An Integrated Commit Protocol For Mobile Network Databases

Joos Hendrik Böse[2] **Stefan Böttcher**[1] **Le Gruenwald**[3]
Sebastian Obermeier[1] **Heinz Schweppe**[2] **Thorsten Steenweg**[1]

[1] University of Paderborn, Computer Science
Fürstenallee 11; 33102 Paderborn; Germany
{stb, so, steenweg}@uni-paderborn.de

[2] Freie Universität Berlin, Institute of Computer Science
Takustr. 9; 14195 Berlin; Germany
{boese, schweppe}@mi.fu-berlin.de

[3] The University of Oklahoma, School of Computer Science
200 Telgar street, Room 116 EL; Norman, OK 73019-3032; USA
{ggruenwald}@ou.edu

Abstract

While traditional fixed-wired network protocols like 2-Phase-Commit guarantee atomicity, we cannot use them in mobile low bandwidth networks where network partitioning, node failure, and message loss may result in blocking. To deploy traditional database applications easily into a mobile environment, there is a demand for a protocol which guarantees an atomic commit of transactions. This paper introduces a protocol which can guarantee such atomic commitment in mobile environments using a combination of commit and consensus protocols. In addition, it takes advantage of mobile network sub-structures like single-hop-environments to reduce message transfer costs.

1 Introduction

Whenever database technology shall be applied to a network of mobile devices, we face a lot of new challenges including guaranteeing transaction atomicity in the presence of lost connections and node failures. This atomic transaction commitment is necessary to deal with concurrent transactions in complex systems like distributed databases, peer-to-peer systems, and service oriented architectures.

The problems like message delay, disconnection of nodes, and network partitioning have been studied and a number of solutions have been proposed to solve these problems. However, in these solutions there is either a large number of messages slowing down the whole system, or the assumptions for the network are so strict that they cannot always be guaranteed (e.g. we cannot always assume a maximum percentage of messages that will not be lost).

For practical applications, we need a non-blocking atomic commit protocol that not only reduces the number of exchange messages but also handles network partitioning and takes ad-

vantage of single hop communication which exists in some network structures. In this paper, we present such an atomic commit protocol for non-compensatable transactions that would not block connected and running nodes.

The rest of the paper is organized as follows. Section 2 describes our assumed system architecture. Section 3 presents our commit protocol and its costs in terms of number of messages. Section 4 discusses related work. Finally Section 5 concludes the paper.

2 Proposed Solution – Architecture and Requirements

This section describes the system architecture for which our protocol is designed. Section 2.2 summarizes the requirements that our commit protocol has to fulfill, while Section 2.3 outlines the assumptions concerning the execution environment of our protocol.

2.1 Architecture

In our system, we assume that there is an *Initiator* of a transaction that divides a transaction into sub-transactions and submits each sub-transaction to a different database. At the end of the sub-transactions, the participating databases start our atomic commit protocol. Furthermore, we assume that there is a group of f nodes in the mobile network, that are relatively stable and in a single-hop distance of each other. Such a group of nodes is chosen to be the *coordinators* that handle transaction commitment. We call them the *cluster of coordinators* and we assume that at any time at least one node of this group will survive. The whole cluster itself acts as a commit coordinator and the databases act as participants in the 2-phase commit (2PC) protocol.

Each coordinator is a representative for the commit decision of one or more *databases*. There is a *main coordinator* that makes the global commitment decision for all coordinators. As illustrated in Figure 1, there are coordinators $C_1 \ldots C_4$ where C_1 is the main coordinator. C_4 is the coordinator for databases DB_5 and DB_6.

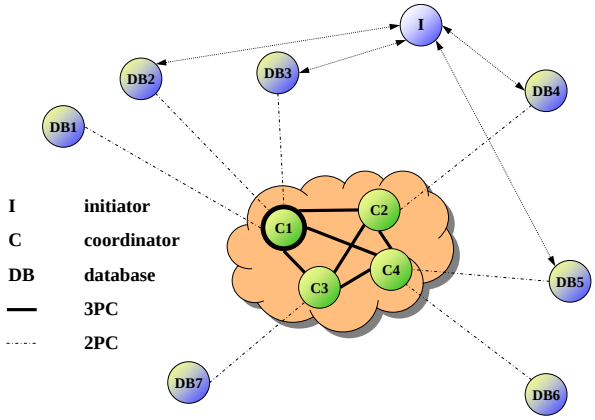

Figure 1: System architecture for our atomic commit protocol

2.2 Requirements

The main requirement is that our commit protocol guarantees the atomic execution of non-compensatable transactions.

