Real-Time Systems, , 1–27 ()

Formal Specification and Compositional Verification of an Atomic Broadcast Protocol

PING ZHOU AND JOZEF HOOMAN wsinjh@win.tue.nl
Department of Mathematics and Computing Science, Eindhoven University of Technology, P.O.Box 513, 5600 MB Eindhoven, The Netherlands

Abstract. We apply a formal method based on assertions to specify and verify an atomic broadcast protocol. The protocol is implemented by replicating a server process on all processors in a network. We show that the verification of the protocol can be done compositionally by using specifications in which timing is expressed by local clock values. First the requirements of the protocol are formally described. Next the underlying communication mechanism, the assumptions about local clocks, and the failure assumptions are axiomatized. Also the server process is represented by a formal specification. Then we verify that parallel execution of the server processes leads to the desired properties by proving that the conjunction of all server specifications and the axioms about the system implies the requirements of the protocol.

Keywords: Formal verification, Protocol verification, Real-time, Fault-tolerance, Broadcast.

1. Introduction

Embedded computing systems are usually composed of hardware and software components which can fail. Component failures might lead to unanticipated behaviour and service unavailability. To achieve high availability of a service despite failures, a key idea is to implement the service by a group of server processes running on distinct processors (Cristian 1990). Replication of service state information among group members enables the group to provide the service even when some of its members fail, since the remaining members have enough information about the service state to continue the service. To maintain the consistency of these replicated global states, any state update must be broadcast to all correct servers such that they all observe the same sequence of state updates. This requires a communication service which can be used by the client processes to deliver updates to their peers. We refer to such a communication service as *atomic broadcast* (also called reliable broadcast). There are two sets of atomic broadcast protocols: *synchronous* protocols, such as the protocols described by Babaoglu and Drumond (1985), Cristian et al. (1985 and 1989), and Cristian (1990), and *asynchronous* protocols, as presented by Birman and Joseph (1987) and Chang and Maxemchuck (1984).

Synchronous atomic broadcast protocols assume that the underlying communication delays between correct processors are bounded. Given this assumption, local clocks of correct processors can be synchronized, see e.g. (Cristian, Aghili, and Strong, 1986 and 1993). Then, following Cristian et al. (1985, 1989), the properties of synchronous atomic broadcast protocols are described in terms of local clocks:

- Termination: every update whose broadcast is initiated by a correct processor at time T on its clock is delivered by all correct processors at time $T+\Delta$ on their own clocks, where Δ is a positive parameter called the *broadcast termination time*.
- Atomicity: if a correct processor delivers an update at time U on its clock, then that update was initiated by some processor and is delivered by each correct processor at time U on its own clock.
- Order: all correct processors deliver their updates in the same order.

Synchronous atomic broadcast protocols provide an upper bound for the broadcast termination time. Hence they can be used in real-time applications where deadlines must always be met, even in the presence of failures. On the other hand, asynchronous broadcast protocols do not assume bounded message transmission delays between correct processors. Thus they cannot guarantee a bound for the broadcast termination time and are not suitable for critical real-time applications.

In order to provide a service despite the presence of faults, real-time systems often adopt fault-tolerance techniques. To achieve fault-tolerance, some kind of redundancy is introduced which will affect the timing behavior of a system. Hence it is a challenging problem to guarantee the correctness of distributed systems in which both real-time and fault-tolerance are important. We investigate the application of a formal verification method to such systems. Since atomic broadcast service is one of the fundamental issues in fault-tolerance, we select an atomic broadcast protocol presented in Cristian et al. (1985, 1989) which tolerates omission failures as our verification example. Henceforth, we use the term *atomic broadcast protocol* to refer to this protocol. An informal description of the protocol, an implementation, and an informal proof which shows that the implementation indeed satisfies the requirement of the protocol are presented in Cristian et al. (1985, 1989). We follow the informal description of these papers as close as possible and show that we can formally reason with specifications expressed in terms of local clocks.

The configuration of the service is illustrated in the figure 1. The atomic broadcast service is implemented by replicating a server process on all distributed processors in a network. Thus any client process on any processor can use this service. We allow more than one client process located on one processor. Assume that there are n processors in the network. Pairs of processors are connected by links which are point-to-point, bi-directional, communication channels. Message transmission time between correct processors is bounded. Each processor has access to a local clock. It is assumed that local clocks of correct processors are approximately synchronized. Further assume that only omission failures occur on processors and links. When a processor suffers an omission failure, it cannot send messages to other processors. When a link suffers an omission failure, the messages traveling along this link may be lost.

To send an update to its peers, a client process initiates the atomic broadcast server process located on the same processor to atomically broadcast that update. After such a request, the server processes execute a certain protocol and finally each

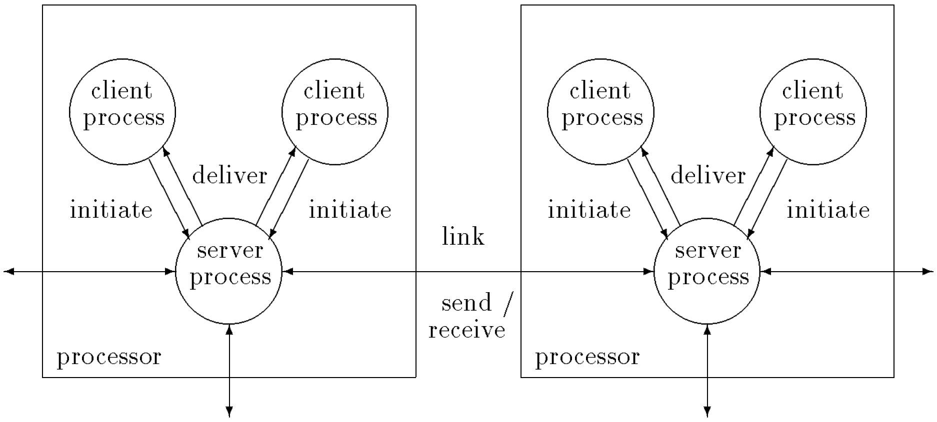

Figure 1. Atomic Broadcast Service Configuration.

server process delivers the update to the client processes located on its processor. To achieve the order property, there is a priority ordering among all processors. If two updates are initiated at different clock times, they will be delivered according to the ordering of their initiation times. If they are initiated at the same clock time on different processors, they will be delivered according to the priority of their initiation processors.

In general, to formally verify a system we need a proof theory which consists of axioms and rules about the system components. To be able to abstract from implementation details, it is often convenient to have a compositional verification method. Compositionality enables us to verify a system by using only specifications of its components without any information about the implementation of those components. Such compositional proof systems have been developed for non-real-time systems, e.g. (Zwiers 1989), and real-time systems, such as (Hooman 1991) and (Zhou and Hooman 1992).

In particular, if the system is composed of parallel components, the proof method should contain a *parallel composition rule.* Let $S(p)$ denote the atomic broadcast server process running on processor p, φ denote a specification written in a formal language based on first-order logic, and $S(p) \;\; \mathbf{sat} \;\; \varphi$ denote that server process $S(p)$ satisfies specification φ. Under the condition of maximal parallelism (i.e., each process runs at its own processor), the parallel composition rule states that if server process $S(p_i)$ satisfies specification φ_i and φ_i only refers to the interface of p_i, for $i = 1, \ldots, n$, then the parallel program $S(p_1) \| \cdots \| S(p_n)$ satisfies $\bigwedge_{i=1}^{n} \varphi_i$. This rule is formalized as follows.

Parallel Composition Rule

$$\frac{S(p_i)\ \mathbf{sat}\ \varphi_i,\ \varphi_i \text{ only refers to the interface of } p_i, \text{ for } i = 1, \ldots, n}{S(p_1) \| \cdots \| S(p_n)\ \mathbf{sat}\ \bigwedge_{i=1}^{n} \varphi_i}$$

We also need a *consequence rule* to weaken a specification and a *conjunction rule* to take the conjunction of specifications. Let S be any process.

Consequence Rule

$$\frac{S \textbf{ sat } \varphi,\ \varphi \rightarrow \psi}{S \textbf{ sat } \psi}$$

Conjunction Rule

$$\frac{S \textbf{ sat } \varphi_1,\ S \textbf{ sat } \varphi_2}{S \textbf{ sat } \varphi_1 \wedge \varphi_2}$$

These rules are used to justify the following steps in the verification of the protocol:

- In section 2, we specify the requirements of the protocol in a formal language based on first-order logic. We call this the *top-level specification* and denote it by ABS. Thus our aim is to prove $S(p_1)\|\cdots\|S(p_n)$ **sat** ABS.

- In section 3, we axiomatize the required assumptions about the system, including underlying communication mechanism, clock synchronization assumption, and failure assumptions. We denote the conjunction of all these axioms by AX.

- The properties of the atomic broadcast server process running on processor p are formalized in section 4. We call this the *server process specification* and denote it by $Spec(p)$. $Spec(p)$ should only refer to the interface of p. We assume $S(p)$ **sat** $Spec(p)$.

- The parallel composition rule leads to $S(p_1)\|\cdots\|S(p_n)$ **sat** $\bigwedge_{i=1}^{n} Spec(p_i)$. By the conjunction rule, we obtain $S(p_1)\|\cdots\|S(p_n)$ **sat** $\bigwedge_{i=1}^{n} Spec(p_i) \wedge AX$. We prove $\bigwedge_{i=1}^{n} Spec(p_i) \wedge AX \rightarrow ABS$ in section 5. Hence the consequence rule leads to $S(p_1)\|\cdots\|S(p_n)$ **sat** ABS.

- Finally section 6 contains concluding remarks, comparing our results with those in (Cristian et al. 1989).

2. Top-Level Specification

In this section we formalize the top-level requirements of the atomic broadcast protocol. After the introduction of a number of notations, primitives, and abbreviations, we express termination by formula $TERM$ in section 2.1, atomicity by $ATOM$ in section 2.2, and order by formula $ORDER$ in section 2.3. Then the top-level specification of the atomic broadcast protocol, ABS, is the conjunction of these three properties:

$ABS \equiv TERM \wedge ATOM \wedge ORDER.$

Let P be a set of processor names and L a set of link names. We assume that all processors and links have unique names. We use $p, q, r, s, \ldots$ to denote elements of P and $l, l_1, \ldots$ to denote elements of L. Let G be the network of processors and links, i.e., $G = P \cup L$.

To denote real times, we use a dense time domain called *RTIME*. The standard arithmetic operators $+$, $-$, $\times$, and the relations $=$, $<$, and $\leq$ are defined on *RTIME*. We use lower case letters, e.g. t, u, v, $\ldots$, to denote variables ranging over *RTIME*.

Each processor has access to a local clock. Let C_p be a function which represents the value of the local clock of processor p, i.e., $C_p(t)$ is the value of the local clock of p at real time t. Suppose local clock values range over a domain called *CVAL*. We assume that $T \geq 0$, for any $T \in CVAL$, and that the operators $+$, $-$, $\times$, and relations $=$, $<$, $\leq$ are defined on *CVAL*. We use capital letters, e.g. T, U, V, $\ldots$, to denote variables ranging over *CVAL*. Further $[U, V]$, $[U, V)$, $(U, V]$, and (U, V) denote, respectively, closed, half-open, and open intervals of clock values.

The atomic broadcast service is implemented by a group of server processes replicated on all processors in the network. When a client process initiates a request to broadcast update σ to a server process running on processor p, we call p the initiator of σ and say that *p initiates σ*. Similarly, when the server process of p delivers an update σ to client processes, we say that *p delivers σ to client processes*.

