Receiving Messages in Their Correct Order: Analyzing Broadcast Protocols in Dynamic Epistemic Logics

Spandan Das[1] and Sujata Ghosh[2]
[1]*Indian Statistical Institute, Kolkata, India*
[2]*Indian Statistical Institute, Chennai, India*

Keywords: Broadcast Protocols, Correctness of Protocols, Dynamic Epistemic Logic, Algorithm, Complexity.

Abstract: In this paper we analyse distributed protocols in a logical framework. In particular, we provide a dynamic epistemic logic analysis of certain broadcast protocols, viz. Birman-Schiper-Stephenson protocol and Lamport's mutual exclusion protocol. In the process, we provide correctness proofs of these protocols via knowledge-based reasoning. We also provide a detailed algorithmic analysis of the logical modeling of these protocols.

1 INTRODUCTION

Distributed systems occur ubiquitously in today's world of computing and as such, a great deal of effort has been provided towards improving our understanding of these systems. A major challenge is to deal with such systems efficiently and effectively. The distributed nature of control and information in these systems lead us to consider the tasks of the systems in terms of the global behavior of the system, even though the actions that an individual processor performs can depend only on its local information.

Since the design of a distributed system involves the behavior and interaction between individual processors in the system, designers frequently find it useful to reason intuitively about processors' states of knowledge at various points in the execution of a system. Over the years, states of knowledge of groups of processors have become useful concepts for the design and analysis of distributed protocols. Epistemic logic and its variants provide us with a proven methodology to analyze these knowledge states of processors and their interactions. A seminal work in this area is (Halpern and Zuck, 1992) which provides a uniform knowledge-based framework to deal with the *sequence transmission* problem in both synchronous and asynchronous settings.

Over the years, such frameworks have been used to model various authentication protocols (for a survey, see (Ahmadi et al., 2019)), and epistemic and temporal epistemic logics played a major role in the analyses of such protocols. Dynamic epistemic logics have also been used to model such protocols, but as described in (Dechesne and Wang, 2010), there are certain limitations to such approaches in terms of handling equivalence of messages. However, these logics are quite adept in handling higher-order uncertainties and partial information in protocols (e.g., see (van Ditmarsch et al., 2014)).

In dynamic epistemic logics, the actions/events bring in changes in the epistemic states of the agents, but these changes happen immediately. Consequently, these logics are generally not appropriate for modelling communication of information where there is a time gap between the send-event and the receive-event, a natural occurrence in asynchronous systems. Recently, some variants of dynamic epistemic logics have been developed to deal with communication in asynchronous systems, e.g., see (Knight et al., 2019; Balbiani et al., 2020), both of which extends the usual dynamic epistemic language so as to express the time-lag in communication.

This work uses a simple variant of dynamic epistemic logic, where the action updates are generally modeled as soft updates (e.g., see (Baltag and Smets, 2008)), i.e., only the agent relations get updated, and there is no change in the set of possible worlds. The logic is used to model communication of information in the form of message passing in asynchronous systems. In such systems, communication speed might differ from channel to channel leading to problems in the arrangement of messages from different processes in their correct temporal order. In case the messages are not executed in their correct temporal order, the system might act erroneously. The goal of distributed protocols such as BSS and Lamport's mutual exclu-

Das, S. and Ghosh, S.
Receiving Messages in Their Correct Order: Analyzing Broadcast Protocols in Dynamic Epistemic Logics.
DOI: 10.5220/0010253608510858
In *Proceedings of the 13th International Conference on Agents and Artificial Intelligence (ICAART 2021) - Volume 2*, pages 851-858
ISBN: 978-989-758-484-8

sion is to choose an execution order so that such inconsistencies do not occur.

The focus is on the modelling of broadcast protocols (Singhal and Shivaratri, 1994) in the framework mentioned above together with the correctness proofs of such protocols in terms of knowledge-based reasoning provided by the framework. In addition, implementation details and complexity analyses are also provided. We basically model the effect of following these protocols on certain message passing systems. The novelty of this work is two-fold. On one hand, broadcast protocols are natural candidates for analyses in terms of dynamic epistemic logics, but to the best of the knowledge of the author(s), such logical analyses of these protocols have not been done before. On the other hand, a simple variant of dynamic epistemic logic has been used to deal with these asynchronous systems, and that was made possible due to the usage of the different possible orderings of the messages as possible worlds of the underlying Kripke model used to describe such systems.

