 1990. doi: http://www.vldb.org/dblp/db/conf/vldb/ElmagarmidLLR90.html.

[3] A. Akhtar, A. A. Minhas, and S. Jabbar. Adaptive intra cluster routing for wireless sensor networks. In *Proceedings of the 2009 International Conference on Hybrid Information Technology*, ICHIT '09, pages 118–123, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-662-5. doi: http://doi.acm.org/10.1145/1644993.1645015. URL http://doi.acm.org/10.1145/1644993.1645015.

[4] I. F. Akyildiz and I. H. Kasimoglu. Wireless sensor and actor networks: research challenges. *Ad Hoc Networks*, 2(4):351 – 367, 2004. ISSN 1570-8705.

[5] J. Al-Karaki and A. Kamal. Routing techniques in wireless sensor networks: a survey. *Wireless Communications*, 11:6–28, 2004.

[6] J. N. Al-Karaki and G. A. Al-Mashaqbeh. Energy-centric routing in wireless sensor networks. *Microprocess. Microsyst.*, 31(4):252–262, 2007. ISSN 0141-9331. doi: http://dx.doi.org/10.1016/j.micpro.2007.02.008.

[7] ARGO. Global ocean sensor network, 2010. URL http://www.argo.ucsd.edu/.

[8] A.-B. Arntsen and R. Karlsen. Reflects: a flexible transaction service framework. In *ARM '05: Proceedings of the 4th workshop on Reflective and adaptive middleware systems*, New York, NY, USA, 2005. ACM. ISBN 1-59593-270-4. doi: http://doi.acm.org/10.1145/1101516.1101520.

[9] A.-B. Arntsen, M. Mortensen, R. Karlsen, A. Andersen, and A. Munch-Ellingsen. Flexible transaction processing in the argos middleware. In S. Apel, M. Rosenmueller,

G. Saake, and O. Spinczyk, editors, *Software Engineering for Tailor-made Data Management*, pages 12–17. University of Magdeburg, 2008.

[10] B. Ayari. *Perturbation-Resilient Atomic Commit Protocols for Mobile Environments*. PhD thesis, Technische Universität Darmstadt, 2010.

[11] B. Ayari, A. Khelil, K. Saffar, and N. Suri. Data-based agreement for inter-vehicle coordination. *Mobile Data Management, IEEE International Conference on*, 0:279–280, 2010. doi: http://doi.ieeecomputersociety.org/10.1109/MDM.2010.26.

[12] B. Ayari, A. Khelil, and N. Suri. Partac: A partition-tolerant atomic commit protocol for manets. In *Proc. of The 11th International Conference on Mobile Data Management (MDM)*, 2010.

[13] H. Baldus, K. Klabunde, and G. Müsch. Reliable set-up of medical body-sensor networks. In *Wireless Sensor Networks*, volume 2920 of *Lecture Notes in Computer Science*, pages 353–363. Springer Berlin / Heidelberg, 2004.

[14] P. H. Bauer, M. Sichitiu, R. S. H. Istepanian, and K. Premaratne. The mobile patient: Wireless distributed sensor networks for patient monitoring and care. In *ITAB-ITIS 2000*, 2000.

[15] U. Berkeley and M. Co. 29 palms fixed/mobile experiment: Tracking vehicles with a uav-delivered sensor network, 2010. URL `http://robotics.eecs.berkeley.edu/~pister/29Palms0103/`.

[16] P. A. Bernstein and N. Goodman. Concurrency control in distributed database systems. *ACM Comput. Surv.*, 13(2):185–221, 1981. ISSN 0360-0300. doi: http://doi.acm.org/10.1145/356842.356846.

[17] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987. ISBN 0-201-10715-5.

[18] J. Beutel, O. Kasten, F. Mattern, K. Römer, F. Siegemund, and L. Thiele. Prototyping wireless sensor network applications with btnodes. In *EWSN*, pages 323–338, 2004.

[19] R. V. Biradar, V. C. Patil, S. R. Sawant, and R. R. Mudholkar. Classification and comparison of routing protocols in wireless sensor networks. *Ubiquitous Computing and Communication Journal*, Ubiquitous Computing Security Systems:704–711, 2009.

[20] J. Blumenthal, M. Handy, F. Golatowski, M. Haase, and D. Timmermann. Wireless sensor networks - new challenges in software engineering. In *Emerging Technologies and Factory Automation*, 2003. URL `citeseer.ist.psu.edu/718537.html`.

[21] C. Bobineau and M. Abdallah. A unilateral commit protocol for mobile and disconnected computing. In *in Proc. 12th International Conference on Parallel and Distributed Computing Systems (PDCS)*, 2000.

[22] P. Bonnet, J. Gehrke, and P. Seshadri. Towards sensor database systems. In *MDM '01: Proceedings of the Second International Conference on Mobile Data Management*, pages 3–14, London, UK, 2001. Springer-Verlag. ISBN 3-540-41454-1.

[23] J.-H. Bose, S. Bottcher, L. Gruenwald, S. Obermeier, H. Schweppe, and T. Steenweg. An integrated commit protocol for mobile network databases. In *IDEAS '05: Proceedings of the 9th International Database Engineering & Application Symposium*, pages 244–250, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2404-4. doi: http://dx.doi.org/10.1109/IDEAS.2005.11.

[24] P. Bose, P. Morin, I. Stojmenovic, and J. Urrutia. Routing with guaranteed delivery in ad hoc wireless networks. *Wireless Networks*, 7(6):609–616, 2001.

[25] S. Bottcher, L. Gruenwald, and S. Obermeier. A failure tolerating atomic commit protocol for mobile environments. In *MDM '07: Proceedings of the 2007 International Conference on Mobile Data Management*, pages 158–165, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 1-4244-1241-2. doi: http://dx.doi.org/10.1109/MDM.2007.31.

[26] D. Braginsky and D. Estrin. Rumor routing algorthim for sensor networks. In *WSNA '02: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 22–31, New York, NY, USA, 2002. ACM. ISBN 1-58113-589-0. doi: http://doi.acm.org/10.1145/570738.570742.

[27] A. Brayner and F. S. Alencar. A semantic-serializability based fully-distributed concurrency control mechanism for mobile multi-database systems. *Database and Expert Systems Applications, International Workshop on*, 0:1085–1089, 2005. ISSN 1529-4188.

[28] A. Brayner and J. A. Filho. Increasing mobile transaction concurrency in dynamically configurable environments. In *Proceedings of the Third International Workshop on Mobile Distributed Computing - Volume 06*, pages 637–641, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2328-5-06.

[29] Y. Breitbart, D. Georgakopoulos, M. Rusinkiewicz, and A. Silberschatz. On rigorous transaction scheduling. *IEEE Trans. Softw. Eng.*, 17:954–960, September 1991. ISSN 0098-5589. doi: 10.1109/32.92915. URL `http://portal.acm.org/citation.cfm?id=126262.126278`.

[30] J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, and J. Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *MobiCom '98: Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking*, pages 85–97, New York, NY, USA, 1998. ACM. ISBN 1-58113-035-X. doi: http://doi.acm.org/10.1145/288235.288256.

