A Compromise Between Synchronous and Asynchronous Systems

Tanisha Saxena
Lexington High School
Lexington, MA, USA

Jun Wan (Mentor)
Massachusetts Institute of Technology
Cambridge, MA, USA

Abstract

In this paper, we introduce a partially synchronous model for distributed systems such that any protocol for our model can be transformed to a corresponding protocol for the asynchronous model. Given a distributed system with n users, we define a normal adversary as one that allows up to f ($f < n/2$) users to send any arbitrary message at any time, and a special adversary that can, additionally, block up to f message channels for any number of users. We prove that, for any synchronous protocol that is resilient to the special adversary, there is an equivalent protocol for the asynchronous model that is resilient to the normal adversary. The special adversary helps us relax the restriction of time-bounded delivery and provides a model that is useful in analyzing if a synchronous protocol can be modified to work correctly in an asynchronous distributed system. Our model provides a basis to use synchronous protocols to function on asynchronous systems such as electronic banking and Blockchain systems distributed across the Internet.

1 Introduction

In this paper, we aim to analyze the relationship between synchronous and asynchronous systems. The synchronous model is useful for theoretical analysis because its strong assumptions of global synchronous clocks and time-bounded message delivery allow for efficient protocols [CD+15], [Cri91]. However, these assumptions are impractical for modelling real-world distributed systems where imposing a consistent time-bound may lead to unrealistic protocols. For example, a time bound that is long enough to accommodate all messages sent across a system distributed across the Internet may significantly degrade the performance of the protocol due to timeouts. Similarly, a time bound that is too short may result in the distributed system violating the model constraints, thus poorly representing the system. Though asynchronous model ([MMR14]) makes no assumptions about synchronicity, it results in protocols that are too complex to be realistic.

Our goal is to create in-between models that can transition synchronous protocols into asynchronous protocols such that writing protocol for distributed systems can be simpler. Our objective is to create a model that can accurately test the resilience of privacy with the addition of a difficult adversary in real-world systems.

2 Background and Definitions

A **distributed system** is a network of n **users** that communicate by sending messages to each other. For our purposes, the main goal of a distributed system is to, eventually, come to a **consensus** in the form of one common decision between all users, once each user has received an initial message.

Distributed systems use clocks to track time and send messages. A **global clock** is a single global counter that tracks time for all users in a distributed system. The value of the global clock *may or may not* be accessible by users of a distributed system. A **local clock** is a counter used by a user to locally simulate time. A user can *always* access the value of its local clock. Local clocks of users are independent of each other.

The existence of clocks in our models allows us to define **events** [**seminalLamport**]. Events can be actions such as the tick of a clock, sending of a message, or delivery of a message. Each user in a distributed system goes through a sequence of events that is ordered based on the clock time (local or global) that that user has access to.

Users of distributed systems send **messages** to other users through communication channels at certain delivery speeds. We define delivery speed as follows.

Definition 2.1. ***Delivery speed*** *of a distributed system with n users is an $n \times n$ array called s where $s[i, j]$ is the inverse of the time it takes to send a message from user i to user j.*

Delivery speed can be asymmetric; the speed at which messages are delivered from user i to user j may be different than the speed of delivery from j to i ($s[i, j] \neq s[j, i]$). To guarantee delivery within a certain time frame, systems can bound their delivery time.

Definition 2.2. *A distributed system is said to be* ***delivery time bounded*** *if there exists a $T_r \in \mathbb{Z}$ such that all messages sent at global time t are guaranteed to be delivered before global time $t + T_r$.*

A **protocol** is an algorithm that determines how each user in a distributed system acts in order to achieve consensus. There are many ways of mathematically describing a protocol. For our purposes. we specify protocols using a function between input and output messages of users.

Definition 2.3. *A protocol can be specified as a function $P(i, s)$ where i is an input vector containing the input messages for each user and s is the delivery speed array for this system. The protocol creates an output vector o that specifies the output generated by each user after they receive their input messages.*

$P(i, s)$ can be successively applied to determine the sequence of outputs a distributed system generates for a given input.

We are now ready to define synchronous and asynchronous distributed systems.

