BOUNDED PROTOCOLS FOR EFFICIENT RELIABLE MESSAGE TRANSMISSION

A Thesis

by

KEISHLA DESIREE ORTIZ-LOPEZ

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Jennifer L. Welch
Committee Members,	Radu Stoleru
	Erick Moreno-Centeno
Head of Department,	Dilma Da Silva

August 2017

Major Subject: Computer Science

ABSTRACT

In the reliable message transmission problem (RMTP) processors communicate by exchanging messages, but the channel that connects two processors is subject to message loss, duplication, and reordering. Previous work focused on proposing protocols in asynchronous systems, where message size is finite and sequence numbers are bounded. However, if the channel can duplicate messages, lose messages, and arbitrarily reorder the messages, the problem is unsolvable. In this thesis, we consider a strengthening of the asynchronous model in which reordering of messages is bounded. In this model, we develop two efficient protocols to solve the RMTP: (1) when messages may be duplicated but not lost and (2) when messages may be duplicated and lost. This result is in contrast to the impossibility of such an algorithm when reordering is unbounded. Our protocols have the pleasing property that no messages need to be sent from the receiver to the sender.

DEDICATION

To my grandmother for being the strongest woman I ever met ♡.

ACKNOWLEDGMENTS

First of all, I would like to thank my advisor Dr. Jennifer Welch for her patience and guidance during these three years and for introducing me to the area of distributed computing. Every time I was stuck with something related to research or the life in academia in general, she always guided me to the right direction. I also want to thank Dr. Radu Stoleru and Dr. Erick Moreno-Centeno for their feedback and for being part of my thesis committee.

Special thanks to my colleagues Saptaparni Kumar and Edward Talmage for discussion, suggestions, advice and our interesting conversations about food.

Finally, I must express my deep gratitude to my family, friends, and my boyfriend Usman for providing me with unfailing support and ongoing encouragement throughout my years of study. This accomplishment would not have been possible without them.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a thesis committee consisting of Dr. Jennifer Welch and Dr. Radu Stoleru of the Department of Computer Science & Engineering and Dr. Erick Moreno-Centeno of the Department of Industrial & Systems Engineering.

The work conducted for the thesis was completed by the student, under the advisement of Dr. Jennifer L. Welch of the Department of Computer Science & Engineering and part of this work was published in [10].

Funding Sources

Graduate study was partially supported by NSF grant 1526725.

NOMENCLATURE

RMTP	Reliable Message Transmission Problem
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

1. INTRODUCTION*

1.1 Problem

Computer communication networks are typically organized as a series of layers to overcome the engineering complexity involved. Each layer provides an interface to the layer above that hides some of the difficulties of the lower layers. An important function of higher layers such as TCP/UDP and data link layer is to mask faults exhibited by a less reliable lower layer. Having reliable message transmission makes it much easier to design correct distributed applications, but actual networks are subject to loss, duplication, and reordering, especially in mobile networks. We call the problem of implementing a reliable layer on top of an unreliable layer the *reliable message transmission problem* (RMTP).

Any practical solution to this problem should use bounded-length messages because real-world communication networks use fixed size messages. In totally asynchronous models, this problem has been shown to be expensive, if not impossible, to solve when messages can be reordered. Historically, less attention has been paid to stronger timing models. Therefore, this paper considers a strengthening of the asynchrony assumption, specifically, bounded reordering, and shows that a bounded solution is now possible.

1.2 Contribution

We present two protocols: the KJ protocol and Extended KJ protocol that solve the RMTP for two processors connected by a channel that is subject to bounded reordering and duplication, and bounded reordering, bounded loss and duplication respectively. In the model for the KJ protocol, which is inspired by that in [11], there is a parameter δ

which bounds how out of order messages can be delivered. Also in our model, messages can experience unbounded but finite duplication. In our algorithm, the sender A appends to each message a bounded counter that ranges from 0 to $M_c - 1$, where $M_c = 2\delta+1$, and sends each message to receiver B. The challenging part is for B to determine if the message received is a new message, in which case the message is added to a local data structure of B waiting to be delivered. One pleasing aspect of our protocol is that the receiver B does not need to send acknowledgments to the sender A, i.e., it is a non ACK-based protocol. In practice, such protocols are faster than ACK-based ones due to less message and time overhead [17]. We present a correctness proof for the KJ protocol. The model of the Extended KJ protocol also allows bounded reordering and duplication with the addition of bounded loss of messages denoted with the parameter λ. In order to tolerate the loss of messages, the sender A needs to send λ copies of each message since the channel is allow to lose at most $\lambda-1$ consecutive messages. The Extended KJ protocol is proved correct by showing that this protocol can be reduced to the KJ protocol.

1.3 Related Work

The reliable message transmission problem (RMTP) has been extensively studied over the past years. The researchers in this area initially focused their work on asynchronous systems, where message loss, duplication and reordering are considered. The three faults together can be tolerated by using unbounded sequence numbers [12], but the use of unbounded sequence numbers is not desirable in practice. The existence of bounded solutions to the RMTP depends on which combination of faults are to be tolerated. For the case of both loss and duplication, there is a well-known protocol called the Alternating Bit Protocol, introduced in Bartlett et al. [3], that also works when loss and duplication are considered individually. The work by Engelhardt and Moses [6] presents a proof assuming both loss and duplication that shows that single-bit messages are insufficient once

channels potentially deliver duplicate messages in a implementation of a data-link layer; thus the messages must have some additional control information such as headers or tags.

For the case of both loss and reordering, there are solutions given in [1, 7, 8, 13, 14, 16] that consider bounded sequence numbers and present lower and upper bounds for communication and space complexity. However, according to the paper by Afek et al. [1] any protocol tolerating both message loss and reordering either requires unbounded sequence numbers or the sender has to send an unbounded number of messages until receiving an acknowledgment from the receiver. For the case of both reordering and duplication, Wang and Zuck [16] prove that no bounded solution can exist. In contrast to this work, which assumes an asynchronous system subject to arbitrary reordering, we consider a system with bounded reordering (duplication and loss) and present two algorithms that are efficient.

Other work that has considered strengthening of the purely asynchronous model includes [8] and [15]. The work in [8] introduces the notion of a probabilistic channel, in which a message can be delayed with some probability. In contrast with [8], our model allows adversarial reordering of messages within the limits of the bound δ. The authors of [15] consider a model in which there is a real-time upper bound on message delay, within which messages can be reordered; however, messages cannot be lost or duplicated. In this model, they give tight bounds on the time complexity of both ACK-based and non ACK-based protocols. In contrast with [15], our model has no real-time aspect and allows unbounded but finite duplication and bounded message loss.

This thesis is organized as follows: Chapter 2 describes the model and defines the problem in more detail. Chapter 3 presents the KJ protocol with algorithms for the sender and receiver, and an algorithm that models and describes the behavior of the channel and its correctness proof. Chapter 4 introduces the Extended KJ protocol that allows message loss with a reduction proof. Finally, Chapter 5 concludes the thesis with further directions.

2. MODEL AND PROBLEM DEFINITION*

2.1 Model

We consider a system consisting of a sender A, a receiver B, and a unidirectional channel C over which A can send messages to B. All three system components are modeled as automata. Each automaton has a (possibly infinite) set of states, with a subset of initial states. A transition from one state to another state is triggered by the occurrence of an event. The possible events are:

1. SEND(m): A learns about message m which is to be sent to B; m is an element of a set M_H of messages, where the set M_H contains the high-level messages.

2. $send(m)$: A transfers message m to C; m is an element of a set M_L of messages, where the set M_L contains the low-level messages.

3. $receive(m)$: C transfers message m to B; m is an element of the set M_L.

4. RECEIVE(m): B delivers m; m is an element of M_H.

Events at an automaton are partitioned into input and output events. An event e is said to be *enabled* in a state s of an automaton if the automaton has a transition from s labeled with e. SEND is an input event of A, $send$ is an output event of A and an input event of C, $receive$ is an output event of C and an input event of B, and RECEIVE is an output event of B.

A *configuration* $\mathscr{C}$ of the system consists of a state for each of A, B and C; in an initial configuration, they are all initial states.

*2017 IEEE. Part of the material presented in this chapter is reprinted, with permission, from Keishla D. Ortiz-Lopez and Jennifer L. Welch, "Bounded Reordering Allows Efficient Reliable Message Transmission", in *Proceedings 31st IEEE International Parallel & Distributed Processing Symposium (IPDPS)*, May 2017.