[31] J.-H. Böse. *Atomic Transaction Processing in Mobile Ad-Hoc Networks*. PhD thesis, Freie Universität Berlin, 2009.

[32] C. Buckl, S. Sommer, A. Scholz, A. Knoll, A. Kemper, J. Heuer, and A. Schmitt. Services to the field: An approach for resource constrained sensor/actor networks. In *Proceedings of the International Conference on Advanced Information Networking and Applications Workshops*, 2009.

[33] J. Burrell, T. Brooke, and R. Beckwith. Vineyard computing: Sensor networks in agricultural production. *IEEE Pervasive Computing*, 3:38–45, 2004. ISSN 1536-1268. doi: http://doi.ieeecomputersociety.org/10.1109/MPRV.2004.1269130.

[34] C. Buschmann. *Zeitliches und raeumliches Kontextbewusstsein in drahtlosen Sensornetzen*. PhD thesis, University of Luebeck, 2008.

[35] C. Buschmann and D. Pfisterer. isense: A modular hardware and software platform for wireless sensor networks. In *6. Fachgespräch Drahtlose Sensornetze der GI/ITG-Fachgruppe Kommunikation und Verteilte Systeme*, 2007.

[36] P. K. Chrysanthis. Transaction processing in mobile computing environment. In *Proceedings of the IEEE Workshop on Advances in Parallel and Distributed Systems*, 1993.

[37] Coalesenses. Coalesenses isense core module, 2009. http://www.coalesenses.com/.

[38] S. Corson and J. Macker. Mobile ad hoc networking (manet): Routing protocol performance issues and evaluation considerations, January 1999.

[39] N. Costa, A. Pereira, and C. Serôdio. A practical solution for automatic service discovery and usage over resource poor ad-hoc sensor networks. In *DASC '09: Proceedings of the 2009 Eighth IEEE International Conference on Dependable, Autonomic and Secure Computing*, pages 777–781, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-0-7695-3929-4. doi: http://dx.doi.org/10.1109/DASC.2009.158.

[40] i. Crossbow Technologies. Micaz, 2010. URL `http://www.cmt-gmbh.de/MICAz.pdf`.

[41] F. C. Delicato, P. F. Pires, L. Pirmez, and L. F. Carmo. A service approach for architecting application independent wireless sensor networks. *Cluster Computing*, 8(2-3):211–221, 2005. ISSN 1386-7857. doi: http://dx.doi.org/10.1007/s10586-005-6186-4.

[42] F. C. Delicato, P. F. Pires, L. Rust, L. Pirmez, and J. F. de Rezende. Reflective middleware for wireless sensor networks. In *SAC '05: Proceedings of the 2005 ACM symposium on Applied computing*, pages 1155–1159, New York, NY, USA, 2005. ACM. ISBN 1-58113-964-0. doi: http://doi.acm.org/10.1145/1066677.1066937.

[43] K. A. Delin, R. P. Harvey, N. A. Chabot, S. P. Jackson, M. Adams, D. W. Johnson, and J. T. Britton. Sensor web in antarctica: Developing an intelligent, autonomous platform for locating biological flourishes in cryogenic environments. In *Lunar and Planetary Science*, 2003.

[44] A. K. Dey. *Providing architectural support for building context-aware applications*. PhD thesis, Georgia Institute of Technology, Atlanta, GA, USA, 2000. Director-Abowd, Gregory D.

[45] Y. Diao, D. Ganesan, G. Mathur, and P. J. Shenoy. Rethinking data management for storage-centric sensor networks. In *CIDR*, pages 22–31. www.crdrdb.org, 2007.

[46] I. Dietrich and F. Dressler. On the lifetime of wireless sensor networks. *ACM Trans. Sen. Netw.*, 5(1):1–39, 2009. ISSN 1550-4859. doi: http://doi.acm.org/10.1145/1464420.1464425.

[47] R. A. Dirckze and L. Gruenwald. A toggle transaction management technique for mobile multidatabases. In *Proceedings of the seventh international conference on Information and knowledge management*, CIKM '98, pages 371–377, New York, NY, USA, 1998. ACM. ISBN 1-58113-061-9. doi: http://doi.acm.org/10.1145/288627.288679. URL `http://doi.acm.org/10.1145/288627.288679`.

[48] R. A. Dirckze and L. Gruenwald. A pre-serialization transaction management technique for mobile multidatabases. *Mob. Netw. Appl.*, 5(4):311–321, 2000. ISSN 1383-469X. doi: http://dx.doi.org/10.1023/A:1019133317760.

[49] R. Dube, C. D. Rais, K.-Y. Wang, and S. K. Tripathi. Signal stability based adaptive routing (ssa) for ad-hoc mobile networks. Technical report, University of Maryland, College Park, MD, USA, 1996.

[50] M. H. Dunham, A. Helal, and S. Balakrishnan. A mobile transaction model that captures both the data and movement behavior. *Mob. Netw. Appl.*, 2:149–162, October 1997. ISSN 1383-469X. doi: http://dx.doi.org/10.1023/A:1013672431080. URL `http://dx.doi.org/10.1023/A:1013672431080`.

[51] A. K. Elmagarmid. *Database Transaction Models for Advanced Applications*. Morgan Kaufmann, 1992. ISBN 1-55860-214-3.

[52] D. Estrin, D. Culler, K. Pister, and G. Sukhatme. Connecting the physical world with pervasive networks. *IEEE Pervasive Computing*, 1(1):59–69, 2002. ISSN 1536-1268. doi: http://dx.doi.org/10.1109/MPRV.2002.993145.

[53] K. P. Eswaran, J. N. Gray, R. A. Lorie, and I. L. Traiger. The notions of consistency and predicate locks in a database system. *Commun. ACM*, 19(11):624–633, 1976. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/360363.360369.

[54] EU. The european project cartalk 2000, 2010. URL `http://www.cartalk2000.net/`.

[55] S. P. Fekete, A. Kröller, S. Fischer, and D. Pfisterer. Shawn: The fast, highly customizable sensor network simulator. In *Proceedings of the 4th International Conference on Networked Sensing Systems (INSS 2007)*, page 299, 2007.

[56] C.-L. Fok, G.-C. Roman, and C. Lu. Agilla: A mobile agent middleware for self-adaptive wireless sensor networks. *ACM Trans. Auton. Adapt. Syst.*, 4(3):1–26, 2009. ISSN 1556-4665. doi: http://doi.acm.org/10.1145/1552297.1552299.

[57] W. Franz, H. Hartenstein, and M. Mauve, editors. *Inter-vehicle-communications based on ad hoc networking principles. The FleetNet Project*. KIT Scientific Publishing, 2005.

[58] D. G. Gaurav Mathur, Peter Desnoyers and P. Shenoy. Ultra-low power storage for sensor networks. In *Proceedings of IEEE/ACM Conference on Information Processing in Sensor Networks - Track on Sensor Platform, Tools and Design Methods for Networked Embedded Systems (IPSN-SPOTS)*, 2006.

[59] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler. The nesc language: A holistic approach to networked embedded systems. In *PLDI '03: Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*, 2003. URL `citeseer.ist.psu.edu/gay03nesc.html`.

