

# Characterizing Distributed Algorithms with Knowledge Theory: A Survey \*

Mark R. Lindsey

April 29, 2003

## 1 Introduction

A distributed system is a collection of computing processes that communicate according to a *protocol* by passing messages [1]. The individual autonomous components of the distributed system may have clocks, and these clocks may be synchronized with one another. The communication within the system may provide guarantees, such as guaranteed reliability and maximum bounds on transmission time. Further, components and communication links within the system may experience failures.

*Knowledge theory* provides a model and fundamental results to describe the distribution of ‘knowledge’ among autonomous, communicating agents [11]. In the context of computing automata, a process  $i$  is said to *know* a fact  $\varphi$  if  $\varphi$  is part of  $i$ ’s local state [10]. Further, we naturally discuss the capabilities of processes by ‘what they know’. Thus, knowledge theory provides a natural framework for distributed systems.

Communication in a distributed system can be modeled as a transformation in the knowledge state of processes in the system [11]. The messages that are passed between processes, together with the guarantees made about the system, determine the knowledge that each process can have. For example: two processes,  $i$  and  $j$ , communicate using message channels that never fail to deliver messages within 5 units time. Further, the processes  $i$

---

\*This work was done for the University of North Carolina at Chapel Hill course COMP 247, *Distributed and Concurrent Algorithms*, taught in the spring, 2003, by Jim Anderson.

and  $j$  never fail. Suppose that  $i$  sends a message  $m$  to  $j$  at time 3, and let the statement  $\varphi$  stand for  $j$  will receive  $m$  before time 9. It is the case that  $i$  knows  $\varphi$ , denoted  $K_i\varphi$ , because of the guarantees of the system, and its own action of sending the message. At time 9,  $j$  receives the message, thus  $K_j\rho$  where  $\rho$  means that  $m$  was sent to  $j$  before time 4. If we assume that  $i$  and  $j$  are the only two members of the distributed system, then  $K_j\tau$ , where  $\tau$  means  $i$  sent  $m$  to  $j$  before time 4. But  $i$  also knows that  $j$  will know this; i.e.,  $K_iv$  where  $v$  means *After time 9*,  $K_j\tau$ .

Things get more interesting when we consider the possibility of failures; for example,  $i$  cannot know that  $j$  received the message until  $i$  receives an acknowledgement of the message from  $j$ . If the message channel does not guarantee a bound on the delivery time, then  $j$  could never know when the message  $m$  was sent, unless it includes a timestamp. And if the message does include a timestamp, then  $j$ 's ability to interpret that timestamp depends on the precision of synchronization between  $i$ 's clock and  $j$ 's clock.

The purpose of this paper is to describe the issues that flow out of this type of reasoning by surveying the fundamental literature in knowledge theory. Section 2 describes the evolution of formal reasoning about knowledge. Section 3 introduces a hierarchy of knowledge levels which correspond to the power of communication channels and the requirements of certain problems. Section 4 describes several variants of one point in this hierarchy, common knowledge, and show how to model realistic situations. Section 5 shows how knowledge theory can be applied to describe realistic protocols, and prove properties of problems.

## 2 Evolution of models

The early formal work on knowledge reasoning is described by Halpern [10]. Epistemology, the study of knowledge, has been studied since the ancient Greeks, though formal analysis about knowledge dates only from the latter half of the twentieth century. It was widely studied by philosophers, though primarily from the perspective of a single agent.

The earliest systematic model for formal reasoning about knowledge is the *possible-worlds* model. In this model, an agent considers all of the possible states of affairs, i.e., the “possible worlds”. Every possible world is compatible with all of the facts that the agent knows; knowing a fact ex-

cludes all possible worlds where that fact does not hold. This model was historically the basis for significant work on the nature of a single agent knowing something.

In a widely-cited early article, Gettier argued about the substance of knowledge with respect to things that are true, but the agent does not know precisely why they are true [4]. Gettier claimed that when an agent is justified in believing a true fact, but that justification is actually faulty, the agent cannot truly be said to know that fact.