An *execution* $\mathscr{E}$ of the system is a sequence $\mathscr{C}_0 e_1 \mathscr{C}_1 e_2 \ldots$ of alternating configurations $\mathscr{C}_i$ and events e_i, beginning with an initial configuration and, if it is finite, ending with a configuration such that:

- For each $i \geq 1$, if e_i is an event of $T \in \{A, B, C\}$, then the state of T in $\mathscr{C}_i$ is the result of T's transition function operating on the state of T in $\mathscr{C}_{i-1}$ and the state of the other components is the same in $\mathscr{C}_i$ as in $\mathscr{C}_{i-1}$.

An execution is *fair* if there is no output event for any component that is continually enabled without occurring. Fairness ensures that if an event is waiting to happen, then eventually it will happen. For example, if A wants to send a message to C, eventually it will be allowed to do so. We will only require correct behavior in fair executions; if an algorithm is prevented from taking its steps, we should not expect it to be able to accomplish its task.

2.2 Problem Definition

We are interested in devising local algorithms for A and B that provide a solution to the RMTP, when the channel C from A to B is subject to bounded out-of-order, loss and duplication. This model is inspired by the *Average Delayed or Dropped* (ADD) system model [11] that deals with message loss and reordering. In our model, the channel may duplicate and lose each message a finite number of times and the degree of reordering is limited by the parameter δ. In more detail, but still informally, the reordering in our model is restricted as follows. Consider the sequence of messages appearing in *send* events in an execution. Suppose m is the i-th in the sequence and m' is the j-th, where $j \geq i+\delta$. Then the first occurrence of *receive*(m') must appear after the last occurrence of *receive*(m). However, messages that are sent at most $\delta-1$ after m are allowed to be received before m is received. For the bounded message loss, denoted by the parameter λ, the channel can lose at most λ-1 consecutive messages sent over it.

We specify the behavior of C with two algorithms (see Algorithm 2 in Figure 3.1 and Algorithm 6 in Figure 4.1). The behavior of C should be consistent with the behavior of these algorithms, depending of the faults to be tolerated, although it does not need to be implemented exactly like one of them. The specification of the first algorithm has the channel keeping messages sent to it from A in a FIFO queue. Each message can be sent to B any number of times. A message can arrive out of order as mentioned before. This is modeled by keeping track of which messages have been sent to B at least once, and any message already received by B that is at most δ from a message being currently received is deleted from the channel C. The intuition behind this is to make sure that a message that is too out of order to be received any more is deleted from the channel. The specification of the second algorithm for C in order to allow message loss is similar to the first one with the following addition: the channel needs to keep track of how many messages are lost in a row. Therefore, if the channel loses $\lambda-1$ messages in a row, then the next message will stay in the channel and then restarts counting lost messages in a row again.

The algorithms for A and B must satisfy the following in every fair execution:

Safety: In any prefix of the execution, the sequence of messages occurring in RECEIVE events is a prefix of the sequence of message occurring in SEND events.

Liveness: The number of RECEIVE events in the execution equals the number of SEND events. Thus, if there are an infinite number of SEND events, then there are an infinite number of RECEIVE events.

We are interested in bounded solutions, which means that if M_H is finite, then M_L is finite. This constraint rules out, for instance, having A tag each message it sends with a counter that grows without bound. However, we use *auxiliary variables* in algorithm for A and B to count the number of messages in M_H sent and received with the purpose to use them in the proofs. Typically, auxiliary variables are used in algorithms to capture properties of execution history. Any operation performed on auxiliary variables must not

affect the values of variables that are part of the actual implementation of the algorithm [9].

3. KJ PROTOCOL*

3.1 Algorithm

The algorithm for the KJ protocol for both the sender A and receiver B, and the behavior of the unreliable channel are shown in Figures 3.1 to 3.3. The main idea of the algorithm is as follows. The sender A accepts a series of messages from its user that are to be sent to the receiver B. As each message arrives, it is assigned a counter from a bounded set of integers that wraps around and is stored in an array waiting for its turn to be sent over the channel. The channel delivers each message sent on it to the receiver B at least once and in roughly the correct order, where the reordering is bounded by δ. B keeps two arrays of messages, one holding those that are waiting to be delivered to the user and another holding the most recent δ messages that have already been delivered. Whenever B receives a message from the channel, it checks whether the bounded counter of the message occurs among the most recent δ messages in the concatenation of the two arrays. If so, then it is a duplicate and is ignored. If not, then this is a new message and it is placed in the proper location in the array of pending messages. The proper location is determined by comparing the bounded counter of the new message with those of the messages already in the array, where the comparison takes into account the fact that the counters are from a set that wraps around.

For purposes of analysis of the algorithm, the algorithm description piggybacks an unbounded counter on the messages as an auxiliary variable [9], but the algorithm never takes an action that depends on the value of an auxiliary variable.

*2017 IEEE. Reprinted, with permission, from Keishla D. Ortiz-Lopez and Jennifer L. Welch, "Bounded Reordering Allows Efficient Reliable Message Transmission", in *Proceedings 31st IEEE International Parallel & Distributed Processing Symposium (IPDPS)*, May 2017.

Algorithm 1: Algorithm for sender A.

1: $counter_A$, an integer, initially 0
2: $send\_pending$, a FIFO queue, initially empty
3: Aux_A, an integer, initially 0
input:
4: **event** SEND(m)
5: **effects:**
6: $send\_pending$.enq($(m, counter_A, Aux_A)$)
7: $counter_A := (counter_A + 1) \bmod M_c$
8: $Aux_A := Aux_A + 1$
9: **end event**
output:
10: **event** $send(m, c, a)$
11: **preconditions:**
12: (m, c, a) is at the head of $send\_pending$
13: **effects:**
14: remove (m, c, a) from $send\_pending$
15: **end event**

Algorithm 2: Algorithm to model behavior of C.

1: $in\_transit$, array, initially empty
input:
2: **event** $send(m, c, a)$
3: **effects:**
4: insert $(m, c, a, False)$ at the end of $in\_transit$
5: **end event**
output:
6: **event** $receive(m, c, a)$
7: **preconditions:**
8: $(m, c, a, y) \in in\_transit$ at some index ℓ for some y
9: for all $(m', c', a', y') \in in\_transit$ with index $\leq \ell - \delta$, $y' = True$
10: **effects:**
11: remove from $in\_transit$ all entries with index $\leq \ell - \delta$
12: $y := True$
13: **end event**

Figure 3.1: KJ protocol: Algorithm for the sender A and the behavior of the channel C [10].

In more detail, the algorithm for A consists of three local variables: $counter_A$, a counter bounded by M_c, where $M_c = 2\delta+1$, $send\_pending$, a FIFO queue used to add the sequence of messages from M_H that need to be sent in order, and the auxiliary counter Aux_A, an unbounded counter. The algorithm that describes the behavior of C consists only of one local variable $in\_transit$, an array used to hold the messages from the set M_L sent by A. The algorithm for B consists of four variables: $counter_B$, a counter bounded by M_c used to determine which message needs to be delivered next to the user, $receive\_pending$ contains all the messages from M_L received that need to be delivered, $delivered$ keeps track of the messages delivered to the user and size is bounded by δ, and Aux_B an unbounded auxiliary counter.

The algorithm for the sender A (Algorithm 1 of Figure 3.1) consists of two events: the input event SEND(m) and output event $send(m, c, a)$. The input event SEND(m) (lines 4-9) takes the message m from the set M_H and simply adds the message to the local FIFO queue $send\_pending$ with the local bounded and auxiliary counters values $counter_A$ and Aux_A respectively. The output event $send(m, c, a)$ (lines 10-15) takes the message that is at the head of $send\_pending$ (if any) and sends the message to the channel and (m, c, a) is removed from $send\_pending$. The message (m, c, a) should be removed from $send\_pending$ to allow A to send the next message in $send\_pending$. Therefore, the task of the sender A is to send each message once with the addition of the bounded counter and auxiliary counter of that particular message.

Algorithm 2 of Figure 3.1 describes the behavior of the unreliable channel C that connects the processors A and B. The channel C consists of two events: $send(m, c, a)$ and $receive(m, c, a)$. The input event is $send(m, c, a)$ (lines 2-5), which simply adds $(m, c, a, False)$ in $in\_transit$, where the fourth component is only used by C to check whether the message was received at least once by B. The output event $receive(m, c, a)$ (lines 6-13) decides what message should be delivered to B, where (m, c, a) cannot be

more than δ apart from the most recent entry in $in\_transit$ (call it $(m', *, *, y')$) that was received at least once by B (i.e. fourth component y' is equal to $True$). In addition, all of the messages before $(m', *, *, y')$ (if any) were also received by B at least once (see precondition in line 9). Since the value of c is bounded by M_c, B could receive a different message with counter value c and later on receive the same (m, c, a) again. In order to avoid this confusion to B, the channel removes all of those messages mentioned above (line 11), thus B never receives a message that is too out-of-order to be received again. Note that if (m, c, a, y) is one of the oldest δ entries in $in\_transit$ then no message is removed from the channel.

