THE COMMUNICATION COMPLEXITY OF ATOMIC COMMITMENT AND OF GOSSIPING

Ouri Wolfson[1,3,4]
Adrian Segall[2,3]

Technical Report CUCS-499-89

ABSTRACT

We consider the problem of atomic commitment of a transaction in a distributed database. This is a variant of the famous gossiping problem (see [HHL] for a survey). Given a set of communication costs between pairs of participant sites, we establish that the necessary communication cost for any atomic commitment algorithm is twice the cost of a certain minimum spanning tree. We also establish the necessary communication time for any atomic commitment algorithm, given a set of communication delays between pairs of participant sites, and the time at which each participant completes its subtransaction. Then we determine that both lower bounds are also upper bounds in the following sense. There is an efficient (i.e. polynomial-time) algorithm that, in the absence of failures, has a minimum communication cost. There is another efficient algorithm that, in the absence of failures, has a minimum communication time. However, unless P=NP, there is no efficient algorithm which has a minimum communication complexity, namely, for which the product of communication cost and communication time is minimum. Then we present a simple, linear time, distributed algorithm, called TREE-COMMIT, whose communication complexity is not worse than p times the minimum complexity, where p is the number of participants. Finally, we demonstrate that TREE-COMMIT is superior to the existing variants of the two-phase commit protocol.

1. This research was supported by the Israeli Foundation for Research in Electronics, Computers, and Communications, by DARPA research grant #F-29601-87-C-0074, and by the Center for Advanced Technology at Columbia University NYSSTF-CAT(89)-5.

2. This research was supported by the Technion Fund for the Promotion of Research.

3. Department of Computer Science, Technion - Israel Institute of Technology, Haifa 32000, Israel

4. Department of Computer Science, Columbia University New York, NY 10027

1. INTRODUCTION

1.1. The problem

In a distributed database, a transaction consists of several subtransactions, each running at a different site that has a local database. When the subtransaction at a site completes, i.e. the corresponding process finishes, the local database manager knows whether it completed successfully or unsuccessfully[1]. The *atomic commitment* problem (see [G]) for the set of local database managers, is to determine the decision of each manager, concerning the changes made by the transaction to the local database. The manager may either validate these changes (commit the subtransaction), or invalidate them (abort the subtransaction). The generally accepted solution to the atomic commitment problem is to commit all the subtransactions, yielding a committed transaction, if all the completions are successful, and abort all of them otherwise, yielding an aborted transaction. To achieve atomic commitment, the local database managers execute a distributed algorithm, exchanging 'yes' and 'no' votes. A 'yes' vote indicates the successful completion of a subtransaction, and a 'no' vote indicates an unsuccessful completion[2].

In this paper we first consider the load that atomic commitment algorithms place on the communication network, in the following sense. Each subtransaction of a given transaction runs at some node, called a *participant* site for that transaction, and with each pair of participants, i and j, is associated the cost, c_{ij}, of sending a message between i and j. The costs may differ from one pair of participants to another. For example, the communication cost for a pair of participants may represent the distance in the communication network between the participants. The communication load that an execution of an algorithm places on the communication network is quantified in terms of its *communication cost*, i.e., the total cost of messages that the algorithm sends among the participants.

Traditionally, the communication load has been quantified in terms of the number of logical messages among participant sites, namely *intersite messages*. (Implicitly, c_{ij} was taken to be one for every pair of participants, i and j.) However, this may be too coarse a measure for comparing the load that different atomic commitment algorithms place on the network. To demonstrate this we shall present next three commitment executions, or instances, for a transaction; they all use the same number of intersite messages, but put different loads on the communication network, since they use different numbers of *network messages* i.e., messages between neighbors in the communication network (regardless of whether they are participants or nonparticipants).

---

1. For example, a subtransaction may complete unsuccessfully as a result of deadlock.

2. Transactions are used in distributed systems other than databases, such as Argus ([LHJLSW]) and Camelot ([Sp]). The atomic commitment problem, as described, arises in such systems as well.

For example, assume that the transaction executes in the computer-communication network given in Fig. 1.1.

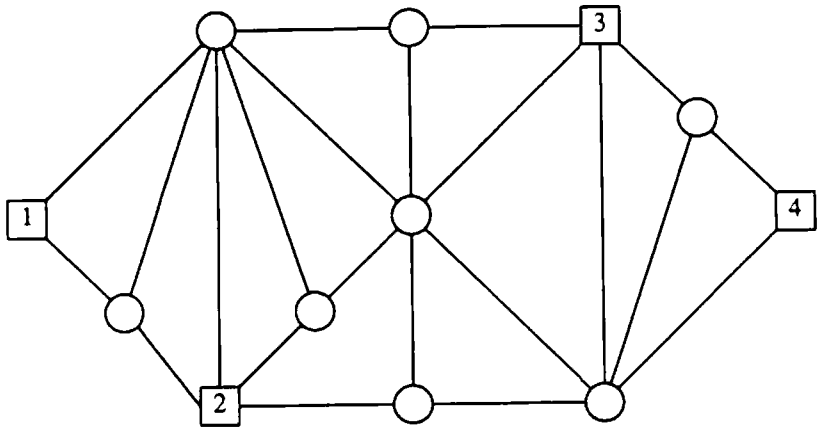


Fig. 1.1

Squares represent participant sites, circles represent nonparticipant sites, and edges represent two-way communication links. A possible commitment instance, based on the "central site" algorithm of Lampson and Sturgis ([LS]), is illustrated in Fig. 1.2a: each one of the participants 2,3, and 4 sends its 'yes' vote directly to participant 1, which also votes 'yes', commits, and sends the commit decision separately to each of participants 2,3, and 4 (each of which commits when it receives the message). This instance uses six intersite messages. Another possible instance, based on the "linear" algorithm of Gray ([G]), is illustrated in Fig. 1.2b: 4 sends its 'yes' vote to 1, which then sends its and 4's 'yes' votes to 3, which then sends its and 1's and 4's 'yes' votes to 2; 2 then votes 'yes', commits, and sends the commit decision to 2, which sends it to 3, which sends it to 4. This instance also uses six intersite messages. In fact, from the lower bound result of Dwork and Skeen [DS1] for atomic commitment, we know that any four-participant atomic commitment execution requires at least six intersite messages. Finally, consider the following instance that also uses six messages (see Fig. 1.2c): 4 sends a 'yes' vote to 3, which sends its and 4's 'yes' votes to 2; meanwhile, 1 also sends a 'yes' vote to 2, which, after receiving the two messages, also votes 'yes' and commits, after which the commit messages travel from 2 to 3 to 4 and from 2 to 1. If we assume that the intersite messages travel through the shortest path in the network, then the first instance takes 20 network messages, the second takes 22, and the latter takes only 14. Since the actual communication load on the network is reflected by the number of network messages, the third instance above imposes the least load on the network.

The other measure that we consider in comparing the performance of commit protocols is the *communication time*. Communication time of an instance is defined as the interval of time starting when the first subtransaction completes, and ending when the last participant commits its subtransaction. In this respect we also depart from traditional models (e.g., [DS1], [R]), which for the purpose of analysis, assume a synchronous communication network, and simultaneous completion of all subtransactions. The synchronous communication implies, in particular, a unit-time intersite message delay, independent of the network location of the sender and receiver.

In our model we allow, more realistically, arbitrary subtransaction completion times as well as different intersite delays. Particularly, different participants may complete their subtransaction at different times, and intersite message delays may differ from one sender-receiver pair to another. Therefore, we dispose of unrealistic assumptions regarding synchronous operation of geographically dispersed processors.

Finally, let us mention a different formulation of the problem addressed. There is a large body of research on "gossiping", and many variants of this problem were studied in the literature (see [HHL] for a survey). Our model and our results apply to the following variant of the gossiping problem; it has not been solved before, and actually, our necessary and sufficient cost results solve an open problem discussed by Cot ([C]). There is a set, A, of individuals, each of which knows a unique piece of a gossip. Additionally, for each pair of individuals i and j, there is a cost, c_{ij}, and a travel-time (corresponding to the intersite delay in atomic-commitment terminology), t_{ij}, associated with a letter (or a message) from i to j. Also, individual i learns its own piece of the gossip at time τ_i (corresponding

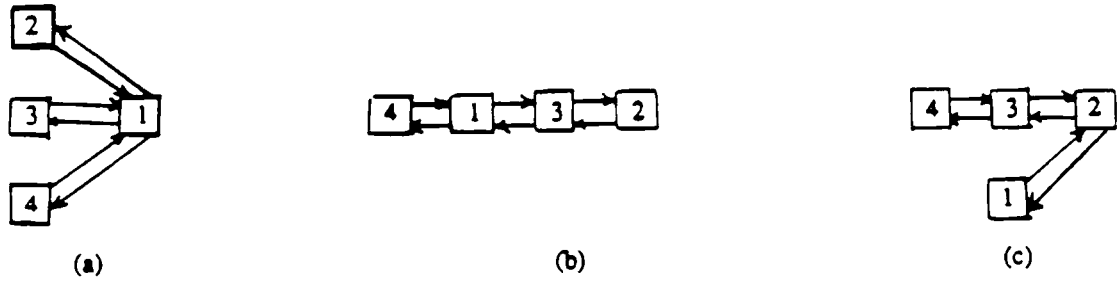


Figure 1.2

to the subtransaction completion time). A solution consists of a partially ordered set of pairs of individuals. Each pair represents a letter sent from the first to the second, and the partial order represents the time at which the letters are sent. Presumably, each letter contains all the pieces of the gossip known to the sender when sending the letter. At the end, each individual must know the whole gossip. In subsection 8.2 we discuss another unsolved variant of gossiping, and the implications of the results in this paper, to that variant.

The atomic-commitment problem is a special case of gossiping, in which the unique pieces of the gossip are the votes, and after collecting all the votes, each individual computes their conjunction. However, although throughout the paper we use the atomic commitment terminology, it is clear that the issues addressed are independent of the function computed by each processor after the vote collection. Therefore, all our results carry over to the general "gossiping" case, with one exception. In some occasions we briefly discuss the abort case of a distributed transaction. In this case a participant does not necessarily have to collect all the votes; when receiving a 'no' vote it already knows the result of the conjunction. Thus, the abort case of atomic-commitment does not have a gossiping counterpart.

1.2. Our results

Given a set of participants, and an associated set of communication costs between every pair of participants, we establish first the necessary communication cost of atomic commitment, namely the lower bound on the communication cost of any atomic commitment algorithm. It is twice the weight of a minimum spanning tree in a complete graph, called the cost graph. It has the participants as nodes, and the costs as the weights of the edges. For example, in Fig. 1.3 we show the cost graph of the participant sites and network of Fig. 1.1, assuming that the communication cost between every pair of participants is the distance between them in the network. We shall explain at the beginning of Section 3, that this result is not trivial because, as we shall show, an atomic commitment execution of minimum communication cost, does not necessarily have a single coordinator (a participant to which all other participants send their votes). In fact, the most difficult part of the lower bound proof is showing that there always exists a minimum communication cost instance with one coordinator.