We note here that these broadcast protocols have been studied quite extensively over the years and correctness proofs are also there in the literature. Our goal here is to showcase the use of dynamic epistemic logic in handling the knowledge-based reasoning of such systems. We believe that this methodology can be adopted for modelling the underlying reasoning processes in different distributed protocols which would provide a better understanding of such protocols applied on asynchronous systems.

Before proceeding further, we list the assumptions on the asynchronous systems that are made for our analyses. These assumptions, without being necessary, enhance the exposition: Each process has a local clock with respect to which it can arrange local *send* and *receive* events in their correct temporal order; inter-process communication channels are First In First Out (FIFO) in nature, that is, two messages from the same process are received by other processes in their order of generation; and, the set of messages generated by all the processes is finite. Additionally, each process has the same initial knowledge about the set of all messages generated in the system, and the actual set of messages generated in the system is a subset of the set of messages known to each of the processes (for a detailed explanation see (Das and Ghosh, 2020)).

2 BROADCAST PROTOCOLS

In asynchronous distributed systems (Singhal and Shivaratri, 1994), when the communication of information is taken care of by message-passing, more often than not, the messages cannot be ordered according to their time of generation, the main reason being the fact that the global clock is unavailable. However, there is a reasonable way in which one can say that a message m_1 is generated before another message m_2. If m_1 is received by the sender of m_2 before m_2 is sent then evidently m_1 is generated before m_2. Such an ordering, defined by Lamport, is given as follows:

Definition 1. (Causal ordering). Let a and b be two events. We say a causally precedes b (notation $a \rightarrow b$) if one of the following three cases happen:

1. a and b are events on the same process and a chronologically precedes b by local clock of the process.
2. a is the send event and b is the receive event of the same message.
3. There exist an event c such that $a \rightarrow c$ and $c \rightarrow b$.

If none of these three cases happen then we say a and b are concurrent.

Various protocols have been developed for such distributed systems so that the causal ordering of messages could be enforced. For example, in Birman-Schiper-Stephenson (BSS) protocol, the processes always execute messages in the correct causal order, whereas, in Lamport's mutual exclusion (LME) protocol critical section requests are granted in their correct causal order. Before moving any further, let us describe those protocols. The BSS protocol helps the processes to execute the messages received in correct causal order. If two messages are concurrent, they are sorted in *first come first serve* basis.

Definition 2. BSS Protocol is defined as follows:

1. When a process broadcasts a message it also broadcasts how many messages it has received from each of the other processes.
2. When a process receives a message from another process it checks whether it has received all previous messages from that same process and also whether it knows about at least as many messages in the system as the sender process does. If both checks are true, the recipient process executes the message, otherwise it puts the message on hold and waits.

The LME protocol helps the processes to decide the order of accessing the critical section (a set of objects which can be modified by at most one process at a time) in case of multiple requests coming from different processes. If one request causally precedes another, the former process is granted access before the latter. Additionally, the protocol assumes that each process has a unique process ID from a totally ordered

set and in case of concurrent requests, it breaks tie using process ID. This means that the protocol gives first access of the critical section to the process with the smallest ID.

Definition 3. LME protocol is defined as follows in terms of the movements around the critical section:

- Requesting the critical section:
  1. When a process wants to access the critical section, it generates a *request* and a timestamp for that *request*. The process keeps this *request* in its own *request* queue sorted according to timestamps and then broadcasts it to all other processes along with the timestamp.
  2. When a process receives a timestamped *request*, it keeps the *request* in its own *request* queue sorted according to timestamps. Then it generates a timestamped *reply* corresponding to the *request* and sends the *reply* to the sender of the *request*.
- Accessing the critical section: When a process observes
  1. its *request* is at the top of its own *request* queue,
  2. it has received at least one message (*request* or *reply* or *release*) with higher timestamp than its *request* from all other processes,

  the process enters the critical section.
- Exiting the critical section:
  1. When a process exits the critical section, it generates a timestamped *release* message and broadcasts it to all other processes. It deletes its own *request* from its *request* queue.
  2. Upon receiving a *release* message, a process removes the *request* of the sender process from its own *request* queue.