[60] J. Gehrke and S. Madden. Query processing in sensor networks. *IEEE Pervasive Computing*, 3:46–55, 2004. ISSN 1536-1268. doi: http://doi.ieeecomputersociety.org/10.1109/MPRV.2004.1269131.

[61] D. K. Gifford. Weighted voting for replicated data. In *Proceedings of the seventh ACM symposium on Operating systems principles*, SOSP '79, pages 150–162, New York, NY, USA, 1979. ACM. ISBN 0-89791-009-5. doi: http://doi.acm.org/10.1145/800215.806583. URL `http://doi.acm.org/10.1145/800215.806583`.

[62] D. Goodman. *Wireless Personal Communications Systems*. Addison-Wesley, 1997.

[63] J. Gray. Notes on data base operating systems. In *Operating Systems, An Advanced Course*, pages 393–481, London, UK, 1978. Springer-Verlag. ISBN 3-540-08755-9.

[64] J. Gray. The transaction concept: virtues and limitations (invited paper). In *VLDB '1981: Proceedings of the seventh international conference on Very Large Data Bases*, pages 144–154. VLDB Endowment, 1981.

[65] J. Gray. Fault-tolerant distributed computing. chapter A comparison of the Byzantine agreement problem and the transaction commit problem, pages 10–17. Springer-Verlag, London, UK, 1990. ISBN 0-387-97385-0. URL `http://portal.acm.org/citation.cfm?id=106729.106731`.

[66] J. Gray and L. Lamport. Consensus on transaction commit. *ACM Trans. Database Syst.*, 31(1):133–160, 2006. ISSN 0362-5915. doi: http://doi.acm.org/10.1145/1132863.1132867.

[67] J. Gray, P. Helland, P. O'Neil, and D. Shasha. The dangers of replication and a solution. In *Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, SIGMOD '96, pages 173–182, New York, NY, USA, 1996. ACM. ISBN 0-89791-794-4. doi: http://doi.acm.org/10.1145/233269.233330. URL `http://doi.acm.org/10.1145/233269.233330`.

[68] L. Gruenwald, Y. Chen, and J. Huang. Effects of update techniques on main memory database system performance. *IEEE Trans. on Knowl. and Data Eng.*, 10(5):859–861, 1998. ISSN 1041-4347. doi: http://dx.doi.org/10.1109/69.729750.

[69] L. Guergen, C. Roncancio, C. Labbé, and V. Olive. Transactional issues in sensor data management. In *DMSN '06: Proceedings of the 3rd workshop on Data management for sensor networks*, pages 27–32, New York, NY, USA, 2006. ACM. doi: http://doi.acm.org/10.1145/1315903.1315910.

[70] Z. J. Haas, J. Y. Halpern, and L. Li. Gossip-based ad hoc routing. *IEEE/ACM Trans. Netw.*, 14(3):479–491, 2006. ISSN 1063-6692. doi: http://dx.doi.org/10.1109/TNET.2006.876186.

[71] T. Haerder. Observations on optimistic concurrency control schemes. *Inf. Syst.*, 9 (2):111–120, 1984. ISSN 0306-4379. doi: http://dx.doi.org/10.1016/0306-4379(84)90020-6.

[72] T. Hara and S. K. Madria. Consistency management strategies for data replication in mobile ad hoc networks. *IEEE Transactions on Mobile Computing*, 8(7):950–967, 2009. ISSN 1536-1233. doi: http://dx.doi.org/10.1109/TMC.2008.150.

[73] T. Hara, R. Hagihara, and S. Nishio. Data replication for top-k query processing in mobile wireless sensor networks. In *SUTC/UMC*, pages 115–122, 2010.

[74] J. R. Haritsa, M. J. Carey, and M. Livny. Dynamic real-time optimistic concurrency control. In *Proc. 11th IEEE Real-Time Systems Symp.*, 1991.

[75] T. He, S. Krishnamurthy, J. A. Stankovic, T. Abdelzaher, L. Luo, R. Stoleru, T. Yan, L. Gu, J. Hui, and B. Krogh. Energy-efficient surveillance system using wireless sensor networks. In *MobiSys '04: Proceedings of the 2nd international conference on Mobile systems, applications, and services*, pages 270–283, New York, NY, USA, 2004. ACM. ISBN 1-58113-793-1. doi: http://doi.acm.org/10.1145/990064.990096.

[76] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *HICSS '00: Proceedings of the 33rd Hawaii International Conference on System Sciences-Volume 8*, page 8020, Washington, DC, USA, 2000. IEEE Computer Society. ISBN 0-7695-0493-0.

[77] A. Helal, T.-H. Ku, R. Elmasri, and S. Mukherjee. Adaptive transaction scheduling. In *CIKM '93: Proceedings of the second international conference on Information and knowledge management*, pages 704–713, New York, NY, USA, 1993. ACM. ISBN 0-89791-626-3. doi: http://doi.acm.org/10.1145/170088.170462.

[78] H. Hellbrück and S. Fischer. Towards analysis and simulation of ad-hoc networks. In *Proceedings of the 2002 International Conference on Wireless Networks (ICWN02)*, 2002.

[79] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. *SIGOPS Oper. Syst. Rev.*, 34:93–104, November 2000. ISSN 0163-5980. doi: http://doi.acm.org/10.1145/384264.379006. URL `http://webs.cs.berkeley.edu/tos/papers/tos.pdf`.

[80] N. Hoeller, C. Reinke, S. Groppe, and V. Linnemann. Xobe sensor networks: Integrating xml in sensor network programming. In *Proceedings of the 5th International Conference on Networked Sensing Systems (INSS 2008)*, page 229, Kanazawa, Japan, June 17 - 19 2008. IEEE. ISBN 978-4-907764-31-9.

[81] N. Hoeller, C. Reinke, J. Neumann, S. Groppe, D. Boeckmann, and V. Linnemann. Efficient xml usage within wireless sensor networks. In X. Wang and N. B. Shroff, editors, *Proceedings of the Fourth International Wireless Internet Conference (WICON 2008)*, ACM International Conference Proceeding Series (AICPS), page Article No: 74, Maui, Hawaii, USA, November 17 - 19 2008. ACM. ISBN 978-963-9799-36-3. doi: http://doi.acm.org/10.1145/1554126.1554218.

[82] N. Hoeller, C. Reinke, J. Neumann, S. Groppe, C. Werner, and V. Linnemann. Xml data management and xpath evaluation in wireless sensor networks. In G. Kotsis, D. Taniar, E. Pardede, and I. Khalil, editors, *Proceedings of the 7th International Conference on Advances in Mobile Computig & Multimedia (MoMM 2009), held in conjunction with iiWAS 2009 conference*, pages 218–227, Kuala Lumpur, Malaysia, December 14 - 16 2009. ACM. ISBN 978-1-60558-659-5. doi: http://doi.acm.org/10.1145/1594187.1594193. This paper received the BEST PAPER AWARD MoMM 2009.

[83] N. Hoeller, C. Reinke, J. Neumann, S. Groppe, C. Werner, and V. Linnemann. Towards energy efficient xpath evaluation in wireless sensor networks. In *ACM International Conference Proceeding Series - Proceedings of the 6th International Workshop on Data Management for Sensor Networks (DMSN 2009)*, pages 1–2, Lyon, Frankreich, August 24 - 28 2009. ACM. ISBN 978-1-60558-777-6. doi: http://doi.acm.org/10.1145/1594187.1594193.