Then, we show that the necessary cost is also sufficient for atomic commitment in the absence of failures, by presenting an algorithm that achieves the lower bound. Subsequently, we establish the necessary communication time for atomic commitment (this is much easier than establishing the communication cost lower bound), and

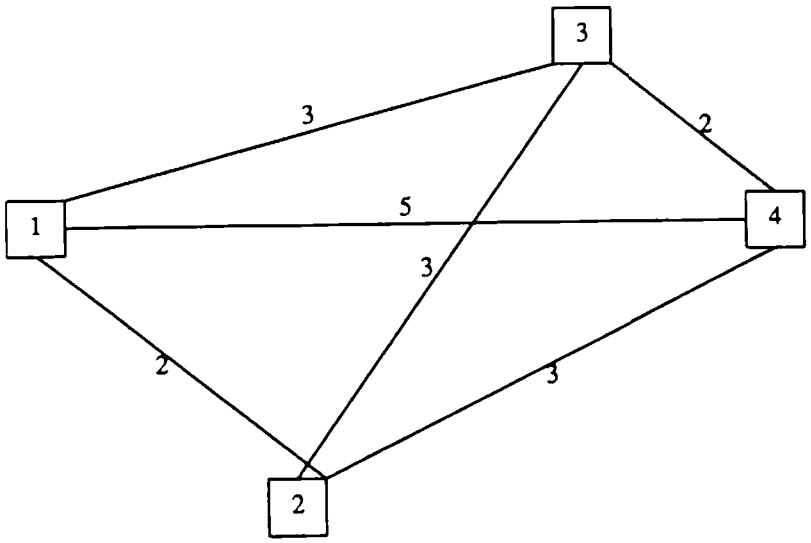


Fig. 1.3

demonstrate that it is also sufficient for atomic commitment in the absence of failures, by demonstrating that the decentralized algorithm (introduced previously by Skeen ([Sk]), and explained briefly in the next subsection), achieves the lower bound. We define the communication-complexity of an instance as the product of its communication-time and its communication-cost, and we point out that it is NP-complete to find the instance of minimum communication-complexity. However, we propose the algorithm TREE-COMMIT, in which messages propagate along the edges of a tree spanning the participants. For a minimum spanning tree of the cost graph, TREE-COMMIT has minimum communication-cost, and its communication-complexity (communication-time) can be worse than the minimum communication-complexity (minimum communication-time) by a factor that is bounded by the number of participants, p. The computation time (as opposed to the communication time) of each one of the p participants executing TREE-COMMIT is linear in the size of the input, which implies that TREE-COMMIT is a distributed variant of a polynomial-time, bounded-error approximation algorithm, for the NP-complete problem of finding the minimum communication complexity instance. Moreover, TREE-COMMIT achieves the bound on the communication-complexity, that we show is tight, without the participants knowing the subtransaction completion times, or the communication delays. Then we analyze the decentralized algorithm, that is communication time optimal. We show that its communication complexity (cost) can be worse than the minimum communication complexity (cost) by a factor of order p^2.

The decentralized algorithm is a variant of the two-phase-commit paradigm. Two other well known variants are the ones mentioned above, namely the central-site and the linear algorithms (they are briefly explained in the next subsection). The algorithm introduced in this paper, TREE-COMMIT, is superior to these two variants in a very strong sense. It can be adapted, by varying its communication tree, to have exactly the same communication cost as the central-site algorithm (alternatively, it can be adapted to have the same communication cost as the linear algorithm). Then, for any set of subtransaction completion times and intersite communication delays, its communication time cannot be higher than that of the central-site algorithm (the linear algorithm). Furthermore, there are cases, i.e., completion times and delays, for which it is half the communication time of the central-site algorithm (the linear algorithm).

In this paper we introduce a novel model of an atomic commitment instance. Although other models, such as finite state automata ([Sk]) and knowledge theoretic ([H2]), exist in the literature, we chose to represent the instance by a directed acyclic graph, representing the time-order of events and messages. It enables us to establish the communication cost and communication time lower bounds, and to analyze the cost and time of algorithms, all in the same formalism. The communication cost is the total length of the arcs, and the communication time is the length of the longest path.

Our discussion is restricted to the case in which no failures occur while the commitment protocol is executed. We mainly analyze the case in which each participant votes to commit the transaction (and thus the transaction commits), for the following reasons. Successful commitment represents the more likely outcome for transactions in most database systems, and it also represents a worst-case scenario from the communication cost viewpoint (see theorem 3).

1.3. Other related work

Atomic commitment is a variant of a fundamental problem in distributed systems, namely distributed consensus. Fischer presents a survey of the subject ([F]), and Dwork and Skeen devise an interesting taxonomy of consensus problems ([DS2]). Hadzilacos presents an illuminating discussion on the applicability of the consensus problem results to the atomic commitment problem ([H1]), particularly the types of failures that are meaningful for each problem. In short, almost all existing research concentrates on the effects of failures on achieving consensus in general, and sometimes atomic commitment in particular.

In contrast, in this paper we concentrate on performance issues. Mohan et. al. also discussed performance issues of commitment protocols ([MLO]), however our work differs from theirs in two respects. First, their atomic commitment algorithms are more complicated mainly because of a more complicated transaction model, allowing for the fact that there may not be any participant that knows the identity of all the other participants in the transaction. Second, [MLO] as the other previous works, has assumed a one-unit cost for each message.

Informal discussions of the performance of atomic commitment algorithms in the absence of failures, also appear in [BHG], [CP], [G], [Sk], mainly in the context of different two-phase-commit variants. One of the most popular is the *central-site* algorithm (see Fig 1.2a); a participant is designated as the "protocol coordinator", and each other participant sends its vote to the coordinator. The coordinator makes the decision, and sends the 'commit' or 'abort' message to all the other participants. Another two-phase-commit variant is the *decentralized* algorithm, in which each participant sends its vote to all the other participants. Based on the received messages each participant makes the 'commit' or 'abort' decision. We shall show that this algorithm minimizes the communication time. Finally, in the *linear* algorithm (Fig. 1.2b), all the participants are sequentially ordered. Each participant sends its vote to the next one in the sequence. The last participant is the protocol coordinator, which reverses the flow direction, by sending the decision message to its predecessor in the sequence. The linear and central-site algorithms require $2(p-1)$ intersite messages, where p is the number of participants. Dwork and Skeen have formally proven that the number of intersite messages required by any atomic commitment protocol, in the absence of failures, is $2(p-1)$ ([DS1]). By contrast, note again that in the present work we consider the total communication cost of intersite messages, rather than their number. In the special case in which the communication cost of every intersite message is one, our cost lower bound result matches the $2(p-1)$ result.

1.4 Paper organization

The rest of this paper is organized as follows. In section 2 we introduce our model of a commit instance. In section 3 we establish the necessary communication cost for the atomic commitment problem, and in section 4 we provide a complete characterization of the minimum communication-cost commit-instances. In section 5 we establish the minimum communication time of an instance, and in section 6 we present the TREE-COMMIT algorithm. In section 7 we analyze TREE-COMMIT and compare it with the other algorithms discussed in this paper. In section 8 we conclude, and discuss future work.

2. COMMIT INSTANCES

In this section we provide some key definitions, particularly, we formally introduce our novel model of a commit instance. Intuitively, the instance executed by an atomic commitment algorithm is represented by the temporal, and thus partial, order of events occurring at the participants. Let P be a set of participants. Formally, an *instance* on P is a directed acyclic graph, $I = (E,A)$ (see Fig. 2.1a). E is a set of nodes, called *events*, and A is a set of arcs (i.e. directed edges). Every event *occurs* at some participant, and all the events occurring at a participant are totally ordered in I. Each event represents zero or more consecutive receives (each of an intersite message) at the participant, without an intervening send, followed by zero or more consecutive sends, without an intervening receive. The first event occurring at a participant also represents the completion of the corresponding subtransaction. Every pair of consecutive events occurring at a participant are connected by an arc called an *order arc* (since it represents the order in which the two events occur at the participant). The other arcs of A are called *messages*. A message is an arc from an event called the *send* of the message, to an event called the *receive* of the message, which occurs at a different participant from the send. Only the last event occurring at a participant may send zero messages, and only the first event occurring at a participant may receive zero messages. Thus, into every event, except possibly the first one occurring at each participant, enters at least one message, and from every event, except possibly the last one occurring at each participant, exits at least one message. If there is a path in I between events a and b, then we say that a *happens* before b (in the sense of Lamport [L]), and b *happens* after a. We assume that a message sent at an event a contains all votes that happened before a. Three possible instances at a set of three participants, are illustrated in Fig. 2.1.

Next, we comment on the representation of several message -receives and -sends by one event. For the purpose of this paper, the order of consecutive message-sends is irrelevant, as is the order of consecutive message-receives. For example, we do not distinguish between two "instances" in which the only difference is that at some participant the order of two consecutive receives is reversed. Only the relative order of blocks of receives and sends is relevant, since we assume that each message sent includes all, and only, the votes received before sending the message, and also includes the vote of the sender.

We assume that a participant may send messages only after its subtransaction completes. Consequently, the first event at each participant represents zero or more message-receives, followed by the corresponding subtransaction completion, followed by zero or more additional message-receives, followed by one or more message-sends.

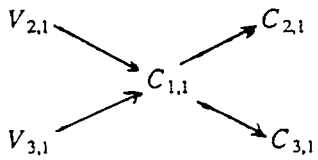
A *commit* instance, I, is an instance that satisfies the following *commit requirement:* At each participant occurs at least one event, e, which happens after the first event occurring at every other participant. Any event such as e, that happens after some, and particularly the first, event at each other participant, is called a *C−event*; informally, when it occurs, its participant, say j, already knows the commit decision. The reason for this is that j has received all the 'yes' votes from the other participants, and it knows that its own vote is 'yes'. The vote of some participant, say i, propagates to j along the path from the first event at i, to the C-event of j. The first C-event occurring at a participant, in addition to message sends and receives, also represents the validation of the changes made by the subtransaction, and the recording in stable storage of the fact that the transaction is committed. A message sent by a C-event is called a *commit* message. Each message that is not a commit message is a 'yes' *vote* message. An event which is not a C-event is called a *V−event.*

The motivation for the commit requirement is that each participant must receive the 'yes' vote of every other participant, and vote 'yes' itself, before knowing that it can commit ([DS1, H2]). For example, the instances illustrated in Fig.2.1 are commit instances. In our figures the V-events (C-events) are denoted by a subscripted V (C). The label $V_{i,j}$ ($C_{i,j}$) of a node indicates that this is the j'th V-event (C-event) occurring at participant i.
Notice that the commit requirement ensures that every commit instance is connected. Notice also that any event which happens after a C-event, must also be a C-event. Another observation is that a C-event may represent the receive of a 'yes' vote message (for example $C_{1,1}$ in Fig 2.1a).

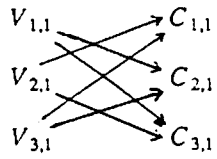
The variants of the two-phase commit protocol mentioned in the introduction are illustrated in terms of our model in Fig. 2.1. In order to prevent cluttering the figures we omit the order arcs; however, remember that any two consecutive events at a participant, are connected by an order arc. An instance executed by the central-site algorithm, with participant 1 as the coordinator, is illustrated in Fig.2.1a. Instances of the decentralized and linear algorithms are illustrated in Fig.2.1b and Fig.2.1c, respectively.