Due to the loss of messages in mobile networks (e.g. link failures) and our goal to minimize the network traffic occurred due to transaction commitment, we allow our protocol to abort a transaction, even if all databases already voted for commit. But we want the protocol to be one-sided correct for the abort decision, i.e. if one participant votes for abort, the decision must be abort.

A database using our protocol should not be blocked even if some other databases or coordinators that are needed to execute the transaction fail. However, there are two scenarios, where we allow blocking, i.e.

- if *all* coordinators fail or

- if network partitioning occurs.

In the first case, we have to wait until one of the coordinators has recovered.

In the second case, when we cannot exclude network partitioning, [ALPS90] proved that there exists no non-blocking atomic commit protocol. However, if network partitioning occurs, we want our protocol to minimize the number of blocked databases and coordinators. If we can totally exclude network partitioning, our protocol should even be non-blocking as long as at least one coordinator still functions.

A database failure after the vote of this database has been received by the cluster of coordinators should not affect the transaction's commit status.

Whenever a mobile network contains a group of nodes in one-hop distance, we want our protocol to take advantage of this locality by preferably using such a group of nodes as atomic commit coordinators.

Our commit protocol should combine the advantages of the 2-phase-commit (2PC), i.e. its practical relevance and the minimum number of messages required, the 3-phase-commit (3PC) [Ske81], i.e. its non blocking behavior, and the Paxos Consensus Protocol [Lam98], i.e. its behavior in case of network partitioning.

We allow nodes to be fail-stopped, that means they stop working after a failure. However, a node may even drop messages if its buffer is full.

2.3 Underlying Assumptions

We assume that no byzantine failures occur, i.e. no node sends fake messages.

When a node is not able to finish the current transaction (e.g. because a loss of power), it tries to inform the other nodes. If it can inform the other participants of this situation, we call this a *controlled failure*.

3 Proposed Solution – Commit Protocol

The problem of using the 3PC protocol for Mobile Ad-hoc Network (MANET) database applications is that the number of messages required is very high. We minimize this disadvantage by using the *cluster of coordinators in single-hop distance* for our commit protocol.

While communication with the databases is done using the popular 2PC protocol, our cluster of coordinators uses the 3PC protocol to eliminate the blocking behavior of 2PC in case of coordinator failure (cf. Figure 1). The advantage of our new protocol lies in the fact that the cluster of coordinators is located in a single-hop environment with fast message transfer. In addition, if we assume that network partitioning cannot split this cluster of coordinators, the protocol can proceed without blocking the coordinators even if some of them fail. The details are given in Section 3.1.

If we cannot safely determine that this cluster of coordinators is not partitioned, or if it seems that the main coordinator has failed, we run a termination protocol which uses the ideas of the Paxos Consensus Protocol ([Lam98]) to get a decision on the transaction, which is described in Section 3.2.

3.1 Normal Case – Failure Free

When there is no node or link failure, our protocol works as follows. As shown in Figure 2, the transaction execution is initiated by an initiator which could be a separate node or a database itself. In order to execute a transaction T, the initiator node divides T into subtransactions and sends the subtransactions to appropriate participating databases. Each participating database executes the subtransaction until it can decide to vote for commit/abort. Then it sends this vote to its associated coordinator. Each associated coordinator