[84] N. Hoeller, C. Reinke, J. Neumann, S. Groppe, F. Frischat, and V. Linnemann. Dacs: A dynamic approximative data caching in wireless sensor networks. In *Proceedings of the 5th International Conference on Digital Information Management (ICDIM 2010)*, pages 339–346, Thunder Bay, Canada, July 5 - 8 2010. IEEE. ISBN 978-1-4244-7571-1.

[85] N. Hoeller, C. Reinke, J. Neumann, S. Groppe, M. Lipphardt, B. Schütt, and V. Linnemann. Stream-based xml template compression for wireless sensor network data management. In *Proceedings of the 4th International Conference on Multimedia and Ubiquitous Engineering (MUE 2010)*, pages 1–9, Cebu, Philippines, August 11 - 13 2010. IEEE. ISBN 978-1-4244-7566-7. doi: http://dx.doi.org/10.1109/MUE.2010.5575094. This paper received the BEST PAPER AWARD MUE 2010.

[86] N. Hoeller, C. Reinke, J. Neumann, S. Groppe, C. Werner, and V. Linnemann. Efficient xml data and query integration in the wireless sensor network engineering process. *International Journal of Web Information Systems*, 6(4):319–358, 2010. doi: http://dx.doi.org/10.1108/17440081011090248. URL `http://www.emeraldinsight.com/products/journals/journals.htm?id=ijwis`.

[87] H.-J. Hof, E.-O. Blaß, and M. Zitterbart. Secure overlay for service centric wireless sensor networks. In *ESAS*, pages 125–138, 2004.

[88] J. W. Hui and D. Culler. The dynamic behavior of a data dissemination protocol for network programming at scale. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, SenSys '04, pages 81–94, New York, NY,

USA, 2004. ACM. ISBN 1-58113-879-2. doi: http://doi.acm.org/10.1145/1031495.1031506. URL http://doi.acm.org/10.1145/1031495.1031506.

[89] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 56–67, New York, NY, USA, 2000. ACM. ISBN 1-58113-197-6. doi: http://doi.acm.org/10.1145/345910.345920.

[90] ISI. The network simulator - ns-2, 2010. URL http://www.isi.edu/nsnam/ns/.

[91] I. T. ISO/IEC JTC1. Open system interconnection (osi) reference model. ISO 7498, 1984.

[92] N. Jain. *Energy aware and adaptive routing protocols in wireless sensor networks*. PhD thesis, University of Cincinnati, Cincinnati, OH, USA, 2004. AAI3141345.

[93] P. Juang, H. Oki, Y. Wang, M. Martonosi, L. S. Peh, and D. Rubenstein. Energy-efficient computing for wildlife tracking: design tradeoffs and early experiences with zebranet. *SIGARCH Comput. Archit. News*, 30(5):96–107, 2002. ISSN 0163-5964. doi: http://doi.acm.org/10.1145/635506.605408.

[94] J. M. Kahn, R. H. Katz, and K. S. J. Pister. Next century challenges: Mobile networking for "smart dust". In *MOBICOM*, pages 271–278, 1999.

[95] H. Karl and A. Willig. A short survey of wireless sensor networks. Technical report, Hasso-Plattner Institute, Potsdam, 2003.

[96] B. Karp and H. T. Kung. Gpsr: greedy perimeter stateless routing for wireless networks. In *Proceedings of the 6th annual international conference on Mobile computing and networking*, MobiCom '00, pages 243–254, New York, NY, USA, 2000. ACM. ISBN 1-58113-197-6. doi: http://doi.acm.org/10.1145/345910.345953. URL http://doi.acm.org/10.1145/345910.345953.

[97] M. Kaur, S. Bhatt, L. S. Golden, and G. R. Iii. An efficient protocol for service discovery in wireless sensor networks. In *Proceedings of the GLOBECOMM Workshops*, 2008.

[98] D. Kotz, C. Newport, and C. Elliot. The mistaken axioms of wireless-network research. Technical report, Dartmouth College Computer Science, 2003.

[99] K.-I. Ku and Y.-S. Kim. Moflex transaction model for mobile heterogeneous multidatabase systems. In *RIDE '00: Proceedings of the 10th International Workshop on Research Issues in Data Engineering*, page 39, Washington, DC, USA, 2000. IEEE Computer Society. ISBN 0-7695-0531-7.

[100] F. Kuhn, R. Wattenhofer, and A. Zollinger. Ad hoc networks beyond unit disk graphs. *Wirel. Netw.*, 14(5):715–729, 2008. ISSN 1022-0038. doi: http://dx.doi.org/10.1007/s11276-007-0045-6.

[101] J. Kulik, W. Heinzelman, and H. Balakrishnan. Negotiation-based protocols for disseminating information in wireless sensor networks. *Wirel. Netw.*, 8(2/3):169–185, 2002. ISSN 1022-0038. doi: http://dx.doi.org/10.1023/A:1013715909417.

[102] V. Kumar, N. Prabhu, M. H. Dunham, and A. Y. Seydim. Tcot-a timeout-based mobile transaction commitment protocol. *IEEE Trans. Comput.*, 51(10):1212–1218, 2002. ISSN 0018-9340. doi: http://dx.doi.org/10.1109/TC.2002.1039846.

[103] H. T. Kung and J. T. Robinson. On optimistic methods for concurrency control. *ACM Trans. Database Syst.*, 6(2):213–226, 1981. ISSN 0362-5915. doi: http://doi.acm.org/10.1145/319566.319567.

[104] S. Kurkowski, T. Camp, and M. Colagrosso. Manet simulation studies: the incredibles. *SIGMOBILE Mob. Comput. Commun. Rev.*, 9(4):50–61, 2005. ISSN 1559-1662. doi: http://doi.acm.org/10.1145/1096166.1096174.

[105] M. Kushwaha, I. Amundson, X. Koutsoukos, S. Neema, and J. Sztipanovits. Oasis: A programming framework for service-oriented sensor networks. *IEEE/Create-Net COMSWARE 2007*, 2007.

[106] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/359545.359563.

[107] L. Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16:133–169, May 1998. ISSN 0734-2071. doi: http://doi.acm.org/10.1145/279227.279229. URL `http://doi.acm.org/10.1145/279227.279229`.

[108] L. Lamport. Paxos made simple. *ACM SIGACT News*, 32(4):18–25, 2001.

[109] K. Lee, J. eon Kim, D. T. Thuy, D. Kim, S. Ahn, and J. Yang. Multi-hop network re-programming model for wireless sensor networks. In *Proceedings of the 4th IEEE Conference on Sensors*, 2006.

[110] M. Lee and S. Helal. Hicomo: High commit mobile transactions. *Distrib. Parallel Databases*, 11:73–92, January 2002. ISSN 0926-8782. doi: 10.1023/A:1013381624108. URL `http://portal.acm.org/citation.cfm?id=509176.509180`.