3. MINIMUM COMMUNICATION COST OF COMMIT INSTANCES

In this section we establish the minimum communication cost of a commit instance. This would have been quite straight forward, had we known that in a minimum communication cost commit instance all votes must be sent



(a)



(b)

$V_{1,1} \rightarrow V_{2,1} \rightarrow C_{3,1} \rightarrow C_{2,1} \rightarrow C_{1,1}$

(c)

Figure 2.1:

to one participant, the coordinator, which then broadcasts the commit decision. This would have meant that the problem of atomic commitment at minimum communication cost can be decomposed into two problems, collect-at-minimum-cost and broadcast-at-minimum-cost, and these two problems can be solved independently. However, as we shall show in theorem 2 (section 4), not every minimum communication cost commit instance consists of a collect-to-one, followed by a broadcast. In lemma 2 (this section) we show that there always exists such an instance with minimum cost, but we do not know this yet. Also, we do not know that the minimum-cost instance has $2(p-1)$ intersite messages. Actually, in subsection 8.2 we discuss a slightly different variant of the atomic commitment problem, in which the minimum cost instance does not have a minimum number of messages. At the end of the section we will present the FIXED COORDINATOR algorithm, that achieves minimum communication cost, and consists of a collect-to-one, followed by a broadcast.

We start with some formal definitions. Let P be a set of participants. We suppose that with P is associated a set, c, of communication costs, consisting of a positive real number, c_{ij}, for each pair of different participants, i and j, in P. For each such pair, c_{ij} is the *communication cost* of a message from i to j, and we assume that it is equal to the cost of a message from j to i, namely, $c_{ij}=c_{ji}$. Given P and c, in every instance on P the communication cost of a message arc from (an event occurring at participant) i to (an event occurring at participant) j is c_{ij}; the communication cost of an order arc is taken to be zero. The communication cost of an instance I, denoted $Cost(I)$, is the total communication cost of all the arcs in I.

In this section we assume a fixed set of participants, P, and a fixed set of associated communication costs, c, and we establish the minimum communication cost of a commit instance on P. A commit instance of such cost will be referred to as a *minimum communication cost* instance. We denote by Ψ the set of commit instances on P.
Given a commit instance I, its C-subgraph, denoted C_I, is defined as the subgraph of I induced by its C-events.

Lemma 1: If I is a minimum communication cost instance of Ψ, then I has only one C-event at every participant. Furthermore, its C-subgraph is a forest of rooted trees.

Proof: Assume that there are two or more C-events at some participant. Then, there must be at least one incoming message into the second C-event, by definition of a commit instance (in particular, that only the first event at a participant may represent zero receives). By omitting such a message, i.e., removing the arc corresponding to some message into the second C-event, a commit instance of strictly lower communication cost can be obtained[1]. The reason that the message can be omitted, is that the commit requirement is satisfied for the participant, by the first C-event occurring at the participant. Consequently, there is only one C-event at every participant.
In order to show that C_I is a forest, it is now sufficient to show that there is no C-event having two incoming commit messages. But this fact is obvious; if there were such an event, then all but one of the incoming commit messages can be omitted, to obtain an instance of lower communication cost. □

The single C-event at every participant, i, will be denoted by C_i. Assume that C_j is a root of C_I, for some minimum communication cost instance, I. Informally, this means that, in the execution I, participant j knows all the votes, without receiving a commit message (that is, j knows all the votes without receiving a message from another participant that knew all the votes). In such a case, we say that participant j is a *coordinator* of the instance I.

Lemma 2: There exists a minimum communication cost instance in Ψ, which has exactly one coordinator.

Proof: Let I be a minimum communication cost instance. We shall show that if I has two or more coordinators, then we can transform it into another commit instance, I', of equal communication cost, but with one less coordinator. The transformation replaces a vote message from some participant, i, to some participant, j, by a commit message from j to i. Since the costs of the two messages are equal, the costs of I and I' are equal. The effect of the transformation is to change the way in which one of the coordinators learns the votes of the other participants; that

1. If the omitted message was the only one exiting its send event, say e, and e is not the last event at its participant, then after the omission the instance has to be adjusted. Adjustment is by collapsing e and its consecutively following event at the participant. Two events e and f are collapsed by omiting the event f, and substituting e for f in the arcs of the instance (the arc (e,e) is omited). A similar adjustment has to occur if the omitted message is the only one entering its receive event.

coordinator becomes a noncoordinator, and learns the votes of the others via a chain of messages from one of the remaining coordinators. We now describe the selection of the vote message which will be replaced.

The C-events at the coordinators of I will be called *boundary C-events*. Consider a V-event, V^o, which satisfies the following condition. It precedes at least two boundary C-events, say C_i and C_j, and, if V^o has any V-event successors, then each one of them precedes only one boundary C-event (for example, V^o is W^o in Fig. 3.1, and the V-event successors are the W^i's, $1 \le i \le n$). It is easy to see that every commit instance with two or more coordinators has a V-event that satisfies the condition. Simply start at the first event at some participant, which by the commit requirement precedes every boundary C-event. Verify whether that event satisfies the above condition. If not, it means that one of its V-event successors precedes two or more boundary C-events. Repeat the verification at that successor, until a V-event with the following property is found: either it has no V-event successors, or, each one of its V-event successors precedes only one boundary C-event.

Suppose that V^o occurs at participant p_o. By Lemma 1 we know the structure of the C-subgraph of I, particularly that there is only one C-event at p_o, C_{p_o}. Assume, without loss of generality, that C_{p_o} is *not* in the tree of C_I rooted at the event C_i. Denote a path from V^o to C_i, by $V^o=W^o, W^1, \ldots, W^n, W^{n+1}=C_i$, where $n \ge 0$. Note that all events on the path, except the last one, are V-events. Let W^r, for $0 \le r \le n$, be the last event on this path for which the following condition is true: W^r occurs at a participant, p_r, for which C_{p_r} is in a tree different than the one rooted at C_i. Since W^o satisfies the condition, there must be such W^r. Denote by p_{r+1} the participant at which W^{r+1} occurs. By the definition of W^r, the event $C_{p_{r+1}}$ is in the tree rooted at C_i. Now to obtain a commit instance of minimum communication cost with one less coordinator, perform the following transformation:

i) the arc $W^r \rightarrow W^{r+1}$ is replaced by a message from C_{p_r} to $C_{p_{r+1}}$,

ii) the direction of the arcs on the path from C_i to $C_{p_{r+1}}$ is reversed, and

iii) if transformations i) and ii) result in any event that consists of message-receives only, then that event and the one immediately following it at the same participant, are collapsed into one event.

Figures 3.1 and 3.2 provide two examples of the transformation. Fig. 3.1 illustrates the modifications performed on I, assuming that W^o does have some V-event successors. Figures 3.2a and 3.2b illustrate an instance, I, before and after the transformation, respectively; W^0 is V_2, and it has no V-event successors.

It is easy to see that the transformation results in a commit instance; denote it I'. The commit requirement is

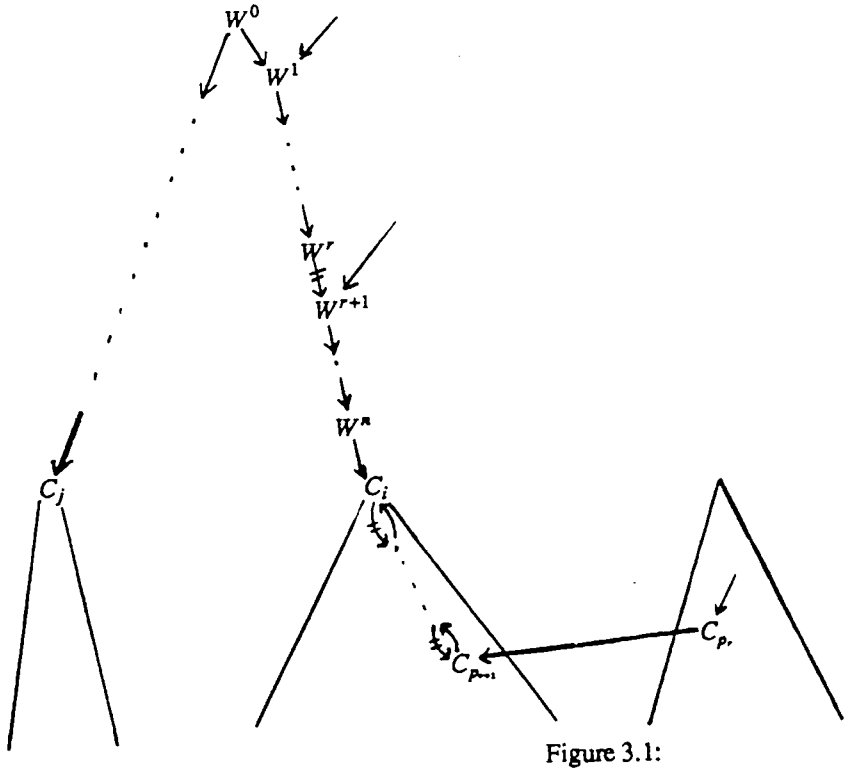


Figure 3.1:

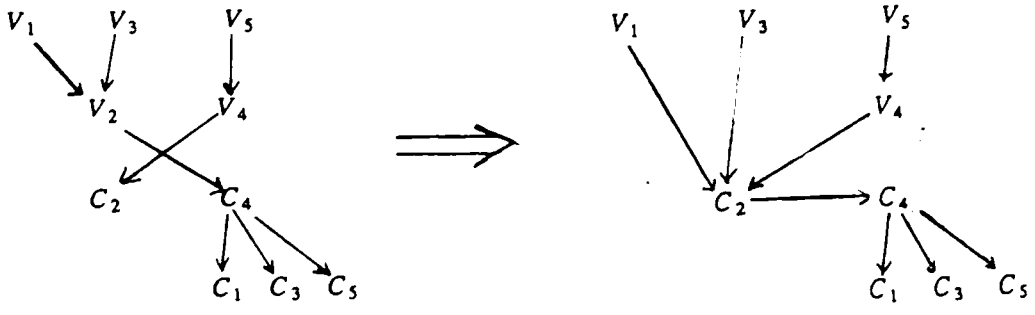


Figure 3.2:

satisfied by I', because, by definition of W^o, the removal of the arc $W^r \rightarrow W^{r+1}$ can only disconnect paths to the C-events in the C-subgraph tree rooted at C_i. However, all those C-events are now preceded by $C_{p_{r+1}}$, which is in a different C-subgraph tree. Therefore, I' is a commit instance, and has the same coordinators as I, except participant

i, that is a coordinator in I, but is not a coordinator in I'. Moreover, the transformation performed on I does not alter the communication cost, hence I' has minimum communication cost.

To summarize, starting with a minimum communication cost instance, I, with two or more coordinators, we obtained a minimum communication cost instance, I', with one less coordinator. We can continue this procedure until a minimum communication cost instance with exactly one coordinator is obtained. □

Next, we will obtain an additional lemma; it will be used in section 4. In a minimum communication cost instance, I, having two or more coordinators, let V^o, p_o, and C_i be as in the proof of Lemma 2. Namely, V^o is a V-event that precedes two or more boundary C-events, but each one of its V-event successors, if any, precedes only one boundary C-event. Such a V-event will be called a boundary *V-event*. We have shown in the proof of Lemma 2 that I must have at least one boundary V-event. Assume, as in the proof of Lemma 2, that C_i is a boundary C-event that is a successor of V^o, is not C_{p_o}, and does not precede C_{p_o}. Such a boundary C-event will be called associated with boundary *V-event V^o*.

