Specifying and Proving Broadcast Properties with TLA

William Hipschman
Department of Computer Science
The University of North Carolina at Chapel Hill

Abstract

Although group communication is vitally important in many distributed systems, it is often reasoned about informally. The seminal papers on broadcast use an operational mindset to argue for correctness. Given their widespread use, it is important to argue formally that the correctness properties claimed for many broadcast variants are actually met by their implementations. Because these ideas got their start from Lamport, we will continue the tradition and use his temporal logic of actions as our proof language. In this paper we prove that reliable broadcast, causal reliable broadcast, and atomic broadcast maintain validity, integrity, and agreement. We also show that causal broadcast fulfills the happened-before order relation and atomic broadcast fulfills a total order relation.

1. Introduction

Group communication is important in distributed systems. Many distributed programs assume there is some underlying method of group communication. Each of Consensus, Dining Philosophers, the Byzantine Generals Problem, and replication, require some method for processes to communicate. In practice, this means that distributed file systems, databases, and transactions often depend on some type of group communication.

Despite this widespread use, operating systems often do not support group communication primitives; they merely provide access to kernel-level send and receive operations. It is then necessary for an application programmer to build their own communication protocols. Because these systems may be safety-critical or have high business impact, it is important to prove that they are correct.

In this paper we consider a common group communication paradigm; broadcast. We use the Temporal Logical of Actions (TLA), devised by Lamport, to rigorously prove correctness and order properties for several types of broadcast.

In Section 2 we review predicate logic, the logic of actions, and temporal logic. We then present a combination of the three, TLA. Section 3 presents the formal model for broadcasts and discusses issues in the system model such as synchrony and process link-interconnects. It also gives an overview of each of the properties that we will prove. Section 4 defines basic broadcast. Sections 5, 6, and 7 define reliable broadcast, causal broadcast, and atomic broadcast. These sections also prove the correctness and order properties for each broadcast variation.

2. Temporal Logic of Actions

We will use the *temporal logic of actions* as our proof language. It is a proof system devised by Lamport that combines predicate logic, the concept of actions, and temporal logic [10].

2.1 Logic

Predicate Logic is a system that assigns Boolean values to predicates. The primary features of the language are conjunction, $\wedge$, disjunction, $\vee$, and negation, $\neg$. Conditional implication, $\rightarrow$, and bi-conditional implication, $\leftrightarrow$, are often used for brevity, instead of their equivalent simplifications. We will also use *first-order-logic*, which, informally, uses universal and existential quantification over sets to define Boolean functions. A reader not familiar with predicate logic can find a cursory description in [10], but is encouraged to consult a discrete mathematics textbook.

2.2 Actions

Process executions can be represented as sequences of states and transitions [9]. A *state* is a non-Boolean expression over variables and constants, while an *action* is a Boolean-value expression over variables that relates an

old state to a new state [10]. An action is *enabled* in a state if that action can be performed and yield a legal state [10]. Often actions are variable assignments. We prime variables to indicate that they refer to a new state, so that $x' = x + 1$ indicates that the new value of x is equal to one more than the old value of x For convenience, when an action, A, may leave variables unchanged, we write $[A]_{\langle t \rangle}$, so that $[A]_{\langle t \rangle} \equiv A \vee (t' = t)$ [10].

2.3 Temporal Logic

Temporal logic is a logic language for reasoning about when actions occur. The primary operator is *always*, written as $\Box$. We say that the predicate $\Box F$ is satisfied if F holds true in any state in a behavior. The operator *eventually*, written as, $\Diamond$, is defined for a formula F as $\Diamond \mathrm{F} = \neg\Box\neg F$. Combining predicate logic, actions, and temporal logic yields *Raw Temporal Logic of Actions* (RTLA), a superset of temporal logic of actions [10].

2.4 Temporal Logic of Actions

The *Temporal Logic of Actions* is a subset of RTLA such every formula can be written in the form $\Box F$ [10]. In this paper, this will not be a limitation. The interested reader should consult [10] for more information. The full logic includes support for fairness and stuttering steps. The programs we deal with will not require these.