The algorithm for A and the description of the channel are simple. However, the algorithm for the receiver B is more complex because it has to make sure that the message received is not a duplicate. The algorithm for B consists of two events: $receive(m, c, a)$ and RECEIVE(m). The input event is $receive(m, c, a)$ (lines 5-16 of Algorithm 3 of Figure 3.2) that checks if there is a message with counter value c (call it (m', c, a')) in $total\_msgs$, which is a concatenation of $delivered$ and $receive\_pending$. B decides whether procedure ADDTORECEIVEPENDING should be invoked in order to add the message (m, c, a). If (m', c, a') is (m, c, a) itself, then there should be at most $\delta-1$ entries following (m', c, a') in $total\_msgs$. Recall that when a message (m', c, a') is received for the first time all the messages that are at least δ apart from (m', c, a') in $in\_transit$ are removed, thus (m', c, a') will be within the oldest δ entries in $in\_transit$. Therefore, when another copy of (m', c, a') (i.e. (m, c, a)) is received, then α, the variable used to determine if (m, c, a) should be added to $receive\_pending$, is set to at most $\delta-1$ which causes that condition in line 13 to evaluate to false and the message (m, c, a) is not added to $receive\_pending$. If (m', c, a') is not a duplicate of (m, c, a) or there is no such entry in $total\_msgs$, then α is set to at least δ (lines 9 and 11) and the message (m, c, a) is added to $receive\_pending$.

Algorithm 4 of Figure 3.3 is the most important one because it ensures that the message

Algorithm 3: Algorithm for receiver B.

1: $counter_B$, an integer, initially 0
2: $receive\_pending$, array, initially empty
3: $delivered$, a FIFO queue, initially empty
4: Aux_B, an integer, initially 0
input:
5: **event** $receive(m, c, a)$
6: **effects:**
7: $total\_msgs := delivered.receive\_pending$
8: **if** $(*, c, *) \notin total\_msgs$ **then**
9: $\alpha := \delta$
10: **else**
11: $\alpha :=$ number of entries after most recent occurrence of $(*, c, *)$ in $total\_msgs$
12: **end if**
13: **if** $\alpha \geq \delta$ **then**
14: ADDTORECEIVEPENDING(m, c, a)
15: **end if**
16: **end event**
output:
17: **event** RECEIVE(m)
18: **preconditions:**
19: (m, c, a) is at the head of $receive\_pending \wedge c = counter_B$
20: **effects:**
21: remove (m, c, a) from $receive\_pending$
22: $counter_B :=$ $(counter_B + 1) \mod M_c$
23: $Aux_B := Aux_B + 1$
24: $delivered$.enq((m, c, a))
25: **if** $\mid delivered \mid = \delta+1$ **then**
26: $delivered$.deq()
27: **end if**
28: **end event**

Figure 3.2: KJ protocol: Algorithm for the receiver B [10].

Algorithm 4: Procedure used by receiver B in the $receive$ event.

```
1:  procedure ADDTORECEIVEPENDING(m, c, a)
2:      if |receive_pending| = 0 then
3:          insert (m, c, a) in receive_pending
4:          return
5:      end if
6:      current := bounded counter of the most recent message in receive_pending
7:      if MLT(current, c) then
8:          insert (m, c, a) at the end of receive_pending
9:          return
10:     end if
11:     for i := |receive_pending| − 1 down to 1 do
12:         current := bounded counter of the message at index i in receive_pending
13:         next := bounded counter of the message at index i − 1 in receive_pending
14:         if MLT(next, c) ∧ MLT(c, current) then
15:             insert (m, c, a) in receive_pending between the messages at in-
                dices i − 1 and i
16:             return
17:         end if
18:     end for
19:     next := bounded counter of the message at index 0 in receive_pending
20:     if MLT(c, next) then
21:         insert (m, c, a) in receive_pending at index 0
22:         return
23:     end if
24: end procedure
25:
26: procedure MLT(c_1, c_2)
27:     if c_1 < c_2 then
28:         return c_2 − c_1 ≤ δ
29:     end if
30:     return c_1 − c_2 > δ
31: end procedure
```

Figure 3.3: KJ protocol: Algorithm for the procedure used by the receiver B [10].

(m, c, a) is added to $receive\_pending$ in the correct order. The algorithm consists of two procedures: ADDTORECEIVEPENDING (lines 1-24) and MLT (lines 26-31). The procedure ADDTORECEIVEPENDING is the main one that adds the message to $receive\_pending$. The main procedure has four conditions (lines 2, 7, 14, and 20). When one of them is evaluated to true, the message (m, c, a) is inserted in $receive\_pending$ once in a certain order (depending on the condition). When $receive\_pending$ is empty, then (m, c, a) is the only message in $receive\_pending$ (line 3). However, when there is at least one message in $receive\_pending$, the help of procedure MLT is needed. MLT takes as parameters two bounded counter values and returns true if and only if the first parameter is less than the second one, taking into account the wrap-around. Since the counters are bounded by M_c, where $M_c = 2\delta+1$, and by the behavior of the channel C, B will never receive an earlier message that is more than δ apart from the most recent message in $receive\_pending$. In fact, the loop in line 11 is never executed more than δ times. Therefore, if the most recent message in $receive\_pending$ has an auxiliary counter a' and bounded counter c', where $a' < a$, then either $c' < c$ and the difference between them is at most δ or $c' > c$ and the difference between them is greater than δ (there is a wrap-around exactly at c or before c). In such case, the condition in line 7 evaluates to true (i.e. MLT returns true) and (m, c, a) is inserted at the end of $receive\_pending$ (line 8). The other case is when $a < a'$, then either $c < c'$ and the difference between them is at most δ or $c > c'$ and the difference between them is greater than δ. Thus either condition in line 14 is true or the condition in line 20 is true. Note that if condition in line 20 is true, then (m, c, a) is inserted at index 0 in $receive\_pending$ (line 21), which implies that there were less than δ entries in $receive\_pending$. If condition in line 14 is true, then the first part of the condition evaluates to true as well. In such case, there is another message (m'', c'', a''), where $a'' < a < a'$, such that either $c'' < c$ and $c - c'' \leq \delta$ or $c'' > c$ and the difference between them is greater than δ. Thus, (m, c, a) is inserted between those two entries in

$receive\_pending$ (line 15). In sum, the procedure ADDTORECEIVEPENDING inserts a message (m, c, a) exactly once and in the correct order by comparing the bounded counter c with at most δ of the bounded counters of the most recent messages in $receive\_pending$.

The output event RECEIVE(m) (lines 17-28 in Algorithm 3) delivers to the user the message m that is at the head of $receive\_pending$. The message is inserted into the queue $delivered$, whose size is bounded by δ. If the size of $delivered$ is equal to $\delta+1$, then the oldest message is removed from $delivered$. Thus, the event RECEIVE(m) delivers each message in the same order that it was sent by checking the current value of $counter_B$. Note that this event is not enabled until there is a message at the head of $receive\_pending$ with counter c equal to $counter_B$.

3.2 Correctness Proof

The correctness of the KJ protocol is proved by showing that it satisfies the safety and liveness properties defined in Chapter 2. The proof is organized as follows: Lemmas 1 and 2 state some basic properties relating the bounded and auxiliary counters of messages; their proofs are simple inductive arguments and are omitted. Lemma 3 states the main invariants on how messages flow from the sender to the receiver; it is proved by induction. Lemma 4 shows that the protocol satisfies the safety property, whereas Lemmas 5 to 8 show that the protocol satisfies the liveness property.

Let e be any fair execution.

Lemma 1. *In every configuration of* e*, (a)* Aux_A *is the number of* SEND *events that have occurred so far in the execution, (b)* $counter_A = Aux_A \mod M_c$*, (c)* Aux_B *is the number of* RECEIVE *events that have occurred so far in the execution, and (d)* $counter_B = Aux_B \mod M_c$.

Proof. Initially $Aux_A = 0 = counter_A$. The only change to Aux_A is when SEND occurs, and the change is to increment it by 1. The only change to $counter_A$ is when SEND occurs,

and the change is to increment it by $1 \mod M_c$.