The fact that I is a minimum communication cost instance, implies that the path in I, denoted p, from V^o to C_i, is unique, and consists of a single message arc, $V^o \rightarrow C_i$. The reason for this is as follows. If this is not the case, then there are more messages on p, or there are additional paths from V^o to C_i. As established in the proof of lemma 2, one of the messages on p is from p_r to p_{r+1}. Then, the transformation in the proof of Lemma 2 can be augmented by the elimination of all messages on p, and on the other paths from V^o to C_i. The resulting graph is a commit instance that has a cost strictly lower than I. This contradicts the fact that I has a minimum communication cost. Therefore, $V^o \equiv W^r$, $W^{r+1} \equiv C_i$, $p_o \equiv p_r$, $i \equiv p_{r+1}$, and the transformation in the proof of Lemma 2 simply replaces a message $V_{p_o,l} \rightarrow C_i$ by a message $C_{p_o} \rightarrow C_i$. We have therefore proved that:

Lemma 3: In any minimum communication cost instance of Ψ, with two or more coordinators, there is at least one boundary V-event. Additionally, there is a unique path from any boundary V-event, V^o, to any associated C-event, C_i, and it consists only of the message $V^o \rightarrow C_i$. □

Now, let us define the cost graph, D, as the complete graph having the set of participants P as its nodes. The weight of each edge (i,j) equals c_{ij}. The cost of a minimum spanning tree in D is denoted by CMST(c,P).

Given T_1 and T_2, two not necessarily different spanning trees of D, we define *a commit instance on T_1 and T_2 coordinated at some participant $k \in P$*. It is denoted by $I(k,T_1,T_2)$, and specified as follows. Denote by

T'_1 the directed graph obtained from T_1 by directing its edges such that from every node there is a path to k (the graph obtained is called an *oriented* tree with sink k). Also, denote by T'_2 the directed graph obtained from T_2 directing its edges to form a rooted tree, with root k.

One obtains the instance $I(k,T_1,T_2)$ as a result of the following modifications:

(1) relabel the nodes from T'_1; node i is relabeled V_i,

(2) relabel the nodes from T'_2; node i is relabeled C_i,

(3) node V_k is omitted, and the arcs entering it are modified to enter C_k instead.

(4) add the order arcs $V_i \rightarrow C_i$ for each i, except the coordinator.

Intuitively, $I(k,T_1,T_2)$ is a commit instance that has vote messages which correspond to the arcs of T'_1, and commit messages which correspond to the arcs of T'_2. The definition of $I(k,T_1,T_2)$ is illustrated in Fig. 3.3. Given minimum spanning trees T_1 (a) and T_2 (b), the instance $I(4,T_1,T_2)$ is illustrated in (c). Clearly, if T_1 and T_2 are minimum spanning trees, then the communication cost of $I(k,T_1,T_2)$ is $2 \cdot CMST(c,P)$.

Theorem 1: Let P be a set of participants, and c a set of associated communication costs. Then $\min_{I \in \Psi} Cost(I) = 2 \cdot CMST(c,P)$.

Proof: Since the cost of an instance $I(k,T_1,T_1)$ for some minimum spanning tree T_1 is $2 \cdot CMST(c,P)$, then clearly $2 \cdot CMST(c,P) \geq \min_{I \in \Psi} Cost(I)$. To obtain the inequality in the other direction, observe that, by Lemma 2, there exists

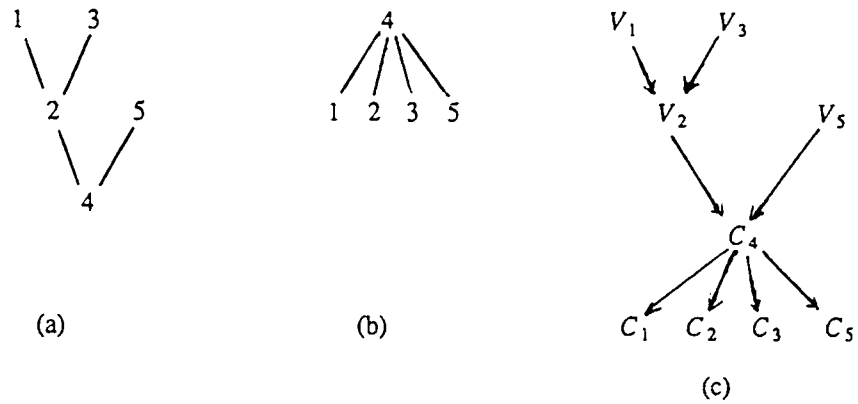


Figure 3.3:

a minimum communication cost instance, I^*, with one coordinator, say k. Based on I^* construct an undirected graph, H, defined as follows. The nodes of H are the participants in P, and edges of H are $\{(i,j) |$ there is a vote message from participant i to participant j in $I^*\}$. Since in I^* there is a path from the first event at every participant to C_k, H must be connected. Therefore its cost is at least $CMST(c,P)$, which in turn implies that the communication cost of vote messages in I^* is at least $CMST(c,P)$. Similarly we can show that the cost of the commit messages of I^* is at least $CMST(c,P)$. Thus, $2 \cdot CMST(c,P) \leq \min_{I \in \Psi} Cost(I)$. □

The result of Dwork and Skeen ([DS1, Theorem 1]) obtained for synchronous networks is extended to asynchronous networks by the following corollary of Theorem 1.

Corollary 1: Assume that the number of participants in a transaction is p. If the communication cost of a message between each pair of participants is one, then it holds true that $\min_{I \in \Psi} Cost(I) = 2(p-1)$. □

Hadzilacos obtained a similar extension of the result in the context of process failures and/or communication failures ([H2, Theorem 6]).

Now we shall present an algorithm that achieves minimum communication cost. In discussing commitment algorithms we assume, as in other works (e.g. [DS2]), that each participant knows the identity of all the participants, and we also assume that it knows the associated set of communication costs. The analysis of this section suggests a very simple minimum communication cost commitment algorithm, which we will call FIXED-COORDINATOR. It proceeds as follows. Each participant constructs some minimum spanning tree, T, of the cost graph, and selects a participant, k, designated as the coordinator. T and k are assumed to be identical at all the participants. This will be the case if the procedure which constructs T and selects k, is identical at all the participants. The algorithm executes an instance with the vote messages corresponding to T' (which is the oriented tree with sink k, obtained by directing the edges of T towards k); the commit messages correspond to T'' (which is the rooted tree obtained from T by directing its edges away from k). Specifically, a participant $i \neq k$ waits until subtransaction completion and until the receipt of all vote messages represented by the arcs incoming into i in T'. Then it will send a 'yes' vote message corresponding to the arc exiting i. Participant k, after completing its subtransaction, waits until receiving the votes from all its neighbors in T, and then commits. The propagation of the commit decision is similar, in the opposite direction.

Observe that the linear and central-site algorithms discussed in the introduction, are special cases of the FIXED-COORDINATOR algorithm, in which the requirement that T is a minimum spanning tree is relaxed. In the central-site case, the tree consists of the coordinator connected by an edge to each other participant. In the linear case, the tree is simply a string, in which the coordinator is one of the leaves.

4. CHARACTERIZATION OF COMMIT INSTANCES WITH MINIMUM COMMUNICATION COST

In this section we provide a complete characterization of all possible commit instances of minimum communication cost (Theorem 2). We determine that if a commit instance has a minimum communication cost, then each participant sends at most one vote message, and receives at most one commit message (a coordinator does not receive a commit messages). Also, in a minimum communication cost instance there are either one or two coordinators. If there are two coordinators, then each participant sends exactly one vote message. If there is only one coordinator, then each participant except the coordinator sends one vote message and receives one commit message; the coordinator does not send a vote message. In either case, the messages of a minimum communication cost instance propagate "along edges" of minimum spanning trees of the cost graph. Specifically, there are two (not necessarily different) minimum spanning trees of the cost graph, such that the vote messages are only sent from a participant to its neighbor in one tree, and commit messages are only sent from a participant to its neighbors in the other. Furthermore, if the instance has two coordinators, then they must be neighbors in both trees, each one of them must send its vote message to the other, and no commit messages are sent between them.

A commit instance on spanning trees T_1 and T_2, coordinated at a participant k, was defined in section 3, and denoted $I(k,T_1,T_2)$. Similarly, we define next a commit instance on two spanning trees, coordinated at two participants. Assume that T_1 and T_2 are two spanning trees of the cost graph, such that participants m and n are neighbors in both trees. A *commit instance on T_1 and T_2 coordinated at m and n*, denoted $I(m,n,T_1,T_2)$, is defined as follows. Denote by T'_1 the graph obtained from T_1 by directing its edges to obtain an oriented tree with sink m, then omitting from it the arc $n \rightarrow m$. Denote by T'_2 the the graph obtained from T_2 by directing its edges to obtain a rooted tree with m as a root, then omitting from it the arc $m \rightarrow n$.

One obtains the instance $I(m,n,T_1,T_2)$ as a result of the following modifications:

(1) relabel the nodes from T'_1; node i is relabeled V_i.

(2) relabel the nodes from T'_2; node i is relabeled C_i,

(3) add the message arcs $V_m \rightarrow C_n$ and $V_n \rightarrow C_m$, and the order arcs $V_i \rightarrow C_i$ for each i.

Intuitively, $I(m,n,T_1, T_2)$ is a commit instance having vote messages that correspond to the arcs of T'_1, plus the arcs $m \rightarrow n$ and $n \rightarrow m$, and commit messages that correspond to the arcs of T'_2. In other words, in $I(m,n,T_1, T_2)$ each participant, including the coordinators, sends exactly one vote message. The vote of n is received by m, and vice versa. Each participant, except the coordinators, receives exactly one commit message. Fig.4.1 demonstrates the definition. It illustrates a minimum communication cost instance on T_1 and T_2 of Fig. 3.3, coordinated at participants 2 and 4. Note that $I(m,n,T_1, T_2)$ is the same graph as $I(n,m,T_1, T_2)$.

Theorem 2: Let P be a set of participants, and let c be a set of associated communication costs. Any minimum communication cost instance $I \in \Psi$, must be of the form $I(k, T_1, T_2)$ or $I(m, n, T_1, T_2)$, for some participants k, m, n, and minimum spanning trees T_1 and T_2 of the cost graph.

In order to prove Theorem 2 we shall show that: a) any one-coordinator, minimum communication cost instance, must be of the form $I(k,T_1,T_2)$; b) any two-coordinator, minimum communication cost instance, must be of the form $I(i,j,T_1,T_2)$; and c) a commit instance with three or more coordinators cannot have minimum communication cost. Fix the set of participants, P, and its associated set of communication costs, c, for the rest of this section.

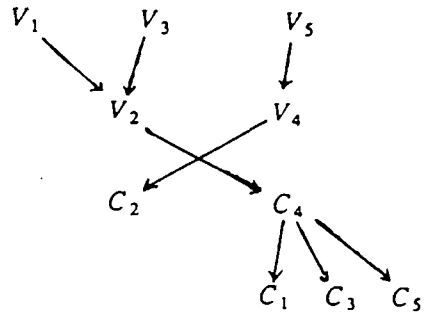


Figure 4.1:

Lemma 4: A one-coordinator minimum communication cost commit instance must be of the form $I(k,T_1,T_2)$ for some minimum spanning trees T_1 and T_2.

Proof: Consider a minimum communication cost instance, I, with one coordinator, k. Following a line of reasoning similar to one used in the proof of Theorem 1, it can be easily seen that the total cost of vote messages is exactly $CMST(c,P)$, and the total cost of commit messages is exactly $CMST(c,P)$. Consider the undirected graph, H, having as nodes the participants, and edges $\{(i,j) |$ there is a vote message from participant i to participant j in $I\}$. H must be connected, its cost is $CMST(c,P)$, therefore it is a minimum spanning tree of the cost graph, T_1. Clearly, the vote messages of I correspond to the oriented tree with sink k, obtained by directing the edges of T_1. Similarly we can show that the commit messages of I correspond to the rooted tree obtained by directing the edges of a minimum spanning tree, T_2. □

In Lemma 2 we presented a procedure to modify any minimum communication cost instance, I, with two or more coordinators, into another commit instance I' with one less coordinator, such that the communication costs of I and I' are identical. A similar procedure will be used here to characterize minimum communication cost instances with two or more coordinators.