3. Model

We will assume a system model that maintains the highest level of asynchronicity. This means that for broadcast, reliable broadcast, and causal broadcast, we assume complete *asynchronicity*, that is, there is no bound on message delay, clock drift, or time required to execute a step. Informally, this means that a process cannot distinguish if another process has failed or if that other process is merely slow-running [5, 6]. For atomic broadcast to be possible, it has been shown that the system must have enough synchrony to be able to solve the consensus problem. While [5] gives this criteria, we will merely assume a solution to the consensus problem, and incorporate it into the atomic broadcast protocol.

We assume that messages are passed by underlying send and receive operations that are provided by the operating system. A *send* operation loads a message onto a link, which transfers it to a destination. The *receive* operation retrieves a message from the link and passes it to the appropriate process. We assume that links do not fail, but may take arbitrarily long to transfer data; this assumption mirrors modern TCP/IP technology and does us no harm with our failure model. Send or receive operation may fail by omission. An *omission* failure causes an operation to be ignored, but leaves the functional unit intact [1, 4]; a send operation may omit sending a message it was asked to, and then continue sending future messages. Therefore, if there are no failures then a message is eventually delivered. Omission failures are the most severe asynchronous benign failure [1]. *Crash* failures, where a process stops running entirely, can be modeled by infinitely many continuous omission failures. We will therefore assume that failures only occur at the send/receive level. If a process attempts a send operation or a receive operation that fails, then that process is in a failed state. Notably, we will not allow *arbitrary*, or Byzantine, failures, where a process may send erroneous or illegal messages nor will we allow *timing failures*, where a process breaks some timing contract [1, 4].

3.1 Formal Model

A system can be described by a set of processes, P, each of which has a clock, C, and an application state, Q. The application state contains a member, the *premature halting state*, $\perp$. We refer to the clock or state of a process, p, by $\boldsymbol{C}(p)$or $\boldsymbol{Q}(p)$, respectively. A message, m, is a tuple $(p, c, data)$ where $p \in P$, $c \in C$, and data is the information to be sent. The set of all messages is denoted by M and the set of all sequences of messages is denoted by M^+. An application protocol, Π, is a relation from a state to a message, and also from a state to a state. That is, $\Pi: P \times C \times Q \times M^+ \to M \cup \emptyset$ and $\Pi: P \times C \times Q \times M^+ \to Q$ [7, 9]. Let the sequence of messages that a process, p, has delivered, be denoted $\boldsymbol{M}^+(p)$

If a process, or if an operating system procedure doing working for that process, fails in any way, then we say that the process is in the premature halting state. We will also assume that sequence numbers may be sent as part of the $data$ field of a message, so that a message can be referred to as $(p, c, seq, data)$. For asynchronous systems the clock will be ignored.

In order to differentiate between programmatic constructs and predicates over variables, we define the actions **D** and **B** over all messages and processes to represent, respectively, the action of delivering and broadcasting a message.

$\mathbf{D}(\mathrm{T}, p, m)$: $\forall p \in P$: $\forall m \in M$: $process\ p\ delivers\ message\ m\ that\ came\ from\ broadcast\ protocol$ T
$\mathbf{B}(\mathrm{T}, p, m)$: $\forall p \in P$: $\forall m \in M$: $process\ p\ broadcasts\ message\ m\ using\ broadcast\ protocol$ T

Each of these constructs corresponds to a type of broadcast, that is:

$\mathrm{T} \in \{\text{Broadcast (B)}, \text{Reliable Broadcast (R)}, \text{Causal Broadcast (C)}, \text{Atomic Broadcast (A)}\}$

When the broadcast type is unnecessary or implied by context, we will omit it and merely write $\mathbf{D}(p, m)$ or $\mathbf{B}(p, m)$. In practice, these actions imply inserting the message into a buffer. If there are no link failures, $\mathbf{B}(p, m) \rightarrow \rightarrow \forall q \in P: \mathbf{Q}(q) \neq \perp: \Diamond \mathbf{D}(q, m)$ and $\mathbf{D}(p, m) \rightarrow m \in \boldsymbol{M}^{+}(p)$

3.2 Correctness and Order Properties

There are three primary correctness properties about which we are interested in reasoning: agreement, integrity, and validity. A protocol satisfies the *agreement* property if, when any correct process delivers a message $m \in M$, then all correct processes eventually deliver m [1, 8]. Formally, this is:

$$\forall p \in P: \mathbf{Q}(p) \neq \perp: \mathbf{D}(p, m) \rightarrow \forall q \in P: \mathbf{Q}(p) \neq \perp: \Diamond \mathbf{D}(q, m) \quad (1)$$