Definition 2.4. *A* ***synchronous system*** *is a delivery time bounded system where each user can access the global clock with zero delay.*

This definition is aligned with the Universal Composablity model framework described in [CD+15] and [Cri91] where users may use this global time as an **event**, i.e use a tick of the global clock to trigger an action. However, there can also be distributed systems without global clocks.

Definition 2.5. *An* ***asynchronous system*** *is a system where users only have access to a local clock, do not have access to a global clock, and the speed of messages is not bounded.*

Therefore, in asynchronous systems, users may send messages to other users with only the assumption that messages will be delivered eventually [MMR14], and asynchronous protocols cannot assume any time bound on message delivery.

Based on these definitions, synchronous and asynchronous systems differ in a few key ways. In synchronous systems, it is easier to define and coordinate instructions between users because there is a guaranteed time when all users will receive all messages. As a result, synchronous protocols are more efficient. However, synchronous models are less practical since guaranteed time-bound delivery is hard to achieve in systems distributed across large networks such as the Internet.

In asynchronous systems, on the other hand, a user may wait indefinitely before they can take an action based on a new message. As a result, asynchronous protocols are more complex since they cannot make any assumptions about when messages are delivered. The complexities of these protocols may make them impractical.

3 Our Problem Statement and Methodology

We aim to find a synchronous model that relaxes the restriction of time-bounded delivery while still being stronger than a pure asynchronous system. Such models can help identify less complex synchronous consensus protocols that can be modified to correctly achieve consensus on a subset of asynchronous systems without time-bounds on message delivery.

We first explore the properties of the asynchronous model with synchronized clocks, but without the restriction of time-bound delivery. We prove that the asynchronous model with synchronous clocks is no different than the asynchronous model with local asynchronous clocks. Therefore, synchronous clocks are not useful in bringing properties of synchronous systems to asynchronous systems.

Next, we explore if we can increase the flexibility of synchronous systems by removing the strong restriction of time-bounded delivery. We accomplish this by using our protocol specification from Definition 2.3 to prove the equivalence of time-based synchronous systems and round-based systems from [Del+07], [BFA21] and [**roundConsensus**]. That enables us to talk about synchronous systems in terms of sequence of events rather than clocks. We leverage this simplification to define a new adversary, resulting in a synchronous model that is stronger than the asynchronous model, yet more flexible than the pure synchronous model.

4 Adding Synchronicity to Asynchronous Systems

To bring synchronicity to the asynchronous models, we will use internal clocks to strengthen the protocol.

Asynchronous and synchronous systems have two main differences: clocks and delivery time. Synchronous systems have synchronous clocks which allow them to coordinate events. They also have finite delivery times such that for any message m sent at time t, it will always be received before time $t + T_r$ where $T_r \in \mathbb{Z}$. Asynchronous systems, on the other hand, do not have clocks nor any finite bounds on delivery time. They simply state that any message m that is sent will eventually be received. In real-world systems, guaranteed delivery in a finite time is not always possible. However, can adding a synchronous clock to an asynchronous system enable us to model systems that are stronger than pure asynchronous systems, even if not as strong as synchronous systems? The answer turns out to be negative. Here we prove that that clocks do not make any difference, and it is in fact the delivery time that most differentiates the synchronous and asynchronous systems.

A standard asynchronous system has asynchronous clocks. Each user gets an individual local clock that is independent of, and may move at speeds different from, the clocks of other users. This is defined as a clock-driven execution of the asynchronous system, and is equivalent to the standard asynchronous model in 2.4.

We now construct an **asynchronous system with synchronous clocks**, and evaluate how it is different from a standard asynchronous system, if at all.

Definition 4.1. *An* ***asynchronous system with synchronous clocks*** *is one where:*

- *Each user has a local clock that is synchronized with clocks of other users, i.e., they tick at exactly the same time.*
- *Users can access the time on their local clock with no time delay.*

Since the clocks of all users are synchronized and can be accessed by users without delay, this system is equivalent to a system with a global clock that all users can read without delay. Both asynchronous models have clocks connected to each user, but the asynchronous clock model can have different speeds for each clock while the synchronous clock model has the same time on all clocks.

Before we compare the models, we want formally define what it means for two systems to be equivalent.