[111] V. C. S. Lee and K.-W. Lam. Optimistic concurrency control in broadcast environments: Looking forward at the server and backward at the clients. In *MDA '99:*

Proceedings of the First International Conference on Mobile Data Access, pages 97–106, London, UK, 1999. Springer-Verlag. ISBN 3-540-66878-0.

[112] V. C. S. Lee, K.-W. Lam, S. H. Son, and E. Y. M. Chan. On transaction processing with partial validation and timestamp ordering in mobile broadcast environments. *IEEE Trans. Comput.*, 51(10):1196–1211, 2002. ISSN 0018-9340. doi: http://dx.doi.org/10.1109/TC.2002.1039845.

[113] P. Levis, N. Patel, D. Culler, and S. Shenker. Trickle: a self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In *NSDI'04: Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation*, pages 2–2, Berkeley, CA, USA, 2004. USENIX Association.

[114] P. Levis, E. Brewer, D. Culler, D. Gay, S. Madden, N. Patel, J. Polastre, S. Shenker, R. Szewczyk, and A. Woo. The emergence of a networking primitive in wireless sensor networks. *Commun. ACM*, 51(7):99–106, 2008. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/1364782.1364804.

[115] X. Li, N. Santoro, and I. Stojmenovic. Localized distance-sensitive service discovery in wireless sensor networks. In *FOWANC '08: Proceeding of the 1st ACM international workshop on Foundations of wireless ad hoc and sensor networking and computing*, pages 85–92, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-149-1. doi: http://doi.acm.org/10.1145/1374718.1374734.

[116] Y. Li, Y. Xiong, L. Zhou, and R. Zhu. Adaptive optimization-based routing in wireless mesh networks. *Wirel. Pers. Commun.*, 56:403–415, February 2011. ISSN 0929-6212. doi: http://dx.doi.org/10.1007/s11277-010-9979-6. URL `http://dx.doi.org/10.1007/s11277-010-9979-6`.

[117] . Lin, . Arai, and . Gunopulos. Reliable hierarchical data storage in sensor networks. *ssdbm*, 00:26, 2007. ISSN 1551-6393. doi: http://doi.ieeecomputersociety.org/10.1109/SSDBM.2007.39.

[118] Y. Lin, B. Li, and B. Liang. Differentiated data persistence with priority random linear codes. page 47, 2007. doi: http://dx.doi.org/10.1109/ICDCS.2007.99.

[119] Y. Lin, B. Liang, and B. Li. Data persistence in large-scale sensor networks with decentralized fountain codes. *IEEE INFOCOM 2007. 26th IEEE International Conference on Computer Communications.*, 2007.

[120] Y. Lin, B. Liang, and B. Li. Data persistence in large-scale sensor networks. 2007. URL `http://www.eecg.toronto.edu/~bli/papers/ylin-infocom07.pdf`.

[121] M. Lipphardt. *Service-orientierte Infrastrukturen und Algorithmen fuer praxistaugliche Sensornetzanwendungen*. PhD thesis, University of Luebeck, 2009.

[122] M. Lipphardt, H. Hellbrück, D. Pfisterer, S. Ransom, and S. Fischer. Practical experiences on mobile inter-body-area-networking. In *Proceedings of the Second International Conference on Body Area Networks (BodyNets'07)*, 2007.

[123] M. Lipphardt, J. Neumann, N. Hoeller, C. Reinke, S. Groppe, V. Linnemann, and C. Werner. Xml und soa als wegbereiter fuer sensornetze in der praxis. *Praxis der Informationsverarbeitung und Kommunikation (PIK)*, 31(3):146–152, Juli - September 2008. URL `http://www.reference-global.com/loi/piko?cookieSet=1`.

[124] M. Lipphardt, N. Glombitza, J. Neumann, and C. Werner. A service-oriented operating system and an application development infrastructure for wireless sensor networks. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems (SenSys 2009)*, pages 309–310, Berkeley, California, November 4 - 6 2009. ACM. ISBN 978-1-60558-519-2. doi: http://doi.acm.org/10.1145/1644038.1644075.

[125] Q. Lu and M. Satyanaranyanan. Isolation-only transactions for mobile computing. *SIGOPS Oper. Syst. Rev.*, 28:81–87, April 1994. ISSN 0163-5980. doi: http://doi.acm.org/10.1145/198153.198164. URL `http://doi.acm.org/10.1145/198153.198164`.

[126] S. Madden. *The Design and Evaluation of a Query Processing Architecture for Sensor Networks*. PhD thesis, MIT, 2003.

[127] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tinydb: an acquisitional query processing system for sensor networks. *ACM Trans. Database Syst.*, 30(1):122–173, 2005. ISSN 0362-5915. doi: http://doi.acm.org/10.1145/1061318.1061322.

[128] S. K. Madria and B. Bhargava. A transaction model to improve data availability in mobile computing. *Distrib. Parallel Databases*, 10:127–160, September 2001. ISSN 0926-8782. doi: 10.1023/A:1019232412740. URL `http://portal.acm.org/citation.cfm?id=568960.568962`.

[129] R. Marin-Perianu, H. Scholten, P. Havinga, and P. Hartel. Energy-efficient cluster-based service discovery in wireless sensor networks. *Local Computer Networks, Annual IEEE Conference on*, 0:931–938, 2006. ISSN 0742-1303. doi: http://doi.ieeecomputersociety.org/10.1109/LCN.2006.322202.

[130] R. Marin-Perianu, H. Scholten, and P. Havinga. Prototyping service discovery and usage in wireless sensor networks. *IEEE Conference on Local Computer Networks*

(LCN), 0:841–850, 2007. ISSN 0742-1303. doi: http://doi.ieeecomputersociety.org/10.1109/LCN.2007.133.

[131] K. Martinez, R. Ong, J. K. Hart, and J. Stefanov. Glacsweb: A sensor web for glaciers. In *Adjunct Proceedings EWSN 2004*, 2004.

[132] A. N. Mian, R. Baldoni, and R. Beraldi. A survey of service discovery protocols in multihop mobile ad hoc networks. *IEEE Pervasive Computing*, 8:66–74, 2009. ISSN 1536-1268. doi: http://doi.ieeecomputersociety.org/10.1109/MPRV.2009.2.

[133] R. Müller, G. Alonso, and D. Kossmann. A virtual machine for sensor networks. *The Proceedings of the EuroSys 2007 Conference*, 2007.

[134] R. Müller, G. Alonso, and D. Kossmann. Swissqm: Next generation data processing in sensor networks. In *CIDR*, pages 1–9. www.crdrdb.org, 2007.

[135] C. Mohan and B. Lindsay. Efficient commit protocols for the tree of processes model of distributed transactions. *SIGOPS Oper. Syst. Rev.*, 19:40–52, April 1985. ISSN 0163-5980. doi: http://doi.acm.org/10.1145/850770.850772. URL `http://doi.acm.org/10.1145/850770.850772`.

[136] C. Mohan, B. Lindsay, and R. Obermarck. Transaction management in the r* distributed database management system. *ACM Trans. Database Syst.*, 11(4):378–396, 1986. ISSN 0362-5915. doi: http://doi.acm.org/10.1145/7239.7266.

[137] J. E. Moss. *Nested transactions: an approach to reliable distributed computing*. Massachusetts Institute of Technology, Cambridge, MA, USA, 1985. ISBN 0-262-13200-1.