Lemma 5: A two-coordinator minimum communication cost commit instance, I, must be of the form $I(i,j,T_1,T_2)$, for some participants i and j, and some minimum spanning trees, T_1 and T_2, both having the edge (i,j).

Proof: Let C_i and C_j be the boundary C-events in I. By replacing one vote message by a commit message, as in Lemma 2, we obtain a one-coordinator instance of minimum communication cost. We then use the characteristics of such an instance given in Lemma 4, to show that I is indeed of the form $I(i,j,T_1,T_2)$.

Let V^o denote a boundary V-event, as defined after the proof of Lemma 2. Since i and j are the only coordinators, V^o precedes both C_i and C_j. Let C_i be the boundary C-event associated with V^o, and p_o the participant at which V^o occurs. Since there are only two boundary C-events, by definition of an "associated boundary C-event", C_{p_o} is in the tree of the C-subgraph which is rooted at C_j.

We first show that, in fact, p_o is the coordinator j. By Lemma 3 observe that the path from V^o to C_i is the message arc $V^o \rightarrow C_i$. Consider the commit instance I' obtained from I by replacing $V^o \rightarrow C_i$ by $C_{p_o} \rightarrow C_i$ (as in the proof of Lemma 2). I' has only one coordinator, j, and has the same cost as I, i.e. minimum communication cost. Therefore, by Lemma 4, the instance I' must be of the form $I(j,T_1,T_2)$ for some minimum spanning trees T_1,T_2. In

particular, in the instance I', the coordinator j does not send a vote message, whereas in I, participant j must send a vote message, because there must be a path from the first event at j to C_i. However, the only vote message deleted in I to obtain I' is $V^o \rightarrow C_i$; hence V^o occurs at participant j, or, in other words, $p_o \equiv j$.

Because of the new arc, $C_{p_o} \rightarrow C_i$, j and i are neighbors in T_2. We shall show that i is a neighbor of j in T_1 as well, or, in other words, that i sends its vote to j in I. Denote the single V-event at each participant r in I by V_r. We have already shown that if V^o has C_i as an associated C-event, then V^o is actually V_j. If there is no other boundary V-event in I, then all V-events must precede V_j, and therefore V_j should be a C- rather than a V-event. Consequently V_j cannot be the only boundary V-event in I. Denote by V^1 another boundary V-event, i.e. $V^1 \neq V^o$. The associated C-event of V^1 cannot be C_i , since otherwise as above, we can show that $V^1 = V_j$, which in turn implies that $V^1 = V^o$. Thus, the associated C-event of V^1 must be C_j, and, as above, V^1 occurs at participant i; that is, V^1 is actually V_i, so participant i sends its vote to participant j.

Now, recall that I' is obtained from I simply by replacing $V_o \rightarrow C_i$ by $C_j \rightarrow C_i$, and that I' has the form $I(j,T_1,T_2)$. Therefore, in the instance I, exactly one vote message is sent by each participant, and I is of the form $I(i,j,T_1,T_2)$ (by definition of such an instance). □

Lemma 6: There are no minimum communication cost commit instances with three or more coordinators.

Proof: From the proof of Lemma 2, we know that for every minimum communication cost instance with two or more coordinators, there exists a minimum communication cost instance with one less coordinator. Consequently, it is sufficient to show that there is no minimum communication cost instance with three coordinators.

Suppose that there exists a minimum communication cost instance, I, with three coordinators i,j,m. Let V^o be a boundary V-event, and C_i an associated boundary C-event. Assume that V^o occurs at participant p_o. By replacing $V^o \rightarrow C_i$ with $C_{p_o} \rightarrow C_i$, we obtain, as in Lemma 2, an instance I' of minimum communication cost with two coordinators, j and m. By Lemma 5, instance I' must have the form $I(j,m,T_1,T_2)$, for some minimum spanning trees T_1 and T_2. Event V^o occurs at exactly one participant, therefore it cannot occur at both coordinators. Assume without loss of generality that it does not occur at coordinator j, i.e. $p_o \neq j$. Then the vote messages exiting V-events occurring at participant j are the same in $I(j,m,T_1,T_2)$ and I. However, by definition, the only such vote message in $I(j,m,T_1,T_2)$ is $V_j \rightarrow C_m$. Therefore, in I, the only V-event occurring at participant j does not precede boundary C-event C_i. Thus I does not satisfy the commit requirement, contradicting the fact that I is a commit instance. □

We have completed the proof of Theorem 2; as an immediate corollary we conclude:

Corollary 2: If the communication cost of some commit instance with p participants is minimum, then it has a minimum number of intersite messages, i.e., 2(p-1). □

Obviously, as demonstrated in the introduction, the converse of corollary 2 is not true, i.e. minimum number of intersite messages does not imply minimum communication cost.

The characterization in Theorem 2 helps us demonstrate that, in general, minimum communication cost cannot be achieved with limited knowledge of participants' identity. For example, consider the network and the participants of Fig. 1.1. Suppose that each one of the participants 2,3 and 4 knows only that 1 is a participant, but does not know of the existence of other participants. Suppose also that 1 knows that 4 is a participant, and that there are two other participants, but 4 does not know their identity. A possible commitment scenario is that participant 4 waits for the votes of all the other participants, and then propagates the commit decision. Another scenario is that 1 only waits until receiving the votes of the two anonymous participants, and then transmits their vote, along with its own, to 4; the latter then propagates the commit decision. In the two scenarios above, as well as in any other possible scenario, a message must be sent between 1 and 4. Since the edge 4-1 in the cost graph does not belong to any minimum spanning tree, minimum communication cost cannot be achieved.

5. COMMUNICATION TIME OF COMMIT INSTANCES

In this section we first define the communication time of an instance, and then we establish the minimum communication time of a commit instance.

Generally, time comparison of instances in a totally asynchronous network is impossible, because each message can have an arbitrarily long delay. Therefore some restrictions on the network behavior must be imposed. The only restriction we impose here is that the delay of a message between every pair of participants is fixed for the duration of any commitment-algorithm execution. Thus, any message from i to j, sent by any algorithm, takes a fixed and finite amount of time to arrive, say t_{ij}.

The communication-time of an instance on a set of participants, P, is defined with respect to a set $\tau = \{\tau_i \mid i \in P\}$ of *subtransaction completion times*, and with respect to a set $t = \{t_{ij} \mid i,j \in P\}$ of *intersite communication delays* (or *intersite delays* for short). For $i \neq j$, the intersite delay, t_{ij}, is a positive real number, and each τ_i is

a nonnegative real number. The smallest τ_i is zero, and so is every t_{ii}. The set t satisfies the triangle inequality, namely, for each i,j,k, $t_{ij} \leq t_{ik}+t_{kj}$. We observe that t_{ij} may be different than t_{ji}. t_{ij} is the interval of time from the send of any message at i, until its receive at j, in any instance on P. Let I be an instance on P. The sending of each message exiting an event, say e, happens at the execution time of e, defined as follows.

Let e be an event at participant i. The *execution time* of e, denoted *time*(e), is the maximum of:

1. The last (highest) receive of a message entering e,
2. The execution time of the event immediately preceding e at participant i, if there is any, and
3. The subtransaction completion time, τ_i.

Observe that the execution times of events on any directed path in I are nondecreasing.

The *communication time* of the instance I, denoted *time*(I), is defined as the maximum execution time of an event in I (i.e., the execution time of the last event). Intuitively, the communication time of I is the interval from the time that the first participant completes its subtransaction (taken to be zero), until the execution time of the last event, assuming the following. A message between every pair of participants, i and j, takes t_{ij} time units, a participant i does not send its first message before time τ_i, and internal processing after subtransaction completion takes zero time at each participant. Clearly, for given sets of subtransaction completion times and intersite communication delays, the communication time may differ from one instance to another. For example, assume that all the subtransaction completion times are zero, and all the intersite delays are one. Then the communication time of the instance in Fig.2.1(a) is two, whereas the communication time of the instance in Fig.2.1(b) is one. Given an instance I and sets τ and t, *time* (I) can be computed in linear time by PERT techniques (see [E]).

Proposition 1: Let P be a set of participants, let τ be a set of subtransaction completion times, and let t be a set of intersite delays. Then the minimum communication time of an instance in Ψ (the set of commit instances for P), with respect to τ and t is: $\max \{\tau_i+t_{ij} \mid i \text{ and } j \text{ are participants}\}$.

Proof: Consider some participant, i. In any instance, I of Ψ, the execution time of the first event, say e, at participant i cannot be smaller than τ_i. Also, there must be a path from e to some event at every other participant. Since t satisfies the triangle inequality, $\{\tau_i+t_{ij} \mid j \text{ is a participant}\}$ is a lower bound on *time*(I). i is an arbitrary participant, thus $\max \{\tau_i+t_{ij} \mid i \text{ and } j \text{ are participants}\}$ is a lower bound on *time*(I). The lower bound is also an upper bound, since it is the communication time of the commit instance in which every participant, i, has one V_i event, and one C_i event, and there is a message from every V_i to every other C_j (e.g. the instance in Fig. 2.1b). (Remark: observe that

if some τ_k is bigger than τ_i+t_{ik}, for each $i \neq k$, then the the execution time of both, V_k and C_k, is τ_k.) □

Consider the *decentralized* algorithm, in which each participant sends its vote to all the other participants when its subtransaction completes. If all the participants vote 'yes', then clearly this algorithm executes a minimum communication time commit instance, for any sets τ and t.

6. THE TREE-COMMIT ALGORITHM

TREE-COMMIT is a distributed, minimum communication cost algorithm, adapted from the PIF (Propagation of Information with Feedback) algorithm of [Se]. Each participant constructs the same minimum spanning tree, T, of the cost graph. In contrast to the fixed-coordinator algorithm, in TREE-COMMIT, a coordinator is not selected when the tree is constructed. The procedure performed by each participant, i, in a committing execution is as follows. After subtransaction completion, it waits until receiving the votes from all its neighbors in T except one, say j, before voting; then it sends its vote to j. If i receives votes from all neighbors in T before it completes its subtransaction, then i commits and becomes the single coordinator. If i receives a vote message from j after having sent its vote to j, then it commits becoming one of two coordinators (j is the other one). If i receives a commit message from j, then it sends commit messages to all its neighbors in T, except j. Therefore, the votes travel from the leaves of T inwards, where one or two coordinators are established. In the commit stage, the commit message is simply propagated along the tree edges, away from the coordinator(s).

A possible situation in the voting stage, i.e. before the coordinators are determined, is illustrated in Fig. 5.1a. In the scenario illustrated, we suppose that participants 1,2, and 5 have completed their subtransactions, and participants 3 and 4 have not. Participant 2 has not voted yet because it has not received the votes of two of its neighbors. Fig. 5.1b, 5.1c illustrate two possible instances executed by TREE-COMMIT, at the completion of the voting stage situation described above. In the first case, 3 completed its subtransaction, and its vote had reached 2 before the vote of 4 did so; furthermore, the votes of 2 and 4 crossed, so they both became coordinators. In the second case, 3 completed its subtransaction after having received the vote of all participants, so 3 became the sole coordinator.