Definition 4.2. *Two systems are said to be* ***equivalent*** *if, for any protocol P for one model, there exists a protocol P' for the other model that maintains the same order of events.*

We now prove that an asynchronous system with synchronous clocks is equivalent to an asynchronous system with asynchronous clocks. By transitivity, an asynchronous system with synchronous clocks is equivalent to the standard asynchronous model. In other words, we will prove that the added benefit of having local clocks that are synchronized across users and accessible without delay does not add any functionality to, or improve, an asynchronous system.

Theorem 4.1. *An asynchronous system with synchronous clocks is equivalent to an asynchronous system with asynchronous clocks.*

Proof. For the purposes of this proof, we require that a message sent by a user contains two pieces of data: the message itself, and the time T_A of user A that is sending the message. This assumption does not change the properties of the protocol because it only increases the communication complexity of a protocol by a constant factor.

Messages in the asynchronous system with synchronous clocks are sent as $P(A, m)$ where A is the user sending the message m, and are stored in the event order as $P(A, m, T_s, T_r)$ where a message m is sent by user A at time T_s and is received at time T_r. Every message in the asynchronous system with asynchronous clocks model is sent as $P'(A, m, T_a)$ where A is sending message m at local clock time T_a, and is stored in the event order as

$P(A, m, T_a, T_b)$ where message m is sent by user A at local clock time T_a and received at another user's local clock time T_b.

Given a protocol P in the asynchronous model with synchronous clocks, with inputs of the starter messages i and delivery speed s, a separate protocol P' can be designed under the asynchronous with asynchronous clocks model such that $P(i, s) = P'(i', s')$. This is because the addition of synchronous clocks is rendered useless by the indefinite delivery time of messages within the asynchronous model, and can be accurately described using the asynchronous model with asynchronous clocks as follows.

We create a global synchronous time for the asynchronous system that can be accessed with no time delay by every user. This is same as the asynchronous system with synchronous clock described earlier because:

1. Sending a message $P(A, m, T_s)$ can be converted to sending a message $P'(A, m, T_a)$ where $T_a = T_s$.

2. When user B receives the message in P', its local clock is changed such that $T_b = T_a + r$ where r is a random number and $r > 0$ and $r \in \mathbb{Z}$.

3. Hence the message that is stored as $P(A, m, T_s, T_r)$ can be converted to $P'(A, m, T_a, T_b)$ after T_b has been adjusted such that $T_b > T_a$ as described earlier,

4. The adjustment of clocks in P' ensures that all saved events of the protocol match directly with the events in P, as a message sent at time T_a is always received after time T_a and is therefore stored in the correct order.

5. Since all events in both protocol occur in the same order, they are proven to be equivalent.

$\square$

We have proven that adding a synchronous clock to an asynchronous model does not change its asynchronicity. So we conclude that adding a synchronous clock to an asynchronous model does not provide us with a model that is stronger than a pure asynchronous system. In the next section, we look towards altering the functionality of the rigid synchronous system to reach a middle ground between the synchronous and asynchronous systems by establishing equivalence between round-based and time-based synchronous systems and by using adversaries.

5 Adding Flexibility to Synchronous Systems

We will attempt to make synchronous systems more flexible with two steps. First, we will simplify synchronous systems using the concept of rounds, and then we will add adversaries to remove the requirement for time-bounded delivery of messages.

5.1 Round-Based Synchronous Systems

The concept of rounds was used in [Del+07] to develop an alternative definition of synchronous distributed systems. In the round model, all users operate in lockstep. During each round, all users receive messages from other users, perform their local computation, and send new messages to other users. Therefore, any message sent in round r will be received before round $r+1$ begins. A protocol may define a constant time interval T where **round** r is defined as the time interval $[i*T,(i+1)*T]$. We will prove that round-based and time-based synchronous systems are equivalent by proving that they have the same set of properties.

A **property** of a protocol can be described as a *matching function* between input vectors i and output vectors o. Specifically, given a function F, we can say that a protocol P satisfies property F if and only if $F(i,o)=1$ for all input and output pairs (i,o) under the protocol P. Notice that, per Definition 2.3, output vector $o=P(i,s)$.