In the remainder of this paper we will model these two protocols in a variant of dynamic epistemic logic, so as to exemplify a novel way of formal modeling such protocols towards proving correctness of the protocols using knowledge-based reasoning.

3 MODELLING PROTOCOLS IN DYNAMIC EPISTEMIC LOGIC

Let us first provide a brief introduction to a relevant variant of dynamic epistemic logic before moving on to the modelling of the protocols. For a detailed exposition, see (van Ditmarsch et al., 2007).

Syntax. Given a finite set of propositions $\mathcal{P}$, a finite set of agents $\mathcal{A}$ and a finite set of actions Act, the language $\mathcal{L}$ is defined as follows:

$$\phi := p \mid \neg\phi \mid \phi \wedge \phi \mid K_a\phi \mid A\phi,$$

where $p \in \mathcal{P}$, $a \in \mathcal{A}$, $A \in \mathsf{Act}$. The formulas $\phi \vee \psi$, $\phi \rightarrow \psi$ are defined as usual. A formula of the form $K_a\phi$ reads as "ϕ is known to agent a" and $A\phi$ reads as "ϕ holds after some action A". The actions are generally considered to be protocol-dependent and changes the model suitably. We use $L_a\phi$ to denote the formula $\neg K_a \neg\phi$.

Semantics. A model M for this language is a tuple (W,R,V) where W denotes a non-empty set of worlds, R denotes a set of agent-relations and V denotes a valuation function. Each agent defines a relation $R_a \in R$ such that $R_a \subseteq W \times W$. In essence, R_a models knowledge of an agent a which can be modified by certain events or actions. A valuation function $V : \mathcal{P} \rightarrow 2^W$ associates propositions to sets of worlds. An action A can modify an agent's knowledge, which we will explain in details while modelling the protocols. The effect of an action A on a model M is given by the model M_A. The truth definition of a formula at a world $w \in W$ in the model M is as follows:

$M,w \models p$ iff $w \in V(p)$,
$M,w \models \neg\phi$ iff $M,w \not\models \phi$,
$M,w \models \phi \wedge \psi$ iff $M,w \models \phi$ and $M,w \models \psi$,
$M,w \models K_a\phi$ iff for all $\tilde{w} \in W$ with $wR_a\tilde{w}$, $M,\tilde{w} \models \phi$
$M,w \models A\phi$ iff $M_A,w \models \phi$.
$M \models \phi$ means that for all $w \in W$, $M,w \models \phi$.

3.1 BSS Protocol

Before we define the language for analyzing BSS protocol, let us first note some important notations and definitions. Let $\mathcal{A}$ denote the set of agents/processes in the distributed system, and let $\mathcal{M}$ denote the set of messages whose causal ordering we are interested in. Each message is denoted by x_i, which can be read as the i^{th} message from agent x.

Definition 4. (Permutation). We say π is a permutation on a finite set S iff $\pi : S \rightarrow \{1,2,\ldots,|S|\}$ is a bijection.

Definition 5. (Allowed permutation). A permutation π on $\mathcal{M}$ is an **allowed permutation** iff $\pi(x_i) < \pi(x_j)$ whenever $i < j$. Here, $\pi(x_i)$ ($\pi(x_j)$) denotes the position of x_i (x_j) in the permutation π.

Definition 6. (MSN). For an agent x, **MSN** (message to send next) of x (denoted $MSN(x)$) is $k \in \mathbb{N}$ if x has sent $x_{k-1} \in \mathcal{M}$ but not sent $x_k \in \mathcal{M}$. However if x has sent all its messages in $\mathcal{M}$ then $MSN(x)$ is ∞.

3.1.1 Language

Now we are ready to define the language $\mathcal{L}$ that can describe the content of the exchanged messages. Let

$\mathcal{P}$ be the set of propositions. We have,

$$\mathcal{P} = \{P^{x,i} \mid x \in \mathcal{A};\ x_i \in \mathcal{M}\} \cup \{Q^{x,j}_{a,i} \mid a,x \in \mathcal{A};\ a_i, x_j \in \mathcal{M};\ a_i \neq x_j\}.$$

Here $P^{x,i}$ reads "agent x has spoken i times" and $Q^{x,j}_{a,i}$ reads "agent a in its i^{th} message notifies that j^{th} message from agent x has reached it". The formulas are given by,

$$\phi := \top \mid p \mid \neg\phi \mid \phi \wedge \phi \mid K_a\phi \mid A^a_\phi\phi.$$