We will denote by TREE-COMMIT(T) the algorithm which uses the tree T. The reader should convince herself/himself that TREE-COMMIT(T) generates an instance of the form $I(k,T,T)$ or $I(m,n,T,T)$, for some coordinators m,n,k.

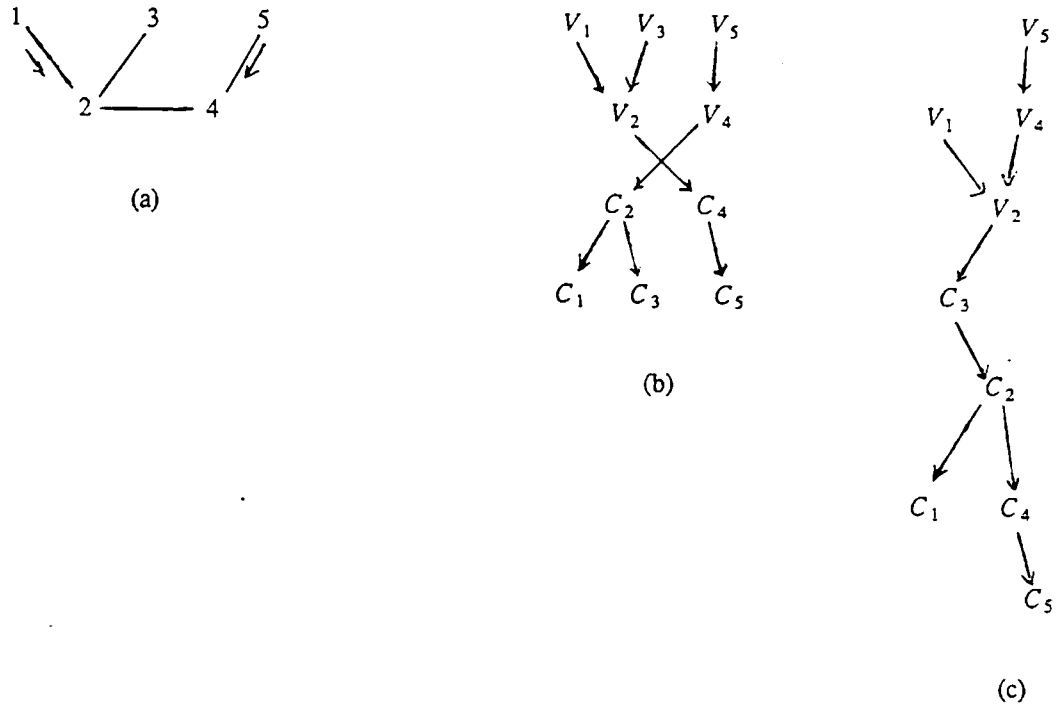


Figure 5.1

The transaction-abort case is handled by TREE-COMMIT as follows: Participant i sends 'abort' messages when the first of the following two cases occurs: i) i unsuccessfully completed its subtransaction, in which case i sends an 'abort' (or a 'no' vote) message to each neighbor in T; ii) i receives the first 'abort' message from a participant, say k, in which case i sends an 'abort' message to each neighbor, except k.

The complete TREE-COMMIT algorithm for each participant, i, is given in Fig. 5.2. Note that TREE-COMMIT uses only 'yes', 'commit', and 'abort' messages, therefore the message length can be restricted to two bits.

We have not formally defined an 'abort' instance, but the next theorem indicates that the lower bound on its communication cost is lower than the commit instance lower bound.

Theorem 3: If some participant sends an 'abort' message, then the communication cost of the instance executed by TREE-COMMIT(T) is at least the cost of T, and at most twice the cost of T.

Proof: Clearly, commit messages are sent only if all the subtransactions completed successfully, and therefore, if

```
TREE-COMMIT(c,P)  /* procedure executed by participant i, given a set of communication costs, c,
                                                        between participants in P */

1. Construct a minimum spanning tree, T, of the cost graph.

2. Wait until subtransaction completion, or receipt of an 'abort' message. /* vote-messages received are saved,
                but they do not wake-up this process; such messages are being considered in steps 5,6 */

3.      if subtransaction completion then do;

4.          if successful completion then do;

5.              if received 'yes' votes from all neighbors in T, then send
                 'commit' messages to all neighbors, and quit.    /* i is a single coordinator */

6.              otherwise wait until receiving the first 'abort' message, or until
                 receiving a 'yes' vote from all neighbors, except one.

7.                      if 'abort' message from a neighbor, say k, then send 'abort' messages to all
                         neighbors, except k, and quit.

8.                      if 'yes' votes from all neighbors, except one, say j, then send a 'yes' vote to j.

9.          end.

10.         otherwise (unsuccessful completion) send 'abort' messages to all neighbors, and quit.

11.     end.

12.     otherwise ('abort' message from a neighbor, say k) send 'abort' messages to all neighbors,
         except k, and quit.

13. Wait until receiving a message.        /* the only way to get here is from step 8,
                                        and the message must have been received from j */

14.     if 'abort' message then send 'abort' messages to all neighbors, except j, and quit.

15.     otherwise ('commit' or 'yes'-vote) send 'commit' messages to all neighbors, except j,
         and quit. /* in the 'yes'-vote case, i and j are coordinators */
```

Figure 5.2

some participant sends an 'abort' message, then no commit messages are sent. Each message in the instance executed by TREE-COMMIT(T), is sent between two neighbors in T, therefore let us consider the edges of T. For each edge (i,j) there is either exactly one abort message from i to j, or exactly one 'yes' vote message from i to j, but not both. Similarly from j to i. Hence the total cost of messages is at most twice the cost of T. Additionally, note that for each edge (i,j) of T there is an 'abort' message from i to j, or from j to i. Hence the total communication cost of

the instance is at least the cost of T. □

It is easy to see that the bounds of Theorem 3 are tight. If all the subtransactions complete unsuccessfully at exactly the same time, then two 'abort' messages are sent along each edge of T, one in each direction, and the communication cost is twice the cost of T. If, on the other hand, some participant completes its subtransaction unsuccessfully, and after this, the 'abort' messages reach each participant before it completed its subtransaction, then one 'abort' message is sent along each edge of T, and the communication cost is the cost of T.

The algorithm TREE-COMMIT assumes that each participant also knows the identities of all the other participants. It is possible that for some transactions this assumption does not hold true, and then TREE-COMMIT cannot be used. However, let us mention two frequent cases in which the assumption does hold true. First, it holds true in a fully replicated database, since then each participant knows that all the other local database managers are participants. Second, for simple update transactions, that accesses only one data item (e.g. add 10,000 dollars to an account which is replicated at four participants), the assumption holds true, since each participant usually stores a directory, indicating where each data-item is replicated.

7. ANALYSIS OF TREE-COMMIT

In this section we first establish the communication time of TREE-COMMIT, and then we show that it is at most p times the minimum communication time commit instance (subsection 7.1). Then we define the communication complexity of an instance, we discuss the communication complexity of TREE-COMMIT, and we point out that it constitutes a polynomial-time approximation for an NP-complete problem (subsection 7.2). Then we consider the other atomic commitment algorithms discussed in this paper, and compare their performance with TREE-COMMIT (subsection 7.3).

7.1 The Communication Time

Given a set of participants and a spanning tree, T, namely a tree in which the nodes are the participants, a *commit instance on T* is a commit instance in which every message is from a processor to one of its neighbors in T. A commit instance on T will be called for short a T-instance. TREE-COMMIT obviously can propagate messages along any spanning tree (simply do not insist on "minimum" in step 1 of Fig. 5.2) and in this section we will speak in this broader context. We shall show that for a spanning tree, T, TREE-COMMIT(T) executes the instance with

the minimum communication time among all T-instances, given any sets of subtransaction completion times and intersite delays.

In this section, the edge (i,j) of an undirected tree often represents the two arcs $i \rightarrow j$ and $j \rightarrow i$; whether it does so or not will be clear from the context. Also, whenever we speak of the length of a path from one node to another in a tree, we assume that the length of each arc, $i \rightarrow j$, is the intersite communication delay t_{ij}. Given a set of intersite communication delays, t, a tree T in which the nodes are the participants, and a participant r, denote by d_r the longest simple path in T, from r to another node.

Lemma 7: Let P be a set of p participants, let T be a tree in which the nodes are the participants, let $\tau = \{\tau_1, \ldots, \tau_p\}$ be a set of subtransaction completion times, and let $t=\{t_{ij} \mid i,j \in P\}$ be a set of intersite communication delays. Then the communication time of a T-instance, I, with respect to τ and t, is at least $\max_{i \in P}\{\tau_i+d_i\}$.

Proof: Let k be an arbitrary participant. There must be a path in I, from $V_{k,1}$ (or $C_{k,1}$, if k is a single coordinator in I) to every other $C_{i,1}$. Since the messages of I are transmitted only between neighbors in T, the communication time of I cannot be lower than $time(V_{k,1})+d_k$ (or $time(C_{k,1})+d_k$). Additionally, $time(V_{k,1})$ and $time(C_{k,1})$ cannot be lower than τ_k. □

Lemma 8: Let P, T, τ and t be as in Lemma 7. Then the communication time of (the instance generated by) TREE-COMMIT(T) for τ and t, is not higher than $\max_{i \in P}\{\tau_i+d_i\}$.

Proof: The instance generated by TREE-COMMIT, denoted I, is of the form $I(k,T,T)$ or $I(m,n,T,T)$, and particularly I has one or two coordinators. If it has one coordinator, k, then the communication time of I is clearly $\{\tau_k+d_k\}$, and the theorem follows. Assume now that I has two coordinators, m and n. We shall show that in this case there also exists some participant, k, such that the communication time of I is $\{\tau_k+d_k\}$.

First, note that the communication time of I equals the execution time of some C-event, say C_q. The C-subgraph of I consists of two rooted trees, one at C_n, and the other at C_m. Suppose that C_q is in the tree rooted at C_n. Then, clearly $time(C_q) = time(C_n)$ + [distance in T from n to q]. Suppose now that the edge (m,n) is removed from T. Denote by T_m and T_n the resulting subtrees that contain m and n respectively. Obviously, q is in T_n. Now comes the simple, but important observation, that lies at the heart of this proof. It is that, since in TREE-COMMIT participant n voted before receiving the vote message from m, then $time(C_n) = time(V_m) + t_{mn}$.

(Note that $time(C_n)$ may be bigger than $time(V_m) + t_{mn}$; it is, if $time(V_n) > time(V_m) + t_{mn}$). This observation is

crucial because, as we shall point out, there is a participant, k, in T_m, such that $time(V_m) = \tau_k$ + [distance in T from k to m]. Since q and k are in T_n and T_m, respectively, and particularly since they are not in the same subtree, then $time(C_q) = \tau_k$ + [distance in T from k to q].

Therefore, left to prove is only that there is a participant, k, in T_m, such that $time(V_m) = \tau_k$ + [distance in T from k to m]. For this, consider $time(V_m)$. By definition, it is either equal to τ_m, in which case the proof is complete, or, there is a message, say $V_j \rightarrow V_m$, such that $time(V_j)+t_{jm}=time(V_m)$. In the latter case, consider V_j. It is either equal to τ_j, in which case, again, the proof is complete, or, there is a message, say $V_u \rightarrow V_j$, such that $time(V_u)+t_{uj}=time(V_j)$. Proceeding in this fashion, we must eventually encounter an event, V_k, such that $time(V_k)=\tau_k$. Then, clearly, $time(V_m) = \tau_k$ + [distance in T from k to m]. □

From Lemmas 7 and 8 we immediately obtain:

Theorem 4: Let P, T, τ and t be as in Lemma 7. Then the communication time of (the instance generated by) TREE-COMMIT(T) for τ and t, is $\max_{i \in P}\{\tau_i+d_i\}$. Furthermore, this communication time is minimum among the communication times of all the T-instances for the same sets τ and t.