Similarly, initially $Aux_B = 0 = counter_B$. The only change to Aux_B is when RECEIVE occurs, and the change is to increment it by 1. The only change to $counter_B$ is when RECEIVE occurs, and the change is to increment it by $1 \mod M_c$. □

Lemma 2. *In every configuration of e, if (m, c, a) or (m, c, a, y) is in $send\_pending$, $in\_transit$, $receive\_pending$, or $delivered$, then (a) $c = a \mod M_c$, and (b) m is the argument of the $(a + 1)$-st* SEND *event that has occurred.*

Proof. By induction on the configurations. In the initial configuration, $send\_pending$, $in\_transit$, $receive\_pending$ and $delivered$ are all empty, thus the lemma is vacuously true.

Suppose the lemma is true in configuration $\mathscr{C}$; we show it is still true in configuration $\mathscr{C}$'. We consider the possibilities for the i-th event.

- **Case 1:** The event is SEND(m).

  (a) By the code, (m, c, a) is put in $send\_pending$, with $c = (counter'_A - 1) \mod M_c$, and $a = Aux'_A - 1$, where $counter'_A = (counter_A + 1) \mod M_c$ and $Aux'_A = Aux_A + 1$ in $\mathscr{C}$'. By Lemma 1(b), $counter'_A = Aux'_A \mod M_c$.

  (b) By Lemma 1(a), the number of SEND events that have occurred before this event, is the value of Aux_A in configuration $\mathscr{C}$; call this value a. This is the value assigned to m, which is the argument of the $(a + 1)$-st SEND event.

- **Case 2:** The event is $send(m, c, a)$. The precondition is that (m, c, a) is at the head of $send\_pending$ in $\mathscr{C}$. By the inductive hypothesis, (a) $c = a \mod M_c$ and (b) m is the argument of the $(a+1)$-st SEND event. By the code, (m, c, a) is removed from $send\_pending$ and added to $in\_transit$. Thus (a) and (b) are true in $\mathscr{C}$'.

- **Case 3:** The event is $receive(m, c, a)$. The preconditions are that (m, c, a, y) is in $in\_transit$ at some index ℓ for some y and that all entries in $in\_transit$ with index $\leq \ell - \delta$, $y' = True$ in $\mathscr{C}$. By the inductive hypothesis, (a) $c = a \mod M_c$ and (b) m is the argument of the $(a+1)$-st SEND event. By the code, all entries with index $\ell - \delta$ are removed from $in\_transit$ and y is set to $True$. In addition, if (m, c, a) is a new message is put into the array $receive\_pending$, otherwise is rejected. Thus (a) and (b) are true in $\mathscr{C}$'.

- **Case 4:** The event is RECEIVE(m). The precondition is that (m, c, a) is at the head of $receive\_pending$ and $c = counter_B$ in $\mathscr{C}$. By the inductive hypothesis, (a) $c = a \mod M_c$ and (b) m is the argument of the $(a+1)$-st SEND event. By the code, (m, c, a) is removed from $receive\_pending$, $counter_B$ and Aux_B are updated to $counter'_B = (counter_B + 1) \mod M_c$ and $Aux'_B = Aux_B + 1$ respectively, and (m, c, a) is put into the $delivered$ queue. If the size of the queue is equal to $\delta+1$, then the oldest message is deleted. Thus (a) and (b) are true in $\mathscr{C}$'.

□

Lemma 3. *In every configuration of* e *there exists integers* u*,* v*, and* w *such that,*

(a) the sequence of auxiliary counters in the FIFO queue $send\_pending$ *at* A *is:* u*,* $u+1$*,...,*$Aux_A - 1$*, where* $u \leq Aux_A$*; (*$u = Aux_A$ *means* $send\_pending$ *is empty)*

(b) the sequence of auxiliary counters in the array $in\_transit$ *in the channel is:* v*,* $v+1$*,...,*$u-2, u-1$*, where* $v \leq u$*; (*$v = u$ *means* $in\_transit$ *is empty)*

(c) If $(m, c, a, True)$ *is in* $in\_transit$*, then* $a - v < \delta$*;*

(d) Suppose (m, c, a, y) *is in* $in\_transit$*. Then* $y = True$ *if and only if* (m, c, a) *is in* $total\_msgs$*;*

(e) the sequence of auxiliary counters in the array $receive\_pending$ *at* B *is:* $Aux_B, Aux_B+1,...,v-1, \mu$*, where* μ *is a (possibly empty) subsequence of* $v, v+1,...,v+\delta-1$ *(*$Aux_B \geq v$ *means the part of the sequence before* μ *is empty);*

(f) the sequence of auxiliary counters in the $delivered$ ***FIFO queue*** *at* B *is:* w, $w+1,...,Aux_B-2, Aux_B-1$*, where* $w = max\{0, Aux_B-\delta\}$*. (*$Aux_B = 0$ ***means*** $delivered$ ***is empty)***

Proof. By induction on the configurations. In the initial configuration, we have that $counter_A$, $counter_B$, Aux_A and Aux_B are set to 0, and $send\_pending$, $receive\_pending$, $in\_transit$, $total\_msgs$, and $delivered$ are set to $\emptyset$. Thus, properties (a) through (f) are true in the initial configuration.

For the inductive step, assume in configuration $\mathscr{C}$ these properties are true. Then, we need to consider all the possibilities for the event that leads to the configuration $\mathscr{C}$'.

- **Case 1:** The event SEND(m) occurs, i.e. lines 4-9 in Algorithm 1 are executed. In configuration $\mathscr{C}$', the message m is put into the $send\_pending$ FIFO queue with $counter_A$ and Aux_A, then $counter'_A = (counter_A + 1) \mod M_c$ and $Aux'_A = Aux_A + 1$. Property (a) is satisfied because after the message $(m, counter_A, Aux_A)$ is put into the $send\_pending$ FIFO queue, the counter $counter_A$ and auxiliary counter Aux_A are updated as mentioned previously to $counter'_A$ and Aux'_A respectively. Therefore, now the message m has an auxiliary counter a equal to $Aux'_A - 1$, which is at the end of $send\_pending$. Since properties (b)-(f) are true in $\mathscr{C}$ by the inductive hypothesis, they are still true in $\mathscr{C}$'.

- **Case 2:** The event $send(m, c, a)$ occurs, i.e. lines 10-15 in Algorithm 1 and lines 2-5 in Algorithm 2 are executed. In configuration $\mathscr{C}$', this event is enabled by its precondition that states that a message (m, c, a) is at the head of $send\_pending$,

thus the message is removed from the FIFO queue and inserted into the channel's array $in\_transit$ with a boolean value set to $False$.

Property (a) is satisfied because the message (m, c, a) is at the head of $send\_pending$ in $\mathscr{C}$ with $a = u$. Note that this implies that in configuration $\mathscr{C}$', $u' = u + 1$ (i.e. the next auxiliary counter in $send\_pending$), $a = u' - 1$ and (m, c, a, y) is inserted at the end of the channel's array $in\_transit$ with y set to $False$. Therefore, property (b) is also satisfied. Since $(m, c, a, False)$ is inserted in $in\_transit$ in $\mathscr{C}$' with $a = u' - 1$, note that this does not affect properties (c)-(e) at all because $y = False$, thus either $a - v < \delta$ or $a - v' \geq \delta$ and (m, c, a) is not in $total\_msgs$ in $\mathscr{C}$' (i.e. is not added to $receive\_pending$ or $delivered$).

Property (f) is true in $\mathscr{C}$ by the inductive hypothesis, so it is true in $\mathscr{C}$'.

- **Case 3:** A $receive(m, c, a)$ event occurs, i.e. lines 6-14 in Algorithm 2, and lines 5-16 in Algorithm 3 are executed and Algorithm 4 is executed. Preconditions are that (m, c, a, y) is in $in\_transit$ at some index ℓ, and all entries in $in\_transit$ with index at most $\ell - \delta$ have last component $True$. The effect on $in\_transit$ is that all entries in $in\_transit$ with index at most $\ell - \delta$ are removed and the last component of the entry for $(m, c, a, *)$ is set to $True$. The effect on $receive\_pending$ is that (m, c, a) is inserted at a certain place if certain conditions are true.

  By the inductive hypothesis for (b), in $\mathscr{C}$, the array $in\_transit$ has auxiliary counters $v, v + 1, ..., u - 1$. Since the change to $in\_transit$ is to remove zero or at most δ entries from the older end, the condition is still true in $\mathscr{C}$', with v' equal to something between v and $v+\delta$, and u' equal to something between v and u.