However, this definition doesn't make sense in the presence of adversaries because they can affect and control the system delivery speed s (see Definition 2.1). For example, an adversary can delay or even corrupt a message, in which case the message may not be delivered at the expected speed, or not delivered at all. To address adversaries, we define a property as follows:

Definition 5.1. *A protocol P satisfies* ***property*** *F if and only if for any possible input vector i and delivery speed s, $F(i,P(i,s))=1$.*

This enables us to define protocol equivalence as follows.

Definition 5.2. *Two protocols P and P' are said to have* ***protocol equivalence*** *if for any input vector i and delivery speed s, there exists a delivery speed s' such that $P(i,s)=P'(i,s')$.*

This implies that for any property F that P' satisfies, P satisfies that property F as well. This is because for any i and s,

$$F(i,P(i,s))=F(i,P'(i,s'))=1.$$

This means that given two systems A and B if, for any protocol P under A, there exists a protocol P' under B such that P and P' are equivalent, then the system B has all the properties F of system A. We will use this approach to prove that the long-known result that a synchronous system with synchronous clocks is equivalent to a round-based system holds even under our formulation of the property in Definition 5.1 [Del+07], [BFA21], [**roundConsensus**]. To establish this equivalence, we first show that given any synchronous system, we can model a round-based system that has all properties of the synchronous system, and then show that given any round-based system, we can create an synchronous model with the same properties.

Lemma 5.1. *For any protocol P in the round-based system, there exists a protocol P' in the synchronous model that is equivalent to P, per Definition 5.2.*

Proof. The intuition is that, for any event in the synchronous system's time model, we can convert any time into a corresponding round within the round model such that the sequence of events remains the same and the output is equal.

The synchronous system contains two key event triggers: receiving a message at any time, and reaching the end of a round. The round model contains two event triggers as well: receiving a message at the beginning of round r and reaching the end of a round. Also recall that in the sync model, every message takes a max of T_r time to be delivered where $T_r\in\mathbb{Z}$.

For convenience, we can denote how a protocol P' reacts to the event "a message m from user i at time T_a" by $P(m,i,T_a)$. Similarly, in the round model, we denote how a protocol P reacts to the event "receiving a message m from user i in round r" by $P(m,i,r)$. We will show how to construct P given P' such that they are equivalent. Specifically, we show that the sequence of events in the synchronous system can be described by a round model that generates the same sequence of events, proving that both are equivalent.

First we reduce protocol P to P' as follows:

1. $P(m,i,T_a)$: reduces to $P'(m,i,\left\lfloor\frac{T_a}{T_r}\right\rfloor)$. In other words, on receiving a message m in time T_a, P simulates what P' would do if it receives m in round $\left\lfloor\frac{T_a}{T_r}\right\rfloor$. This ensures that any message sent in the synchronous system can be converted to an accurate counterpart in the round-based model.

2. Users only send message during the period $T = 2k \cdot T_r, T = (2k+1) \cdot T_r$.

3. Reaching end of time T event: if $T = 2k \cdot T_r$ for some constant k, then simulate what P would do at the end of round k. Otherwise, don't do anything.

Next, we describe the actions of users in the synchronous system. We expand on how, given the time it is sent and the time the message is received, any series of events in the synchronous system can be described by the round-based model while fitting the restrictions in the correct order. This means that for all events A in protocol P, there will be a corresponding even B in protocol P'

Now we establish that P' will generate events in the same sequence as P.

1. Using an integer $k \in \mathbb{Z}$, we can clarify time transformations between the time model and round model: in round k, any event that is taken within the round must end before the next round starts, equal to the time interval $(2kT_r, (2k+1)T_r)$ in the time model.

2. For event $R(A_i)$ in the time model where user A is sending a message at time t, this can be converted where $R(A_i) = (send, A, \lfloor\frac{t}{2T_r}\rfloor, m)$ where $2kT_r \leq t \leq (2k+1)T_r$, "send" is the action, and m is the message.

3. For event $R(B_j)$ in the time model where user B is receiving a message from user A at time t', $R(B_j) = (receive, B, A, \lfloor\frac{t'}{2T_r}\rfloor, m)$ where $t' < t + T_r \leq (2k+2)T_r$.

4. The described restrictions for t' also mean that $\lfloor\frac{t'}{2T_r}\rfloor < k+1$, therefore, $R(B_j) = (deliver, B, A, k, m)$.