Next we shall show that the communication time of TREE-COMMIT is at most p times bigger than the minimum communication time of an instance in Ψ.

Theorem 5: Let P, T, τ and t be as in Lemma 7. Denote by $I_{\min}$ the instance in Ψ that has minimum communication time with respect to τ and t, and denote by I_{TC} the instance executed by TREE-COMMIT(T) for τ and t. Then $time(I_{TC}) \,/\, time(I_{\min}) \le p$.

Proof: Denote by i the participant for which $\tau_i+d_i = time(I_{TC})$. By Lemmas 7 and 8, we know that there is such a participant. By Proposition 1 we know that $time(I_{\min}) = \max_{k,j \in P}(\tau_k+t_{kj})$. Denote by $q \rightarrow r$ the longest arc in some longest simple path from i, in T (i.e., some path of length d_i). Denote by A the ratio $\tau_i+d_i \,/\, \max_{k,j \in P}(\tau_k+t_{kj})$. Remember that there are at most $p-1$ arcs on a simple path from i. Therefore, if $\tau_i \ge t_{qr}$, then the following holds true. $A \le p \cdot \tau_i \,/\, \max_{k,j \in P}(\tau_k+t_{kj})$; obviously, $\max_{k,j \in P}(\tau_k+t_{kj}) \ge \tau_i$; and consequently, $A \le p$. On the other hand, if $\tau_i < t_{qr}$, then the following holds true. $A \le p \cdot t_{qr} \,/\, \max_{k,j \in P}(\tau_k+t_{kj})$; obviously, $\max_{k,j \in P}(\tau_k+t_{kj}) \ge t_{qr}$; and again, $A \le p$. □

Next we demonstrate that the bound on the ratio between the communication time of TREE-COMMIT and the minimum communication time is tight. Assume that all the subtransaction completion times are zero, all the

intersite delays are one, and the tree, T_0, along which TREE-COMMIT propagates messages is a string from 1 to p, namely, 1—2—3—,....,—p. Then the time of TREE-COMMIT is $p-1$, whereas the minimum communication time is one.

7.2 The Communication Complexity

Let P be a set of participants, let c be a set of communication costs, let τ be a set of subtransaction completion times, and let t be a set of intersite communication delays. We define the *communication complexity* of an instance I on P, denoted $com(I)$, to be the product $cost(I) \cdot time(I)$. Note that it is possible for the minimum-communication-complexity commit-instance not to be a minimum-communication-cost commit-instance, nor a minimum-communication-time commit-instance.

Since TREE-COMMIT on a minimum spanning tree has a minimum communication cost, we obtain as an immediate consequence of Theorem 5 that:

Corollary 3: Let P, τ, and t be as in lemma 7. Let c be a set of costs associated with P, let I_{TC} be the instance generated by TREE-COMMIT on a minimum spanning tree of the costs graph, and denote by $I_{\min}$ the instance in Ψ that has a minimum communication complexity with respect to c, τ, and t. Then $com(I_{TC}) \,/\, com(I_{\min}) \le p$.

It is easy to see that $O(p)$ is a tight bound on the ratio between the complexity of TREE-COMMIT and the optimal complexity. Simply consider again the tree T_0 (namely, 1—2—3—,....,—p), and assume that all the communication costs and intersite delays are one, and all the subtransaction completion times are zero. Then the complexity of TREE-COMMIT(T_0) is $2(p-1)^2$, whereas the following instance has a complexity of $4(p-1)$. All the participants send their vote to 1, and 1 sends the commit message to all the other participants (see Fig 2.1a).

Given an integer, K, it is NP-complete to determine whether there exists an instance I in Ψ, for which $com(I) \le K$. In other words, the problem of finding the minimum complexity instance (MCI) is NP-complete.

For proof, observe first that the MCI problem is obviously in NP. A nondeterministic algorithm need guess an instance, and check whether its communication complexity is $\le K$. The MCI problem reduces to the MINIMUM RADIUS MINIMUM SPANNING TREE (MRMST[1]) problem defined as follows.

Input: A connected graph $H=(V,E)$, a vertex $r \in V$ (the root), a (weight) function, w, that maps each edge of E to an

1. The MRMST problem was shown NP-complete by Itai ([I]). Note that the problem closely resembles the BOUNDED DIAMETER SPANNING TREE problem (ND4 in [GJ]).

integer, and that satisfies the triangle inequality, and an integer k.

Question: Does H have a minimum spanning tree, T, such that the distance from r to any node is less than, or equal to, k.

The reduction is straight-forward. Given an MRMST instance we construct an MCI instance as follows. Assume that m is the total weight of a minimum spanning tree of H. Let z be a node that is not in H. Define $P = V \bigcup \{z\}$. The set of communication costs, c, is defined as follows. For any pair of nodes, i and j that are both in V, such that (i,j) is in E, $c_{ij}=c_{ji}=w(i,j)$. If i is z and j is r, or vice versa, then $c_{ij}=c_{ji}=1$. Each one of the remaining costs, c_{ij}, is simply the length of the shortest path between i and j, in the incomplete graph created thus far. The set of intersite delays, t, is defined as follows. For any pair of nodes, i and j that are both in V, $t_{ij}=c_{ij}$. Additionally, $t_{zr}=t_{rz}=k$. For each one of the remaining intersite delays, $t_{ij}=t_{ji}=$*length of the shortest path between* i and j, *where the length of each edge* (r,q) *is* t_{rq}. The set of subtransaction completion times, τ, is defined as follows. If $i \neq r$ then $\tau_i=0$; otherwise, $\tau_r=(2m)^4$. Finally, $K=2(m+1)[(2m)^4+k]$.

It is straight-forward to see, using Theorem 2, that the MCI instance has a solution if and only if the MRMST instance has a solution. Actually, the same transformation can be used to show that the following two problems are also NP-complete. Finding, among all the instances that have minimum communication cost, one that has a minimum communication time. And, finding among all the instances that have minimum communication time, one that has a minimum communication cost.

Let us consider now the computation time complexity of TREE-COMMIT, i.e., its time-complexity assuming that the communication time is zero (in contrast, remember that the communication-time is defined assuming that the computation- or internal processing- time was zero). The computation time is dominated by the complexity of finding a minimum spanning tree, and consequently is $O(p^2)$. Corollary 3 means that the following algorithm is a polynomial-time, bounded error approximation, for the MCI (minimum complexity instance) problem. Given P, c, τ, and t, first find a minimum spanning tree, T, of the cost graph. Then simulate TREE-COMMIT(T) to find the coordinator(s), and establish the instance.

7.3 Comparison with Other Algorithms

First, consider the communication complexity and the communication cost of the decentralized algorithm (see section 5), that has minimum communication time. We shall demonstrate that the complexity (cost) can be $O(p^2)$ times the minimum communication complexity (cost). Assume that T_0 = 1—2—3—,....,—p is a minimum spanning tree of the cost graph, and the cost of each edge in T_0 is one. For every pair of participants, i and j, let the communication cost c_{ij} be the distance in T_0 between i and j. Then it is easy to see that the communication cost of the instance executed by the decentralized algorithm is $O(p^3)$ whereas the minimum communication cost is $2(p-1)$.

For demonstrating the complexity of the decentralized algorithm, let us continue this example, and suppose that for every pair of participants, i and j, $t_{ij}=c_{ij}$, and all subtransaction completion times are zero, except for the completion time of participant p, that is a very large number, say p^5. Then the communication complexity of the decentralized algorithm is $O(p^2)$ times the minimum communication complexity. An easy way to see this is to observe that TREE-COMMIT(T_0), in addition to having minimum communication cost, also has a minimum communication time.

Actually, it is easy to see that $O(p^2)$ is a bound on the ratio of the communication complexity (cost) of the decentralized algorithm, to the minimum communication complexity (cost), assuming that the costs satisfy the triangle inequality. The reason for this is that the cost each message is bounded by the cost of a minimum spanning tree of the cost graph, and the number of messages sent by the algorithm is $p(p-1)$.

Consider now the other two atomic commitment algorithms mentioned in the introduction, namely the *central-site* and the *linear* algorithms. As explained in section 3, they are both special cases of the FIXED COORDINATOR algorithm. The communication cost of the FIXED-COORDINATOR algorithm, propagating messages along the edges of some tree, is equal to the communication cost of TREE-COMMIT, propagating messages along the edges of the same tree. Based on Theorem 4, the communication time of TREE-COMMIT(T) is never higher than the communication time of an arbitrary algorithm, particularly FIXED COORDINATOR, in which messages are sent only between neighbors in T. Next, we shall establish the communication time of the FIXED COORDINATOR algorithm. For this we need to define, for a given a set of intersite communication delays, t, a tree T in which the nodes are the participants, and two participants r and k, the tree-distance from k to r, denoted d_{kr}; it is the length of the path along the tree edges, from k to r (remember, the length of $i \rightarrow j$ is t_{ij}).

Theorem 6: Let P be a set of participants, let r be a participant, and let T be a tree in which the nodes are the parti-

cipants. Denote by I the instance generated by the FIXED-COORDINATOR algorithm that has the coordinator r, and propagates messages along the edges of the tree T. For any set of subtransaction completion times, τ, and any set of intersite delays, t, $time(I)$ is $\max_{i \in P}\{\tau_i+d_{ir}\} + d_r$.

Proof: By definition, $time(I)$ is $time(C_r) + d_r$. Additionally, we shall show that $time(C_r) = \max_{i \in P}\{\tau_i+d_{ir}\}$. The proof for this is as follows. Obviously, $time(C_r)$ cannot be lower than $\max_{i \in P}\{\tau_i+d_{ir}\}$, since the vote of every participant must reach r along a path in the tree, and a participant, i, cannot send its vote message before time τ_i. However, $time(C_r)$ is not higher than $\max_{i \in P}\{\tau_i+d_{ir}\}$ for the following reason. By definition, $time(C_r)$ is either equal to τ_r, in which case the proof is complete, or, there is a message, say $V_j \rightarrow C_r$, such that $time(V_j)+t_{jr}=time(C_r)$. In the latter case, consider V_j. It is either equal to τ_j, in which case again the proof is complete, or, there is a message, say $V_u \rightarrow V_j$, such that $time(V_u)+t_{uj}=time(V_j)$. Proceeding in this fashion, we must eventually encounter an event, V_k, such that $time(V_k)=\tau_k$. Then, $time(C_r) = \tau_k$ + [distance in T from k to r]. □

Therefore, the communication complexity (time) of FIXED-COORDINATOR on some tree, T, can be twice the communication complexity (time) of TREE-COMMIT(T). This happens if $\tau_r=0$, and $\max_{i \in P}\{\tau_i+d_i\} = d_r$, and there is some participant k, such that $\{\tau_k+d_{kr}\} = d_r$. For example, consider the linear algorithm on some string, T, and assume that the set t is such that $t_{ij}=t_{ji}=1$ for each edge (i,j) of T, and the subtransaction completion time of each participant is zero. Then, assuming that there are p participants, the communication time of TREE-COMMIT is $p-1$, whereas the communication time of the linear algorithm is $2(p-1)$. The communication time of the central site algorithm cannot be exactly twice the communication time of TREE-COMMIT (since the requirement $\{\tau_k+d_{kr}\} = d_r$ necessitates that the coordinator is a leaf), but it can be arbitrarily close to twice the communication time of TREE-COMMIT. This happens, for example, if there is some participant, i, such that $t_{ir}=t_{ri}=1$, for each other participant, j, $t_{jr}=t_{rj}=\varepsilon$, and all subtransaction completion times are zero. Then the communication time of TREE-COMMIT is $1+\varepsilon$, whereas the communication time of the central-site algorithm is two.