  In order to prove property (c), we need to show that after (m, c, a) is received by B its distance to v' is less than δ in $\mathscr{C}$'. We need to consider two cases: 1) $\ell - \delta < 0$ and 2) $\ell - \delta \geq 0$. In case 1) no message is removed from $in\_transit$ in $\mathscr{C}$', which

implies that $v' \leq a \leq v'+\delta-1$, where $v' = v$. Note that if either y is $False$ or $True$ in $\mathscr{C}$, the message (m, c, a, y) is within δ from the old end of $in\_transit$ in $\mathscr{C}$' as well (i.e. $a - v' < \delta$), thus property (c) is satisfied. In case 2) all of the messages with indices of at most $\ell-\delta$ should be removed from $in\_transit$ in $\mathscr{C}$' because they were already received by B. Therefore in $\mathscr{C}$, $v+\delta \leq a \leq v + 2\delta-1$. Due to the fact that just messages with indices at most $\ell-\delta$ are removed from $in\_transit$, then $v' = (a-\delta) + 1$ in $\mathscr{C}$'. To prove this, first assume that $a = v+\delta$ in $\mathscr{C}$, then v' should be at least $v + 1$ in $\mathscr{C}$' because only one message is removed, thus $v' = (a-\delta) + 1 = ((v+\delta-\delta)) + 1 = v + 1$. Now, assume that $a = v + 2\delta-1$ in $\mathscr{C}$, implying that δ messages are removed from $in\_transit$. Then v' is at most $v+\delta$ in $\mathscr{C}$', so $v' = (a-\delta) + 1 = ((v + 2\delta-1)-\delta) + 1 = v+\delta$. Thus, it follows that $v' \leq a \leq v'+\delta-1$, where $v + 1 \leq v' \leq v+\delta$. Since in $\mathscr{C}$', the value of v changes to v', we have that $in\_transit$ consists of $v', v' + 1, ..., u' - 2, u' - 1$ and y is set to $True$. In this scenario, the only possible value of y in $\mathscr{C}$ is $False$, so at least one message is removed (i.e. $a = v+\delta$) and at most δ messages are removed (i.e. $a = v + 2\delta-1$) from $in\_transit$. Thus $a - v' = (v+\delta) - (v + 1) = \delta-1 < \delta$ and $a - v' = (v + 2\delta-1) - (v+\delta) = \delta-1 < \delta$ and property (c) is satisfied.

To prove property (d) for the case when $y = False$ in $\mathscr{C}$, we need to prove that (m, c, a) is put in $receive\_pending$. Note that procedure ADDTORECEIVEPENDING is only invoked in line 13 of Algorithm 3 if the value of α is set to at least δ. The easiest case is when there is no occurrence of $(*, c, *)$ in $total\_msgs$, so α is set to δ and the message (m, c, a) is added to $receive\_pending$. The remaining case is when there is a occurrence of $(*, c, *)$ in $total\_msgs$ and α is set to the number of entries after most recent occurrence of such message in $total\_msgs$. We need to prove that in fact α is set to at least δ, such that (m, c, a) is put in $receive\_pending$.

Assume by contradiction that there are less than δ entries after the most recent occurrence of $(*, c, *)$, call it (m', c, a'), in $total\_msgs$ and (m, c, a) is not added to $receive\_pending$ in $\mathscr{C}$'. Furthermore, let this be the first $receive$ with this property. By the precondition of $receive(m, c, a)$ all the entries in $in\_transit$ with indices at most $\ell - \delta$ have last component $True$. By Lemma 5, there is a previous $receive$ event for all those entries, and by the assumption that $receive(m, c, a)$ is the first receive whose message is not added to $receive\_pending$, all those entries have already been added to $receive\_pending$. Since those messages are also deleted from $in\_transit$, then by property (c) the distance between $a - v'$ is less than δ. Therefore, the number of auxiliary counters between a and a' is less than 2δ, less than δ in $in\_transit$ and less than δ in $total\_msgs$). By Lemma 2, $c = a \mod M_c$ and $c = a' \mod M_c$, which implies that a and a' leave the same remainder when divided by M_c. Since $M_c = 2\delta + 1$, then the difference between a and a' is M_c, which is greater than 2δ. Therefore, it is impossible that there are less than δ entries following (m', c, a') in $total\_msgs$, thus α is set to at least δ and (m, c, a) is added to $receive\_pending$.

The changes that can affect the validity of property (e) are the possible removal of messages from $in\_transit$ and the possible insertion of (m, c, a) in $receive\_pending$. Suppose v' is the oldest auxiliary counter in $in\_transit$ in $\mathscr{C}$', after the removal of messages. If a message is removed from $in\_transit$, then it has last component $True$ in $\mathscr{C}$. By inductive hypothesis for property (d), for every removed message there is a corresponding entry in $total\_msgs$ in $\mathscr{C}$, including one for $v' - 1$. Those messages remain in $total\_msgs$ in $\mathscr{C}$', so there is no gap in auxiliary counters between $receive\_pending$ and $in\_transit$. Now let's show that if (m, c, a) is inserted into $receive\_pending$, it goes in the correct order of the auxiliary counters. Since (m, c, a) is inserted, it means that, in $\mathscr{C}$, either there is no occurrence of c

in $total\_msgs$, or there are at least δ entries in $total\_msgs$ following the most recent occurrence of c. By the precondition, in $\mathscr{C}$ all entries in $in\_transit$ that are at least δ older than (m, c, a, y) have last component $True$. By inductive hypothesis for property (d), in $\mathscr{C}$ there is an entry in $total\_msgs$ for each one of these messages. By inductive hypothesis for properties (e) and (f), in $\mathscr{C}$ these messages appear in $total\_msgs$ in sorted order of auxiliary counters. By Lemma 2(a), in $\mathscr{C}$ the corresponding bounded counters of these messages appear in $total\_msgs$ in sorted order (subject to wrap-around). Lines 7-24 in Algorithm 4 insert (m, c, a) into $receive\_pending$ in correct sorted order with respect to the bounded counters. Note that the procedure MLT(c_1, c_2) returns whether or not bounded counter c_1 is less than bounded c_2, when considering the wrap around. Thus, those facts imply that (m, c, a) is in proper sorted order with respect to the auxiliary counters and property (e) is true in $\mathscr{C}$'.

By inductive hypothesis, properties (a) and (f) are true in $\mathscr{C}$, then they are still true in $\mathscr{C}$'.

- **Case 4:** A RECEIVE(m) event occurs, i.e. lines 17-28 in Algorithm 3 are executed. The preconditions are that (m, c, a) is at the head of $receive\_pending$ and $c = counter_B$. The message (m, c, a) is moved from $receive\_pending$ to $delivered$, $counter_B$ and Aux_B are updated to $counter'_B = (counter_B + 1) \mod M_c$ and $Aux'_B = Aux_B + 1$ respectively. This event may cause the drop of the oldest element in $delivered$ with auxiliary counter w if the size of $delivered$ becomes $\delta+1$ in $\mathscr{C}$'. These changes do not affect properties (a)-(c).

  We can prove that property (d) remains true in $\mathscr{C}$' even if the oldest message is removed from $delivered$ because there is no corresponding entry in $in\_transit$. Suppose in contradiction there is a corresponding entry in $in\_transit$ for the oldest

entry in $delivered$ with auxiliary counter w'. Since auxiliary counters in $in\_transit$ are consecutive, there are entries in $in\_transit$ for all the δ elements of $delivered$ as well as for the message being transferred from $receive\_pending$ to $delivered$ (i.e. $\delta+1$ messages). By the inductive hypothesis for property (d), all these entries have last component $True$. However, this contradicts the inductive hypothesis for property (c), which states there are at most $\delta-1$ entries in $in\_transit$ following the oldest one with last component $True$.

By inductive hypothesis for property (e), in $\mathscr{C}$ the auxiliary counters in $receive\_pending$ are $b, b+1, ..., v-1, \mu$, where b is value of Aux_B in $\mathscr{C}$, and μ is a (possibly empty) subsequence of $v, v+1, ..., v+\delta-1$, where v is the smallest auxiliary counter in $in\_transit$ in $\mathscr{C}$. Since the changes include removing the head of $receive\_pending$, in $\mathscr{C}$' the auxiliary counters in $receive\_pending$ are $b+1, ..., v-1, \mu$. Since the changes include incrementing Aux_B by one, in $\mathscr{C}$' the value of Aux_B is $b+1$. The only other changes (incrementing $counter_B$ and changes to $delivered$) do not affect the validity of property (e). Thus property (e) is true in $\mathscr{C}$'.