Here $K_a\phi$ reads "agent a knows ϕ". We include $A^a_\phi\psi$ in the language only for Boolean formulas ϕ (no restriction on ψ) and not for more complex formulas since they are not needed for modeling the BSS protocol. For a Boolean formula ϕ and a general formula ψ in the language, $A^a_\phi\psi$ reads "ψ is true after ϕ is received by a". Let us now have the following definitions for the Boolean formulas in the language:

Definition 7. (Sender of a formula). The **sender** of a Boolean formula ϕ is defined recursively,

- $sender(\phi) = x$ if $\phi = P^{x,i}$.
- $sender(\phi) = a$ if $\phi = Q^{x,j}_{a,i}$.
- $sender(\neg\phi) = sender(\phi)$.
- $sender(\phi \wedge \psi) = sender(\phi)$ if $sender(\phi) = sender(\psi)$ and undefined otherwise.

Definition 8. (Index of a formula). The **index** of a Boolean formula ϕ is defined recursively,

- $index(\phi) = i$ if $\phi = P^{x,i}$.
- $index(\phi) = i$ if $\phi = Q^{x,j}_{a,i}$.
- $index(\neg\phi) = index(\phi)$.
- $index(\phi \wedge \psi) = index(\phi)$ if $index(\phi) = index(\psi)$ and undefined otherwise.

3.1.2 Semantics

We define the model M as the tuple (W,R,V) where, $W = \{\pi \mid \pi$ is an allowed permutation on the underlying message set $\}$,
$R = \{R_a \mid a \in \mathcal{A}; R_a$ is a binary relation on $W\}$,
$V : \mathcal{P} \to 2^W$ is given as follows,

$V(P^{x,i}) = W$ for all $x_i \in \mathcal{M}$

$V(Q^{x,j}_{a,i}) = \{\pi \in W \mid \pi(x_j) < \pi(a_i)\}$ for all $a_i, x_j \in \mathcal{M}$.

Here $\pi(x_j)$ ($\pi(a_i)$) denotes the position of message x_j (a_i) in the permutation π and the order between two positions is shown by the relation '$<$'.

Definition 9. (Semantics). Let a model M be given. We inductively define the interpretation of formula $\phi \in \mathcal{L}$ on (M,π) as follows,

$M,\pi \models \top$, always ,
$M,\pi \models p \in \mathcal{P}$ iff $\pi \in V(p)$,
$M,\pi \models \neg\phi$ iff $M,\pi \not\models \phi$,
$M,\pi \models \phi \wedge \psi$ iff $M,\pi \models \phi$ and $M,\pi \models \psi$,
$M,\pi \models K_a\phi$ iff for all $\tilde{\pi}$ with $\pi R_a \tilde{\pi}$, $M,\tilde{\pi} \models \phi$,
$M,\pi \models A^a_\phi\psi$ iff $M_{\phi,a} \models \psi$,

where $M_{\phi,a}$ is the updated model under the action A^a_ϕ. We now provide the details of the model updates (actions) that we consider for BSS protocol.

In BSS protocol the content of each message is a formula in $\mathcal{L}$ either of the form $\phi = P^{x,i}$ or of the form $\phi = P^{x,i} \wedge \bigwedge_{y \neq x} Q^{y,j}_{x,i}$, since the sender also broadcasts the messages she has received from each of the other agents along with the current message. We now concentrate on such formulas to model the BSS protocol. The initial model is $M = (W,R,V)$ where W is the set of all allowed permutations on $\mathcal{M}$ and for each agent x, R_x is a universal relation on W, i.e., $R_x = W \times W$. Suppose the formula ϕ is received by agent x. Then the corresponding action will transform the current model M into the model $M_{\phi,x} = (W,R^{\phi,x},V)$ where only the agent relations are modified. The new set of agent relations is $R^{\phi,x} = \{R'_a \mid a \in \mathcal{A}\}$ where,

- For each $y \neq x$, $R'_y = R_y$,
- R'_x is the intersection of R_x and the complete relation on E' where, $E' = \{\pi \in W \mid \pi(sender(\phi)_{index(\phi)}) < \pi(x_{MSN(x)})$ and $M,\pi \models \phi\}$.