For a formal description of the properties of the protocol, we define the primitives:

- $correct(p)$ **at** t: processor p is correct at real time t.
- $correct(l)$ **at** t: link l is correct at real time t.
- $initiate(p, \sigma)$ **at** t: processor p finishes with receiving a request to broadcast update σ from a client process located on p at real time t, i.e., p initiates σ at real time t.
- $deliver(p, \sigma)$ **at** t: processor p starts to send update σ to client processes at real time t.

Henceforth we use $\equiv$ to denote syntactic equality. We define the following abbreviations, for any primitive φ **at** t,

- $correct(p) \equiv \forall t : correct(p)\ \mathbf{at}\ t$
- $correct(l) \equiv \forall t : correct(l)\ \mathbf{at}\ t$
- $\varphi\ \mathbf{at_p}\ T \equiv \exists t : \varphi\ \mathbf{at}\ t \wedge C_p(t) = T$
- $\varphi\ \mathbf{by_p}\ T \equiv \exists T_0 : \varphi\ \mathbf{at_p}\ T_0 \wedge T_0 \leq T$
- $\varphi\ \mathbf{before_p}\ T \equiv \exists T_0 : \varphi\ \mathbf{at_p}\ T_0 \wedge T_0 < T$
- $\varphi\ \mathbf{in_p}\ I \equiv \exists T \in I : \varphi\ \mathbf{at_p}\ T$, where $I \subseteq CVAL$.

In (Cristian et al. 1989), assumptions about the system are simplified. For instance, it is assumed that message processing time on a correct processor is zero. In this paper, however, we will take the time spent by a correct processor into account. Then the termination and atomicity properties can only be described by using an upper bound and an interval, respectively, instead of precise time points as in (Cristian et al. 1989).

2.1. Termination

The property of termination is stated as follows: every update whose broadcast is initiated by a correct processor s at clock value T will be delivered at all correct processors by clock value $T + D_1$ on their own clocks, where D_1 is a positive constant called the broadcast termination time. This property is formally expressed as follows, with the usual convention that any free variable occurring in a formula is universally quantified.

$$TERM \equiv initiate(s,\sigma)\ \mathbf{at_s}\ T \wedge correct(s) \wedge correct(q) \rightarrow deliver(q,\sigma)\ \mathbf{by_q}\ T+D_1$$

2.2. Atomicity

The atomicity property requires that if a correct processor p delivers an update at clock value U, then that update was initiated by some processor s (at some local time T) and is delivered by all correct processors at some local clock value between $U - D_2$ and $U + D_2$, where D_2 is a positive constant and indicates the difference of delivery times of an update by two correct processors. Formally,

$$\begin{aligned} ATOM \equiv\ & deliver(p,\sigma)\ \mathbf{at_p}\ U \wedge correct(p) \wedge correct(q) \rightarrow \\ & \exists s, T : initiate(s,\sigma)\ \mathbf{at_s}\ T \wedge deliver(q,\sigma)\ \mathbf{in_q}\ [U-D_2, U+D_2] \end{aligned}$$

Notice that the atomicity property does not follow from the termination property, because the initiator s need not be correct.

2.3. Order

The order property is expressed in (Cristian et al. 1989) as: all correct processors deliver their updates in the same order. To formalize this we consider the sequence of updates of a processor. Let U be any clock value. If $\langle\sigma_1, \ldots, \sigma_k\rangle$ is a sequence of updates delivered by processor p before local time U, then there should exist a clock value V such that $\langle\sigma_1, \ldots, \sigma_k\rangle$ has also been delivered by any other processor q before local time V. Notice that U and V can be different. Furthermore, there is no reason to exclude the possibility that more than one update is delivered at the same time by a processor. This is modelled by using a *set* of sequences to represent the updates of a processor and to include all possible interleavings for simultaneous updates. We define the following abbreviation to express that no update has been delivered in a certain interval I.

- $\neg deliver(p)\ \mathbf{in_p}\ I \equiv \neg\exists\sigma : deliver(p,\sigma)\ \mathbf{in_p}\ I.$

Let $I\!N$ denote the set of all natural numbers (including 0) and $I\!N^+ = I\!N \setminus \{0\}$. Next define $List(p, U)$ to be the set of all possible sequences of updates delivered by p before local time U.

Definition. For any processor p and any clock value $U \in CVAL$, define
$List(p, U) = \{\langle\sigma_1, \ldots, \sigma_k\rangle \mid$ there exist $k \in I\!N^+$, $U_1, \ldots, U_k \in CVAL$ such that

$U_1 \leq \ldots \leq U_k < U$,
$deliver(p, \sigma_i)$ $\mathbf{at_p}$ U_i, for $i = 1, \ldots, k$,
$\neg deliver(p)$ $\mathbf{in_p}$ (U_j, U_{j+1}), for $j = 1, \ldots, k-1$, and
$\neg deliver(p)$ $\mathbf{in_p}$ $[0, U_1)$.}

Then the order property is formalized as follows:

$ORDER \equiv correct(p) \wedge correct(q) \rightarrow \forall U \exists V : List(p, U) \subseteq List(q, V)$

By this property, we obtain that, for any pair of correct processors p and q, $\forall U \exists V : List(p, U) \subseteq List(q, V)$ and also $\forall U' \exists V' : List(q, U') \subseteq List(p, V')$. Hence p and q deliver their updates in the same order.

3. System Assumptions

Next we axiomatize the assumptions about the system, that is, properties of processors and links in section 3.1, axioms about the underlying communication mechanism in section 3.2, properties of local clocks in section 3.3, and the failure hypothesis in section 3.4. The conjunction of all these axioms is denoted by AX.

3.1. Processors and Links

We first consider the topology of the network. Define the primitive.

- $link(l, p, q)$: l is a physical communication channel between p and q.

Define $Link(p) = \{l \mid \exists q : link(l, p, q)\}$, representing the set of links that connect p with other processors. Two processors connected by a link are called neighbors. For any p, q, and l, if $l \in Link(p)$, $l \in Link(q)$, and $p \not\equiv q$, then p and q are connected by l. This is expressed by the following axiom.

Axiom 1 (Link) $\quad l \in Link(p) \wedge l \in Link(q) \wedge p \not\equiv q \rightarrow link(l, p, q)$

We also assume that a link connects at most two processors.

Axiom 2 (Point-to-Point) $\quad link(l, p, q) \wedge link(l, p, r) \rightarrow q \equiv r$

Let $FP = \{p \mid \neg correct(p)\}$ and $FL = \{l \mid \neg correct(l)\}$. Define $F = FP \cup FL$. Thus F denotes the set of processors and links which are not always correct.

An important assumption about the network is that during any execution of the protocol all correct processors remain connected via correct links. Recall that G is the set of all processors and links, that is, $G = P \cup L$. Then we have $G \setminus F = \{p \mid correct(p)\} \cup \{l \mid correct(l)\}$, denoting the set of correct processors and links. $G \setminus F$ can be considered as a graph in which processors are vertices and links are edges. We call $G \setminus F$ *connected* if and only if there exists a path between any two processors in $G \setminus F$. Now we can give the axiom for connectivity.

Axiom 3 (Connectivity) $G \setminus F$ is connected.

Given axiom 3, $d(p, q)$ is used to denote the *distance* between p and q and d to represent the diameter of $G \setminus F$.

3.2. Bounded Communication

To axiomatize the underlying communication mechanism, we define two primitives:

- $send(p, m, l)$ **at** t to denote that processor p starts to send message m along link l at real time t.
- $receive(p, m, l)$ **at** t to denote that processor p finishes with receiving message m along link l at real time t.

The abbreviations defined in section 2 are also used for these two primitives. When $send(p, m, l)$ **at** t or $receive(p, m, l)$ **at** t holds, l must be a link of p.

Axiom 4 (Connected Link)

$$send(p, m, l) \ \mathbf{at_q} \ T \ \vee \ receive(p, m, l) \ \mathbf{at_q} \ T \rightarrow l \in Link(p)$$

Two processors can send messages to each other if they are connected by a link. Communication along links is synchronous in the sense that the duration of the transmission of a message is bounded by two parameters γ and δ with $\gamma, \delta \in CVAL$, $0 < \gamma \leq \delta$. Let p and q be two correct processors connected by a correct link l. Let r be any correct processor to be used as reference. If p sends message m along link l at clock value U according to the clock of r, then q will receive m along l at some clock value in the interval $[U + \gamma, U + \delta]$ according to the clock of r.

Axiom 5 (Bounded Communication)

$$\begin{aligned} & send(p, m, l) \ \mathbf{at_r} \ U \wedge correct(p) \wedge correct(q) \wedge link(l, p, q) \wedge \\ & \quad correct(l) \wedge correct(r) \rightarrow receive(q, m, l) \ \mathbf{in_r} \ [U + \gamma, U + \delta] \end{aligned}$$

3.3. Clock Synchronization

Assume that clocks of correct processors are synchronized within a parameter ϵ.

Axiom 6 (Clock Synchronization)

$$correct(p) \ \mathbf{at} \ t \wedge correct(q) \ \mathbf{at} \ t \rightarrow |C_p(t) - C_q(t)| < \epsilon$$

Lemma 1 (Clock Synchronization)

$$correct(p) \wedge correct(q) \rightarrow |C_p(t) - C_q(t)| < \epsilon$$

We also assume that local clocks are monotonic.

Axiom 7 (Monotonic Clock)

$$t_1 \leq t_2 \leftrightarrow C_p(t_1) \leq C_p(t_2)$$

According to Cristian (1993), in the paper (Cristian et al. 1989) an implicit assumption was used, namely that any clock on a correct processor has a speed that differs from the speed of any other clock on a correct processor by a very small quantity ρ, $\rho \geq 0$. This drift ρ was neglected in (Cristian et al. 1989), resulting in the following approximation: while a message travels between two processors the clocks of the two processors will keep their distance constant. Here, however, this factor ρ is taken into account, leading to the following assumption:

Axiom 8 (Relative Speed)

$$\begin{array}{l} correct(p) \wedge correct(q) \wedge t_1 \leq t_2 \rightarrow \\ \qquad (1-\rho)(C_p(t_2) - C_p(t_1)) \leq C_q(t_2) - C_q(t_1) \leq (1+\rho)(C_p(t_2) - C_p(t_1)) \end{array}$$

3.4. Failure Assumptions

The atomic broadcast protocol verified in this paper tolerates omission failures. When a processor suffers an omission failure, it cannot send out messages. More precisely, if a processor p is not correct at real time t, then p is not able to send any message m along any link l at time t. This is also called the *fail silence* property of processors.

Axiom 9 (Fail Silence)

$$\neg correct(p) \ \mathbf{at_q} \ T \rightarrow \neg send(p, m, l) \ \mathbf{at_q} \ T$$

We assume that during any protocol execution there can be at most m processors that suffer omission failures, where $m \in I\!N$.

When a link suffers an omission failure, the messages entrusted on that link may be lost. But if a message has been received by a processor along a (possibly faulty) link, then that message should have been correctly transmitted by that link. For instance, the message is not corrupted and there are no timing errors. Therefore, if a processor q receives a message m along link l at clock value V on the clock of r, then there exists another processor p which has sent that message earlier along l at some time between $[V - \delta, V - \gamma]$ according to the clock of r.