$K_i\varphi$  indicates that the agent  $i$  knows a fact,  $\varphi$ . In the possible-worlds model,  $\neg K_i\varphi$  is interpreted as possibility; i.e., agent  $i$  considers  $\varphi$  to be possible when it does not know that it holds.

## 2.1 Bounded agents

The possible-worlds model suggests *logical omniscience* – agents know all of the consequences of their knowledge, and all tautologies; e.g., when  $K_i\varphi$  and  $K_i(\varphi \Rightarrow \rho)$ , then  $K_i\rho$ . However, in many useful systems, the agents have bounded resources. For example, if agents are computationally unbounded, then they can factor large numbers and decrypt messages as a matter of course.

Several approaches are taken to address this; in one approach, an agents knowledge is strictly confined to a set of formulas, but not necessarily their implications. In another, an agent’s knowledge is not described by a set of formulas, but rather by all of the possible worlds that the agent knows. Analysis is done in terms of the set of possible worlds for an agent rather than by implications from its set of formulas.

In yet another approach, truth in all possible worlds is not a sufficient condition for knowledge [10]. This scheme suggests a *logic of general awareness*, wherein an agent must be aware of a fact (e.g., by having computed it) in addition to the fact’s deducability. For example, the plain-text of an encrypted message may be deducable, but the agent may not be aware of it unless it possesses the key and is aware of it.

## 2.2 Multiple agents

Since each such *modal* operator  $K_i$  is a fact, they are combined to express knowledge about members of a group. For example,  $K_jK_i\varphi$  holds when

agent  $i$  knows fact  $\varphi$ , and agent  $j$  knows that agent  $i$  knows  $\varphi$ .

The possible-worlds model is thus extended to provide a suitable framework for reasoning about multi-agent systems. This model also provides a way to talk about knowledge within a group.  $E_G\varphi$  holds when every agent in a set  $G$  knows a fact  $\varphi$ .

For *agreement* to occur in a distributed system, every processor must be certain of the decision that will have been made by every other processor when the protocol eventually terminates. Further, from any one processor's perspective, every other processor needs to be sure that it knows that they are all going to make that decision. A special case of group knowledge called *common knowledge* describes this. A fact  $\varphi$  is common knowledge when everyone in  $G$  knows that everyone in  $G$  knows that . . . everyone in  $G$  knows  $\varphi$  [10]. However, it turns out that *common knowledge* is not as important as it may first appear. Perhaps the chief contribution of knowledge theory to general analysis is the a careful analysis of states of knowledge that are similar to common knowledge, and how they relate to the power of communication channels.

### 2.3 Reasoning about protocols

Instead of reasoning only about the base facts and implications that an agent knows, it is useful to think about the protocol by which an agent acquires knowledge. The *coordinated attack* problem, introduced by Gray [5], motivates these issues. This is the problem faced by two generals camped on hilltops overlooking a valley. The generals wish to attack simultaneously, and need to coordinate their attack, for neither general will attack unless he is certain the other will attack at the same time. They communicate by sending a messenger from one side to the other, but this messenger may get lost or be captured.

Suppose that General  $A$  sends a message, "Attack at noon", to General  $B$ . When General  $B$  receives the message, he sends the messenger back with an acknowledgement, because he knows that General  $A$  will not attack unless he knows that General  $B$  will attack simultaneously. But when General  $A$  receives the acknowledgement, he must acknowledge, for he knows that General  $B$  will not attack unless he is sure that General  $A$  received that acknowledgement. Even if no message is ever lost, the possibility that it

could be lost prevents either general from being certain that the other general will attack; each round of acknowledgements gets the generals ‘closer’, but they never reach the required state of knowledge.

Halpern and Moses [11] show that in any system of unbounded message delays, it is impossible to achieve the required state of knowledge to attack. It turns out that common knowledge is required to solve the coordinated attack problem, but that common knowledge cannot be achieved in realistic message-passing systems such as the one provided by the vulnerable messenger.