Property (f) is satisfied in $\mathscr{C}$' because $c = (counter'_B - 1) \mod M_c$ and $a = Aux'_B - 1$ and (m, c, a) is inserted at the end of $delivered$. As mentioned previously, if the size of $delivered$ is $\delta+1$ in $\mathscr{C}$', then only the oldest message with auxiliary counter w is removed from the FIFO queue, thus $|delivered| = \delta$ at the end of $\mathscr{C}$'. Since $w' = w + 1$, then $w' = Aux'_B - \delta$ if $|delivered| = \delta$, otherwise $w' = 0$ if $|delivered| < \delta$ in $\mathscr{C}$' because the message with auxiliary counter w' is not removed from $delivered$.

□

Lemma 4. *In every configuration of e, the sequence of messages in the* RECEIVE *events that have occurred so far is a prefix of the sequence of messages in the* SEND *events that*

have occurred so far.

Proof. Let m be the argument of the i-th RECEIVE event and show m is the argument of the i-th SEND event. The precondition for the i-th RECEIVE event is that (m, c, a) is at the head of $receive\_pending$ and $c = counter_B$.

By Lemma 1(d), $counter_B = Aux_B \mod M_c$ and by the precondition $c = counter_B$. In addition, by Lemma 2, $c = a \mod M_c$ and m is the argument of the $(a+1)$-st SEND event.

Thus $a \mod M_c = c = counter_B = Aux_B \mod M_c$, i.e., $a \mod M_c = Aux_B \mod M_c$.

We have to show that $a = Aux_B$. Suppose in contradiction $a \neq Aux_B$. By Lemma 3(e) (that auxiliary counters in $receive\_pending$ are increasing from Aux_B) it must be that $a > Aux_B$, and in fact $a \geq Aux_B + M_c$. Since a is at the head of $receive\_pending$, it follows that $receive\_pending$ is missing entries with auxiliary counters between Aux_B and $a - 1$.

Lemma 3(e) states that $receive\_pending$ has this sequence of auxiliary counters: (recall that v is the smallest (oldest) auxiliary counter in $in\_transit$) $Aux_B, Aux_B+1, ..., v-1$, μ, where μ is a (possibly empty) subsequence of $v, v+1, ..., v+\delta-1$. If $Aux_B \geq v$, this means this part of the sequence is empty.

Thus it must be that $Aux_B \geq v$ (otherwise Aux_B, ... would be in $receive\_pending$). Thus $a \geq Aux_B + M_c$ which implies $a \geq v + M_c$, i.e., $a - v \geq M_c$.

Since $a > v$ and v is the oldest entry in $in\_transit$, it cannot be that the entry for $(m, c, a, *)$ has already been removed from $in\_transit$. Thus (m, c, a, y) is in $in\_transit$. By Lemma 3(d), $y = True$ and by Lemma 3(c), $a - v < \delta$. However, this contradicts the fact above that $a - v \geq M_c$, since $M_c > \delta$.

By Lemma 1(c), Aux_B is the number of RECEIVE events so far, therefore $Aux_B =$

$i-1$. Since $a = Aux_B$, it is also true that $a = i-1$. Therefore, m is the argument of the $(a+1)$-st = i-th SEND event. □

Lemma 5. *Let $\mathscr{C}$ be any configuration of e.*

(a) Suppose (m, c, a, y) is in $in\_transit$ in $\mathscr{C}$. Then, $y = True$ if and only if there is a previous $receive(m, c, a)$ event in e.

(b) Suppose there is no entry with auxiliary counter a in $in\_transit$ in $\mathscr{C}$. Then there exists a previous $receive(m, c, a)$ in e if and only if $a < v$, where v is the smallest auxiliary counter in $in\_transit$ in $\mathscr{C}$.

Proof. By induction on the configurations. In the initial configuration, we have that $in\_transit$ is empty and thus the lemma follows. For the inductive step, assume in configuration $\mathscr{C}$ the lemma is true, where property (a) is affected by events $send$ and $receive$ and property (b) is only affected by the $receive$ event. Then, we need to consider all the possibilities for the event leading to the next configuration $\mathscr{C}$' that can affect both properties.

- **Case 1:** A $send(m, c, a)$ event occurs, lines 10-15 in Algorithm 1 and lines 1-5 of Algorithm 2 are executed. This event affects property (a), so we need to show that (a) still holds in $\mathscr{C}$', as the validity of property (b) is unaffected.

  The effect of the send event is that $(m, c, a, False)$ is put in $in\_transit$ in $\mathscr{C}$'. We must show that there is no previous (with respect to $\mathscr{C}$') $receive(m, c, a)$ event in e. By Lemma 3, the auxiliary counters in $send\_pending$ and $in\_transit$ are consecutive and increasing in both $\mathscr{C}$ and $\mathscr{C}$'. Thus a cannot be less than v, and so by inductive hypothesis for property (b), there is no previous $receive(m, c, a)$ event in e with respect to $\mathscr{C}$. Thus the same is true for $\mathscr{C}$'.

- **Case 2:** A $receive(m, c, a)$ event occurs, lines 6-13 of Algorithm 2 and lines 5-16

of Algorithm 3 are executed. Properties (a) and (b) are affected by this event, thus we need to show that both properties hold in $\mathscr{C}$'.

The effect of the receive event is that the last component of the entry for (m, c, a) in $in\_transit$ is set to $True$. Since obviously a $receive(m, c, a)$ just occurred, property (a) is true in $\mathscr{C}$'.

Now we show that property (b) continues to be true in $\mathscr{C}$'. We must show that for every entry that is removed from $in\_transit$, call it $(m', c', a', True)$, there is a previous $receive(m', c', a')$ and $a' < v'$, where v' is the smallest auxiliary counter in $in\_transit$ in $\mathscr{C}$'. Inductive hypothesis for property (a) implies that, since $(m', c', a', True)$ is in $in\_transit$, there is a previous $receive(m', c', a')$ event. Thus with respect to $\mathscr{C}$' also, there is a previous $receive(m', c', a')$ event. By Lemma 3, auxiliary counters in $in\_transit$ are consecutive and increasing, thus removing one or more from the smaller end of $in\_transit$ does not affect the validity of a' being less than v'.

□

Lemma 6. *For every $send(m, c, a)$ event in e, there is at least one subsequent $receive(m, c, a)$ event.*

Proof. Suppose in contradiction that not every send event has a subsequent receive event with the same arguments. Let $send(m, c, a)$ be the send event with the smallest value of a without a matching receive event.

When $send(m, c, a)$ occurs, $(m, c, a, False)$ is put in $in\_transit$. By the code, as long as the fourth component is $False$, this entry stays in $in\_transit$. Lemma 5 implies that the fourth component for (m, c, a) stays $False$ throughout the execution.

By the choice of (m, c, a), all entries in $in\_transit$ with smaller values of the auxiliary

counter have a matching receive. Thus by Lemma 5 all those entries eventually have last component equal to $True$.

Thus eventually $receive(m, c, a)$ becomes enabled and stays enabled forever, without ever occurring. This contradicts the assumed fairness of the execution. □

Lemma 7. *When $receive(m, c, a)$ occurs for the first time, (m, c, a) is added to $receive\_pending$.*

Proof. Let $\mathscr{C}$ be the configuration just before the first occurrence of $receive(m, c, a)$ and $\mathscr{C}$' be the following configuration. By Lemma 5, the entry in $in\_transit$ in $\mathscr{C}$ is $(m, c, a, False)$. Thus by Lemma 3(d), there is no (m, c, a) in $total\_msgs$ in $\mathscr{C}$. By the code, in $\mathscr{C}$' the entry in $in\_transit$ becomes $(m, c, a, True)$. Therefore, by Lemma 3(d), there is an entry for (m, c, a) in $total\_msgs$ in $\mathscr{C}$'. By the code, the only way an entry can be added to $total\_msgs$ is for it to be added to $receive\_pending$. □

Lemma 8. *If there are at least i* SEND *events, then there are at least i* RECEIVE *events, for all $i \geq 1$.*

Proof. Suppose in contradiction there exists an i such that there are at least i SEND events, but only $i - 1$ RECEIVE events.

Let m be the argument of the i-th SEND event. By the code, when SEND(m) occurs, $(m, (i-1) \mod M_c, i-1)$ is put in $send\_pending$. By admissibility of the execution, eventually $send(m, (i-1) \mod M_c, i-1)$ occurs. By Lemma 6, eventually $receive(m, (i-1) \mod M_c, i-1)$ occurs at least once.