Axiom 10 (Only Omission Failure)

$$receive(q, m, l) \ \mathbf{at_r} \ V \wedge correct(r) \rightarrow \exists p \not\equiv q : send(p, m, l) \ \mathbf{in_r} \ [V - \delta, V - \gamma]$$

4. Server Process Specification

In this section, we characterize $S(p)$, i.e., the atomic broadcast server process running on p. Notice that in the top-level specification only delivery of updates is important and thus a primitive $deliver(p, \sigma)$ **at** t is used. In the server process specification, however, information about the initiation time T and the initiator s of an update σ is needed to implement the top-level specification. Therefore we define a primitive

- $convey(p, < T, s, \sigma >)$ **at** t to denote that processor p starts to send message $< T, s, \sigma >$ to client processes at real time t.

Then the relation between $deliver(p, \sigma)$ **at** t and $convey(p, < T, s, \sigma >)$ **at** t is clear:

- $deliver(p, \sigma)$ **at** $t \leftrightarrow \exists s, T : convey(p, < T, s, \sigma >)$ **at** t

Assume that any correct processor can send a message to all its neighbors within $T_s \in CVAL$ time units and any correct processor can convey all the updates initiated at the same clock time to client processes within $T_c \in CVAL$ time units. Let $T_r \in CVAL$, $T_r \geq T_s$, be the time to ensure that all correct processors have received a message containing an update after it is initiated. These parameters will be used to determine the values of D_1 and D_2 occurring in the top-level specification.
Then the server specification is described by the following requirements.

- Initiation requirement.
  When p initiates an update σ at clock time T, it will send message $< T, p, \sigma >$ to all its neighbors within T_s. After T_r time units, assuming that then all correct processors have received that message, p will convey $< T, p, \sigma >$ to client processes. This is formalized by the following formula:

  $$\begin{aligned} Start(p) \equiv initiate(p, \sigma)\ \mathbf{at_p}\ T \rightarrow & \\ & \forall l \in Link(p) : send(p, < T, p, \sigma >, l)\ \mathbf{in_p}\ [T, T + T_s] \wedge \\ & convey(p, < T, p, \sigma >)\ \mathbf{in_p}\ [T + T_r, T + T_r + T_c] \end{aligned}$$

- Relay requirement.
  When p receives a message $< T, s, \sigma >$, it will relay this message on all its links except the one along which it received this message. As in the initiator's case, when its clock reaches $T + T_r$, p will convey $< T, s, \sigma >$ to client processes.

  $$\begin{aligned} Relay(p) \equiv receive(p, < T, s, \sigma >, l)\ \mathbf{at_p}\ U \rightarrow & \\ & \forall l_1 \in Link(p) \backslash \{l\} : send(p, < T, s, \sigma >, l_1)\ \mathbf{in_p}\ [U, U + T_s] \wedge \\ & convey(p, < T, s, \sigma >)\ \mathbf{in_p}\ [T + T_r, T + T_r + T_c] \end{aligned}$$

- Convey requirement.
  If processor p conveys a message $< T, s, \sigma >$ at clock time U, then we have $U \in [T + T_r, T + T_r + T_c]$ and either p initiated σ itself at local clock time T or p is different from s and has received $< T, s, \sigma >$ at some clock value V.
  When p initiates σ at local time T or it receives $< T, s, \sigma >$ at some local time

V, we say that p *learns of* message $< T, s, \sigma >$ and define:

$$Learn(p, < T, s, \sigma >) \equiv (initiate(p, \sigma) \ \mathbf{at_p} \ T \wedge p \equiv s) \vee \\ (\exists l, V : receive(p, < T, s, \sigma >, l) \ \mathbf{at_p} \ V \wedge p \not\equiv s)$$

Then the requirement is formalized by the formula $Origin(p)$:

$$Origin(p) \equiv convey(p, < T, s, \sigma >) \ \mathbf{at_p} \ U \rightarrow \\ Learn(p, < T, s, \sigma >) \wedge U \in [T + T_r, T + T_r + T_c]$$

- Ordering requirement.
  If two messages are conveyed by processor p, then they will be conveyed in the order of the initiation times contained in these two messages. If these initiation times are the same, then they will be conveyed according to the priority of the initiators. Therefore it is assumed that there is a total order $\prec$ on the set of processor names P. This total order specifies a priority ordering among processors. A lexicographical ordering $\sqsubset$ on pairs $< T, s >$ is defined, for any two pairs (T_1, s_1) and (T_2, s_2), by
  $(T_1, s_1) \sqsubset (T_2, s_2)$ iff $(T_1 < T_2) \vee (T_1 = T_2 \wedge s_1 \prec s_2)$.

  Then the fourth requirement is formalized by the formula $Sequen(p)$:

  $$Sequen(p) \equiv convey(p, < T_1, s_1, \sigma_1 >) \ \mathbf{at_p} \ V_1 \wedge convey(p, < T_2, s_2, \sigma_2 >) \ \mathbf{at_p} \ V_2 \\ \rightarrow (V_1 < V_2 \leftrightarrow (T_1, s_1) \sqsubset (T_2, s_2))$$

The requirements mentioned above only hold for correct processors. Since omission failures are allowed, we still need to define the acceptable behaviour for faulty processors. This is captured by the following requirement, which holds for any arbitrary processor p.

- Failure requirement.
  When p sends a message $< T, s, \sigma >$ to a neighbor at local time U, there can be only two possibilities: either p initiated σ itself at local time T and $U \in [T, T + T_s]$ holds, or p received $< T, s, \sigma >$ at some local time V and $U \in [V, V + T_s]$ holds.

  $$Source(p) \equiv send(p, < T, s, \sigma >, l) \ \mathbf{at_p} \ U \rightarrow \\ (initiate(p, \sigma) \ \mathbf{at_p} \ T \wedge U \in [T, T + T_s] \wedge p \equiv s) \vee \\ (\exists l_1, V : receive(p, < T, s, \sigma >, l_1) \ \mathbf{at_p} \ V \wedge U \in [V, V + T_s] \wedge p \not\equiv s)$$

  Observe that $send(p, < T, s, \sigma >, l) \ \mathbf{at_p} \ U$ implies, by the fail silence axiom 9, that $correct(p) \ \mathbf{at_p} \ U$ holds. But $correct(p) \ \mathbf{at_p} \ U$ does not imply $correct(p)$, and it is quite possible that p is faulty at some other time. Therefore this requirement should hold for any processor p and not only for correct ones.

Now we assume that server process $S(p)$ satisfies specification $Spec(p)$ with

$$Spec(p) \equiv [correct(p) \rightarrow Start(p) \wedge Relay(p) \wedge Origin(p) \wedge Sequen(p)] \wedge Source(p).$$

Then a processor p is specified by the fail silence axiom 9 and the next axiom.

Axiom 11 (Server Process Specification) $\quad S(p) \ \mathbf{sat} \ Spec(p)$

5. Verification

As explained in the introduction, our aim is to prove $\bigwedge_{i=1}^{n} Spec(p_i) \wedge AX \rightarrow ABS$, where AX is the conjunction of all the axioms and ABS is the top-level specification of the protocol. Thus we assume $\bigwedge_{i=1}^{n} Spec(p_i) \wedge AX$ and prove ABS, that is, we prove $TERM$ in section 5.1, $ATOM$ in section 5.2, and $ORDER$ in section 5.3. As a preparation we first give a number of lemmas.

Since we have assumed $\bigwedge_{i=1}^{n} Spec(p_i) \wedge AX$, we can rewrite a part of the specification $Spec(p)$ to a more general form in which the clock values are measured on an arbitrary correct processor r.

Lemma 2 (Modified Server Specification)

$$correct(r) \rightarrow [correct(p) \rightarrow Forward(p,r)] \wedge NSource(p,r)$$

where $Forward(p,r)$, which generalizes part of $Relay(p)$, is defined by

$$\begin{aligned} Forward(p,r) \equiv\ & receive(p, < T, s, \sigma >, l) \ \mathbf{at_r}\ U \rightarrow \\ & \forall l_1 \in Link(p) \setminus \{l\} : send(p, < T, s, \sigma >, l_1) \ \mathbf{in_r}\ [U, U + (1+\rho)T_s] \end{aligned}$$

and $NSource(p,r)$ is a general form of $Source(p)$:

$$\begin{aligned} NSource(p,r) \equiv\ & send(p, < T, s, \sigma >, l) \ \mathbf{at_r}\ U \rightarrow \\ & (initiate(p,\sigma) \ \mathbf{at_p}\ T \wedge U \in (T - \epsilon, T + T_s + \epsilon) \wedge p \equiv s) \vee \\ & \exists l_1, V : (receive(p, < T, s, \sigma >, l_1) \ \mathbf{at_r}\ V \wedge \\ & \qquad U \in [V, V + (1+\rho)T_s] \wedge p \not\equiv s) \end{aligned}$$

Proof: The proof of this lemma consists of two parts.

- First, we prove $correct(r) \wedge correct(p) \rightarrow Forward(p,r)$.
  Assume $correct(r) \wedge correct(p) \wedge receive(p, < T, s, \sigma >, l) \ \mathbf{at_r}\ U$.
  Let t_1 be the real time such that $C_r(t_1) = U$. Suppose $C_p(t_1) = U_1$.
  Then we have $receive(p, < T, s, \sigma >, l) \ \mathbf{at_p}\ U_1$. By $Relay(p)$, we obtain
  $\forall l_1 \in Link(p) \setminus \{l\} : send(p, < T, s, \sigma >, l_1) \ \mathbf{in_p}\ [U_1, U_1 + T_s]$.
  Let t_2 be the real time such that $C_p(t_2) = U_1 + T_s$. Thus we have
  $\forall l_1 \in Link(p) \setminus \{l\} : send(p, < T, s, \sigma >, l_1) \ \mathbf{in}\ [t_1, t_2]$.
  Since $T_s \geq 0$, we obtain $C_p(t_1) \leq C_p(t_2)$. By the monotonic clock axiom 7, we have $t_1 \leq t_2$. Then by the relative speed axiom 8, we obtain
  $(1-\rho)(C_p(t_2) - C_p(t_1)) \leq C_r(t_2) - C_r(t_1) \leq (1+\rho)(C_p(t_2) - C_p(t_1))$.
  Hence $C_r(t_2) \leq U + (1+\rho)T_s$. Thus we obtain
  $\forall l_1 \in Link(p) \setminus \{l\} : send(p, < T, s, \sigma >, l_1) \ \mathbf{in_r}\ [U, U + (1+\rho)T_s]$
  and then $Forward(p,r)$ holds.