### 3 Hierarchy of knowledge states

Common knowledge is a very strong sort of knowledge among a group of agents. Halpern and Moses [11] detail a hierarchy of states of knowledge that expands on the classical approach described above, and fills in the space below common knowledge. One extension is *Distributed Knowledge*, denoted as  $D_G\varphi$  when the group  $G$  has distributed knowledge of a fact  $\varphi$ . Distributed knowledge is the knowledge that a single agent would have if it knew everything that every member of  $G$  knows. Slightly stronger, we say that “someone knows  $\varphi$ ” as  $S_G\varphi$  when one agent knows the fact  $\varphi$ .

When everyone in a group knows  $\varphi$ , this is  $E_G\varphi$ . It is useful to reason about what everyone knows about everyone else, so this is extended to include  $E^k$ -knowledge;  $\varphi$  is  $E^k$ -knowledge in group  $G$  when “everyone in  $G$  knows that everyone in  $G$  knows that . . . that everyone in  $G$  knows that  $\varphi$  holds”, where “everyone in  $G$  knows that” appears  $k$  times in the sentence [11].

These types of knowledge among a set of agents form a hierarchy:

$$C\varphi \supset \dots E^{k+1}\varphi \supset \dots E\varphi \supset S\varphi \supset D\varphi$$

In message-passing systems, the state of knowledge can move up the hierarchy toward common knowledge by employing a protocol. In shared-memory systems, this all processes can access all of each others’ state, this hierarchy collapses [11].

The muddy-children problem highlights several points in this hierarchy. Suppose that a group of children are playing, and some of them get mud on their foreheads. A child cannot tell the he has mud on his own forehead,

although he can see the mud on the foreheads of any other muddy children. The father announces that “at least one of you has a dirty forehead”, and then he asks “do any of you know that you have mud on your forehead?”

The fact  $m$ , “at least one child has a muddy forehead”, has become common knowledge, because every child hears the father, and knows that every other child has heard the father, and knows that every other child knows that every other child heard the father, and so on. Suppose that there is only one muddy child, and that he can see all of the other children. He knows that none of them have muddy foreheads, so he answers ‘yes’ to this question.

But if there were two dirty children, then both would legitimately answer this first question ‘no’, because each must assume that  $m$  applies to the other dirty child. But when dirty child  $i$  hears the other dirty child,  $j$ , answer ‘no’ to the first question,  $i$  will realize that  $j$  must have answered ‘no’ because  $j$  saw another dirty forehead; since  $i$  can see only one other dirty forehead, he correctly deduces that his own forehead is dirty, and thus answers ‘yes’ to the second question. Muddy child  $j$  reasons similarly, and also answers ‘yes’ to the second question. This logic is used to prove by induction on  $k$ , the number of dirty children, that all children will answer ‘no’ for the first  $k - 1$  queries of the father, and all of the the dirty children will answer ‘yes’ on the  $k$ ’th query [11]. Clearly, these are insightful children – far more insightful than this author would have been as a child. This issue hilights the need to think about agents with limited capabilities, as is described in Section 2.1.

When  $k > 1$ , the father’s statement did not introduce any new information, because every child could see a muddy forehead. But every child did not know that every child could see a dirty forehead, because each child is not aware of the muddiness (or cleanliness) of his own forehead, and thus cannot know what others know about his own forehead. Thus, the father’s statement of  $m$  transformed the state of knowledge from  $E^{k-1}m$  to  $E^k m$ .

In the coordinated-attack problem, each round of acknowledgements increases the  $k$  for which  $E^k m$  holds, where  $m$  is the fact that General  $A$  sent the message “attack at noon”. As discussed above, there is no  $k$  for which both generals can be certain that it is safe to attack. Halpern and Moses show that the generals actually need common knowledge, and that all processes’ state must *simultaneously* change to reflect that [10]. In general, when attaining common knowledge requires guaranteed communication, or

some common source of information. For example, if a global clock is available, then the clock's reading is common knowledge.