When a message having content ϕ is received by an agent x, the corresponding action A^x_ϕ is performed on the current model M, transforming it to $M_{\phi,x}$. We note that the operator A^x_ϕ acts globally. After all messages are received, the set of permutations can be partitioned into two sets. In one set, each permutation π is isolated with respect to every R_x, i.e., $\pi \not{R_x} \sigma$ for all $\sigma \in W$ and for all x. The other set is universal with respect to every R_x (i.e., for any π, σ in the set, $\pi R_x \sigma$ for all x) and in each of them the messages appear in correct causal order. Formally, $x_i \to y_j$ if and only if we have that in the final model M_{fin}, $M_{fin} \models \wedge_{a \in \mathcal{A}}(L_a\top \to K_a Q^{x,i}_{y,j})$. For a worked out example, see (Das and Ghosh, 2020).

3.2 LME Protocol

For the analysis of LME Protocol, we will develop the language corresponding to *request* messages only, as the other kinds of messages do not play a role in

generating the order of these messages. The ordering of *request* messages generated by LME Protocol is an index-based ordering where ties between *request* messages with equal indices are broken based on the IDs of the requesting processes. We will later show that this is actually a causal ordering of the *request* messages. Let $\mathcal{A}$ denote the set of agents/processes in the distributed system, and $\mathcal{M}$ denote the set of *request* messages whose causal ordering we are interested in. Each *request* is of the form x_i, which means that it is the i^{th} request made by agent x. Permutations and allowed permutations are defined as earlier.

Definition 10. (Index of a message). **Index of a message** generated by a process is 1 if the process has not sent or received any other message before. Otherwise it is $k+1$ where k is the maximum index of all messages received or sent by the process before.

Definition 11. (Timestamp of a message). **Timestamp of a message** is generated by concatenating its index (definition 10) at the front of the ID of the sender process.

3.2.1 Language

We will now define the language for modeling the LME protocol. The set of propositions is given by,

$$\mathcal{P} = \{Q_{x,i}^{y,j} \mid x,y \in \mathcal{A};\ x_i, y_j \in \mathcal{M};\ x_i \neq y_j\}.$$

Here, $Q_{x,i}^{y,j}$ reads "*request* x_i causally precedes *request* y_j". The formulas are given by,

$$\phi := \top \mid p \mid \neg\phi \mid \phi \wedge \phi \mid K_a\phi \mid A_s^{x_i,j}\phi \mid A_{r,y}^{x_i,j}\phi,$$

where, $p \in \mathcal{P}$. As earlier, the formula $K_a\phi$ reads as "agent a knows ϕ". $A_s^{x_i,j}\phi$ reads "after *request* x_i with index j is sent, ϕ holds" and $A_{r,y}^{x_i,j}\phi$ reads "after *request* x_i with index j is received by y, ϕ holds".

3.2.2 Semantics

We define A model M as a tuple (W,R,Ind,V) where,
$W = \{\pi \mid \pi$ is an allowed permutation on $\mathcal{M}\}$,
$R = \{R_a \mid a \in \mathcal{A};\ R_a$ is a binary relation on $W\}$,
$Ind = \{Ind_a \mid a \in \mathcal{A}\}$ where, $Ind_a = \{(a_i, j) \mid a_i \in \mathcal{M}, index(a_i) = j \in \mathbb{N}\}$ (Ind_a stores the *requests* observed by agent a and their corresponding indices),
$V : \mathcal{P} \to 2^W$ is given by $V(Q_{x,i}^{y,j}) = \{\pi \mid \pi(x_i) < \pi(y_j)\}$.

Definition 12. (Semantics). Let a model M be given. The truth definition of the formulas ϕ of the language above at (M,π) is given as follows,

$M,\pi \models \top$, always ,
$M,\pi \models Q_{x,i}^{y,j}$ iff $\pi \in V(Q_{x,i}^{y,j})$
$M,\pi \models \neg\phi$ iff $M,\pi \not\models \phi$,
$M,\pi \models \phi \wedge \psi$ iff $M,\pi \models \phi$ and $M,\pi \models \psi$,
$M,\pi \models K_a\phi$ iff for all $\tilde{\pi}$ with $\pi R_a \tilde{\pi}$, $M,\tilde{\pi} \models \phi$,
$M,\pi \models A_s^{x_i,j}\psi$ iff $M',\pi \models \psi$,
$M,\pi \models A_{r,y}^{x_i,j}\psi$ iff $M'',\pi \models \psi$.