By Lemma 7, when $receive(m, (i-1) \mod M_c, i-1)$ occurs for the first time, $(m, (i-1) \mod M_c, i-1)$ is put into $receive\_pending$.

By the assumption that there are only $i-1$ RECEIVE events and Lemma 4, $(m, (i-1) \mod M_c, i-1)$ is never removed from $receive\_pending$.

By Lemma 1(c) and the assumption that exactly $i-1$ RECEIVE events occur, eventually

$Aux_B = i - 1$ and never changes afterwards. By Lemma 1(d), eventually $counter_B = (i-1) \mod M_c$ and never changes afterwards.

Therefore, eventually RECEIVE(m) is continuously enabled, but never occurs. This contradicts the assumed admissibility of the execution. □

Theorem 1. *The Algorithm for unbounded but finite duplication and bounded reordering is correct.*

Proof. Follows from Lemmas 4 and 8. □

4. EXTENDED KJ PROTOCOL*

4.1 Algorithm

As presented in Chapter 3, the KJ protocol only works when bounded message reordering and duplication are allowed. However, the algorithm for the sender only sends one copy of each message, thus if there is message loss in the channel, the protocol will violate the safety and liveness properties. In order to model message loss, the bounded parameter λ is used, which is defined in Chapter 2. Recall that the channel is allowed to lose at most $\lambda-1$ consecutive messages sent over it and the bounded parameter λ is known to the sender.

The algorithm for the Extended KJ protocol is shown in Figure 4.1. The algorithm for the sender A is shown in Algorithm 5, where in the event SEND(m) (lines 4-11), the sender adds λ copies of each message in the FIFO queue $send\_pending$, instead of a single copy. The event $send(m, c, a)$ (lines 12-17) does not require any modification.

The algorithm to model the behavior of the channel C is shown in Algorithm 6. This algorithm requires an additional local variable, $loss\_counter$, which is used to keep track of how many messages are lost in a row. When a $send(m, c, a)$ event occurs (lines 3-16), it checks the value of $loss\_counter$ to determine if the message (m, c, a) should be inserted into $in\_transit$ or not. In more detail, if the value of $loss\_counter$ is equal to $\lambda-1$, then the message is inserted into $in\_transit$ and the variable is reset to 0. If the value of $loss\_counter$ is a number between 0 and $\lambda-2$ inclusive, then the channel makes a non-deterministic choice to lose the message or not. If the decision is to lose the message, then $loss\_counter$ is incremented by one, otherwise the message is inserted

*2017 IEEE. Part of the material reported in this chapter is reprinted, with permission, from Keishla D. Ortiz-Lopez and Jennifer L. Welch, "Bounded Reordering Allows Efficient Reliable Message Transmission", in *Proceedings 31st IEEE International Parallel & Distributed Processing Symposium (IPDPS)*, May 2017.

into $in\_transit$. The event $receive(m, c, a)$ does not require any modification and the algorithm for the receiver B is the same algorithm presented in Figures 3.2 and 3.3. Also since the channel can lose at most $\lambda-1$ consecutive messages sent over it, the modified behavior of A ensures that at least one copy of each high level message is received by B.

4.2 Reduction Proof

We will show that the Extended KJ protocol is correct by reducing it to the KJ protocol which was proven to be correct in Chapter 3. The idea is to show that the sequence of SEND and RECEIVE events in every execution of the Extended KJ protocol is the same as the sequence of SEND and RECEIVE events in some execution of the KJ protocol.

Lemma 9. *For every execution $\mathscr{E}$ of the Extended KJ protocol in a system with parameters λ and δ, there exists an execution $\mathscr{E}'$ of the KJ protocol in a system with parameter δ, such that $\mathscr{E}$, restricted to events and removing duplicate $send$ events is the same as $\mathscr{E}'$ restricted to events.*

Proof. Let $\mathscr{E} = \mathscr{C}_0 e_1 \mathscr{C}_1 ...$ be an execution of the Extended KJ protocol with parameters δ and λ. We will inductively construct an execution $\mathscr{E}' = f(\mathscr{E})$ of the KJ protocol with parameter δ as follows. Let $\mathscr{E}_i$ be the prefix of $\mathscr{E}$ up to $\mathscr{C}_i$ and let τ' be the transition function of the KJ protocol.

Basis: $i = 0$. $f(\mathscr{E}_0) = f(\mathscr{C}_0)$ is defined to be the (unique) initial configuration of the KJ protocol.

Induction: Suppose that $f(\mathscr{E}_{i-1})$ is defined and has been shown to be an execution of the KJ protocol. We now define $f(\mathscr{E}_i)$ and show it is an execution of the KJ protocol. In all the cases, let $\mathscr{C}$' be the last configuration in $f(\mathscr{E}_{i-1})$.

- **Case 1:** e_i = SEND(m). Define $f(\mathscr{E}_i)$ to be $f(\mathscr{E}_{i-1})$ SEND(m) $\mathscr{C}''$, where $\mathscr{C}'' = \tau'(\mathscr{C}')$. Then we append SEND($m$) to the execution of the KJ protocol that we are

constructing and let the state change according to the KJ protocol code. Therefore, $f(\mathscr{E}_i)$ is an execution of the KJ protocol because SEND(m) is an input event and input events are always enabled.

- **Case 2:** $e_i = send(m, c, a)$. If the effect of e_i in $\mathscr{E}_i$ is to add (m, c, a) to $in\_transit$, and this is the first time that (m, c, a) is added to $in\_transit$, then define $f(\mathscr{E}_i)$ to be $f(\mathscr{E}_{i-1})send(m, c, a)\,\mathscr{C}''$, where $\mathscr{C}'' = \tau'(\mathscr{C}')$; otherwise define $f(\mathscr{E}_i)$ to be $f(\mathscr{E}_{i-1})$. That is, if this is the first time that $send(m, c, a)$ succeeds, then reflect this $send$ in the execution of the KJ protocol being constructed, and otherwise ignore the event.

  To show that $send(m, c, a)$ is enabled in $\mathscr{C}'$, we note that in the KJ protocol execution being constructed, the previous SEND(m) event puts one copy of (m, c, a) in $send\_pending$. Since we only append $send(m, c, a)$ to the KJ protocol execution the first time that $send(m, c, a)$ succeeds in the Extended KJ protocol execution, that copy of (m, c, a) is at the head of $send\_pending$ in $\mathscr{C}'$.

- **Case 3:** $e_i = receive(m, c, a)$. Define $f(\mathscr{E}_i)$ to be $f(\mathscr{E}_{i-1})receive(m, c, a)\mathscr{C}''$, where $\mathscr{C}'' = \tau'(\mathscr{C}'')$, i.e., append $receive(m, c, a)$ to the execution of the KJ protocol that we are constructing and let the state change according to the KJ protocol code. The precondition of this event is that all entries in $in\_transit$ with index $\leq \ell - \delta$ have been delivered at least once to B. This event is enabled because the duplicated messages put into $in\_transit$ in the Extended KJ protocol does not affect the algorithm for the receiver V, as their delivery could just as well be explained as duplicate delivery of a single message in the KJ protocol. In fact the level of reordering in the Extended KJ protocol is never more than that in the KJ protocol because of the duplicated messages that are sent consecutively. We now proceed in more detail.

  The following facts can be proved by induction:

Fact 1. *In $\mathscr{C}_i$, $in\_transit$ consists of the sequence $m_1^{a_1}, m_2^{a_2}, ..., m_{k-1}^{a_{k-1}}, m_k^{a_k}$ for some k, where each $m_h^{a_h}$ indicates a_h copies of the message m_h, for some a_h between 1 and λ (ignoring the values of the fourth component). Furthermore, if a message has fourth component $True$, then it is at index at most $\delta-1$.*

Fact 2. *In $\mathscr{C}'$, $in\_transit$ consists of the sequence $m_1, m_2, ..., m_{k-1}, m_k$ (ignoring the values of the fourth component), i.e., there is exactly one copy of each unique message in $\mathscr{C}_i$. Furthermore, if a message has fourth component $True$, then it is at index at most $\delta-1$.*

Since $receive(m, c, a)$ is enabled in $\mathscr{C}_i$, the code of the Extended KJ protocol implies that (m, c, a, y) is in $in\_transit$ at some index ℓ for some y, and every entry in $in\_transit$ at index at most $\ell-\delta$ has fourth component $True$.