A protocol satisfies the *integrity* property if, for any message $m \in M$, every process delivers m at most once, and only if some process previously broadcast it [1, 8]. Formally, this is:

$$\forall m, m' \in M: \forall p \in P: \mathbf{Q}(p) \neq \perp: \mathbf{D}(p, m) \wedge \mathbf{D}(p, m') \rightarrow (m \neq m' \wedge \exists q \in P: \mathbf{B}(q, m)) \quad (2)$$

A protocol satisfies the *validity* property if, when any correct process broadcasts a message $m \in M$, then all correct processes eventually deliver m [1, 8]. Formally, this is:

$$\forall p \in P: \mathbf{Q}(p) \neq \perp: \mathbf{B}(p, m) \rightarrow \forall q \in P: \mathbf{Q}(q) \neq \perp: \Diamond \mathbf{D}(q, m) \quad (3)$$

While correctness properties are important, they do not guarantee anything about message order. We are primarily interested in causal and total orders. We will reason about both by associating a sequence number with every message [2]. We will refer to the sequence number of a message, $m \in M$ by $Seq(m)$. Then a behavior satisfies the *causal order* property iff :

$$\forall p \in P: \mathbf{Q}(p) \neq \perp: \forall m \in M: \mathbf{D}(p, m) \rightarrow \forall m' \in M: seq(m') < seq(m): m' \in \boldsymbol{M}^{+}(p) \quad (4)$$

That is, a causal order is a partial order over all messages in the system [1, 8, 9]. In practice, causal orders are often informally defined as the *happened before* relation, where a process must deliver all messages in the order that they happened, and deliver simultaneous messages in arbitrary order. It is important to note that two different messages can have the same sequence number. A total order does not allow this equality [1, 8]. A behavior satisfies the *total order* property iff:

$$\forall j, k \in M: seq(j) = seq(k) \rightarrow j = k \wedge$$
$$\forall p \in P: \mathbf{Q}(p) \neq \perp: \forall m \in M: \mathbf{D}(p, m) \rightarrow \forall m' \in M: seq(m') < seq(m) \rightarrow m' \in \boldsymbol{M}^{+}(p) \quad (5)$$

It is worth noting that these correctness and order properties do not gives us all the requirements we might desire. It possible for processes to become inconsistent while meeting each of these properties. We will not discuss inconsistency, contamination, or uniformity in this paper; the interested reader can find more information in [7].

4. Broadcast

The most basic form of broadcast, hereafter referred to merely as *broadcast*, is a form of one-to-N communication in which one process sends a message to all other processes. There are no imposed properties, though our implementation maintains validity, which we will prove in Section 4.2. Traditional pseudocode for broadcast is given in Figure 1.

Every Process p executes the following:

to execute **broadcast**(B, m):
    for each *process* i:
        send m *to* i

deliver(B, m) *occurs as follows*:
    upon recipt of m
        deliver m

Figure 1. Broadcast

4.1 Specification

Broadcast can be written in TLA as:

$\mathbf{B}(\mathrm{B}, p, m) \equiv \forall q \in P: \Diamond \mathbf{send}(p, q, m)$
$\mathbf{D}(\mathrm{B}, p, m) \equiv \forall q \in P: \mathbf{receive}\ (p, q, m) \rightarrow \Diamond \mathbf{D}(B, q, m)$

4.2 Proof

Note that to prove that broadcast is valid, we need only concern ourselves with processes where $\mathbf{Q}(p) \neq \perp$. Then validity is:

$$\forall p \in P: \mathbf{B}(p, m) \rightarrow \forall q \in P: \Diamond \mathbf{D}(q, m)$$

Lemma 1: Broadcast is valid

$\mathbf{B}(\mathrm{B}, p, m) \equiv \forall q \in P: \Diamond \mathbf{send}(p, q, m)$	Broadcast Definition
$\mathbf{B}(\mathrm{B}, p, m) \rightarrow \forall q \in P: \Diamond \mathbf{receive}\ (p, q, m)$	Links Do Not Fail
$\mathbf{B}(\mathrm{B}, p, m) \rightarrow \forall q \in P: \Diamond \mathbf{D}(B, q, m)$	Broadcast Delivery Definition
Q.E.D.	