- Second, we prove $correct(r) \rightarrow NSource(p,r)$.
  Assume $correct(r)$ and $send(p, < T, s, \sigma >, l) \ \mathbf{at_r}\ U$.
  Let t_1 be the real time such that $C_r(t_1) = U$. Suppose $C_p(t_1) = U_1$.
  Then we have $send(p, < T, s, \sigma >, l) \ \mathbf{at_p}\ U_1$. By $Source(p)$, we obtain
  $$initiate(p,\sigma) \ \mathbf{at_p}\ T \wedge U_1 \in [T, T + T_s] \wedge p \equiv s \qquad (1)$$

or

$$\exists l_1, V_1 : receive(p, < T, s, \sigma >, l_1) \ \mathbf{at_p} \ V_1 \wedge U_1 \in [V_1, V_1 + T_s] \wedge p \not\equiv s \quad (2)$$

Assume (1) holds. From $send(p, < T, s, \sigma >, l)$ **at** t_1, by the fail silence axiom 9, we have $correct(p)$ **at** t_1. By $correct(r)$ and the clock synchronization axiom 6, we obtain $|C_r(t_1) - C_p(t_1)| < \epsilon$. Since $C_p(t_1) = U_1 \in [T, T + T_s]$, this leads to $C_r(t_1) \in (T - \epsilon, T + T_s + \epsilon)$. From (1), we obtain

$$initiate(p, \sigma) \ \mathbf{at_p} \ T \wedge U \in (T - \epsilon, T + T_s + \epsilon) \wedge p \equiv s \quad (3)$$

Suppose that (2) holds. Let t_2 be the real time such that $C_p(t_2) = V_1$. Then there exists a V such that $C_r(t_2) = V$. Since $C_p(t_2) \leq C_p(t_1)$, by the monotonic clock axiom 7, we have $t_2 \leq t_1$. By the relative speed axiom 8, we obtain
$(1 - \rho)(C_p(t_1) - C_p(t_2) \leq C_r(t_1) - C_r(t_2) \leq (1 + \rho)(C_p(t_1) - C_p(t_2))$.
From $U_1 \in [V_1, V_1 + T_s]$, we have $0 \leq C_p(t_1) - C_p(t_2) \leq T_s$ and then
$C_r(t_2) \leq C_r(t_1) \leq C_r(t_2) + (1 + \rho)T_s$, i.e., $U \in [V, V + (1 + \rho)T_s]$.
With (2), this leads to

$$\exists l_1, V : receive(p, < T, s, \sigma >, l_1) \ \mathbf{at_r} \ V \wedge U \in [V, V + (1 + \rho)T_s] \wedge p \not\equiv s.$$

Together with (3) this proves $NSource(p, r)$.

■

The second lemma expresses that if a correct processor p receives a message $< T, s, \sigma >$ at time V measured on the clock of a correct processor r, then a correct neighbor q which is not s will receive $< T, s, \sigma >$ by $V + (1 + \rho)T_s + \delta$ measured on the clock of r.

Lemma 3 (Propagation)

$$receive(p, < T, s, \sigma >, l_1) \ \mathbf{at_r} \ V \wedge correct(p) \wedge correct(q) \wedge link(l_2, p, q) \wedge$$
$$correct(l_2) \wedge q \not\equiv s \wedge \ correct(r) \rightarrow$$
$$\exists l : receive(q, < T, s, \sigma >, l) \ \mathbf{by_r} \ V + (1 + \rho)T_s + \delta$$

Proof: Assume that the premise of the lemma holds.
Since $receive(p, < T, s, \sigma >, l_1)$ $\mathbf{at_r}$ V holds, there are two possibilities.

- If $l_1 \not\equiv l_2$ then, by $Forward(p, r)$, p will send the message $< T, s, \sigma >$ to q along link l_2 within $(1 + \rho)T_s$ time units as measured on the clock of r. Thus we have $send(p, < T, s, \sigma >, l_2)$ $\mathbf{in_r}$ $[V, V + (1 + \rho)T_s]$.
  Then there exists an V_1 such that
  $send(p, < T, s, \sigma >, l_2)$ $\mathbf{at_r}$ $V_1 \wedge V_1 \in [V, V + (1 + \rho)T_s]$.
  By the bounded communication axiom 5, we obtain
  $receive(q, < T, p, \sigma >, l_2)$ $\mathbf{in_r}$ $[V_1 + \gamma, V_1 + \delta]$.
  Together with $V_1 \leq V + (1 + \rho)T_s$, we obtain
  $\exists l : receive(q, < T, s, \sigma >, l)$ $\mathbf{by_r}$ $V + (1 + \rho)T_s + \delta$.
- If $l_1 \equiv l_2$ then p receives $< T, p, \sigma >$ from link l_2 and thus we have
  $receive(p, < T, s, \sigma >, l_2)$ $\mathbf{at_r}$ V.

By the only omission failure axiom 10, there exists a p_1 such that
$p_1 \not\equiv p \wedge send(p_1, < T, s, \sigma >, l_2)$ $\mathbf{in_r}$ $[V - \delta, V - \gamma]$.
By the connected link axiom 4, we have $l_2 \in Link(p) \wedge l_2 \in Link(p_1)$.
Since $p \not\equiv p_1$, by the link axiom 1, we obtain $link(l_2, p, p_1)$. By assumption, $link(l_2, p, q)$ and thus the point-to-point axiom 2 leads to $p_1 \equiv q$.
Thus $send(q, < T, s, \sigma >, l_2)$ $\mathbf{in_r}$ $[V - \delta, V - \gamma]$, that is, there exists a U such that $send(q, < T, s, \sigma >, l_2)$ $\mathbf{at_r}$ $U \wedge U \in [V - \delta, V - \gamma]$.
Since $q \not\equiv s$, by $NSource(q, r)$, we obtain
$\exists l, V' : (receive(q, < T, s, \sigma >, l)$ $\mathbf{at_r}$ $V' \wedge U \in [V', V' + (1 + \rho)T_s])$.
From $V' \leq U$ and $U \leq V - \gamma$, we obtain $V' \leq V - \gamma$ and thus
$V' \leq V + (1 + \rho)T_s + \delta$. Hence
$\exists l : receive(q, < T, s, \sigma >, l)$ $\mathbf{by_r}$ $V + (1 + \rho)T_s + \delta$.

■

If correct processor s initiates update σ at local time T, then any another correct processor q will receive $< T, s, \sigma >$ by time $T + d(s, q)((1 + \rho)T_s + \delta)$ measured on the clock of s. Recall that $d(s, q)$ denotes the distance between s and q.

LEMMA 4 (BOUNDED RECEIVING)

$$initiate(s, \sigma)\ \mathbf{at_s}\ T \wedge correct(s) \wedge correct(q) \wedge q \not\equiv s \rightarrow$$
$$\exists l : receive(q, < T, s, \sigma >, l)\ \mathbf{by_s}\ T + d(s, q)((1 + \rho)T_s + \delta)$$

Proof: Assume that the premise of the lemma holds. We prove this lemma by induction on the distance between s and q. Since $s \not\equiv q$, we start with $d(s, q) = 1$.

- $d(s, q) = 1$. Since both s and q are correct processors, by the definition of $d(s, q)$, they are connected by some correct link. Let l be that link. Then we obtain $link(l, s, q) \wedge correct(l)$. Since $correct(s)$ holds, $Spec(s)$ leads to $Start(s)$. From $Start(s)$ and $initiate(s, \sigma)$ $\mathbf{at_s}$ T, we obtain that s will send the message $< T, s, \sigma >$ to processor q along link l, that is,
  $send(s, < T, s, \sigma >, l)$ $\mathbf{in_s}$ $[T, T + T_s]$.
  By definition, there exists a U such that
  $send(s, < T, s, \sigma >, l)$ $\mathbf{at_s}$ $U \wedge U \in [T, T + T_s]$.
  By the bounded communication axiom 5, we obtain
  $receive(q, < T, s, \sigma >, l)$ $\mathbf{in_s}$ $[U + \gamma, U + \delta]$.
  From $U \leq T + T_s$, we obtain
  $receive(q, < T, s, \sigma >, l)$ $\mathbf{by_s}$ $T + T_s + \delta$.
  Since $\rho \geq 0$, we have
  $\exists l : receive(q, < T, s, \sigma >, l)$ $\mathbf{by_s}$ $T + d(s, q)((1 + \rho)T_s + \delta)$.
- $d(s, q) = k + 1$ with $k \geq 1$. By definition, there exist a link l_2 and a processor q_1 such that $link(l_2, q_1, q) \wedge correct(l_2) \wedge correct(q_1) \wedge d(s, q_1) = k \wedge d(q_1, q) = 1$.
  By the induction hypothesis, we have

$\exists l_1 : receive(q_1, < T, s, \sigma >, l_1)$ $\mathbf{by_s}$ $T + k((1+\rho)T_s + \delta)$.
By definition, there exists a V_1 such that
$\exists l_1 : (receive(q_1, < T, s, \sigma >, l_1)$ $\mathbf{at_s}$ $V_1 \wedge V_1 \leq T + k((1+\rho)T_s + \delta)$).
By the propagation lemma 3, we have
$\exists l : receive(q, < T, s, \sigma >, l)$ $\mathbf{by_s}$ $V_1 + (1+\rho)T_s + \delta$, i.e.,
$\exists l : receive(q, < T, s, \sigma >, l)$ $\mathbf{by_s}$ $T + (k+1)((1+\rho)T_s + \delta)$.
Hence we have proved
$\exists l : receive(q, < T, s, \sigma >, l)$ $\mathbf{by_s}$ $T + d(s,q)((1+\rho)T_s + \delta)$.

■

5.1. Verification of Termination

In this section we prove the termination property of the protocol, starting with a lemma which expresses that if a correct processor s initiates σ at local time T, then every correct processor q will convey $< T, s, \sigma >$ in the interval $[T+T_r, T+T_r+T_c]$ according to its own clock.

LEMMA 5 (CONVEY)

$$\begin{aligned} &initiate(s,\sigma)\ \mathbf{at_s}\ T \wedge correct(s) \wedge correct(q) \rightarrow \\ &\qquad convey(q, < T, s, \sigma >)\ \mathbf{in_q}\ [T+T_r, T+T_r+T_c] \end{aligned}$$

Proof: Assume that the premise of the lemma holds. We consider two cases.

- $d(s,q) = 0$. Thus, by definition, $s \equiv q$.
  By $correct(s)$ we obtain $Start(s)$, and thus $initiate(s,\sigma)$ $\mathbf{at_s}$ T leads to
  $convey(s, < T, s, \sigma >)$ $\mathbf{in_s}$ $[T+T_r, T+T_r+T_c]$. Thus, using $s \equiv q$,
  $convey(q, < T, s, \sigma >)$ $\mathbf{in_q}$ $[T+T_r, T+T_r+T_c]$.

- $d(s,q) > 0$. By definition, $s \not\equiv q$. By the bounded receiving lemma 4, we obtain
  $\exists l : receive(q, < T, s, \sigma >, l)$ $\mathbf{by_s}$ $T + d(s,q)((1+\rho)T_s + \delta)$.
  By the clock synchronization lemma 1, this leads to
  $\exists l : receive(q, < T, s, \sigma >, l)$ $\mathbf{before_q}$ $T + d(s,q)((1+\rho)T_s + \delta) + \epsilon$.
  Thus there exists a V such that $\exists l : receive(q, < T, s, \sigma >, l)$ $\mathbf{at_q}$ V.
  By $Relay(q)$, we obtain $convey(q, < T, s, \sigma >)$ $\mathbf{in_q}$ $[T+T_r, T+T_r+T_c]$.

■

Next we prove that the termination property $TERM$ follows from the axioms and lemmas given before.

THEOREM 1 (TERMINATION) If $D_1 \geq T_r + T_c$, then

$$initiate(s,\sigma)\ \mathbf{at_s}\ T \wedge correct(s) \wedge correct(q) \rightarrow deliver(q,\sigma)\ \mathbf{by_q}\ T + D_1$$

Proof: Assume that the premise of this theorem holds. By the convey lemma 5, we obtain $convey(q, < T, s, \sigma >)\ \mathbf{in_q}\ [T + T_r, T + T_r + T_c]$. As observed in section 4, this implies $deliver(q,\sigma)\ \mathbf{in_q}\ [T + T_r, T + T_r + T_c]$. Using $D_1 \geq T_r + T_c$, this leads to $deliver(q,\sigma)\ \mathbf{by_q}\ T + D_1$. ■

5.2. Verification of Atomicity

In this section, we prove the atomicity property of the atomic broadcast protocol, again using a number of lemmas. The first lemma states that if correct processor p receives message $< T, s, \sigma >$ at local time V, then that update σ was initiated by processor s at local time T.