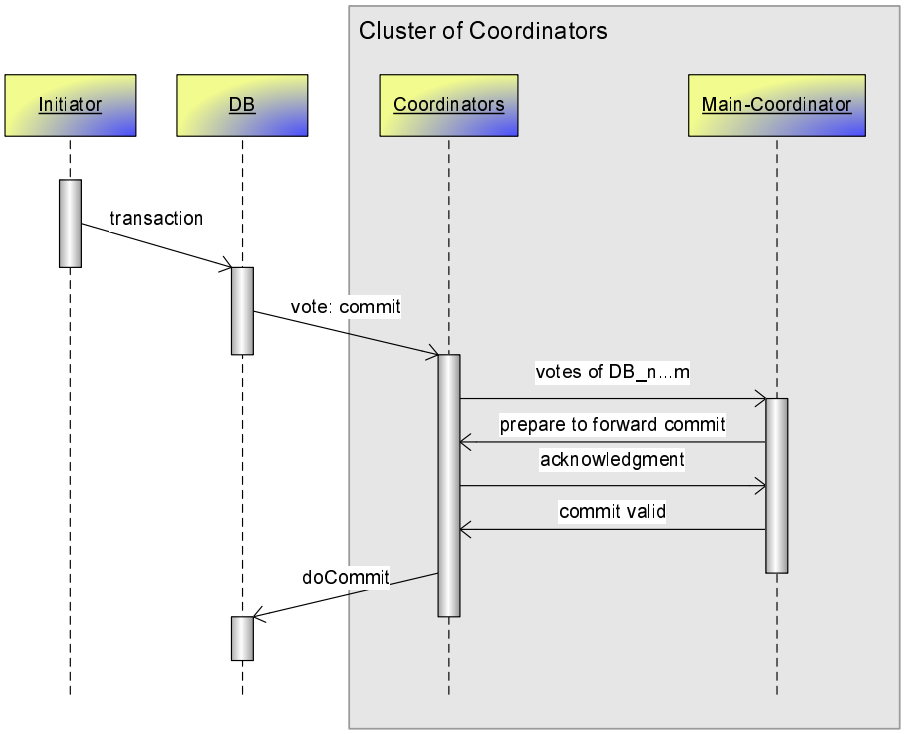

Figure 2: Sequence diagram of the failure free case

starts its timer and accepts votes from other participating databases for a certain period of time. When the time has passed, it will no longer accept messages. The coordinators then bundle their collected commit/abort votes and forward them to the main coordinator. After a specified period of time, the main coordinator decides on a global commit/abort and informs the coordinators. The main coordinator's decision is to commit the transaction if all votes are to commit; and is to abort otherwise. The coordinators then wait for a "prepare-to-forward-commit message" from the main coordinator, and upon receiving this message, they send an acknowledgment message to the main coordinator. After having received all acknowledgment messages, the main coordinator sends a "commit valid" message to the coordinators. Each coordinator forwards the decision to its associated databases (or the databases themselves will ask for the decision in case of message loss), so that the databases can perform the necessary actions.

In summary, the whole cluster of coordinators acts as a commit coordinator and the databases as the participants in the 2PC protocol, while the main coordinator acts as the commit coordinator and the individual coordinators act as the participants in the 3PC protocol.

3.2 Failure Handling

In this section, we describe failures occurring during the protocol execution and how the protocol handles them. Mobile applications would require our commit protocol to be strongly non-blocking and work one-sided correctly.

3.2.1 Individual Coordinator Failures

If one or more coordinators except the main coordinator fail or do not acknowledge the decision message, databases have to contact and communicate with the next coordinator $C_{((i+1) \bmod f)+1}$. From the databases' point of view, the databases themselves are the active components and are asked to find a running coordinator.

3.2.2 Loss Of Votes

We allow global aborts even if all databases voted for commit. Such an abort can occur if a coordinator fails after it has acknowledged its associated databases' commit/abort vote. Consequently, this vote is lost and cannot be forwarded to the main coordinator, which, after a timeout, will eventually decide on abort since it does not have all necessary commit votes. The abort information is sent to all remaining coordinators which acknowledge and wait for a "declare valid" message before forwarding this information to their associated databases. We must allow this abort decision in order to ensure termination of a transaction if one coordinator fails before forwarding votes.

To describe the transaction termination protocol, we have to distinguish between the databases' physical commit command and the coordinators' commit decision. The latter must be forwarded to the databases which can then complete the transaction. In the following, we will refer to the coordinators' commit decision.

3.2.3 The Main Coordinator's Failure And Network Partitioning

This section explains the coordinators' behavior in case they cannot communicate with the main coordinator anymore. Our protocol uses two concepts which are used in the Paxos Commit ([Lam98]) to terminate a transaction even in the case of network partitioning with at least $(1/2f + 1)$ coordinators in one partition: Quorums and version numbers. Quorums enable a decision to become valid after a majority of coordinators have agreed on the decision. The concept of version numbers is introduced in order to have a unique interim coordinator every time. These two concepts are used to guarantee that no new interim main coordinator can change a decision that has been accepted by the majority. However, a proposal of an interim coordinator can be rejected by a new interim coordinator as long as the majority has not accepted the decision.

Whenever a coordinator does not get a decision message from the main coordinator after a time out, this coordinator decides that the main coordinator may have failed and takes over its role. It must assign itself a version number and query at least $(1/2f + 1)$ other coordinators for the latest state they have reached in the 3PC protocol. It will then take over the previous main coordinator and adopt the latest proposal in terms of commit/abort decision of which it has knowledge.