Further, they show that common knowledge can only be achieved in systems with *perfect* synchrony, because all processes must know when all other processes will be aware that some fact is common knowledge. Before that time, it is not common knowledge, so no system can be the 'first' to take the fact as common knowledge. Real systems do not provide such temporal perfection.

## 4 Variants on common knowledge

We have seen that common knowledge is impractical; but this leaves us without a model of general cooperative knowledge. In this section, we describe several variants on common knowledge that reflect the realities of communication guarantees and synchrony.

### 4.1 $\epsilon$ -common knowledge

For a fact to become common knowledge in a group  $G$ , it must be simultaneously known to become common knowledge. In many applications, however, it is acceptable for systems to come to agree on a fact over a period of time, so long as that period is known to end. After that period has ended, then every agent has certainty about the knowledge of every other agent.

This type of knowledge is modeled as  $\epsilon$ -common knowledge. Instead of insisting that every agent know a fact at precisely the same time, we can accept that every agent will know a fact within  $\epsilon$  time units. Whereas common knowledge describes simultaneous actions,  $\epsilon$ -common knowledge describes actions that are guaranteed to occur within  $\epsilon$  time units of one another [10].

### 4.2 Timestamped common knowledge

*Timestamped* common knowledge is another useful weakening of common knowledge proposed by Halpern and Moses [10]. It models the case where clocks within the distributed system are synchronized to some relative precision. We call *Timestamped* knowledge to be statements of the form "When

agent  $i$ 's clock reads  $T$ , agent  $i$  knows  $\varphi$ ," written as  $K_i^T \varphi$  and  $@_i^T \varphi$  alternately in the literature [10, 13]. Timestamped common knowledge is defined in the expected way, and can be achieved in many practical systems; e.g., the *atomic broadcast protocol* achieves timestamped common knowledge [3].

Timestamped common knowledge is stronger than  $\epsilon$ -common knowledge. It is possible when clocks in the distributed system are synchronized to some precision, and communication channels never fail to deliver messages within a guaranteed bound. By contrast,  $\epsilon$ -common knowledge requires only that messages will always be successfully delivered within a bounded time,  $\epsilon$ . Halpern and Moses prove that links that do not provide delay guarantees are not useful for establishing  $\epsilon$ -common knowledge.

### 4.3 Eventual common knowledge

$\epsilon$ -common knowledge and Timestamped common knowledge both put stringent demands on the underlying communication channels; many useful systems never achieve either. Therefore, Halpern and Moses also define *eventual* common knowledge, written as  $\diamond$ -common knowledge. A fact is said to be  $\diamond$ -common knowledge if it will *eventually* be known to all processes. This case models Byzantine Agreement where individual processors do not fail, because all processors ultimately decide on the same value [10].

### 4.4 Continual common knowledge

When we allow for crash and omission failures, the Byzantine Agreement problem can be partitioned into two versions: simultaneous Byzantine Agreement (SBA), where all nonfaulty processors must decide simultaneously; and eventual Byzantine Agreement (EBA), where all nonfaulty processors must eventually decide on the same value. Whereas the simultaneity of SBA requires common knowledge, Halpern, Moses and Waarts show that none of the variants of common knowledge describe EBA [8].

The apparent choice would be eventual common knowledge, but Halpern *et al.* show that this does not work [8]. In the presence of sending omissions, eventual common knowledge does not guarantee consistency for a nontrivial EBA protocol. They define *continual* common knowledge, which relates to a set  $S$  of nonfaulty processors. For different executions of the protocol,  $S$  may contain different processors. If at every state in the execution of

the protocol, every processor in  $S$  knows that every processor in  $S$  at that point in execution knows that ...  $\varphi$  holds, then  $\varphi$  is continual common knowledge, written as  $C_S^\square \varphi$ . The complicating factor is the nonrigidity of the set of nonfaulty processors,  $S$ . Halpern *et al.* show that continual common knowledge is actually a strengthening of common knowledge [8].