LEMMA 6 (INITIATION)

$$receive(p, < T, s, \sigma >, l)\ \mathbf{at_p}\ V \wedge correct(p) \rightarrow initiate(s,\sigma)\ \mathbf{at_s}\ T$$

Proof: Assume that the premise of the lemma holds. By the only omission failure axiom 10, there exist s_1 and U_1 such that

$$s_1 \not\equiv p \wedge send(s_1, < T, s, \sigma >, l)\ \mathbf{at_p}\ U_1 \wedge U_1 \in [V - \delta, V - \gamma]. \qquad (1)$$

By $NSource(s_1, p)$, there exist l_1 and V_1 such that

$$initiate(s_1,\sigma)\ \mathbf{at_{s_1}}\ T \wedge s_1 \equiv s \qquad (2)$$

or

$$receive(s_1, < T, s, \sigma >, l_1)\ \mathbf{at_p}\ V_1 \wedge s_1 \not\equiv s \wedge U_1 \in [V_1, V_1 + (1+\rho)T_s]. \qquad (3)$$

If (2) holds, then clearly $initiate(s,\sigma)\ \mathbf{at_s}\ T$.

If (2) does not hold, then s_1 is not the initiator of σ and (3) holds.
From (1), we have $U_1 \leq V - \gamma$, i.e., $V \geq U_1 + \gamma$, and (3) implies $U_1 \geq V_1$.
Thus $V \geq V_1 + \gamma$, that is, $V - V_1 \geq \gamma$.
Using $receive(s_1, < T, s, \sigma >, l_1)\ \mathbf{at_p}\ V_1$, we can again use axiom 10 and $NSource$ to obtain another processor $s_2 \not\equiv s_1$. In case s_2 is not the initiator of σ we can repeat these steps to obtain s_3, etc. Let $k \in I\!N$, $k \geq 2$, such that $k > V/\gamma$ (notice that $\gamma > 0$). Then there are two possibilities:

- either there exists a $i < k$ such that s_i is the initiator of σ and $s_i \equiv s$. Hence we have obtained $initiate(s,\sigma)\ \mathbf{at_s}\ T$;
- or there does not exist a $i < k$ such that s_i is the initiator of σ.
  Thus the processors $s_1, \ldots, s_{k-1}$ are not the initiator of σ.
  Then, for any $i = 2, 3, \ldots, k-1$, there exist l_i and V_i such that
  $$s_i \not\equiv s_{i-1}\ \wedge\ receive(s_i, < T, s, \sigma >, l_i)\ \mathbf{at_p}\ V_i\ \wedge\ s_i \not\equiv s\ \wedge\ V_{i-1} - V_i \geq \gamma$$
  Together with $V - V_1 \geq \gamma$ we obtain $V - V_i \geq i\gamma$, for any $i = 1, \ldots, k-1$.
  From $receive(s_{k-1}, < T, s, \sigma >, l_{k-1})\ \mathbf{at_p}\ V_{k-1}$, by the only omission failure axiom 10, there exists a processor $s_k \not\equiv s_{k-1}$ such that

$send(s_k, < T, s, \sigma >, l_{k-1})$ $\mathbf{in_p}$ $[V_{k-1} - \delta, V_{k-1} - \gamma]$.
By $NSource(s_k, p)$, there exist l_k and V_k such that

$$initiate(s_k, \sigma) \ \mathbf{at_{s_k}} \ T \wedge s_k \equiv s \tag{5}$$

or

$$receive(s_k, < T, s, \sigma >, l_k) \ \mathbf{at_{s_k}} \ V_k \wedge s_k \not\equiv s \tag{6}$$

If (6) holds, then we can derive $V_{k-1} - V_k \geq \gamma$ as before. From $V - V_i \geq i\gamma$, we obtain $V - V_k \geq k\gamma$. Since $k > V/\gamma$, we have $V - V_k > V$ and thus $V_k < 0$. Recall that all local clock values are nonnegative. Hence (6) does not hold. Therefore (5) must hold, i.e., s_k is the initiator of σ and $s_k \equiv s$.

■

We define an abbreviation $Firstrec(p, < T, s, \sigma >, l)$ $\mathbf{at_r}$ V, which expresses that p receives $< T, s, \sigma >$ at time V measured on the clock of a correct processor r and p is one of the first correct processors which have received $< T, s, \sigma >$ according to the clock of r, that is, any other correct processor has not received $< T, s, \sigma >$ earlier.

$$\begin{aligned} &Firstrec(p, < T, s, \sigma >, l) \ \mathbf{at_r} \ V \equiv \\ &\quad receive(p, < T, s, \sigma >, l) \ \mathbf{at_r} \ V \wedge correct(r) \wedge correct(p) \ \wedge \\ &\quad \forall p', l', V' : (correct(p') \wedge p' \not\equiv p \wedge receive(p', < T, s, \sigma >, l') \ \mathbf{at_r} \ V' \rightarrow V' \geq V) \end{aligned}$$

The next lemma expresses that if p receives $< T, s, \sigma >$ at time V measured on the clock of a correct processor r, p is one of the first correct processors which have received $< T, s, \sigma >$, and s is faulty, then any processor q which is not p and has sent $< T, s, \sigma >$ earlier than V is a faulty processor.

LEMMA 7 (FAULTY SENDER)

$$\begin{aligned} &Firstrec(p, < T, s, \sigma >, l_1) \ \mathbf{at_r} \ V \wedge send(q, < T, s, \sigma >, l_2) \ \mathbf{at_r} \ U \ \wedge \\ &\quad p \not\equiv q \ \wedge \neg correct(s) \wedge U < V \rightarrow \neg correct(q) \end{aligned}$$

Proof: Assume that the premise of the lemma holds.
From $send(q, < T, s, \sigma >, l_2)$ $\mathbf{at_r}$ U, by $NSource(q, r)$, we obtain

$$initiate(q, \sigma) \ \mathbf{at_q} \ T \wedge q \equiv s \tag{1}$$

or there exist l' and V' such that

$$receive(q, < T, s, \sigma >, l') \ \mathbf{at_r} \ V' \wedge q \not\equiv s \wedge U \in [V', V' + (1 + \rho)T_s]. \tag{2}$$

Then there exist two possibilities:

- if (1) holds, then $q \equiv s$ and thus, by assumption, $\neg correct(q)$;
- if (2) holds, we have $V' \leq U$. Since $U < V$, we obtain $V' < V$.
  Suppose $correct(q)$. Then $Firstrec(p, < T, s, \sigma >, l)$ $\mathbf{at_r}$ V implies $V' \geq V$, and thus leads to a contradiction. Hence $\neg correct(q)$.

■

The following lemma shows that if p receives $< T, s, \sigma >$ at time V measured on the clock of a correct processor r, p is one of the first correct processors which have received $< T, s, \sigma >$, and s is faulty, then $V < T + m((1+\rho)T_s + \delta) + \epsilon$, where m is the maximum number of faulty processors in the network.

Lemma 8 (First Correct Receiving)

$$\begin{aligned} &Firstrec(p, < T, s, \sigma >, l) \ \mathbf{at_r}\ V \wedge \neg correct(s) \rightarrow \\ &\quad V < T + m((1+\rho)T_s + \delta) + \epsilon \end{aligned}$$

Proof: Assume that the premise of the lemma holds.
From $Firstrec(p, < T, s, \sigma >, l)$ $\mathbf{at_r}$ V, we obtain $receive(p, < T, s, \sigma >, l)$ $\mathbf{at_r}$ V.
By the only omission failure axiom 10, there exist s_1 and U_1 such that
$s_1 \not\equiv p \wedge send(s_1, < T, s, \sigma >, l)$ $\mathbf{at_r}$ $U_1 \wedge U_1 \in [V - \delta, V - \gamma]$. Thus we have

$$V \leq U_1 + \delta \text{ and } U_1 \leq V - \gamma. \qquad (1)$$

Then we obtain $V \geq U_1 + \gamma$. Since $\gamma > 0$, this leads to

$$V > U_1. \qquad (2)$$

Since $Firstrec(p, < T, s, \sigma >, l)$ $\mathbf{at_r}$ V, the faulty sender lemma 7 implies that s_1 is a faulty processor, i.e., $\neg correct(s_1)$. By $NSource(s_1, r)$, either

$$initiate(s_1, \sigma) \ \mathbf{at_{s_1}}\ T \wedge s_1 \equiv s \wedge U_1 \in (T - \epsilon, T + T_s + \epsilon) \qquad (3)$$

or there exist l_1 and V_1 such that

$$receive(s_1, < T, s, \sigma >, l_1) \ \mathbf{at_r}\ V_1 \wedge s_1 \not\equiv s \wedge U_1 \in [V_1, V_1 + (1+\rho)T_s] \qquad (4)$$

Then there are two possibilities.

- If (3) holds, then s_1 is the initiator of σ and we have $U_1 < T + T_s + \epsilon$.
  Together with (1), this leads to $V < T + (1+\rho)T_s + \delta + \epsilon$.
  Since $\neg correct(s)$, there is at least one faulty processor, i.e., $m \geq 1$.
  Thus we obtain $V < T + m((1+\rho)T_s + \delta) + \epsilon$.

- If (4) holds, we have $U_1 \leq V_1 + (1+\rho)T_s$. Using (1), this leads to

  $$V \leq V_1 + (1+\rho)T_s + \delta. \qquad (5)$$

  From $receive(s_1, < T, s, \sigma >, l_1)$ $\mathbf{at_r}$ V_1, by the only omission failure axiom 10, there exist s_2 and U_2 such that s_2 has sent $< T, s, \sigma >$ to s_1 along link l_1 at time U_2 measured on the clock of r. As before, we have $U_2 \in [V_1 - \delta, V_1 - \gamma]$, i.e., $U_2 \leq V_1 - \gamma$.
  From (4), $V_1 \leq U_1$ and thus $U_2 \leq U_1 - \gamma$. By (2), $U_1 < V$ and then $U_2 < V - \gamma$.
  Hence $V > U_2$. Then by the faulty sender lemma 7, $\neg correct(s_2)$ holds.
  By $NSource(s_2, r)$, we obtain formulae similar to (3) and (4) for s_2.
  If s_2 is not the initiator of σ, we follow the above steps and then obtain another s_3 which is also a faulty processor. Since there are at most m faulty processors, we cannot continue this procedure infinitely. We must obtain a s_k which is the initiator of σ with $k \leq m$. Then, for any $i = 2, 3, \ldots, k-1$, by the only omission

failure axiom 10 and $NSource(s_i, r)$, there exist l_i and V_i such that
$s_i \not\equiv s_{i-1} \wedge receive(s_i, < T, s, \sigma >, l_i)$ $\mathbf{at_r}$ $V_i \ \wedge \ s_i \not\equiv s \wedge$
$V_{i-1} \leq V_i + (1+\rho)T_s + \delta$.
This leads to
$$V_1 \leq V_{k-1} + (k-2)((1+\rho)T_s + \delta). \quad (6)$$
From $receive(s_{k-1}, < T, s, \sigma >, l_{k-1})$ $\mathbf{at_r}$ V_{k-1}, by the only omission failure axiom 10, there exists a U_k such that
$s_k \not\equiv s_{k-1} \wedge send(s_k, < T, s, \sigma >, l_{k-1})$ $\mathbf{at_r}$ $U_k \wedge U_k \in [V_{k-1} - \delta, V_{k-1} - \gamma]$.
Then we obtain $V_{k-1} \leq U_k + \delta$. Together with (6) this leads to
$$V_1 \leq U_k + (k-2)(1+\rho)T_s + (k-1)\delta. \quad (7)$$
Since s_k is the initiator of σ, by $NSource(s_k, r)$, we have
$initiate(s_k, \sigma)$ $\mathbf{at_{s_k}}$ $T \wedge s_k \equiv s \wedge U_k \in (T - \epsilon, T + T_s + \epsilon)$.
Together with (7) this leads to
$$V_1 < T + (k-1)((1+\rho)T_s + \delta) + \epsilon. \quad (8)$$
Combining (5) and (8), we obtain $V < T + k((1+\rho)T_s + \delta) + \epsilon$.
Since $k \leq m$, this leads to $V < T + m((1+\rho)T_s + \delta) + \epsilon$.