The purpose of the version number is that the coordinator with the highest version number is the currently valid main coordinator. If there are any coordinators with lower self-assigned version numbers, other coordinators with higher numbers may have assumed that the previous main coordinator has failed and therefore assigned themselves a higher number. Every coordinator can only communicate with the interim main coordinator with the highest number of which the coordinator has knowledge. This ensures that we have versioned decisions and that the decisions will not change.

A decision becomes implicitly valid when the $(1/2f + 1)$th coordinator has received it. At this time, it is ensured that any new main coordinator will receive the latest decision since it also has to query $(1/2f + 1)$ coordinators.

3.2.4 Majority Decisions In Case Of Network Partitioning

This section explains the message flow in case a coordinator assumed the main coordinator to have failed. After one coordinator decides that the main coordinator may have failed, it declares itself to be an interim main coordinator. For this purpose, it performs the following four steps.

1. The new interim main coordinator has to assign itself a version number greater than the maximum of 0 and the version number of any other interim coordinator of which it has knowledge.

2. Then it queries every other coordinator for the state it has reached during the execution of the 3PC protocol (cf. Figure 3) and the version number of the previous main coordinator with which it communicated. The new main coordinator needs at least $(1/2f+1)$ responses to proceed, otherwise it has to wait.

   The coordinators must communicate only with the coordinator having the highest version number. If a coordinator has to reject a message of another coordinator with a lower version number, it informs the other coordinator of the higher version number. If a coordinator gets two messages with the same version number from different coordinators, it must also reject the messages.

3. The new main coordinator takes those responses with the *highest version number* and decides based on the states of the responses.

   If the responses

   - contain at least one proposal, i.e. vote commit or vote abort, of any other interim coordinator: it adopts this proposal.
   - contain only wait states in the 3PC protocol: it proposes abort
   - contain at least one "Prepare to Commit" state in the 3PC protocol: it proposes "prepare" and continues the normal execution of the 3PC protocol

   This means the new main coordinator will either adopt the latest proposal the previous main coordinator made, or, if there was no proposal, it will abort the transaction.

4. The interim main coordinator sends the new decision to those $(1/2f + 1)$ coordinators it has just informed that it is the new main coordinator.

A non-interim coordinator behaves in the following way: If it gets a state query message, it responds only if the version number of the new self-appointed main coordinator is greater than the version number of any other main coordinator to which it has communicated. It will reject messages from coordinators with lower or equal version numbers.

The consensus protocol works because a decision is valid at the time when the $(1/2f+1)$th coordinator has gotten the decision. Because a new main coordinator has to query more than half of the other coordinators, it is guaranteed that if there was a valid decision, the new main coordinator will get this decision in its first querying phase. Because the new main coordinator must adopt the latest proposal it has received, a valid decision will not be changed anymore.

In addition, we rely on the fact proved in [Ske81] that in 3PC no coordinator can be two states in advance. This includes, for example, a Prepare state which guarantees that no coordinator can have a valid commit state while another one is in a wait state. Since the main

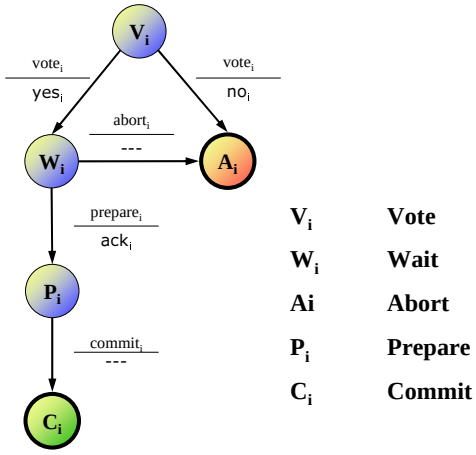

Figure 3: States of 3-Phase-Commit

coordinator sends a Prepare to Commit message only after all participants have voted for commit, it is sufficient to have one coordinator in this prepare state to be sure that every database voted for commit and no coordinator can be in an abort state.

Figure 3 illustrates the coordinators' states and the reaction to the main coordinator's messages. If the coordinators' bundled vote messages do not contain an abort decision, the coordinators proceed to the wait state W_i and wait for the main coordinator's decision. If this decision is Commit, the main coordinator sends a "Prepare" message to the coordinators, which then proceed to the state P_i and send an acknowledgment to the main coordinator. After the main coordinator received all acknowledge messages, it sends the commit decision to the coordinators which forward this decision to their associated databases.