Collectively, these works show that the hierarchy of states of knowledge is expanded thus:

$$C^\square \varphi \supset C\varphi \supset C^T \varphi \supset C^\epsilon \varphi \supset C^\diamond \varphi \supset \dots E^{k+1} \varphi \supset \dots E\varphi \supset S\varphi \supset D\varphi$$

## 5 Applying knowledge theory

Many problems can be characterized without appealing to variants of common knowledge. This section describes some of that analysis.

### 5.1 Atomic Commitment

The *atomic commitment problem* [6] occurs in distributed database applications. Each processor in the system must decide irrevocably whether to COMMIT or to ABORT. Each processor votes either *yes* or *no* about the decision to commit. Only if all processors are known to vote *yes* can any processor decide to COMMIT, and every processor must make the same decision. For nontriviality, we require that when every processor votes *yes* and there are no failures every processor will COMMIT.

Hadzilacos discusses protocols for solving this problem in a knowledge-theoretic framework [7]. Two-phase commit (2PC) [5] is the most popular protocol for solving atomic commit. Hadzilacos shows that each processor in 2PC decides COMMIT as soon as it has knowledge that every other processor has voted *yes*. This is the minimum knowledge necessary, therefore 2PC is shown to exchange the least number of messages as is possible [7].

This result does not say anything about global level of knowledge after the votes have been cast; indeed, in 2PC, individual processors may fail after the votes are collected, and no guarantees are made as to precisely *when* every processor will come to know that every processor has voted *yes*.

Hadzilacos also shows more information is useful for constraining this state of knowledge. Note, as in the case of the failed processor in 2PC, atomic commitment is solved if every non-faulty process will *eventually* know that

everyone voted *yes*. Hadzilacos shows that in 3PC, an atomic commitment protocol designed to avoid blocking, each processor may COMMIT when it has enough knowledge to do so and avoid blocking. Therefore, 3PC exchanges the minimum number of messages necessary to COMMIT and avoid blocking.

## 5.2 Secrecy

Secrecy for purposes of security in multi-agent systems can be described in terms of the facts shared between agents, and the knowledge held by each of these agents. A *high-level agent* is one that knows facts that cannot be available to a *low-level agent*. Models for secrecy in multi-agent systems are used for specifying and checking them. Halpern and O'Neill claim that knowledge theory provides a model that matches intuition particularly well [9].

The strongest type of secrecy is *total secrecy*; this means that a low-level agent cannot determine anything about the state of a high-level agent. In a knowledge-theoretic sense, this means that a low-level agent must not be able to determine that a high-level agent's state is incompatible with its own. Yet total secrecy is too strong for many applications; for example, in a synchronous system, total secrecy excludes a low-level agent from knowing a high-level agent's clock reading. An 'information function' defines a view into non-secret part of the high-level agent's state. Halpern and O'Neill show that this can model the allowability that we intuitively expect.

It is important to be able to prove secrecy in these systems. Therefore, they also show that the following are equivalent:

1. Agent  $i$  does not know any formulas that depend only on the state of  $j$ .
2. Agent  $j$ 's state is kept secret from agent  $i$ .

*Distributed knowledge* is also shown to be a useful way to describe secrets within a group. Previous models of secrecy tend to focus on pair-wise secrets, but this is awkward for reasoning about system in which the agents could 'put their heads together' to determine some fact. Distributed knowledge, described in section 3, does just this.

Knowledge theory also provides a ready model for describing probabilistic secrecy. We do not want a low-level agent to determine the probability of any component of a high-level agent's state. To do this, it is shown that

a low-level agent must never learn anything about the probability of a high-level agent being in a given state [9]. Ultimately, total probabilistic secrecy between two agents  $i$  and  $j$  requires that the probability of each of  $i$ 's local states be independent of the probability of each of  $j$ 's local states.

### 5.3 Distributed Problem Solving

Wooldridge proposes that knowledge theory provides a useful alternate perspective to distributed problem solving, such as theorem proving [14]. Normally, the paradigm of *distributed search* of a problem's solution space is used to define the computation. Instead, he proposes that distributed problem-solving agents have a knowledge base, and that facts are sent between agents which construct that knowledge base.