5. Reliable Broadcast

Reliable broadcast guarantees integrity, validity, and agreement. It is accomplished by having a process broadcast a message. Whenever a process delivers a message from the broadcast protocol, it then broadcasts it to all other processes before delivering it in the reliable broadcast protocol [1, 2, 8]. We show pseudocode for a version of reliable broadcast in Figure 2.

Every Process p executes the following:

to execute **broadcast**(R, m):
    broadcast(B, m)

deliver(R, m) *occurs as follows*:
    upon deliver(B, m) **do**:
        if p has not previously executed **deliver**(R, m)
            then
                if $sender(m) \neq p$ **then broadcast**(B, m)
                deliver(R, m)

Figure 2. Reliable Broadcast [1]

5.1 Specification

Reliable broadcast can be written in TLA as:

$\mathbf{B}(\mathrm{R}, p, m) \equiv \mathbf{B}(B, p, m)$
$M1 \equiv \forall q \in P: \forall m \in M: (\mathbf{D}\ (\mathrm{B}, q, m) \rightarrow \Diamond \mathbf{D}(R, q, m) \vee m \in \boldsymbol{M}^{+}(q)$
$M2 \equiv \forall q \in P: \forall m \in M: m \in \boldsymbol{M}^{+}(q) \rightarrow \neg \Diamond \mathbf{D}(R, q, m)$
$\mathbf{D}(\mathrm{R}, p, m) \equiv M1 \wedge \mathrm{M2}$

5.2 Proof

Note that to prove that broadcast maintains validity, integrity, and agreement, we need only concern ourselves with processes where $\mathbf{Q}(p) \neq \perp$. Then we have:

Lemma 2: Reliable Broadcast is valid

This follows from the equivalence of reliable broadcast and broadcast.

Lemma 3: Reliable Broadcast maintains agreement

$\mathbf{D}(\mathrm{R}, p, m) \equiv M1 \wedge \mathrm{M2}$	Reliable Broadcast Delivery Definition
$\mathbf{D}(\mathrm{R}, p, m) \rightarrow \mathbf{B}(R, p, m)$	Links Do Not Fail, Closure
$\mathbf{B}(R, p, m) \rightarrow \forall q \in P: \Diamond \mathbf{D}(B, q, m)$	Validity of Reliable Broadcast
Q.E.D.	

Lemma 4: Reliable Broadcast maintains integrity

$\mathbf{D}(\mathrm{R}, p, m) \wedge \mathbf{D}(\mathrm{R}, p, m') \equiv$	Reliable Broadcast Delivery Definition
$\quad \mathbf{D}(\mathrm{B}, p, m) \rightarrow (\Diamond \mathbf{D}(R, p, m) \vee m \in \boldsymbol{M}^{+}(p))$	
$\quad \wedge\, m \in \boldsymbol{M}^{+}(p) \rightarrow \neg \Diamond \mathbf{D}(R, p, m)$	
$\quad \wedge\, \mathbf{D}(\mathrm{B}, q, m') \rightarrow (\Diamond \mathbf{D}(R, q, m') \vee m' \in \boldsymbol{M}^{+}(q))$	
$\quad \wedge\, m' \in \boldsymbol{M}^{+}(q) \rightarrow \neg \Diamond \mathbf{D}(R, q, m')$	
$\rightarrow$	
$\quad \neg \mathbf{D}(\mathrm{B}, p, m) \vee \Diamond \mathbf{D}(R, p, m) \vee m \in \boldsymbol{M}^{+}(p)$	Conditional Definition
$\quad \wedge\, m \notin \boldsymbol{M}^{+}(p) \vee \neg \Diamond \mathbf{D}(R, p, m)$	
$\quad \wedge\, \neg \mathbf{D}(\mathrm{B}, q, m') \vee \Diamond \mathbf{D}(R, q, m') \vee m' \in \boldsymbol{M}^{+}(q)$	
$\quad \wedge\, m' \notin \boldsymbol{M}^{+}(q) \vee \neg \Diamond \mathbf{D}(R, q, m')$	
$\rightarrow$	
$\quad \Diamond \mathbf{D}(R, p, m) \vee m \in \boldsymbol{M}^{+}(p)$	Eliminate Vacuous Cases
$\quad \wedge\, m \notin \boldsymbol{M}^{+}(p) \vee \neg \Diamond \mathbf{D}(R, p, m)$	
$\quad \wedge\, \Diamond \mathbf{D}(R, q, m') \vee m' \in \boldsymbol{M}^{+}(q)$	
$\quad \wedge\, m' \notin \boldsymbol{M}^{+}(q) \vee \neg \Diamond \mathbf{D}(R, q, m')$	
$\rightarrow \Diamond \mathbf{D}(R, p, m) \oplus m \in \boldsymbol{M}^{+}(p) \wedge \Diamond \mathbf{D}(R, q, m') \oplus m' \in \boldsymbol{M}^{+}(q)$	Exclusive Or Definition
$\rightarrow m \neq m'$	Predicate Logic
$\rightarrow m \neq m' \wedge \exists q \in P: \mathbf{B}(q, m)$	Messages Must Originate At Some Process