■

Next we show that if p receives $< T, s, \sigma >$ at time V measured on the clock of a correct processor r and s is faulty, then any other correct processor q will receive $< T, s, \sigma >$ by time $V + d(p,q)((1+\rho)T_s + \delta)$ measured on the clock of r.

Lemma 9 (Correct Receiving)

$$receive(p, < T, s, \sigma >, l')\ \mathbf{at_r}\ V \wedge \neg correct(s) \wedge correct(q) \wedge p \not\equiv q \rightarrow$$
$$\exists l : receive(q, < T, s, \sigma >, l)\ \mathbf{by_r}\ V + d(p,q)((1+\rho)T_s + \delta)$$

Proof: Assume that the premise of the lemma holds. We prove this lemma by induction on the distance between p and q. Since $p \not\equiv q$, we start with $d(p,q) = 1$.

- $d(p,q) = 1$. By definition, p and q are connected by some correct link. Let l be that link. Then we have $link(l, p, q) \wedge correct(l)$.
  From $receive(p, < T, s, \sigma >, l')$ $\mathbf{at_r}$ V, by the only omission failure axiom 10, there exist a p_1 and a U_1 such that
  $p_1 \not\equiv p \wedge send(p_1, < T, s, \sigma >, l')$ $\mathbf{at_r}$ $U_1 \wedge U_1 \in [V - \delta, V - \gamma]$.
  Since $U_1 \leq V - \gamma$ and $\gamma > 0$, we have $V \geq U_1 + \gamma$ and then $V > U_1$.
  By the faulty sender lemma 7, we have $\neg correct(p_1)$.
  Thus the correct processor q is different from the sender p_1.
  By $Forward(p, r)$, p will send $< T, s, \sigma >$ to q along link l within $(1+\rho)T_s$ time units. Thus we have $send(p, < T, s, \sigma >, l)$ $\mathbf{in_r}$ $[V, V + (1+\rho)T_s]$. By definition, there exists an X such that $send(p, < T, s, \sigma >, l)$ $\mathbf{at_r}$ $X \wedge X \in [V, V + (1+\rho)T_s]$.
  By the bounded communication axiom 5, we obtain
  $receive(q, < T, s, \sigma >, l)$ $\mathbf{in_r}$ $[X + \gamma, X + \delta]$.

Together with $X \leq V + (1+\rho)T_s$, we have proved
$\exists l : receive(q, < T, s, \sigma >, l)$ $\mathbf{by_r}$ $V + (1+\rho)T_s + \delta$, i.e.,
$\exists l : receive(q, < T, s, \sigma >, l)$ $\mathbf{by_r}$ $V + d(p,q)((1+\rho)T_s + \delta)$.

- $d(p,q) = k+1$ with $k \geq 1$. Then there exist a processor q_1 and a link l_2 such that $correct(q_1) \wedge correct(l_2) \wedge link(l_2, q_1, q) \wedge d(p, q_1) = k \wedge d(q_1, q) = 1$.
  By the induction hypothesis, we have
  $\exists l_1 : receive(q_1, < T, s, \sigma >, l_1)$ $\mathbf{by_r}$ $V + k((1+\rho)T_s + \delta)$.
  By definition, there exists a V_1 such that
  $\exists l_1 : receive(q_1, < T, s, \sigma >, l_1)$ $\mathbf{at_r}$ $V_1 \wedge V_1 \leq V + k((1+\rho)T_s + \delta)$.
  Since $correct(q)$ and $\neg correct(s)$, we obtain $q \not\equiv s$.
  Then by the propagation lemma 3, we have
  $\exists l : receive(q, < T, s, \sigma >, l)$ $\mathbf{by_r}$ $V_1 + (1+\rho)T_s + \delta$, i.e.,
  $\exists l : receive(q, < T, s, \sigma >, l)$ $\mathbf{by_r}$ $V + (k+1)((1+\rho)T_s + \delta)$.
  Hence we have proved
  $\exists l : receive(q, < T, s, \sigma >, l)$ $\mathbf{by_r}$ $V + d(p,q)((1+\rho)T_s + \delta)$.

■

The next lemma expresses that if correct processor p learns of $< T, s, \sigma >$, then any correct processor q also learns of $< T, s, \sigma >$.

LEMMA 10 (ALL LEARN)

$$Learn(p, < T, s, \sigma >) \wedge correct(p) \wedge correct(q) \rightarrow Learn(q, < T, s, \sigma >)$$

Proof: Assume that the premise of the lemma holds. By $Learn(p, < T, s, \sigma >)$, we have

$$initiate(p, \sigma) \ \mathbf{at_p} \ T \wedge p \equiv s \tag{1}$$

or

$$\exists l_1, V_1 : receive(p, < T, s, \sigma >, l_1) \ \mathbf{at_p} \ V_1 \wedge p \not\equiv s \tag{2}$$

From (2), by the initiation lemma 6, we obtain $initiate(s, \sigma)$ $\mathbf{at_s}$ T.
Since either (1) or (2) holds, we obtain $initiate(s, \sigma)$ $\mathbf{at_s}$ T from the premise.
We have to prove $Learn(q, < T, s, \sigma >)$, i.e.,

$$initiate(q, \sigma) \ \mathbf{at_q} \ T \wedge q \equiv s \tag{3}$$

or

$$\exists l_2, V_2 : receive(q, < T, s, \sigma >, l_2) \ \mathbf{at_q} \ V_2 \wedge q \not\equiv s. \tag{4}$$

There are two possibilities:

- if $s \equiv q$, then we have $initiate(q, \sigma)$ $\mathbf{at_q}$ $T \wedge q \equiv s$ holds, i.e., (3) holds;
- if $s \not\equiv q$, we prove that (4) holds by considering the following two cases.

  1. If $correct(s)$ holds, by the bounded receiving lemma 4, we obtain
     $\exists l_2 : receive(q, < T, s, \sigma >, l_2)$ $\mathbf{by_s}$ $T + d(s,q)((1+\rho)T_s + \delta)$.

By the clock synchronization lemma 1, we have
$\exists l_2 : receive(q, < T, s, \sigma >, l_2)$ **before$_\mathbf{q}$** $T + d(s, q)((1 + \rho)T_s + \delta) + \epsilon$, i.e.,
$\exists l_2, V_2 : receive(q, < T, s, \sigma >, l_2)$ **at$_\mathbf{q}$** $V_2 \wedge q \not\equiv s$, and thus (4).

2. If $\neg correct(s)$ holds, then $correct(p)$ leads to $p \not\equiv s$ and thus (1) does not hold. From (2), we have $receive(p, < T, s, \sigma >, l_1)$ **at$_\mathbf{p}$** V_1.
Then there exists a V_1' such that
$receive(p, < T, s, \sigma >, l_1)$ **at$_\mathbf{q}$** $V_1' \wedge V_1' \in (V_1 - \epsilon, V_1 + \epsilon)$.
Hence there must exist a processor p_1 which is one of the first correct processors that have received $< T, s, \sigma >$ according to the clock of q. Thus there exist l_3 and V such that $Firstrec(p_1, < T, s, \sigma >, l_3)$ **at$_\mathbf{q}$** V and hence $receive(p_1, < T, s, \sigma >, l_3)$ **at$_\mathbf{q}$** V. By the first correct receiving lemma 8, we obtain $V < T + m((1 + \rho)T_s + \delta) + \epsilon$. There are again two possibilities:
   - if $q \equiv p_1$, then we have $receive(q, < T, s, \sigma >, l_3)$ **at$_\mathbf{q}$** V, i.e.,
     $\exists l_2, V_2 : receive(q, < T, s, \sigma >, l_2)$ **at$_\mathbf{q}$** $V_2 \wedge$
     $V_2 < T + m((1 + \rho)T_s + \delta) + \epsilon$;
   - if $q \not\equiv p_1$, by the correct receiving lemma 9, we have
     $\exists l_2 : receive(q, < T, s, \sigma >, l_2)$ **by$_\mathbf{q}$** $V + d(p, q)((1 + \rho)T_s + \delta)$, i.e.,
     $\exists l_2, V_2 : receive(q, < T, s, \sigma >, l_2)$ **at$_\mathbf{q}$** $V_2 \wedge$
     $V_2 < T + (d(p, q) + m)((1 + \rho)T_s + \delta) + \epsilon$.

   Both possibilities lead to
   $\exists l_2, V_2 : receive(q, < T, s, \sigma >, l_2)$ **at$_\mathbf{q}$** $V_2 \wedge q \not\equiv s$, i.e., (4).

■

If correct processor p conveys $< T, s, \sigma >$ at local time U, then any correct processor q conveys $< T, s, \sigma >$ in the interval $[T + T_r, T + T_r + T_c]$ on its own clock.

LEMMA 11 (ALL CONVEY)

$$convey(p, < T, s, \sigma >) \ \mathbf{at_p}\ U \wedge correct(p) \wedge correct(q) \rightarrow$$
$$convey(q, < T, s, \sigma >) \ \mathbf{in_q}\ [T + T_r, T + T_r + T_c]$$

Proof: Assume that the premise of this lemma holds. From $correct(p)$, we have $Origin(p)$. By $convey(p, < T, s, \sigma >)$ **at$_\mathbf{p}$** U, we obtain $Learn(p, < T, s, \sigma >)$. Then the all learn lemma 10 leads to $Learn(q, < T, s, \sigma >)$, i.e.,

$$initiate(q, \sigma) \ \mathbf{at_q}\ T \wedge q \equiv s \quad (1)$$

or

$$\exists l, V : receive(q, < T, s, \sigma >, l) \ \mathbf{at_q}\ V \wedge q \not\equiv s. \quad (2)$$

If (1) holds, $Start(q)$ leads to $convey(q, < T, s, \sigma >)$ **in$_\mathbf{q}$** $[T + T_r, T + T_r + T_c]$.
If (2) holds, $Relay(q)$ leads to $convey(q, < T, s, \sigma >)$ **in$_\mathbf{q}$** $[T + T_r, T + T_r + T_c]$.
Hence in both cases we obtain $convey(q, < T, s, \sigma >)$ **in$_\mathbf{q}$** $[T + T_r, T + T_r + T_c]$.
■

Finally we prove a theorem which shows that the atomicity property $ATOM$ follows from the axioms and lemmas given before.