Wooldridge proves that any problem that has a solution can be solved by agents that send their complete state in every message. However, because this is impractical for real systems, each agent program can only decide which is the best message to send using its own knowledge set. Determining the most-useful fact to assert is the role of each agent.

Each message sent is an assertion of truth by the sender. When an agent receives a message asserting a fact, it removes from its possibility set any facts that are incompatible with the newly-learned fact. If we assume that every agent in the problem-solving system starts with the same information, the knowledge set is the conjunct of all messages received thus far. Wooldridge shows that the *distributed* knowledge of such a system is non-diminishing; i.e., facts are not lost.

### 5.4 Routing

Packet-switched networks are those in which messages can be exchanged only between neighbors, called *routers*. Messages traverse the network by being forwarded from one neighbor to the next. At each step, the router holding the message must determine the best step that it should take next. We can assign each edge in the network a cost, and charge each message the cumulative cost of all of the edges that it traverses. The lowest-cost path between any two neighbors is called the 'shortest path', because cost is often a nondecreasing function of distance, and only links incur cost.

To achieve this shortest path for every message, every node must forward each message  $m$  along the path that will incur the lowest cost to  $m$ . The Bellman-Ford *distance-vector* algorithm is designed for computing just this [2]. Malkin specifies an Internet-standard routing protocol for such networks, called the Routing Information Protocol (RIP) [12].

Each node computes the shortest paths from itself to each node in the system, sending messages only to its neighbors. The only information sent by node  $i$  to its neighbors is the list of estimated distances from  $i$  node to each other node. Eventually, each node knows the shortest path to each other node.

Let us observe some properties of knowledge within the network. RIP specifies that messages are sent according to a fixed period, so let us assume that all devices send messages at the same time. At time 0, no messages have been sent; at time 1, one message has been sent by every router and received by its neighbors; at time  $t$ ,  $t$  messages have been sent.

Let  $\delta_{i,j}$  be the cost of the shortest between  $i$  and  $j$ . Because edges are undirected,  $\delta_{i,j} = \delta_{j,i}$ .

At time 0, each router knows only the distance to each of its own neighbors. The shortest path between any two neighbors is distributed knowledge at this point, because an agent with access to all of this internal state could compute the shortest path between every two nodes; i.e.,  $D\delta_{i,j}$  for every pair of routers  $i$  and  $j$ .

A router must wait until  $d$  to get information message rounds that must elapse to get information about an edge that is  $d$  routers away. At time 1, every router knows of a distance to each every router that is two network edges away. At time  $k + 1$ , every router knows the distance to every router that is  $k$  or fewer hops away. If  $d$  is the maximum number of hops between any two nodes (i.e., the diameter of the network) then at time  $d - 1$ , the shortest path to every other node will be known to each node. That is, at time  $d - 1$ ,  $K_i\delta_{i,j}$  for all routers  $i$  and  $j$ .

Interestingly, at time  $d - 2$ , the shortest path between  $i$  and all other routers is known to all of  $i$ 's adjacent neighbors that are along that path; i.e.,  $K_h\delta_{i,j}$  for all  $i$  and  $j$ , where  $h$  is adjacent to  $i$ . This is true because  $h$  has heard about all of the messages that are  $d - 2$  hops away from itself, and also knows about its own link to  $i$ . This shows that  $S\delta_{i,j}$ .

RIP is a *full-information* protocol, meaning that each message includes

all of the sender’s local (routing-information) state. Because of this, details about the complete network are eventually distributed to every router in the network. Ultimately,  $E\delta_{i,j}$ , but  $i$  may be the last to know.

So far, we have seen a simple case where a protocol transforms distributed knowledge to someone-knows knowledge, and eventually to everyone-knows knowledge, assuming that no failures occur. In fact, under these assumptions, we actually have timestamped common knowledge – i.e.,  $C^{d-1}\delta_{i,j}$ . However, this is not realistic for deployed RIP, because RIP does not use reliable communication channels and routers crash.