[138] L. Nachman, R. Kling, R. Adler, J. Huang, and V. Hummel. The intel mote platform: a bluetooth-based sensor network for industrial monitoring. In *IPSN '05: Proceedings of the 4th international symposium on Information processing in sensor networks*, page 61, Piscataway, NJ, USA, 2005. IEEE Press. ISBN 0-7803-9202-7.

[139] C.-S. Nam, H.-J. Jeong, and D.-R. Shin. The adaptive cluster head selection in wireless sensor networks. *Semantic Computing and Applications, IEEE International Workshop on*, 0:147–149, 2008. doi: http://doi.ieeecomputersociety.org/10.1109/IWSCA.2008.14.

[140] J. Neumann, C. Reinke, N. Hoeller, and V. Linnemann. Adaptive quality-aware replication in wireless sensor networks. In *Communication and Networking, Proceedings of the 2009 International Workshop on Wireless Ad Hoc, Mesh and Sensor Networks (WAMSNET09), held in conjunction with the 2009 International Conference on Future Generation Communication and Networking(FGCN 2009)*, volume 56 of *Communications in Computer and Information Science (CCIS)*, pages 413–420,

Jeju Island, Korea, Dezember 10 - 12 2009. Springer. ISBN 978-3-642-10843-3 (Print), 978-3-642-10844-0 (Online). doi: http://www.springerlink.com/content/v8n275618q8l3677/.

[141] J. Neumann, N. Hoeller, C. Reinke, and V. Linnemann. Redundancy infrastructure for service-oriented wireless sensor networks. In *Proceedings of the 9th International Symposium on Network Computing and Applications (NCA 2010)*, pages 269–274, Cambridge, Massachusetts, USA, July 15 - 17 2010. IEEE. ISBN 978-0-7695-4118-1.

[142] C. E. Nishimura and D. M. Conlon. Iuss dual use: Monitoring whales and earthquakes using sosus. *Marine Technology Society Journal*, 27(4):13–21, 1994. URL `http://www.pmel.noaa.gov/vents/acoustics/sosus.html`.

[143] N. Nouali, A. Doucet, and H. Drias. A two-phase commit protocol for mobile wireless environment. In *ADC '05: Proceedings of the 16th Australasian database conference*, pages 135–143, Darlinghurst, Australia, Australia, 2005. Australian Computer Society, Inc. ISBN 1-920-68221-X.

[144] N. Nouali-Taboudjemat, F. Chehbour, and H. Drias. On performance evaluation and design of atomic commit protocols for mobile transactions. *Distrib. Parallel Databases*, 27(1):53–94, 2010. ISSN 0926-8782. doi: http://dx.doi.org/10.1007/s10619-009-7055-6.

[145] P. F. Nuno Santos, Luis Veiga. Transaction policies for mobile networks. In *POLICY '04: Proceedings of the Fifth IEEE International Workshop on Policies for Distributed Systems and Networks*, page 55, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2141-X.

[146] OASIS. Reference model for service oriented architecture 1.0, August 2006. URL `http://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf`.

[147] OASIS. Web services atomic transaction (ws-atomictransaction), 2009. URL `http://docs.oasis-open.org/ws-tx/wsat/2006/06`.

[148] S. Obermeier. *Database Transaction Management in Mobile Ad-Hoc Networks*. PhD thesis, University of Paderborn, 2008.

[149] S. Obermeier and S. Boettcher. Avoiding infinite blocking of mobile transactions. In *IDEAS*, pages 63–71, 2007.

[150] S. Obermeier, J.-H. Böse, S. Böttcher, P. K. Chrysanthis, A. Delis, L. Gruenwald, A. Mondal, A. Ouksel, G. Samaras, and S. Viglas. Atomicity in mobile networks. In S. Böttcher, L. Gruenwald, P. J. Marrón, and E. Pitoura, editors, *Scalable Data*

Management in Evolving Networks, number 06431 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2007. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany. URL http://drops.dagstuhl.de/opus/volltexte/2007/952.

[151] S. Obermeier, S. Boettcher, and D. Kleine. Clcp - a distributed cross-layer commit protocol for mobile ad-hoc networks. In *Proceedings of the 2008 IEEE International Symposium on Parallel and Distributed Processing with Applications*, volume 0, pages 361–370, Los Alamitos, CA, USA, 2008. IEEE Computer Society. ISBN 978-0-7695-3471-8. doi: http://doi.ieeecomputersociety.org/10.1109/ISPA.2008.31.

[152] S. Obermeier, S. Böttcher, and D. Kleine. A cross-layer atomic commit protocol implementation for transaction processing in mobile ad-hoc networks. *Distributed and Parallel Databases*, 26(2-3):319–351, 2009.

[153] J. Paradiso, G. Borriello, L. Girod, and R. Han. Applications: Beyond dumb data collection (panel). EmNets, 2006.

[154] J. Payton, C. Julien, and G.-C. Roman. Automatic consistency assessment for query results in dynamic environments. In *ESEC-FSE '07: Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 245–254, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-811-4. doi: http://doi.acm.org/10.1145/1287624.1287659.

[155] C. E. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers. *SIGCOMM Comput. Commun. Rev.*, 24(4):234–244, 1994. ISSN 0146-4833. doi: http://doi.acm.org/10.1145/190809.190336.

[156] C. E. Perkins and E. M. Royer. Ad-hoc on-demand distance vector routing. *Mobile Computing Systems and Applications, IEEE Workshop on*, 0:90, 1999. doi: http://doi.ieeecomputersociety.org/10.1109/MCSA.1999.749281.

[157] D. Pfisterer. *Comprehensive Development Support for Wireless Sensor Networks*. PhD thesis, University of Luebeck, 2007.

[158] E. Pitoura and B. Bhargava. Maintaining consistency of data in mobile distributed environments. *Distributed Computing Systems, International Conference on*, 0:0404, 1995. doi: http://doi.ieeecomputersociety.org/10.1109/ICDCS.1995.500045.

[159] E. Pitoura and B. Bhargava. Data consistency in intermittently connected distributed systems. *IEEE Trans. on Knowl. and Data Eng.*, 11:896–915, November 1999. ISSN 1041-4347. doi: http://dx.doi.org/10.1109/69.824602. URL http://dx.doi.org/10.1109/69.824602.

[160] J. Polastre, R. Szewczyk, and D. Culler. Telos: enabling ultra-low power wireless research. In *IPSN '05: Proceedings of the 4th international symposium on Information processing in sensor networks*, page 48, Piscataway, NJ, USA, 2005. IEEE Press. ISBN 0-7803-9202-7.

[161] J. M. Prinsloo, C. L. Schulz, D. G. Kourie, W. H. M. Theunissen, T. Strauss, R. V. D. Heever, and S. Grobbelaar. A service oriented architecture for wireless sensor and actor network applications. In *SAICSIT '06: Proceedings of the 2006 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries*, pages 145–154, , Republic of South Africa, 2006. South African Institute for Computer Scientists and Information Technologists. ISBN 1-59593-567-3. doi: http://doi.acm.org/10.1145/1216262.1216278.