This proves that the events causing a message sent at time t and received at t' all take place within the same round k, meaning that any message events in the synchronous system can be converted to the Round-based model while fitting the respective restrictions. Therefore, given any protocol P for the round-based model, we have defined a protocol P' for the synchronous system that is equivalent to P, hence proving the lemma. □

Now we prove the other direction, that is, for any protocol under the round-based model, we can create an equivalent protocol in the synchronous system.

Lemma 5.2. *Given any protocol P in the synchronous system, there exists a protocol P' in the round-based model that is equivalent to P per Definition 5.2.*

Proof. Given a protocol P for the synchronous system, we construct P' for the round-based model as follows:

1. Sending a message at the start of round r, represented by $P(m, i, r)$ can be reduced to $P'(m, i, r * T_r)$ where the round model starts at round 0.

2. Reaching the end of round r is equivalent to starting round $r+1$, so the same method as above can be applied to transform the protocol.

If we send a message at the beginning of the round, it will be delivered by the end of the round. So after reduction, the message takes at most T_r time to deliver. This implies that P and P' are equivalent. □

Theorem 5.3. *Given any synchronous system S, there is an equivalent round-based model R, such that if $F(i, o)$ is a property of S then it is also a property of R, and vice-versa.*

Proof. Lemma 5.2 shows that given any synchronous system, we can create a round-based such that all protocols in the synchronous system have an equivalent protocol in the round-based model. That means, given any synchronous system, we can create a round-based model that has all the properties of the synchronous system, which proves the theorem in one direction. Lemma 5.1 similarly shows that given any round-based model, we can create a synchronous system with the same properties. □

Since we have already proven that the synchronous and round-based models are equivalent, we can simply use round-based models to describe synchronous systems. In our proofs, we described round-based models using order of events. This is a useful technique which we will now generalize, and formally define so we can use it to define all models.

Definition 5.3. *An **Event-order-based (EOB) Description** is a description of a model using a sequence of the following events:*

- *Send(i, j, m) where user i sends user j a message m*
- *Deliver(i, j, m) where user j receives the message m sent by user i*
- *Round(begin/end, r) where round r either begins or ends in that event*
- *Block(i, j) where the messages that user i sends to j are blocked in one direction.*

Since the output of a computation in a distributed system is determined by the ordering of events executed by the model, we can define special types of distributed systems by simply placing restrictions on these events. For example, as used in our proof, the EOB description to define synchronous systems is as follows:

Definition 5.4. *A synchronous system places the following restrictions on the sequence of events:*

1. *Send(i, j, m) must occur after Round(begin, r)*
2. *Deliver(i, j, m) must come after Send(i, j, m)*
3. *Deliver(i, j, m) must come before Round(end, r)*

We can also use EOB model to describe Asynchronous models as follows:

Definition 5.5. *An asynchronous system places the following restriction on the sequence of events:*

1. *Send(i, j, m) must come before Deliver(i, j, m)*

Now that we have proven that round-based systems are equivalent to synchronous systems, and used the EOB description, borrowed from round-based systems to describe synchronous systems, we are ready to introduce adversaries to increase the flexibility of synchronous systems so we can used them to model real-world distributed systems.

5.2 Using Adversaries to Remove Synchronicity

We know that asynchronous systems are too broad, and synchronous systems are too restrictive to model real-world systems. We will show how the concept of adversaries can be used to define a hybrid distributed system that is more flexible than synchronous systems, but not as broad as an asynchronous system.

Adversaries are a new type of user that do not follow the restrictions in Definitions 5.4 and 5.5. To distinguish adversaries from other users, we now call users that follow restrictions imposed by their distributed system as **honest users**. We further define two types of adversaries - *normal*, and *special*, and will prove that a synchronous model with a special adversary is equivalent to an asynchronous model with a normal adversary.

Definition 5.6. *A* ***normal adversary*** *can effect (corrupt) at most f users, and allows them to send any arbitrary message at any time. That is, up to f users can Send(i, j, m) at any time with any j and any m.*

A normal adversary cannot prevent a system from reaching consensus because it does not block any messages , simply adds messages.