Due to the behavior of the main coordinator, which proceeds only to the next state if all coordinators send their votes or all coordinators send their acknowledgment, any two coordinators cannot be more than one state apart. Therefore we can decide on the fate of the transaction at any time.

3.2.5 Minority Termination Decisions

If we are able to guarantee that network partitioning would not occur, no coordinator can be disconnected from the whole system while communicating with other coordinators. This is possible if commit coordinators provide controlled failures, i.e. inform the other coordinators before they fail (e.g. because of low energy). This knowledge that some coordinators have failed can be used to reduce the number of coordinators needed to come to a majority decision. In other words, it is sufficient to have the majority of all nodes except those which are known to be down. In this case, we do not need a majority of $(1/2f + 1)$ remaining coordinators to complete the transaction. In an extreme case, it is even possible to come to a commit decision when all nodes except one perform controlled failures.

If we cannot exclude network partitioning but are able to detect it, a new interim main coordinator which does not get the necessary majority of $(1/2f + 1)$ remaining coordinators can try to check that a network partitioning did not occur. If network partitioning can be excluded because of controlled termination of some coordinators, the interim main coordinator can continue because now we are sure that sufficiently many other coordinators have failed and they cannot communicate with the others anymore.

3.3 Communication Costs

We consider the number of messages in the normal case where a transaction is committed successfully. We assume that we have n databases and f coordinators. When the Initiator distributes the transaction to the participating databases, the necessary transaction information (e.g. who is the main coordinator; who are the other coordinators) is sent within the same message, i.e. it does not require any ex-

tra message. Figure 2 shows the message flow. In the failure free case we have at most

- n messages of the Initiator to the databases, including the subtransaction and the request to send a vote about the subtransaction to one coordinator
- n messages "vote: commmit" or "vote: abort" from the databases to a coordinator
- f messages from one coordinator to the main coordinator (votes from the databases associated with a coordinator are bundled in one message for the coordinator)
- $3f$ messages using the 3PC (f messages from the main coordinator to the coordinator: "prepared?"; f messages are sent back from the databases to the main coordinator: "prepared!"; f messages from the main coordinator to the coordinators: "commit/send decision to your databases")
- n messages "do Commit" from the coordinators to the databases.
- 1 message from the main coordinator to the Initiator ("transaction is committed/aborted").

Summing up all messages, we have a total of $(3n + 4f + 1)$ messages. $4f$ messages are sent within the cluster of coordinators where message transfer is very fast because of one-hop communication among coordinators. The remaining messages are the costs of the normal 2-Phase-Commit which is implemented in many database applications.

4 Related work

The non-blocking behavior of commit protocols has been studied in the literature. Message loss and node failure turned out to be the two main problems for atomic commit. [Gra78] even proved that a commit decision is not possible under the assumption of message loss within the well known coordinated attack scenario. In this scenario the commit decision is that two generals must agree on a time for a common attack using an unreliable communication channel.

While our approach does not fulfill some requirements of this coordinated attack scenario (e.g. strict time constraints), other approaches attempt to identify message loss and node disappearance with special cross-layer protocols ([VLdL04], [CMTG04]). However, in current mobile networks, these protocols are difficult to implement because they need new communication protocols. Therefore, we focus on atomic commit protocols that work on the TCP/IP layer and are applicable to mobile networks regardless of the underlying transport protocol.

Of course, there are attempts to guarantee message delivery on the TCP/IP layer, mainly by the use of "reliable multicast protocols" (e.g. [GM98] and [SS93]). This group of protocols uses a reliable uniform multicast protocol for sending messages. Therefore, the protocols can guarantee that certain messages arrive even if a node disconnects temporarily. To do so, they flush the transaction log before sending a message or commit decision so that other nodes do not need to wait for the recovery of the disconnected node. However, these protocols assume that there is a limit on the maximum number of messages lost. They also cause an enormous traffic due to the multicasting of messages. This impedes practical usage in dynamic mobile networks with low bandwidth.

Our approach aims at a practically useful solution and follows [Ske81] in the way that we

use 3PC) for the communication of coordinators. However, we combine the use of 3PC with 2PC in order to reduce the amount of messages transferred.