[162] E. Rahm. *Mehrrechner-Datenbanksysteme - Grundlagen der verteilten und parallelen Datenbankverarbeitung*. Addison-Wesley, 1994. ISBN 3-89319-702-8. URL `http://dbs.uni-leipzig.de/buecher/mrdbs/index.html`.

[163] Raumcomputer, 2010. URL `www.raumcomputer.com`.

[164] Y. Raz. The principle of commitment ordering, or guaranteeing serializability in a heterogeneous environment of multiple autonomous resource mangers using atomic commitment. In *VLDB '92: Proceedings of the 18th International Conference on Very Large Data Bases*, pages 292–312, San Francisco, CA, USA, 1992. Morgan Kaufmann Publishers Inc. ISBN 1-55860-151-1.

[165] C. Reinke. Adaptive service migration and transaction processing in wireless sensor networks. In *Proceedings of the 7th Middleware Doctoral Symposium (MDS 2010)*, pages 8–13, Bangalore, India, November 29 - December 03 2010. ACM. ISBN 978-1-4503-0457-3. doi: http://doi.acm.org/10.1145/1891748.1891750.

[166] C. Reinke, N. Hoeller, M. Lipphardt, J. Neumann, S. Groppe, and V. Linnemann. Integrating standardized transaction protocols in service oriented wireless sensor networks. In *Proceedings of the 24th ACM Symposium on Applied Computing*, pages 2202–2203, Honolulu, Hawaii, USA, March 8 - 12 2009. ACM. ISBN 978-1-60558-166-8. doi: http://doi.acm.org/10.1145/1529282.1529768.

[167] C. Reinke, N. Hoeller, and V. Linnemann. Adaptive atomic transaction support for service migration in wireless sensor networks. In *Proceedings of the Seventh IEEE and IFIP International Conference on Wireless and Optical Communications Networks (WOCN2010)*, Colombo, Sri Lanka, September 6 - 8 2010. IEEE. ISBN 978-1-4244-7203-1. doi: http://dx.doi.org/10.1109/WOCN.2010.5587366.

[168] C. Reinke, N. Hoeller, J. Neumann, S. Groppe, S. Werner, and V. Linnemann. Analysis and comparison of atomic commit protocols for adaptive usage in wireless sensor

networks. In *Proceedings of the 2010 IEEE International Conference on Sensor Networks, Ubiquitous and Trustworthy Computing (SUTC 2010)*, pages 138–145, Newport Beach, California, USA, June 7 - 9 2010. IEEE. ISBN 978-0-7695-4049-8. doi: http://doi.ieeecomputersociety.org/10.1109/SUTC.2010.12.

[169] C. Reinke, N. Hoeller, S. Werner, S. Groppe, and V. Linnemann. Consistent service migration in wireless sensor networks. In *Proceedings of the 2011 International Conference on Wireless and Optical Communications (ICWOC 2011)*, 2011.

[170] O. Riva, T. Nadeem, C. Borcea, and L. Iftode. Context-aware migratory services in ad hoc networks. *IEEE Transactions on Mobile Computing*, 6(12):1313–1328, 2007. ISSN 1536-1233. doi: http://dx.doi.org/10.1109/TMC.2007.1053.

[171] K. Römer, P. Blum, and L. Meier:. *Handbook of Sensor Networks: Algorithms and Architectures*, chapter Time Synchronization and Calibration in Wireless Sensor Network, pages 199–237. Wiley and Sons, 2005.

[172] K. Römer. Programming paradigms and middleware for sensor networks. In *GI/ITG Workshop on Sensor Networks*, 2004. URL `http://www.vs.inf.ethz.ch/publ/papers/middleware-kuvs.pdf`.

[173] K. Römer. *Time Synchronization and Localization in Sensor Networks*. PhD thesis, Swiss Federal Institute of Technology Zurich (ETH Zurich), 2005.

[174] D. J. Rosenkrantz, R. E. Stearns, and P. M. Lewis, II. System level concurrency control for distributed database systems. *ACM Trans. Database Syst.*, 3:178–198, June 1978. ISSN 0362-5915. doi: http://doi.acm.org/10.1145/320251.320260. URL `http://doi.acm.org/10.1145/320251.320260`.

[175] R. Rouvoy, P. Serrano-Alvarado, and P. Merle. Towards context-aware transaction services. In *DAIS*, pages 272–288, 2006.

[176] F. Sailhan and V. Issarny. Scalable service discovery for manet. In *PERCOM '05: Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications*, pages 235–244, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2299-8. doi: http://dx.doi.org/10.1109/PERCOM.2005.36.

[177] G. Schiele, C. Becker, and K. Rothermel. Energy-efficient cluster-based service discovery for ubiquitous computing. In *EW 11: Proceedings of the 11th workshop on ACM SIGOPS European workshop*, page 14, New York, NY, USA, 2004. ACM. doi: http://doi.acm.org/10.1145/1133572.1133604.

[178] G. Schlageter. Problems of optimistic concurrency control in distributed database systems. In *ACM SIGMOD Record*, volume 12, pages 62–66, New York, NY, USA, 1982. ACM. doi: http://doi.acm.org/10.1145/984505.984510.

[179] C. Scholliers, T. Van Cutsem, and W. De Meuter. Ambient transactors. In *MPAC '08: Proceedings of the 6th international workshop on Middleware for pervasive and ad-hoc computing*, pages 49–53, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-364-8. doi: http://doi.acm.org/10.1145/1462789.1462798.

[180] P. Serrano-Alvarado. *Transactions adaptables pour les environments mobiles*. PhD thesis, Fourier University, Grenoble, France, 2004.

[181] P. Serrano-Alvarado, C. Roncancio, and M. Adiba. A survey of mobile transactions. *Distrib. Parallel Databases*, 16(2):193–230, 2004. ISSN 0926-8782. doi: http://dx.doi.org/10.1023/B:DAPD.0000028552.69032.f9.

[182] P. Serrano-Alvarado, R. Rouvoy, and P. Merle. Self-adaptive component-based transaction commit management. In *ARM '05: Proceedings of the 4th workshop on Reflective and adaptive middleware systems*, New York, NY, USA, 2005. ACM. ISBN 1-59593-270-4. doi: http://doi.acm.org/10.1145/1101516.1101527.

[183] S. D. Servetto and G. Barrenechea. Constrained random walks on random graphs: routing algorithms for large scale wireless sensor networks. In *WSNA '02: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 12–21, New York, NY, USA, 2002. ACM. ISBN 1-58113-589-0. doi: http://doi.acm.org/10.1145/570738.570741.

[184] K. Sha, W. Shi, and O. Watkins. Using wireless sensor networks for fire rescue applications: Requirements and challenges. In *IEEE International Conference on Electro/information Technology*, 2006.

[185] G. Simon, M. Maróti, A. Lédeczi, G. Balogh, B. Kusy, A. Nádas, G. Pap, J. Sallai, and K. Frampton. Sensor network-based countersniper system. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 1–12, New York, NY, USA, 2004. ACM. ISBN 1-58113-879-2. doi: http://doi.acm.org/10.1145/1031495.1031497.