6. Causal Broadcast

Causal broadcast is the first ordered broadcast we will cover. Figure 3 helps illustrate how causal broadcast functions; each message is delivered only after all messages that it is causally dependent on are delivered. The construction assumes an underlying FIFO reliable broadcast. While it is not discussed at length here, a FIFO reliable broadcast guarantees reliability in addition to a FIFO delivery of messages that come from the same process [1, 2, 8].

6.1 Specification

In TLA, causal broadcast can be written as:

Every Process p executes the following:
Initialization:
    $prevDlvrs = \perp$

to execute **broadcast**(C, m):
    broadcast(F, $< prevDlvrs || m >$)
    $prevDlvrs := \perp$

deliver(C, m) *occurs as follows*:
    upon deliver(F, $\langle m_1 \ldots m_l \rangle$) **do**:
        for i := 1 ... l **do**
            if p has not previously
                executed **deliver**(C, m_i)
            then
                deliver(C, m_i)
                $prevDlvrs := prevDlvrs || m_i$

Figure 3. Causal Broadcast [1]

$\mathbf{B}(\mathrm{C}, p, m) \equiv \mathbf{B}(F, p, m) \wedge \forall m' \in M: seq(m') < seq(m): \mathbf{B}(\mathrm{C}, p, m')$
$\mathbf{D}(\mathrm{C}, p, m) \equiv (\mathbf{D}\ (\mathrm{F}, q, m) \rightarrow \Diamond \mathbf{D}(\mathrm{C}, q, m)) \wedge (\mathbf{D}(\mathrm{C}, q, m) \rightarrow \forall m' \in M: seq(m') < seq(m): m' \in \boldsymbol{M}^{+}(p))$

6.2 Proof

Note that to prove that causal broadcast maintains causal order, we need only concern ourselves with processes where $\mathbf{Q}(p) \neq \perp$. Then we have:

Lemma 5: Causal Broadcast is valid

This follows trivially from the equivalence of causal broadcast and FIFO reliable broadcast.

Lemma 6: Reliable Broadcast maintains agreement

This follows trivially from the equivalence of causal broadcast and FIFO reliable broadcast.

Lemma 7: Reliable Broadcast maintains integrity

This follows trivially from the equivalence of causal broadcast and FIFO reliable broadcast.

Lemma 8: Reliable Broadcast maintains causal order

The causal order property is explicitly stated in the definition of causal delivery.

7. Atomic Broadcast

Atomic broadcast guarantees a total order over all messages. Because consensus can be reduced to atomic broadcast, atomic broadcast can only be implemented in a system where there is a solution to consensus. We assume a consensus solution that generates unique sequence numbers for each message [1, 2, 3, 8].

7.1 Specification

Atomic broadcast is specified in TLA as:

$\mathbf{B}(\mathrm{A}, p, m) \equiv \mathbf{B}(R, p, \boldsymbol{Consensus}(p, m) || m)$
$\mathbf{D}(\mathrm{A}, p, m) \equiv (\mathbf{D}\ (\mathrm{R}, q, m) \rightarrow \Diamond \mathbf{D}(\mathrm{C}, q, m)) \wedge$
$(\mathbf{D}(\mathrm{A}, q, m) \rightarrow \forall m' \in M: seq(m') <$
$seq(m): m' \in \boldsymbol{M}^{+}(p))$

Where $seq(m) = \boldsymbol{Consensus}(p, m)$ is prepended to each message. Note that the delivery specification is similar to the causal broadcast deliver specification. The only difference is in the underlying broadcast type and the sequence number generation. In the preceding section, sequence numbers fulfilled a causal partial order, while here they are defined by consensus to fulfill a total order.