We want to define a special adversary as one that can corrupt users to send arbitrary messages like a normal adversary and, additionally, can also block messages. That is, for any number of honest users, it can block up to k messages from other users in each round. Note that, by blocking messages with the special adversary, we break the time-bounded restriction over the synchronous model and help introduce asynchronous behaviour. However, such a special adversary is problematic because it can prevent a distributed system from reaching consensus. In order for a system to reach consensus, each user must be able to reach a conclusion. Therefore, we allow each honest user to define their own set of "safe" users whose messages they need to reach a conclusion.

Definition 5.7. *A* ***special adversary*** *can have two effects. First is the same effect as the normal adversary. Additionally, for any number of honest users, it can block up to f messages from other users in each round, but not from "safe" users of that honest user. That is, affected users can Block(i, j) for any j and i up to k times per round, except when i is in the "safe" user list of j.*

We now prove that special adversaries cannot prevent systems from reaching a consensus.

Theorem 5.4. *A special adversary cannot prevent a distributed system from reaching consensus under dishonest minority ($n > 2f$).*

Proof. By definition, each user of a synchronous model with n users must wait for $n - f$ messages where at least $n-2f$ messages are honest. If this condition is broken, the protocol cannot reach a conclusion because there will not be agreement across users. Since special adversaries cannot block safe users, we place at least f users in the safe list of every users so that they will be able to receive at least $n - 2f$ honest users' messages within $n - f$ message every round. This guarantees that a system-wide agreement is met. $\square$

Now we can prove what we set out to do, namely that introduction of a special adversary to a synchronous system results in a system that is less restrictive than a pure synchronous system, but not as open as a pure asynchronous system. In fact, it is exactly equivalent to an asynchronous system with normal adversaries. To show this, we prove equivalence in both directions.

Lemma 5.5. *Given any normal adversary A applied to the asynchronous system, there exists a corresponding special adversary A' that can be applied to the synchronous system to create the same output.*

Proof. We construct a special adversary A' from a normal adversary A as follows:

1. Based on Definition 5.6, a normal adversary A can corrupt up to f users. Let's label these users $U_1, U_2, U_3, \ldots, U_f$.

2. To match the blocked channels of the asynchronous system, we create a special adversary A' which also corrupts the exact same set of users, $U_1, U_2, U_3, \ldots, U_f$.

3. Now, to simulate the indeterminate delivery speed of an asynchronous system, we use the special adversary to block the channels from which the users do not receive messages in the asynchronous systems.

As shown in Theorem 5.4, a special adversary may not block enough users to prevent a user from receiving $n - 2f$ messages to reach a conclusion. Therefore, we can guarantee that the number of users from which another user does not receive a message will not exceed f.

As proven by the above paragraphs, the special adversary is able to impose the same restrictions on the synchronous system as the normal adversary does on the asynchronous system in addition to the restrictions of the asynchronous system itself. This proves that for any normal adversary there exists a special adversary that can simulate the effects of the asynchronous system and the normal adversary onto the synchronous systems. Since the restrictions from the effects are equivalent, the outputs with also be the same, proving the lemma.

$\square$

Lemma 5.6. *For any special adversary A applied to the synchronous system, there exists a corresponding normal adversary A' that can be applied to the asynchronous system to create the same output.*

Proof. We construct a normal adversary and an asynchronous system as follows:

1. Let all users of the synchronous system corrupted by the special adversary A be marked $U_1, U_2, U_3, \ldots, U_f$.

2. To simulate the same corrupted users, we construct A' such that it also the same set of users $U_1, U_2, U_3, \ldots, U_f$ in the asynchronous system.

3. Let the users of the synchronous system blocked by A be $U_{b1}, U_{b2}, U_{b3}, \ldots, U_{br}$.

4. To simulate blocked users, we create an asynchronous system where messages sent by users $U_{b1}, U_{b2}, U_{b3}, \ldots, U_{br}$ are not received before consensus is reached, and hence their message are effectively disregarded to simulate blocking.

We can guarantee that the last step above is possible in an asynchronous system because we have shown in Theorem 5.4 that an adversary may not block any users that would prevent the system from reaching a conclusion, therefore the number of users blocked will not exceed f.