THEOREM 2 (ATOMICITY) If $D_2 \geq T_c$, then

$$\begin{aligned}&deliver(p,\sigma)\ \mathbf{at_p}\ U \wedge correct(p) \wedge correct(q) \rightarrow\\ &\qquad \exists s, T : initiate(s,\sigma)\ \mathbf{at_s}\ T \wedge deliver(q,\sigma)\ \mathbf{in_q}\ [U - D_2, U + D_2]\end{aligned}$$

Proof: Assume that the premise of the theorem holds. From $deliver(p,\sigma)\ \mathbf{at_p}\ U$, by definition, there exist s and T such that $convey(p, < T, s, \sigma >)\ \mathbf{at_p}\ U$ holds. By the server process specification axiom 11 and $correct(p)$, we have $Origin(p)$. By $Origin(p)$, we obtain $Learn(p, < T, s, \sigma >) \wedge U \in [T + T_r, T + T_r + T_c]$, i.e.,

$$initiate(p,\sigma)\ \mathbf{at_p}\ T \wedge p \equiv s \tag{1}$$

or

$$\exists l, V : receive(p, < T, s, \sigma >, l)\ \mathbf{at_p}\ V \wedge p \not\equiv s \wedge \tag{2}$$

$$U \in [T + T_r, T + T_r + T_c]. \tag{3}$$

If (1) holds, then $initiate(s,\sigma)\ \mathbf{at_s}\ T$.
If (2) holds, the initiation lemma 6 leads to $initiate(s,\sigma)\ \mathbf{at_s}\ T$.
Hence both cases lead to

$$\exists s, T : initiate(s,\sigma)\ \mathbf{at_s}\ T. \tag{4}$$

From $convey(p, < T, s, \sigma >)\ \mathbf{at_p}\ U$, by the all convey lemma 11, we obtain $convey(q, < T, s, \sigma >)\ \mathbf{in_q}\ [T + T_r, T + T_r + T_c]$.
By (3), we have $T \in [U - T_r - T_c, U - T_r]$.
Hence $convey(q, < T, s, \sigma >)\ \mathbf{in_q}\ [U - T_c, U + T_c]$.
Thus, by definition, $deliver(q,\sigma)\ \mathbf{in_q}\ [U - T_c, U + T_c]$. Since $D_2 \geq T_c$ this leads to $deliver(q,\sigma)\ \mathbf{in_q}\ [U - D_2, U + D_2]$. Together with (4) this proves the theorem.
∎

5.3. Verification of Order

The order property of the atomic broadcast protocol will be proved in this section. We first prove a lemma which expresses that, for any correct processors p and q, if p conveys $< T, s, \sigma >$ at local time U, q conveys $< T, s, \sigma >$ at local time V, and no update is delivered by p in the interval $[0, U)$, then there is also no update delivered by q in the interval $[0, V)$.

LEMMA 12 (FIRST DELIVERY)

$$\begin{aligned}&convey(p, < T, s, \sigma >)\ \mathbf{at_p}\ U \wedge convey(q, < T, s, \sigma >)\ \mathbf{at_q}\ V \wedge correct(p) \wedge\\ &\qquad correct(q)\ \wedge \neg deliver(p)\ \mathbf{in_p}\ [0, U) \rightarrow \neg deliver(q)\ \mathbf{in_q}\ [0, V)\end{aligned}$$

Proof: Assume that the premise of this lemma holds.
Suppose $deliver(q)\ \mathbf{in_q}\ [0, V)$ holds. Then there exist s_0, T_0, and V_0 such that $convey(q, < T_0, s_0, \sigma_0 >)\ \mathbf{at_q}\ V_0 \wedge V_0 \in [0, V)$.

By assumption, we have $convey(q, < T, s, \sigma >)$ $\mathbf{at_q}$ V.
From $V_0 < V$, by $Sequen(q)$, we obtain $(T_0, s_0) \sqsubset (T, s)$.
By the all convey lemma 11, we have
$convey(p, < T_0, s_0, \sigma_0 >)$ $\mathbf{in_p}$ $[T_0 + T_r, T_0 + T_r + T_c]$, i.e.,
there exists a $U_0 \in CVAL$ such that $convey(p, < T_0, s_0, \sigma_0 >)$ $\mathbf{at_p}$ U_0.
By assumption, we have $convey(p, < T, s, \sigma >)$ $\mathbf{at_p}$ U.
Since $(T_0, s_0) \sqsubset (T, s)$, $Sequen(p)$ leads to $U_0 < U$.
From $U_0 \in CVAL$, we have $U_0 \geq 0$ and thus $U_0 \in [0, U)$. Hence
$convey(p, < T_0, s_0, \sigma_0 >)$ $\mathbf{at_p}$ $U_0 \wedge U_0 \in [0, U)$, i.e., $deliver(p, \sigma_0)$ $\mathbf{in_p}$ $[0, U)$.
Since by assumption, $\neg deliver(p)$ $\mathbf{in_p}$ $[0, U)$, this leads to a contradiction. Thus $deliver(q)$ $\mathbf{in_q}$ $[0, V)$ does not hold, i.e., $\neg deliver(q)$ $\mathbf{in_q}$ $[0, V)$ holds. ■

The next lemma expresses that, for any pair of correct processors p and q, if p conveys $< T_1, s_1, \sigma_1 >$ at local time U_1 and $< T_2, s_2, \sigma_2 >$ at local time U_2, q conveys $< T_1, s_1, \sigma_1 >$ at local time V_1 and $< T_2, s_2, \sigma_2 >$ at local time V_2, and there is no update delivered by p in the interval (U_1, U_2), then there is also no update delivered by q in the interval (V_1, V_2).

LEMMA 13 (NO DELIVERY)

$$\begin{aligned}
&convey(p, < T_1, s_1, \sigma_1 >) \ \mathbf{at_p} \ U_1 \wedge convey(p, < T_2, s_2, \sigma_2 >) \ \mathbf{at_p} \ U_2 \wedge \\
&convey(q, < T_1, s_1, \sigma_1 >) \ \mathbf{at_p} \ V_1 \wedge convey(q, < T_2, s_2, \sigma_2 >) \ \mathbf{at_p} \ V_2 \wedge \\
&correct(p) \ \wedge correct(q) \ \wedge \neg deliver(p) \ \mathbf{in_p} \ (U_1, U_2) \rightarrow \\
&\qquad \neg deliver(q) \ \mathbf{in_q} \ (V_1, V_2)
\end{aligned}$$

Proof: Assume that the premise of this lemma holds.
Suppose $deliver(q)$ $\mathbf{in_q}$ (V_1, V_2) holds.
By definition, there exist s and T such that $convey(q, < T, s, \sigma >)$ $\mathbf{in_q}$ (V_1, V_2).
Then there exists a V such that $convey(q, < T, s, \sigma >)$ $\mathbf{at_q}$ $V \wedge V \in (V_1, V_2)$.
By assumption, we have $convey(q, < T_1, s_1, \sigma_1 >)$ $\mathbf{at_p}$ V_1.
Since $V_1 < V$, $Sequen(q)$ leads to $(T_1, s_1) \sqsubset (T, s)$.
Similarly, assumption $convey(q, < T_2, s_2, \sigma_2 >)$ $\mathbf{at_p}$ V_2 and $V < V_2$
lead with $Sequen(q)$ to $(T, s) \sqsubset (T_2, s_2)$.
From $convey(q, < T, s, \sigma >)$ $\mathbf{at_q}$ V, by the all convey lemma 11, we have
$convey(p, < T, s, \sigma >)$ $\mathbf{in_p}$ $[T + T_r, T + T_r + T_c]$,
i.e., there exists a U such that $convey(p, < T, s, \sigma >)$ $\mathbf{at_p}$ U holds.
Assumption $convey(p, < T_1, s_1, \sigma_1 >)$ $\mathbf{at_p}$ U_1 and $(T_1, s_1) \sqsubset (T, s)$
lead by $Sequen(p)$ to $U_1 < U$.
Similarly, assumption $convey(p, < T_2, s_2, \sigma_2 >)$ $\mathbf{at_p}$ U_2 and $(T, s) \sqsubset (T_2, s_2)$
lead by $Sequen(p)$ to $U < U_2$.
Thus $convey(p, < T, s, \sigma >)$ $\mathbf{at_p}$ $U \wedge U \in (U_1, U_2)$.
Then, by definition, $deliver(p, \sigma)$ $\mathbf{in_p}$ (U_1, U_2).
But, by assumption, we have $\neg deliver(p)$ $\mathbf{in_p}$ (U_1, U_2), leading to a contradiction.
Hence $deliver(q, \sigma)$ $\mathbf{in_q}$ (V_1, V_2) does not hold, i.e., $\neg deliver(q)$ $\mathbf{in_q}$ (V_1, V_2).
■

This leads to the following theorem.

THEOREM 3 (ORDER)

$$correct(p) \wedge correct(q) \rightarrow \forall U \exists V : List(p, U) \subseteq List(q, V)$$

Proof: For any clock value $U \in CVAL$, assume $\langle \sigma_1, \ldots, \sigma_k \rangle \in List(p, U)$.
We prove that there exists a V such that $\langle \sigma_1, \ldots, \sigma_k \rangle \in List(q, V)$.
By definition, there exist $k \in I\!N^+$, $U_1, \ldots, U_k$ such that
$U_1 \leq \ldots \leq U_k < U$, $deliver(p, \sigma_i)$ $\mathbf{at_p}$ U_i, for $i = 1, \ldots, k$,
$\neg deliver(p)$ $\mathbf{in_p}$ (U_j, U_{j+1}), for $j = 1, \ldots, k-1$, and $\neg deliver(p)$ $\mathbf{in_p}$ $[0, U_1)$.
From $deliver(p, \sigma_i)$ $\mathbf{at_p}$ U_i, there exist s_i and T_i such that
$convey(p, < T_i, s_i, \sigma_i >)$ $\mathbf{at_p}$ U_i holds. Let $V = U + T_c$.
We show, by induction on k, that there exist $V_1, V_2, \ldots, V_k$ such that
$V_1 \leq V_2 \leq \ldots \leq V_k < V$, $convey(q, < T_i, s_i, \sigma_i >)$ $\mathbf{at_q}$ V_i, for $i = 1, \ldots, k$,
$\neg deliver(q)$ $\mathbf{in_q}$ (V_j, V_{j+1}), for $j = 1, \ldots, k-1$, and $\neg deliver(q)$ $\mathbf{in_q}$ $[0, V_1)$.

- $k = 1$. By assumption, we have $convey(p, < T_1, s_1, \sigma_1 >)$ $\mathbf{at_p}$ U_1 and
  $\neg deliver(p)$ $\mathbf{in_p}$ $[0, U_1)$. By the all convey lemma 11, we obtain
  $convey(p, < T_1, s_1, \sigma_1 >)$ $\mathbf{in_p}$ $[T_1 + T_r, T_1 + T_r + T_c]$ and
  $convey(q, < T_1, s_1, \sigma_1 >)$ $\mathbf{in_q}$ $[T_1 + T_r, T_1 + T_r + T_c]$.
  Thus $U_1 \in [T_1 + T_r, T_1 + T_r + T_c]$. Since $U_1 < U$, we obtain $T_1 + T_r < U$.
  Then there exists a $V_1 \in CVAL$ such that
  $convey(q, < T_1, s_1, \sigma_1 >)$ $\mathbf{at_q}$ $V_1 \wedge V_1 \in [T_1 + T_r, T_1 + T_r + T_c]$.
  Thus $V_1 \leq T_1 + T_r + T_c$ and hence $V_1 < U + T_c$, i.e., $V_1 < V$.
  By the first deliver lemma 12, we also obtain $\neg deliver(q)$ $\mathbf{in_q}$ $[0, V_1)$.