Every process p executes the following:
$broadcast := \emptyset$
$delivered := \emptyset$
$k := 0$

to execute **broadcast**(A, m)*:*
    broadcast(R, m)

deliver(A, m)*occurs as follows*:
    upon deliver(R, m) **do**
        $broadcast := broadcast \cup m$
        if $broadcast - delivered = \emptyset$
            end
        $k := k + 1$
        $undelivered := broadcast - delivered$
        $msgs_k = \boldsymbol{Consensus}(k, undelivered)$
        deliver all $m' \in msgs_k$ in some deterministic order
        $delivered := delivered \cup msgs_k$

Figure 4. Atomic Broadcast [3]

7.2 Proof

Note that to prove that atomic broadcast maintains total order, we need only concern ourselves with processes where $\mathbf{Q}(p) \neq \perp$. Then we have:

Lemma 9: Causal Broadcast is valid

This follows trivially from the equivalence of atomic broadcast and reliable broadcast.

Lemma 10: Reliable Broadcast maintains agreement

This follows trivially from the equivalence of atomic broadcast and reliable broadcast.

Lemma 11: Reliable Broadcast maintains integrity

This follows trivially from the equivalence of atomic broadcast and reliable broadcast.

Lemma 12: Reliable Broadcast maintains total order

Total order follows trivially from the uniqueness of sequence numbers and the atomic broadcast delivery definition.

7. Conclusion

Broadcast is a vital operation in any distributed system. It can be used for message passing, agreement protocols, replication, and recovery. Despite this widespread use, most textbooks and papers do not rigorously prove broadcast. They either use operational proofs or omit proofs entirely. We proved that TLA specifications for broadcast, reliable broadcast, causal broadcast, and atomic broadcast fulfill the requirements for several delivery correctness and order properties. Many of these proofs were simplified by the use of predicate logic in TLA specifications; the proof obligations were often present in their entirety within the program specification. It would be worthwhile to perform assertational proofs of broadcast protocols written in pseudocode, as opposed to TLA. Additionally, commercial grade broadcast protocols should be proved correct, as they are more complicated than the textbook broadcast variants provided here.

References

[1] P. Bernstein, V. Hadzilacos, and N. Goodman, "Distributed Systems," Addison-Wesley, 1987.

[2] K. Birman, T. Joseph, "Reliable Communication in the Presence of Failures," ACM Transactions on Computer Systems, Vol. 5, Issue 1, Feb. 1987, pp.47-76.

[3] T. Chandra, S. Toueg, "Unreliable Failure Detectors for Reliable Distributed Systems," Journal of the ACM, Vol. 43, No. 2, March 1996, pp. 225-267.

[4] G. Coulouris, J. Dollimore, T. Kindberg, and G. Blair, "Distributed Systems: Concepts and Design," Edition Five, Addison-Wesley, 2012.

[5] D. Dolev, C. Dwork, and L. Stockmeyer, "On the Minimal Synchronism Needed for Distributed Consenus," Journal of the ACM, Vol. 34, No. 1, 1987, pp. 77-97.

[6] M. Fischer, N. Lynch, and M. Patterson, "Impossibility of Distributed Consensus with One Faulty Process," Journal of the ACM, Vol. 32, No. 2, April 1985, pp. 374-382.

[7] A. Gopal, "Fault-Tolerant Broadcasts and Multicasts: The Problem of Inconsistency and Contamination," Cornell University, Ph.D. Dissertation, 1991.

[8] V. Hadzilacos, and S. Toueg, "A Modular Apprach to Fault-Tolerant Broadcasts and Related Problems," Technical Report, Department of Computer Science, University of Toronto.

[9] L. Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System," Communications of the ACM, Vol. 21, No. 7, July 1978, pp. 558-565.

[10] L. Lamport, “The Temporal Logic of Actions,” ACM Transactions on Programming Languages and Systems, Vol. 16, No. 3, May 1994, pp. 872-923.