Finally, we shall point out that the communication time of the FIXED-COORDINATOR algorithm cannot be more than twice the communication time of TREE-COMMIT. To realize that observe that $\max_{i \in P}\{\tau_i+d_{ir}\} \leq \max_{i \in P}\{\tau_i+d_i\}$, and also $d_r \leq \max_{i \in P}\{\tau_i+d_i\}$.

8. DISCUSSION

8.1 Conclusion

In this paper we discussed the communication cost, the communication time, the communication complexity, and the computation time complexity of atomic commitment algorithms. We established that the lower bound on the communication cost for solving the atomic commitment problem is twice the weight of a minimum spanning tree of the cost graph. Given a set of intersite delays, t, and a set of subtransaction completion times, τ, the lower bound on the communication time is max $\{\tau_i+t_{ij} \mid i$ and j are participants $\}$. TREE-COMMIT, a new atomic commitment algorithm introduced in this paper, achieves, in the absence of failures, minimum communication cost. The decentralized algorithm, achieves, in the absence of failures, minimum communication time. We also characterized the minimum communication cost commit instances, and showed that each such instance must propagate messages along two (not necessarily different) minimum spanning trees of the cost graph, and it that it must have one or two coordinators.

Then we analyzed TREE-COMMIT, and we compared it with existing variants of the two-phase commit paradigm, namely the decentralized, the linear, and the central-site algorithms. The communication time of TREE-COMMIT is at most p times the minimum communication time, where p is the number of participants. Consequently, its communication complexity, the product of its communication cost and its communication time, is also at most p times the minimum communication complexity (an NP-complete concept). Furthermore, the computation time of TREE-COMMIT is polynomial in the size of the input. The minimum communication time algorithm, the decentralized, is also an approximation of the minimum communication complexity, but its error can be of order p^2.

When compared with the linear and the central-site algorithms, that are special cases of the FIXED-COORDINATOR algorithm, TREE-COMMIT is not only better in the worst case, but it is better in any case, in the following sense. Its communication cost can be made equal to the communication cost of FIXED-COORDINATOR, by parameterizing TREE-COMMIT to propagate its messages along the same spanning tree as FIXED-COORDINATOR. Then, the communication-time of TREE-COMMIT cannot be worse than the communication time of FIXED-COORDINATOR, for any sets of the intersite delays and subtransaction completion times. Furthermore, for some sets of the intersite delays and subtransaction completion times, the communication-time of TREE-COMMIT is half the communication time of FIXED-COORDINATOR.

8.2 Future Work

Society as a whole becomes increasingly dependent communication and information dissemination. Also, in business, government, and military, transaction processing gains ground, often at the expense of other types of processing. Therefore, we feel that the issues of gossiping, atomic commitment, and similar problems, will become increasingly important, and we intend to continue the study initiated in this paper.

First, it is important to extend our results to handle failures. Some related questions are the following. What is the necessary communication cost and communication time of atomic commitment algorithms under different failure assumptions? Are they sufficient? How do existing algorithms (e.g. three phase commit) approximate the optimal communication complexity?

Second, it is interesting to determine the bounds on performance of algorithms having different levels of knowledge. For example, it is easy to extend our model to define abort instances. Intuitively, it is clear that if there is exactly one 'no'-voter, then a communication cost of one minimum spanning tree is necessary for abort. However, it is also intuitively clear that unless the participants know that there is one 'no'-voter, in which case, obviously executing an atomic commitment algorithm is superfluous, this necessary cost is not sufficient. We do not know how to prove this yet. Or, consider the example at the end of section 4. How do we determine the performance lower bounds, given limited knowledge of participants identity? We feel that the interesting approach taken by Hadzilacos in [H2] provides a solid foundation for solving these problems.

Finally, on a more technical note, we will point out that the results in this paper suggest the solution to another variant of the "gossiping" problem (see subsection 1.3) that is unsolved in the literature (see [HHL]). In this variant, the pairs of individuals communicate by telephone calls, or two-way sessions, as opposed to messages, or one-way sessions; in one call the two individuals *exchange* the pieces of the gossip known to them, rather than one transmitting to the other. The problem is, again, to find a solution that has minimum total cost (now c_{ij} is the cost of a telephone call between i and j). Notice that modelling an instance by a directed acyclic graphs, is inappropriate for communication by two-way sessions. When the cost of each call is one (or when counting the number of calls), then the following algorithm achieves the minimum (necessary) cost, i.e. $2p-4$ calls (see [BK]).

MINIMUM-NUMBER-OF-CALLS:

1. Partition the individuals into four groups, A,B,C, and D, and appoint a leader in each group. Denote the leaders

by a,b,c, and d, respectively.

2. Each leader calls every member in its group, collecting the information-piece from each one ($p-4$ calls).

3. a and b exchange information in one phone call, and so do c and d (2 calls).

4. a and c exchange information in one phone call, and so do b and d. At this point a,b,c, and d know the whole gossip.

5. Each leader calls every member in its group, telling each one the whole gossip ($p-4$ calls).

When the telephone-call costs differ from one pair to another, we conjecture that determining the minimum cost of a solution is NP-complete, and the reason will become clear in the discussion below. (Remember, for one-way communication the minimum cost can be determined in linear time.) Furthermore, we conjecture that there are two possible structures for the pattern of telephone calls that achieves minimum cost. The first is a pattern that resembles the one that has a minimum number of calls. Specifically, the individuals are partitioned into four groups as in the algorithm MINIMUM-NUMBER-OF-CALLS. However, the information-piece (corresponding to the vote) of each member of a group, rather than being communicated directly to the group leader, flows to it along the edges of a minimum spanning tree. Then the leaders exchange information as in steps 3 and 4 of MINIMUM-NUMBER-OF-CALLS, and afterwards, the whole gossip flows back to all the individuals along the edges of the same four minimum spanning trees. We conjecture that finding a partition as above, for which the total cost of the telephone calls is minimum, is NP-complete.

The second possible structure for the pattern of minimum cost, is one that corresponds to the edges of a spanning tree, T, of the cost graph. The gossip pieces flow inwards, towards the coordinators, that are the neighbors on both sides of the costliest edge in T. After a coordinator receives the pieces of the gossip from all its neighbors in the tree, except from the other coordinator, it calls the other coordinator, and they exchange information. At this point, the coordinators, and only them, know the whole gossip, the total cost is equal to the cost of T, and the number of calls is $p-1$. Subsequently, the gossip flows to the rest of the participants along the edges of T, requiring $p-2$ additional calls, at a cost of:

(the cost of T) – (the cost of the edge between the coordinators).

Next we will make a few remarks about the second pattern of minimum cost mentioned above. First, note that the total cost of the pattern above is:

8.2.1 (twice the cost of T) – (the cost of the edge between the coordinators)

and this is the reason for choosing the coordinators as the endpoints of the costliest edge in T. Second, observe intuitively, that there are are cases in which the pattern above indeed achieves minimum communication cost. For example, it does so if there is a unique minimum spanning tree of the cost graph, and any telephone call between participants that are not neighbors in the tree, is prohibitively expensive. Third, the pattern above achieves minimum cost with $p-3$ calls, and this is higher than the minimum number of calls, that is $p-4$. This corroborates the argument made in section 3, that the communication pattern having a minimum number of messages (or calls) does not necessarily have a minimum cost. Fourth, we conjecture that in general, finding the spanning tree for which formula 8.2.1 is minimum, is NP-complete.

In conclusion, we feel that much remains to be done in order to prove the conjectures above, and then, to incorporate communication time considerations into the solutions. We believe that TREE-COMMIT will provide a handle on an approach for this incorporation.

ACKNOWLEDGEMENT

We wish to thank Alon Itai for very helpful discussions and insightful remarks. We also thank the referees for detailed and thorough reports, that helped us improve this paper significantly.

REFERENCES

[BHG] P. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems* Addison Wesley, 1987.

[BK] B. Baker and R. Shostak, "Gossips and Telephones," *Discr. Math.* 2, pp. 191-193, 1972.

[C] N. Cot, "Extensions of the Telephone Problem," *Proc. 7th SE Conf. on Combinatorics, Graph Theory and Computing* Utilitas Mathematica, Winnipeg, pp. 239-256, 1976.

[CP] S. Ceri and G. Pelagatti, *Distributed Database Principles and Systems*, McGraw-Hill, 1984.

[DS1] C. Dwork and D. Skeen, "The Inherent Cost of Nonblocking Commitment", *Prod. 2nd ACM Symp. on PODC*, pp. 1-11, 1983.

[DS2] C. Dwork and D. Skeen, "Patterns of Communication in Consensus Protocols", *Proc. 3rd ACM Symp. on PODC*, pp. 143-153, 1984.

[E] S. Even, *Graph Algorithms*, Computer Science Press, 1979.

[F] M. Fischer, *The Consensus Problem in Unreliable Distributed Systems (a brief survey)*, Technical Report YALEU/DCS-/RR-273, Yale University, June 1983.

[G] J.N. Gray, "Notes on Database Operating Systems," *Operating Systems: An Advanced Course*, Springer-Verlag, 1979.

[GJ] M.R. Garey and D.S. Johnson, *Computers and Intractability*, Freeman, 1979.

[H1] V. Hadzilacos, "On the Relationship between the Atomic Commitment and Consensus Problems," *Proc. of the Workshop on Fault-Tolerant Distributed Computing* Springer Verlag, 1986.

[H2] V. Hadzilacos, "A Knowledge Theoretic Analysis of Atomic Commitment Protocols," *Proc. 6th ACM Symp. on PODS*, pp. 129-134, 1987. A revised version has been submitted for publication.

[HHL] S. Hadetniemi, S. Hadetniemi, and A. Liestman, " A Survey of Gossiping and Broadcasting in Communication Networks," *Networks Vol. 18* pp. 319-349, 1988.

[I] A. Itai, Unpublished result, 1986.

[L] L. Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System," *CACM*, 21(7), pp. 558-565, 1978.

[LHJLSW] B. Liskov, M. Herlihy, P. Johnson, G. Leavens, R. Scheifler, and W. Weihl, "Preliminary Argus Reference Manual", Programming Methodology Group Memo 39, 1983.

[LS] B. Lampson and H. Sturgis, "Crash Recovery in a Distributed Database System," TR, Xerox PARC, 1976.

[MLO] C. Mohan, B. Lindsay, and R. Obermack, "Transaction Management in the R* Distributed Database Management System," *TODS*, 11(4), pp. 378-396, 1986.

[R] K.V.S. Ramarao, "On the Complexity of Commit Protocols," *Proc. 4th ACM Symp. on PODS*, pp. 235-244, 1985.

[Se] A. Segall, "Distributed Network Protocols," IEEE Trans. Inform. Theory, Vol. IT-29, No.1, pp. 23-35, Jan. 1983.

[Sk] D. Skeen, "Nonblocking Commit Protocols," *Proc. ACM SIGMOD*, pp. 133-142, 1981.

[Sp] A. Spector, "Modular Architectures for Distributed and Database Systems," *Proc 8th ACM Symp. on PODS*, pp. 217-224, 1989.