Here M' denotes the updated model under $A_s^{x_i,j}$ and M'' denotes the updated model under $A_{r,y}^{x_i,j}$ (with respect to the current model M). We now provide the details of model updates under these actions.

The model M changes when a *request* is generated or received by a process. The corresponding action transforms the model M to the model $M' = (W,R',Ind',V)$ where only the agent relations and Ind sets corresponding to the agents are modified.

- Suppose *request* x_i is generated by agent x.

- If $y \neq x$ then $Ind'_y = Ind_y$.
- $Ind'_x = Ind_x \cup \{(x_i, j)\}$ where $j = index(x_i)$.
- If $y \neq x$ then $R'_y = R_y$.
- R'_x is the intersection of R_x and the complete relation on E' where
  $E' = \{\pi \in W \mid \forall (y_j,k) \in Ind_x; \pi(y_j) < \pi(x_i)\}$.

- Suppose *request* x_i is received by agent y.

- If $z \neq y$ then $Ind'_z = Ind_z$.
- $Ind'_y = Ind_y \cup \{(x_i, j)\}$ where $j = index(x_i)$.
- If $z \neq y$ then $R'_z = R_z$.
- R'_y is the intersection of R_y and the complete relation on E' where,
  $E' = \{\pi \in W \mid \forall (z_j,k) \in Ind_y; \pi(x_i) < \pi(z_j)$ iff $index(x_i) < k$ and $\pi(x_i) > \pi(z_j)$ iff $index(x_i) > k\}$.

The initial model is $M = (W,R,Ind,V)$ where W is the set of all allowed permutations on $\mathcal{M}$, for each agent x, R_x is a universal relation on W (i.e., $R_x = W \times W$) and $Ind_a = \emptyset$ for all $a \in \mathcal{A}$. When an agent x sends its i^{th} *request* or when an agent y receives the *request* x_i, the corresponding actions $A_s^{x_i,j}$ or $A_{r,y}^{x_i,j}$ is performed on the current model respectively. The model update happens according to the rules defined above. After all *requests* are received, the set of permutations gets partitioned into two sets, as in the case of BSS Protocol. In one set, each permutation is isolated with respect to every R_x (i.e., $\pi \not{R_x} \sigma$ for all $\sigma \in W$ and for all x) and the other set is universal with respect to every R_x (i.e., for any π, σ in the set, $\pi R_x \sigma$ for all x)

and in each world of it, the *requests* appear in correct causal order. Observe that $index(x_i) < index(y_j)$ iff in the final model $M_{fin} \models \wedge_{a\in\mathcal{A}}(L_a\top \rightarrow K_aQ^{y,j}_{x,i})$, i.e., every agent considers only those permutations where the *requests* are ordered according to their indices. For a worked out example, see (Das and Ghosh, 2020).

4 CORRECTNESS OF THE PROTOCOLS

In the following, we prove the correctness of the protocols, using knowledge-based reasoning as provided by the formal frameworks we just discussed.

4.1 Correctness of BSS Protocol

It is claimed that in BSS protocol, each agent executes messages in the correct causal order. To prove this we will show that after all the messages are received, each agent knows the causal order between any two messages. In logical language, we will show that the final model $M_{fin} \models \wedge_{a\in\mathcal{A}}(L_a\top \rightarrow K_aQ^{x,i}_{y,j})$ iff $x_i \rightarrow y_j$. The following theorems prove this.

Theorem 1. (Agreement). After all the messages are received, an agent a is sure that x_i precedes y_j iff every other agent in the system is sure about this. Formally, we have that $M_{fin} \models (L_a\top \rightarrow K_aQ^{x,i}_{y,j}) \leftrightarrow (\wedge_{b\in\mathcal{A};b\neq a}(L_b\top \rightarrow K_bQ^{x,i}_{y,j}))$.

Proof. The detailed proof of this theorem is provided in (Das and Ghosh, 2020). □

Definition 13. (Basic precedence). We say a causal precedence $a_i \rightarrow b_j$ is a basic precedence iff either $a = b$ and a_i is sent before a_j or $a \neq b$ and a_i is received by b before b_j is sent.