## 6 Conclusion

We have discussed the historical underpinnings of knowledge theory as described by the multiple-worlds model, and the direction of the early work in epistemic reasoning. We have shown how it was extended to describe multi-agent systems, and specifically distributed systems. We described *common knowledge*, explained why it is impractical, and discussed several variants of common knowledge that can describe realistic systems: eventual common knowledge, timestamped common knowledge, and  $\epsilon$ -common knowledge. We cited the necessity of a strengthening of common knowledge to a form called continual common knowledge, which can represent the knowledge present in a system with a nonrigid set of faulty processors. We then showed how knowledge theory has been applied to model and analyze eventual byzantine agreement, atomic commitment, secrecy, and routing.

As Halpern, the most widely-cited researcher in this field, says in [10] –

[I]n this model, knowledge is an “external” notion. We don’t imagine a process scratching its head wondering whether or not it knows a certain fact  $\varphi$ . Rather, a programmer reasoning about a particular protocol would say, from the outside, that the process knows  $\varphi$  because in all global states consistent with its current state ...  $\varphi$  is true.

As demonstrated by the examples in this survey, knowledge theory can provide an interesting and sometimes useful tool for discussing protocols in distributed systems. The strong results concerning common knowledge appear to be the greatest value, because they provide ready boundaries to

prevent hyperbolic expectations of any realistic protocol.

## References

- [1] Free online dictionary of computing. <http://www.foldoc.org/>, April 2003.
- [2] CORMEN, T. E., LEISERSON, C. E., RIVEST, R. L., AND STEIN, C. *Introduction to Algorithms*, 2 ed. McGraw Hill, 2001.
- [3] CRISTIAN, F., AGHALI, H., STRONG, R., AND DOLEV, D. Atomic broadcast: From simple message diffusion to byzantine agreement. In *Proc. 15th Int. Symp. on Fault-Tolerant Computing (FTCS-15)* (Ann Arbor, MI, USA, 1985), IEEE Computer Society Press, pp. 200–206.
- [4] GETTIER, E. L. Is justified true belief knowledge? In *Analysis*, vol. 23. 1963, pp. 121–123.
- [5] GRAY, J. N. Notes on database operating systems. In *Operating Systems: An Advanced Course* (1978), vol. 60, Springer-Verlag.
- [6] HADZILACOS, V. On the relationship between the atomic commitment and consensus problems. pp. 201–208.
- [7] HADZILACOS, V. A knowledge-theoretic analysis of atomic commitment protocols. In *Proceedings of the sixth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems* (1987), ACM Press, pp. 129–134.
- [8] HALPERN, J., MOSES, Y., AND WAARTS, O. A characterization of eventual byzantine agreement. In *Proceedings of the 9th Annual ACM Symposium on Principles of Distributed Computing* (Québec City, Québec, Canada, 1990), C. Dwork, Ed., ACM Press, pp. 333–346.
- [9] HALPERN, J., AND O’NEILL, K. Secrecy in multi-agent systems. In *15th IEEE Computer Security Foundations Workshop* (2002), pp. 32–46.
- [10] HALPERN, J. Y. Reasoning about knowledge: A survey. In *Handbook of Logic in Artificial Intelligence and Logic Programming, Volume 4*:

*Epistemic and Temporal Reasoning*, D. M. Gabbay, C. J. Hogger, and J. A. Robinson, Eds. Clarendon Press, Oxford, 1995, pp. 1–34.

- [11] HALPERN, J. Y., AND MOSES, Y. Knowledge and common knowledge in a distributed environment. In *Journal of the ACM* (1990), no. 37:3, pp. 549–587.
- [12] MALKIN, G. RIP version 2. Internet Engineering Task Force RFC 2453, 1998.
- [13] NEIGER, G., AND TOUEG, S. Simulating synchronized clocks and common knowledge in distributed systems. *Journal of the ACM (JACM)* 40, 2 (1993), 334–367.
- [14] WOOLDRIDGE, M. A knowledge-theoretic approach to distributed problem solving. In *Proceedings of the Thirteenth European Conference on Artificial Intelligence* (1998), John Wiley & Sons.