[186] D. Skeen. Nonblocking commit protocols. In *SIGMOD '81: Proceedings of the 1981 ACM SIGMOD international conference on Management of data*, pages 133–142, New York, NY, USA, 1981. ACM. ISBN 0-89791-040-0. doi: http://doi.acm.org/10.1145/582318.582339.

[187] D. Skeen and M. Stonebraker. A formal model of crash recovery in a distributed system. *IEEE Transactions on Software Engineering*, vol 9, no.3:pp.219–228., 1983.

[188] S. Sommer, A. Scholz, I. Gaponova, A. Knoll, A. Kemper, C. Buckl, G. Kainz, J. Heuer, and A. Schmitt. Service migration scenarios for embedded networks. In

Advanced Information Networking and Applications Workshops, International Conference on, volume 0, pages 502–507, Los Alamitos, CA, USA, 2010. IEEE Computer Society. ISBN 978-0-7695-4019-1. doi: http://doi.ieeecomputersociety.org/10.1109/WAINA.2010.84.

[189] I. Sun Microsystems. Rpc: Remote procedure call protocol specification version 2. Network Working Group Request For Comments: 1057, June 1988. URL `http://www.ietf.org/rfc/rfc1057.txt`.

[190] V. Sundramoorthy, P. Hartel, and J. Scholten. A taxonomy of self-configuring service discovery systems. Technical report, Centre for Telematics and Information Technology, University of Twente, 2007.

[191] A. R. G. U. S. A. Systems, 2010. URL `http://www.globalsecurity.org/intell/systems/arguss.htm`.

[192] R. Szewczyk, A. Mainwaring, J. Polastre, J. Anderson, and D. Culler. An analysis of a large scale habitat monitoring application. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 214–226, New York, NY, USA, 2004. ACM. ISBN 1-58113-879-2. doi: http://doi.acm.org/10.1145/1031495.1031521.

[193] F. L. Tang, M. L. Li, M. Y. Guo, and I. You. An adaptive context-aware transaction model for mobile and ubiquitous computing. *Computing and Informatics*, 27(5):785–798, 2008.

[194] Y.-C. Tseng, S.-Y. Ni, Y.-S. Chen, and J.-P. Sheu. The broadcast storm problem in a mobile ad hoc network. *Wireless Networks*, 8(2/3):153–167, 2002. ISSN 1022-0038. doi: http://dx.doi.org/10.1023/A:1013763825347.

[195] C. Tuerker and G. Zini. A survey of academic and commercial approaches to transaction support in mobile computing environments. Technical Report 429, ETH Zürich, Institute of Information Systems, November 2003.

[196] G. D. Walborn and P. K. Chrysanthis. Supporting semantics-based transaction processing in mobile database applications. In *Proceedings of the 14TH Symposium on Reliable Distributed Systems*, SRDS '95, pages 31–, Washington, DC, USA, 1995. IEEE Computer Society. ISBN 0-8186-7153-X. URL `http://portal.acm.org/citation.cfm?id=829520.830874`.

[197] G. D. Walborn and P. K. Chrysanthis. Pro-motion: management of mobile transactions. In *Proceedings of the 1997 ACM symposium on Applied computing*, SAC '97, pages 101–108, New York, NY, USA, 1997. ACM. ISBN 0-89791-850-9. doi: http://doi.acm.org/10.1145/331697.331718. URL `http://doi.acm.org/10.1145/331697.331718`.

[198] P. Wang and T. Wang. Adaptive routing for sensor networks using reinforcement learning. In *Proceedings of the Sixth IEEE International Conference on Computer and Information Technology*, CIT '06, pages 219–, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2687-X. doi: http://dx.doi.org/10.1109/CIT.2006.34. URL `http://dx.doi.org/10.1109/CIT.2006.34`.

[199] Q. Wang, Y. Zhu, and L. Cheng. Reprogramming wireless sensor networks: challenges and approaches. *IEEE Network*, 20(3):48–55, 2006.

[200] M. Weiser. The computer for the 21st century. *Scientific American*, 265(3):91–104, September 1991.

[201] E. H.-K. Wu and Y.-Z. Huang. Dynamic adaptive routing for a heterogeneous wireless network. *Mob. Netw. Appl.*, 9:219–233, June 2004. ISSN 1383-469X. doi: http://dx.doi.org/10.1145/1012113.1012119. URL `http://dx.doi.org/10.1145/1012113.1012119`.

[202] W. Xie. *Supporting Distributed Transaction Processing Over Mobile and Heterogeneous Platforms*. PhD thesis, Georgia Institute of Technology, 2005.

[203] Z. Xing, L. Gruenwald, and S. Song. An optimistic concurrency control algorithm for mobile ad-hoc network databases. In *Proceedings of the Fourteenth International Database Engineering & Applications Symposium*, IDEAS '10, pages 199–204, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-900-8.

[204] F. Xue and P. R. Kumar. The number of neighbors needed for connectivity of wireless networks. *Wirel. Netw.*, 10(2):169–181, 2004. ISSN 1022-0038. doi: http://dx.doi.org/10.1023/B:WINE.0000013081.09837.c0.

[205] Y. Yao and J. Gehrke. The cougar approach to in-network query processing in sensor networks. *SIGMOD Rec.*, 31(3):9–18, 2002. ISSN 0163-5808. doi: http://doi.acm.org/10.1145/601858.601861.

[206] D. Y. Ye, M. C. Lee, and T. I. Wang. Mobile agents for distributed transactions of a distributed heterogeneous database system. In *DEXA '02: Proceedings of the 13th International Conference on Database and Expert Systems Applications*, pages 403–412, London, UK, 2002. Springer-Verlag. ISBN 3-540-44126-3.

[207] Y. Yu, L. J. Rittle, V. Bhandari, and J. B. LeBrun. Supporting concurrent applications in wireless sensor networks. In *Proceedings of the 4th international conference on Embedded networked sensor systems*, SenSys '06, pages 139–152, New York, NY, USA, 2006. ACM. ISBN 1-59593-343-3. doi: http://doi.acm.org/10.1145/1182807.1182822. URL `http://doi.acm.org/10.1145/1182807.1182822`.

[208] Y. Zaslavsky, L. H. Yeo, and A. Zaslavsky. Submission of transactions from mobile workstations in a cooperative multidatabase processing environment. In *In Proceedings of the 14th International Conference on Distributed Computing Systems*, pages 372–379, 1994.

[209] G. Zhou, T. He, S. Krishnamurthy, and J. A. Stankovic. Impact of radio irregularity on wireless sensor networks. In *MobiSys '04: Proceedings of the 2nd international conference on Mobile systems, applications, and services*, pages 125–138, New York, NY, USA, 2004. ACM. ISBN 1-58113-793-1. doi: http://doi.acm.org/10.1145/990064.990081.

[210] X.-l. Zhou and M. Wu. Service discovery protocol in wireless sensor networks. In *SKG '06: Proceedings of the Second International Conference on Semantics, Knowledge, and Grid*, page 101, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2673-X. doi: http://dx.doi.org/10.1109/SKG.2006.92.

[211] M. Zuniga and B. Krishnamachari. Analyzing the transitional region in low power wireless links. In *In First IEEE International Conference on Sensor and Ad hoc Communications and Networks (SECON)*, pages 517–526, 2004.