It is easy to observe from the definition of causal order that if $a_i \rightarrow b_j$ then either it is a basic precedence or there are $c^{(1)}_{i_1}, \ldots, c^{(m)}_{i_m}$ such that $a_i \rightarrow c^{(1)}_{i_1} \rightarrow \cdots \rightarrow c^{(m)}_{i_m} \rightarrow b_j$ where each precedence is a basic precedence. Let us use this observation to show that each agent knows all the causal orders when BSS protocol is used.

Theorem 2. If $x_i \rightarrow y_j$ is a basic precedence then agent y knows x_i precedes y_j. In other words, in the final model $M_{fin} \models L_y\top \rightarrow K_yQ^{x,i}_{y,j}$.

Proof. We have provided the detailed proof in (Das and Ghosh, 2020). □

Theorems 1 and 2 show that each basic precedence is known to all agents. Since any causal precedence can be built up from basic precedence only, we get that all causal precedence are known to every agent. Thus the claim that each agent knows all and only the causal orders after message passing, is true. Thus, no agent can violate the causal order while executing messages. We also observe that BSS protocol generates a unique allowed permutation iff for any two messages x_i and y_j, either every agent knows $x_i \rightarrow y_j$ or every agent knows $y_j \rightarrow x_i$. In logical language, BSS protocol generates a unique permutation iff the final model $M_{fin} \models \wedge_{x_i\neq y_j}((\wedge_{a\in\mathcal{A}}(L_a\top \rightarrow K_aQ^{x,i}_{y,j})) \vee (\wedge_{a\in\mathcal{A}}(L_a\top \rightarrow K_aQ^{x,i}_{y,j})))$.

4.2 Correctness of LME Protocol

For LME protocol, we have to show that no two *requests* can access the critical section at the same time. To prove this we show that all agents agree on a unique permutation of the *requests* and the critical section is accessed in that order. Thus there cannot be any conflict between the decisions of any two agents. We will also show that the permutation each agent agrees on, is actually a causal order of the *requests*.

Theorem 3. If two *requests* x_i and y_j are such that $x_i \rightarrow y_j$, then $index(x_i) < index(y_j)$.

Proof. We have provided a detailed proof in (Das and Ghosh, 2020). □

Theorem 4. (Agreement). Each agent agrees on permutations where *requests* appear in correct causal order and all agents agree on same set of permutations.

Proof. We have provided a detailed proof in (Das and Ghosh, 2020). □

Now if every agent breaks tie between *requests* with same index using process ID (i.e., the process with smallest ID wins in case of a tie in indices), they must agree on a unique permutation since without using process ID they agree on same set of permutations. Thus LME protocol leads to a unique permutation of *requests* which is agreed upon by all agents. This proves the correctness of the protocol. In other words, each agent knows the order in which the *requests* should access the critical section.

5 ON IMPLEMENTATION AND TIME COMPLEXITY

In all these models we have only used allowed permutations as states. Let us first provide an estimate

for the number of allowed permutations. Let for each agent $a \in \mathcal{A}$, $\mathcal{M}_a$ be the set of messages sent by agent a. Then $\mathcal{M} = \cup_{a\in\mathcal{A}}\mathcal{M}_a$. Since $\mathcal{M}_a \cap \mathcal{M}_b = \emptyset$ for $a \neq b$, $|\mathcal{M}| = \sum_{a\in\mathcal{A}}|\mathcal{M}_a|$. Let SA be the set of allowed permutations. Then,

$$|SA| = \frac{|\mathcal{M}|!}{\prod_{a\in\mathcal{A}}|\mathcal{M}_a|!}$$

In case each agent sends only one message, we have $|SA| = |\mathcal{M}|!$. Once again, the detailed analysis is provided in (Das and Ghosh, 2020).

5.1 Time Complexity for BSS Protocol

In this section we provide a possible implementation of the action update model for BSS protocol and its time complexity. The implementation has two steps:

1. Given set of agents $\mathcal{A}$ and set of messages $\mathcal{M}$ build the initial model according to BSS protocol.
2. Update the model according to the BSS protocol as messages are received by agents until each message is received by all the other agents except for the sender.