In addition, we employ a termination protocol which uses the idea of the Paxos Consensus ([Lam98] and [PLL97]). However, we do not follow Lamport by using the Paxos Commit Protocol ([GL04]) to get a consensus on the commit state as this approach does not take advantage of one-hop-environments and has a lot of message overhead when there are many coordinators. In addition, if the coordinators know that there is no network partitioning in the cluster of coordinators, our protocol is able to be non-blocking even with only one remaining coordinator (the Paxos Commit Protocol needs at least $(\frac{1}{2}f + 1)$ remaining coordinators, where f denotes the total number of coordinators). If we cannot exclude or safely detect a network partitioning, our protocol needs $(\frac{1}{2}f+1)$ coordinators to be running in the same network partition to get a commit decision for the transaction. Otherwise, [ALPS90] proved that it is inevitable to wait until the network is connected again. In this case, the number of remaining coordinators is the same as that in the Paxos Commit Protocol.

5 Summary and Conclusion

We have presented an atomic commit protocol which is designed for the use in mobile networks and takes advantage of special mobile network structures like single-hop-communication. Because the traditional 2-Phase-Commit protocol is not suitable in mobile networks due to the blocking behavior in case of node or link failures, our protocol is aimed at handling new challenges including network partitioning, node failures and message loss in a blocking-free manner. This is ensured by the use of the 3-Phase-Commit protocol in the cluster of coordinators. If we can only detect that no majority of coordinators is reachable, but can not exclude the occurrence of network partitioning because of controlled termination of coordinators, our protocol is still better than previous commit protocols in the following ways. Our protocol blocks only in situations when we do not have more than half of the coordinators which are working or in an unknown state in one network partition. However, this blocking in split asynchronous networks has been proved to be unavoidable ([ALPS90]).

Compared to the Paxos commit protocol, our protocol uses consensus only in case of failure for termination purposes. As a result, we need fewer messages for communication between the coordinators and the databases.

In addition, our protocol's logic resides in the cluster of coordinators; so we can keep the widely used 2PC of database applications and use our protocol as an extension. As only slight modifications are needed for standard databases, our protocol is easy to implement as an extension to the 2-Phase-Commit protocol. Therefore, we consider our approach to be a useful contribution for transferring fixed-wired applications into mobile environments.

References

[ALPS90] P. Ancilotti, B. Lazzerini, C. A. Prete, and M. Sacchi. A distributed commit protocol for a multicomputer system. *IEEE Trans. Comput.*, 39(5):718–724, 1990.

[CMTG04] Marco Conti, Gaia Maselli, Giovanni Turi, and Silvia Giordano. Cross-layering in mobile ad hoc network design. *IEEE Computer*, 37(2):48–51, 2004.

[GL04] Jim Gray and Leslie Lamport. Consensus on transaction commit. *CoRR*, cs.DC/0408036, 2004.

[GM98] Laurent George and Pascale Minet. A uniform reliable multicast protocol with guaranteed response times. In *LCTES '98: Proceedings of the ACM SIGPLAN Workshop on Languages, Compilers, and Tools for Embedded Systems*, pages 65–82. Springer-Verlag, 1998.

[Gra78] Jim Gray. Notes on data base operating systems. In *Operating Systems, An Advanced Course*, pages 393–481. Springer-Verlag, 1978.

[Lam98] Leslie Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, 1998.

[PLL97] Roberto De Prisco, Butler W. Lampson, and Nancy A. Lynch. Revisiting the paxos algorithm. In *Distributed Algorithms, 11th International Workshop, WDAG '97, Saarbrücken, Germany*, Lecture Notes in Computer Science, pages 111–125. Springer, 1997.

[Ske81] Dale Skeen. Nonblocking commit protocols. In Y. Edmund Lien, editor, *Proceedings of the 1981 ACM SIGMOD International Conference on Management of Data, Ann Arbor, Michigan*, pages 133–142. ACM Press, 1981.

[SS93] A. Schiper and A. Sandoz. Uniform reliable multicast in a virtually synchronous environment. In *Proceedings of the 13th International Conference on Distributed Computing Systems (ICDCS-13)*, pages 561–568, Pittsburgh, Pennsylvania, USA, 1993. IEEE Computer Society Press.

[VLdL04] Alex Varshavsky, Baochun Li, and Eyal de Lara. Cross-layer flow control in lightly-loaded multi-hop ad hoc networks. In *ICPP Workshops*, pages 315–321, 2004.