- $k > 1$. By the induction hypothesis, there exist $V_1, V_2, \ldots, V_{k-1}$ such that
  $V_1 \leq V_2 \leq \ldots \leq V_{k-1}$, $convey(q, < T_i, s_i, \sigma_i >)$ $\mathbf{at_q}$ V_i, for $i = 1, \ldots, k-1$,
  $\neg deliver(q)$ $\mathbf{in_q}$ (V_j, V_{j+1}), for $j = 1, \ldots, k-2$, and $\neg deliver(q)$ $\mathbf{in_q}$ $[0, V_1)$.
  By assumption, we have $convey(p, < T_k, s_k, \sigma_k >)$ $\mathbf{at_p}$ U_k.
  By the all convey lemma 11, there exists a V_k such that
  $convey(q, < T_k, s_k, \sigma_k >)$ $\mathbf{at_q}$ $V_k \wedge V_k \in [T_k + T_r, T_k + T_r + T_c]$.
  Since $U_{k-1} \leq U_k$, we prove $V_{k-1} \leq V_k$ by the following two cases.

  1. Assume $U_{k-1} < U_k$. By assumption,
     $convey(p, < T_{k-1}, s_{k-1}, \sigma_{k-1} >)$ $\mathbf{at_p}$ U_{k-1} and
     $convey(p, < T_k, s_k, \sigma_k >)$ $\mathbf{at_p}$ U_k.
     Since $U_{k-1} < U_k$, by $Sequen(p)$, we obtain $(T_{k-1}, s_{k-1}) \sqsubset (T_k, s_k)$.
     Since we have $convey(q, < T_{k-1}, s_{k-1}, \sigma_{k-1} >)$ $\mathbf{at_q}$ V_{k-1} and
     $convey(q, < T_k, s_k, \sigma_k >)$ $\mathbf{at_q}$ V_k, this leads by $Sequen(q)$ to $V_{k-1} < V_k$.
  2. Assume $U_{k-1} = U_k$. Suppose $V_{k-1} < V_k$. Then, as above, we obtain $U_{k-1} < U_k$ and hence a contradiction. Similarly, if we assume $V_{k-1} > V_k$ then $U_{k-1} > U_k$ and also a contradiction. Hence $V_{k-1} = V_k$ holds.

Combining these two cases, we obtain $V_{k-1} \leq V_k$.
Similar to the case $k = 1$, we have $U_k \in [T_k + T_r, T_k + T_r + T_c]$ and $U_k < U$. Thus $T_k + T_r < U$. Since $V_k \leq T_k + T_r + T_c$, we have $V_k < U + T_c$, i.e., $V_k < V$. By assumption, $\neg deliver(p)$ $\mathbf{in_p}$ (U_{k-1}, U_k). Then by the no delivery lemma 13, we obtain $\neg deliver(q)$ $\mathbf{in_q}$ (V_{k-1}, V_k).

Hence we have proved that there exist $V_1, V_2, \ldots, V_k$ such that
$V_1 \leq V_2 \leq \ldots \leq V_k < V$, $convey(q, < T_i, s_i, \sigma_i >)$ $\mathbf{at_q}$ V_i, for $i = 1, \ldots, k$,
$\neg deliver(q)$ $\mathbf{in_q}$ (V_j, V_{j+1}), for $j = 1, \ldots, k-1$, and $\neg deliver(q)$ $\mathbf{in_q}$ $[0, V_1)$.
Since $convey(q, < T_i, s_i, \sigma_i >)$ $\mathbf{at_q}$ V_i implies $deliver(q, \sigma_i)$ $\mathbf{at_q}$ V_i, we obtain $deliver(q, \sigma_i)$ $\mathbf{at_q}$ V_i, for $i = 1, \ldots, k$. Therefore $\langle \sigma_1, \ldots, \sigma_k \rangle \in List(q, V)$.
Hence for any U there exists a V, i.e., $V = U + T_c$, such that $List(p, U) \subseteq List(q, V)$.
■

6. Comparison and Conclusion

We have formally proved that the termination, atomicity, and order properties of the protocol hold, provided

1. $D_1 \geq T_r + T_c$, where D_1 is the broadcast termination time (as specified by $TERM$), T_r is the time to ensure that all correct processors have received a message containing an update after it is initiated, and T_c is the time for a correct processor to convey updates to its client processes;

2. $D_2 \geq T_c$, where D_2 is the difference of delivery time of an update by two correct processors (as specified by $ATOM$);

3. $T_r \geq T_s \geq 0$, $T_c \geq 0$, $\delta \geq \gamma > 0$, $\epsilon > 0$, and $\rho \geq 0$, where T_s is the time for a correct processor to send a message to its neighbors, γ and δ are the lower and upper bounds, respectively, of message transmission delay along a link, ϵ and ρ are the maximal deviation and speed difference, respectively, of local clocks of correct processors.

Comparing our paper with (Cristian et al. 1989), the basic ideas of proving properties of the protocol are similar. In their algorithm a processor only relays a message to its neighbors if the message is received by the processor for the first time and it is not a "late message". Actually these two factors do not affect the correctness of the protocol. Adding them to the algorithm only improves the efficiency of the implementation. Thus the informal proof in that paper verifies the protocol without taking these factors into account. We also did this in our formal proof, as can be seen from the $Relay(p)$ property.

From the first correct receiving lemma 8 and the correct receiving lemma 9, we observe that if an update σ is initiated by a processor s at local clock time T, then any correct processor p will receive the message $< T, s, \sigma >$ in less than $(d + m)((1 + \rho)T_s + \delta) + \epsilon$ time units measured on its own clock, where d is the maximal distance between two correct processors and m is the maximal number of faulty processors. Thus $T_r \geq (d+m)((1+\rho)T_s + \delta) + \epsilon$. The corresponding time bound in (Cristian et al. 1989) is $(d+m)\delta+\epsilon$, which equals our bound if we assume $T_s = 0$ and $\rho = 0$ as in (Cristian et al. 1989). Notice that the condition on T_r is only needed for the implementation of the server specification $Spec(p)$, not for the correctness proof of the protocol.

In (Cristian et al. 1989), clock synchronization is assumed for always correct processors. To give a precise proof of the protocol, e.g. in the proof of lemma 2, we needed a more refined clock synchronization assumption for processors which are correct at some time points. Thus we took this assumption as an axiom and derived the assumption of (Cristian et al. 1989) as a lemma.

To prove the atomicity property, we have to show that if a correct processor p delivers σ at some time U, then σ was initiated by some processor s at some clock time T. This has not been proved in (Cristian et al. 1989). We have proved it (in lemma 6) by means of available timing information, using a lower bound for message transmission delay between two correct processors. Therefore we have added a lower bound γ in the bounded communication axiom 5.

There is an implicit assumption in (Cristian et al. 1989) about the drift of local clocks. We have formalized this assumption in axiom 8. This axiom is used in lemma 2 to formulate part of the server specification in terms of the local clock of any correct processor. Together with the other axioms about local clocks, i.e., the synchronization axiom 6 and the monotonic clock axiom 7, this makes it possible to perform the verification in terms of local clock values, similar to the informal reasoning in (Cristian et al. 1989). In contrast with most formal methods, see e.g. (de Bakker et al. 1991), there is no need to refer to global times during the protocol verification. This leads to a convenient and natural calculus.

There is quickly growing literature on the formal verification of real-time and fault-tolerant distributed systems. Closely related to our approach is the work on the interactive proof checker EHDM and its successor PVS. Rushby and von Henke (Rushby and von Henke 1993) use EHDM to check the proofs the interactive convergence clock synchronization algorithm of Lamport and Melliar-Smith (1985). Mechanical verification of a generalized protocol for Byzantine fault-tolerant clock synchronization (Schneider 1987) by using EHDM is described in (Shankar 1992). In future applications of our approach we will certainly investigate the use of such an interactive proof checker.

Observe that the formal method used in our paper is compositional. It enables us to reason with only specifications and abstract from the implementation details. A natural continuation of this work is to derive a correct implementation which satisfies the server specification.

Acknowledgements

Many thanks go to Flaviu Cristian for his interest, stimulation, and valuable comments on preliminary versions of this paper.

References

O. Babaoglu and R. Drumond. Streets of Byzantium: Network architectures for fast reliable broadcast. *IEEE Transactions on Software Engineering 11(6):546–554*, 1985.

J.W. de Bakker, C. Huizing, W.-P. de Roever, and G. Rozenberg(Eds.). *Real-Time: Theory in Practice*. LNCS 600, Springer-Verlag, 1991.

K. Birman and T. Joseph. Reliable communication in the presence of failures. *ACM Transactions on Computer Systems 5(1)*, pages 47–76, 1987.

F. Cristian, H. Aghili, and R. Strong. Approximate clock synchronization despite omission and performance failures and processor joins. In *The 16th International Symposium on Fault-Tolerant Computing*. Wien, Austrian, 1986.

F. Cristian, H. Aghili, and R. Strong. Clock synchronization in the presence of omission and performance failures, and processor joins. In *Global States and Time in Distributed Systems*. Z. Yang and T.A. Marsland (Eds.), IEEE Computer Society Press, 1993.

F. Cristian, H. Aghili, R. Strong, and D. Dolev. Atomic broadcast: From simple message diffusion to Byzantine agreement. In *The 15th Annual International Symposium on Fault-Tolerant Computing*, pages 200 – 206. Ann Arbor, USA, 1985.

F. Cristian, H. Aghili, R. Strong, and D. Dolev. Atomic broadcast: From simple message diffusion to Byzantine agreement. Research Report RJ 5244, IBM Almaden Research Center, 1989.

J.M. Chang and N. Maxemchuck. Reliable broadcast protocols. *ACM Transactions on Computer Systems 2(3)*, pages 251–273, 1984.

F. Cristian. Synchronous atomic broadcast for redundant broadcast channels. *Real-Time Systems 2*, pages 195–212, 1990.

F. Cristian. Comments. *Private Correspondence*, 1993.

J. Hooman. *Specification and Compositional Verification of Real-Time Systems*. LNCS 558, Springer-Verlag, 1991.

L. Lamport and P.M. Melliar-Smith. Synchronizing clocks in the presence of faults. *Journal of the ACM*, 32(1):52–78, 1985.

J. Rushby and F. von Henke. Formal verification of algorithms for critical systems. *IEEE Transactions on Software Engineering*, 19(1):13–23, 1993.

F.B. Schneider. Understanding protocols for Byzantine clock synchronization. Technical report 87-859, Dept. of Computer Science, Cornell University, 1987.

N. Shankar. Mechanical verification of a generalized protocol for Byzantine fault tolerant clock synchronization. In *Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 217–236. J. Vytopil(Ed.), LNCS 571, Springer-Verlag, 1992.

P. Zhou and J. Hooman. A proof theory for asynchronously communicating real-time systems. In *Proc. of the 13th IEEE Real-Time Systems*, pages 177–186. IEEE Computer Society Press, 1992.

J. Zwiers. *Compositionality, Concurrency and Partial Correctness*. LNCS 321, Springer-Verlag, 1989.