The time complexity of each of the above steps is $O(|\mathcal{A}||\mathcal{M}|!)$. Thus total time complexity of the entire implementation is $O(|\mathcal{A}||\mathcal{M}|! + |\mathcal{A}||\mathcal{M}|!)$, i.e., $O(|\mathcal{A}||\mathcal{M}|!)$ (for a detailed analysis see (Das and Ghosh, 2020)).

5.2 Time Complexity for LME Protocol

In this section we provide a possible implementation of the action update model for LME protocol and its time complexity. The implementation has two steps:

1. Given a set of agents $\mathcal{A}$ and a set of *requests* $\mathcal{M}$ build the initial model according to LME protocol.
2. Update the model according to LME protocol as messages are sent and received by agents until each message is received by all the other agents except the sender.

The time complexity of each of the above steps is $O(|\mathcal{A}||\mathcal{M}|!)$. Thus total time complexity of the entire implementation is $O(|\mathcal{A}||\mathcal{M}|! + |\mathcal{A}||\mathcal{M}|!)$, i.e, $O(|\mathcal{A}||\mathcal{M}|!)$ (for a detailed analysis see (Das and Ghosh, 2020)).

6 CONCLUSION AND FUTURE DIRECTIONS

To summarize, we have shown that simple variants of dynamic epistemic logic can indeed be used to model distributed protocols in asynchronous settings. The knowledge-based framework can be used to model the effect of these protocols, and the underlying reasoning process becomes explicit. In some sense, the action update operators model the effect of receipt of messages, and can be considered as private announcements to the individual processes/agents. However, the logical languages developed here are protocol-specific and quite ad hoc in nature. It would be interesting to see whether a uniform framework could be developed and how that framework would relate to the existing literature on announcement logics. We leave this for the future.

Another direction to work on is to bring other distributed protocols in the purview of the logical framework described here. To this end, one can consider Maekawa's quorum based mutual exclusion algorithm (Singhal and Shivaratri, 1994), where each process consults only a proper subset of all processes before accessing the critical section. It would be interesting to see the variant of dynamic epistemic logic which could be used to model the underlying reasoning process of the protocol.

REFERENCES

Ahmadi, S., Fallah, M. S., and Pourmahdian, M. (2019). On the properties of epistemic and temporal epistemic logics of authentication. *Informatica*, 43(2).

Balbiani, P., van Ditmarsch, H., and González, S. F. (2020). From public announcements to asynchronous announcements. In Giacomo, G. D., Catala, A., Dilkina, B., Milano, M., Barro, S., Bugarín, A., and Lang, J., editors, *Proceedings of the 24th European Conference on Artificial Intelligence*, volume 325 of *Frontiers in Artificial Intelligence and Applications*.

Baltag, A. and Smets, S. (2008). A qualitative theory of dynamic interactive belief revision. In Bonanno, G., van der Hoek, W., and Wooldridge, M., editors, *Logic and the Foundations of Game and Decision Theory (LOFT07)*, volume 3 of *Texts in Logic and Games*, pages 9–58. Amsterdam University Press, Amsterdam.

Das, S. and Ghosh, S. (2020). Receiving messages in their correct order: Analyzing broadcast protocols in dynamic epistemic logic (a technical report). Technical report, Indian Statistical Institute.

Dechesne, F. and Wang, Y. (2010). To know or not to know: epistemic approaches to security protocol verification. *Synthese*, 177(1):51–76.

Halpern, J. Y. and Zuck, L. D. (1992). A little knowledge goes a long way: knowledge-based derivations and correctness proofs for a family of protocols. *Journal of the ACM (JACM)*, 39(3):449–478.

Knight, S., Maubert, B., and Schwarzentruber, F. (2019). Reasoning about knowledge and messages in asyn-

chronous multi-agent systems. *Mathematical Structures in Computer Science*, 29(1):127–168.

Singhal, M. and Shivaratri, N. (1994). *Advanced Concepts in Operating Systems: Distributed, Database, and Multiprocessor Operating Systems*. Computer Science Series. McGraw-Hill.

van Ditmarsch, H., Ghosh, S., Verbrugge, R., and Wang, Y. (2014). Hidden protocols: Modifying our expectations in an evolving world. *Artificial Intelligence*, 208:18–40.

van Ditmarsch, H., van der Hoek, W., and Kooi, B. (2007). *Dynamic Epistemic Logic*, volume 337 of *Synthese Library*. Springer Verlag, Berlin.