By the facts above, the unique entry in $in\_transit$ in $\mathscr{C}'$ that corresponds to (m, c, a, y) is at index at most ℓ, and every entry in $in\_transit$ at index at most $\ell-\delta$ has fourth component $True$. Therefore, $receive(m, c, a)$ is enabled in $\mathscr{C}'$.

- **Case 4:** $e_i = \text{RECEIVE}(m)$. Define $f(\mathscr{E}_i)$ to be $f(\mathscr{E}_{i-1})\text{RECEIVE}(m)\mathscr{C}''$, where $\mathscr{C}'' = \tau'(\mathscr{C}'')$, i.e., append $\text{RECEIVE}(m)$ to the execution of the KJ protocol that we are constructing and let the state change according to the KJ protocol code. The precondition of this event is that (m, c, a) is at the head of $receive\_pending$ and $c = counter_B$. Therefore, this event is enabled in $\mathscr{C}'$ because B's code is the same in the Extended KJ protocol as in the KJ protocol, and thus the evolution of the state of B in $\mathscr{E}$ is the same as the evolution of the state of B in $f(\mathscr{E})$. Since $\text{RECEIVE}(m)$ is enabled in $\mathscr{C}_{i-1}$, it is also enabled in $\mathscr{C}'$.

□

Theorem 2. *The algorithm for unbounded but finite duplication, bounded reordering and bounded loss is correct.*

Proof. Follows from Lemma 9, since it was shown that $\mathscr{E}$ and $f(\mathscr{E})$ have the same sequence of SEND and RECEIVE events and since $f(\mathscr{E})$ is an execution of the KJ protocol which we already proved was correct, it follows that $\mathscr{E}$ is also correct. □

Algorithm 5: Algorithm for sender A.

$counter_A$, an integer, initially 0
$send\_pending$, a FIFO queue, initially empty
Aux_A, an integer, initially 0
input:
event SEND(m)
    effects:
        for $i = 1$ **to** λ **do**
            $send\_pending$.enq($(m, counter_A, Aux_A)$)
        end for
        $counter_A$:= $(counter_A + 1) \bmod M_c$
        $Aux_A := Aux_A + 1$
end event
output:
event $send(m, c, a)$
    preconditions:
        (m, c, a) is at the head of $send\_pending$
    effects:
        remove (m, c, a) from $send\_pending$
end event

Algorithm 6: Algorithm to model behavior of C.

$in\_transit$, array, initially empty
$loss\_counter$, an integer, initially 0
input:
event $send(m, c, a)$
    effects:
        if $loss\_counter = \lambda - 1$ **then**
            insert $(m, c, a, False)$ at the end of $in\_transit$
            $loss\_counter := 0$
        else
            make a non-deterministic choice to lose (m, c, a) or not
            if $lose$ **then**
                $loss\_counter := loss\_counter + 1$
            else
                insert $(m, c, a, False)$ at the end of $in\_transit$
            end if
        end if
end event
output:
event $receive(m, c, a)$
    preconditions:
        $(m, c, a, y) \in in\_transit$ at some index ℓ for some y
        for all $(m', c', a', y') \in in\_transit$ with index $\leq \ell - \delta$, $y' = True$
    effects:
        remove from $in\_transit$ all entries with index $\leq \ell - \delta$
        $y := True$
end event

Figure 4.1: Extended KJ Protocol: Algorithm for the sender A and the behavior of the channel C.

5. CONCLUSIONS AND FUTURE WORK*

In this thesis, we have presented and proved the correctness of a non ACK-based protocol, called KJ protocol, to solve the RMTP when there is a single channel, from the sender to the receiver, in which message reordering is bounded by a parameter δ and message duplication is unbounded but finite. The KJ protocol relies on tagging each message sent with a counter bounded by M_c, where $M_c = 2\delta+1$. The receiver has the responsibility to determine if a message received is new or old by ordering each message in the same order they were sent with respect to the bounded counters and considering the cases when the counters wrap around. We also provided another protocol, called the Extended KJ protocol, that allows bounded message loss λ and a reduction proof showing that the Extended KJ protocol can be reduced to the KJ protocol, since the algorithm for the receiver does not need any modification.

5.1 Future Work

Future work includes providing an algorithm or impossibility proof when the parameters δ and λ are unknown to the processors. Another interesting direction is to consider extensions to multi-hop communication in a general network, including the possibility of dynamic changes to the network topology. Prior work [2, 4, 5] has studied this problem in the fully asynchronous case; improvements might be possible when there are bounds on the amount of loss, duplication and reordering.

*2017 IEEE. Part of the material presented in this chapter is reprinted, with permission, from Keishla D. Ortiz-Lopez and Jennifer L. Welch, "Bounded Reordering Allows Efficient Reliable Message Transmission", in *Proceedings 31st IEEE International Parallel & Distributed Processing Symposium (IPDPS)*, May 2017.

REFERENCES

[1] Y. Afek, H. Attiya, A. Fekete, M. Fischer, N. Lynch, Y. Mansour, D.-W. Wang, and L. Zuck, "Reliable Communication over Unreliable Channels," *Journal of the ACM*, vol. 41, no. 6, pp. 1267–1297, Nov. 1994.

[2] Y. Afek, B. Awerbuch, E. Gafni, Y. Mansour, A. Rosén, and N. Shavit, "Slide–the key to polynomial end-to-end communication," *Journal of Algorithms*, vol. 22, no. 1, pp. 158–186, 1997.

[3] K. A. Bartlett, R. A. Scantlebury, and P. T. Wilkinson, "A note on reliable full-duplex transmission over half-duplex links," *Communications of the ACM*, vol. 12, no. 5, pp. 260–261, May 1969.

[4] S. Dolev and J. L. Welch, "Crash resilient communication in dynamic networks," *IEEE Transactions on Computers*, vol. 46, no. 1, pp. 14–26, 1997.

[5] S. Dolev, A. Hanemann, E. M. Schiller, and S. Sharma, "Self-stabilizing end-to-end communication in (bounded capacity, omitting, duplicating and non-FIFO) dynamic networks," in *Symposium on Self-Stabilizing Systems*, 2012, pp. 133–147.

[6] K. Engelhardt and Y. Moses, "Single-bit messages are insufficient in the presence of duplication," in *International Workshop on Distributed Computing*, 2005, pp. 25–31.

[7] R. E. Ladner, A. LaMarca, and E. Tempero, "Counting Protocols for Reliable End-to-End Transmission," *Journal of Computer and System Sciences*, vol. 56, no. 1, pp. 96–111, Feb. 1998.

[8] Y. Mansour and B. Schieber, "The intractability of bounded protocols for on-line sequence transmission over non-FIFO channels," *Journal of the ACM (JACM)*, vol. 39, no. 4, pp. 783–799, Oct. 1992.

[9] E. R. McCurley, "Auxiliary variables in partial correctness programming logics,"

Information Processing Letters, vol. 33, no. 3, pp. 131–133, 1989.

[10] K. D. Ortiz-Lopez and J. L. Welch, "Bounded Reordering Allows Efficient Reliable Message Transmission," in *Proceedings of the 31st IEEE International Parallel & Distributed Processing Symposium (IPDPS 2017)*, May 2017.

[11] S. Sastry and S. M. Pike, "Eventually perfect failure detectors using ADD channels," in *International Symposium on Parallel and Distributed Processing and Applications*, 2007, pp. 483–496.

[12] N. V. Stenning, "A data transfer protocol," *Computer Networks*, vol. 1, no. 2, pp. 99–110, 1976.

[13] E. Tempero and R. Ladner, "Tight bounds for weakly bounded protocols," in *Proceedings of the Ninth Annual ACM Symposium on Principles of Distributed Computing*, 1990, pp. 205–218.

[14] E. D. Tempero and R. E. Ladner, "Recoverable sequence transmission protocols," *Journal of the ACM (JACM)*, vol. 42, no. 5, pp. 1059–1090, Sep. 1995.

[15] D.-W. Wang and L. Zuck, "Real-time sequence transmission problem," in *Proceeding PODC '91 Proceedings of the Tenth Annual ACM Symposium on Principles of Distributed Computing*, 1991, pp. 111–123.

[16] D.-W. Wang and L. D. Zuck, "Tight bounds for the sequence transmission problem," in *Proceedings of the Eighth Annual ACM Symposium on Principles of Distributed Computing*, 1989, pp. 73–83.

[17] G. Xylomenos and G. C. Polyzos, "TCP and UDP performance over a wireless LAN," in *INFOCOM'99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 2. IEEE, 1999, pp. 439–446.