Similar to lemma 5.5 this ensures that any restriction imposed by the special adversary A onto the synchronous system can be simulated by creating a specific normal adversary A' and applying it to a specific asynchronous system. Since the restrictions on the systems are equivalent they will both produce the same output, hence proving this lemma.

□

With these two lemmas proven, it is now easy to show that:

Theorem 5.7. *A synchronous system with a special adversary is equivalent to an asynchronous system with a normal adversary.*

Proof. Lemma 5.5 and 5.6 show that given any asynchronous system and normal adversary, we can construct an equivalent synchronous system and special adversary that produce the same output. □

With this theorem, we can conclude that a special adversary, when applied to a synchronous model, gives us a model that is equivalent to an asynchronous model with a normal adversary. That is, it gives us a model that is less restrictive than a pure synchronous model, yet more realistic than a pure asynchronous model. This is the main result of our paper.

6 Conclusion

In this paper we identified the aspects of a synchronous system that affect its synchronicity. We also defined a "special adversary" and showed that a synchronous system with a special adversary is stronger than a pure asynchronous system yet less restrictive than a pure synchronous system. Adding a special adversary helped us remove the requirement to deliver messages in a bounded time from synchronous models. This allows us to transform synchronous protocols into protocols that can work on asynchronous systems.

Our construct of adversaries also helped us create a hierarchy of distributed system models based on how restrictive they are. Our results show that the asynchronous model is the least restrictive, followed by the two equally restrictive asynchronous model with a normal adversary and synchronous model with a special adversary which are less restrictive than the synchronous model with a normal adversary, and lastly, the most restrictive model is the pure synchronous model.

7 Future Work

We want to consider and apply some changes to our initial problem regarding the adversaries research. Specifically, we want to investigate if it is possible to remove the normal adversary all together (given that the special adversary is equivalent to the normal adversary with some properties added) and exclusively apply the second part of the special adversary (block up to f channels for each user) to a synchronous model and make it equivalent to the plain asynchronous model.

Beyond that, we also question the practical effects of the normal adversary on the asynchronous model. As the asynchronous model is defined in 2, it is not clear whether adding the normal adversary affects the performance of the users, as they are allowed to send messages at anytime, regardless of the addition of the normal adversary.

On a larger scale, we want to continue our research on adversaries and explore the effects they can have on systems when they effect more than just the quantity of messages and the delivery time. We aim to determine if it is possible to generalize adversary effects such that, given a criteria on how strong a model should be, an adversary can be created with a limited selection of events it can manipulate that will accurate change the strength of any model to match the criteria.

Given that our work so far is theoretical, we would like to identify the practical applications and prove the functionality of our work in a more tangible way. To do this, we would run algorithms on distributed systems simulations to observe the efficiency of our adversaries and their ability to convert synchronous protocols to asynchronous protocols.

8 Acknowledgements

I would like to thank my mentor Jun Wan for providing his valuable insight and guidance as well as his time to help support me through this research. I would also like to thank Yu Xia, for his help in revising our research, and Srini Devadas, Slava Gerovitch, and the entire MIT PRIMES program for giving me the opportunity to participate in this research.

References

[BFA21] Raul Barbosa, Alcides Fonseca, and Filipe Araujo. “Reductions and abstractions for formal verification of distributed round-based algorithms”. In: *Software Quality Journal* (2021), pp. 1–27.

[CD+15] Ronald Cramer, Ivan Bjerre Damgård, et al. *Secure multiparty computation.* Cambridge University Press, 2015.

[Cri91] Flaviu Cristian. “Reaching agreement on processor-group membership in synchronous distributed systems”. In: *Distributed Computing* 4.4 (1991), pp. 175–187.

[Del+07] Carole Delporte-Gallet, Hugues Fauconnier, Rachid Guerraoui, and Bastian Pochon. “The perfectly synchronized round-based model of distributed computing”. In: *Information and Computation* 205.5 (2007), pp. 783–815.

[MMR14] Achour Mostefaoui, Hamouma Moumen, and Michel Raynal. "Signature-free asynchronous Byzantine consensus with t¡ n/3 and O (n2) messages". In: *Proceedings of the 2014 ACM symposium on Principles of distributed computing.* 2014, pp. 2–9.