Fault-Tolerant Broadcasts and Multicasts: The Problem of Inconsistency and Contamination

Ajei Sarat Gopal
Ph.D Thesis

92-1272
March 1992

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

FAULT-TOLERANT BROADCASTS AND MULTICASTS: THE PROBLEM OF INCONSISTENCY AND CONTAMINATION

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Ajei Sarat Gopal

January 1992

FAULT-TOLERANT BROADCASTS AND MULTICASTS:
THE PROBLEM OF INCONSISTENCY AND CONTAMINATION

Ajei Sarat Gopal, Ph.D.
Cornell University 1992

An increasingly important paradigm for designing fault-tolerant applications for distributed systems is based on processes that communicate exclusively via fault-tolerant broadcasts and multicasts. Most broadcasts that are described in the literature, such as *reliable broadcast*, *causal broadcast*, *atomic broadcast* and the corresponding *multicasts*, specify the behavior of *correct* processes, but do not impose requirements on the behavior of *faulty* processes. Such specifications allow a process that fails during a broadcast to reach an "inconsistent" state (*e.g.*, by omitting the delivery of a message), and to continue execution from that state. This faulty process may later broadcast messages that "contaminate" the correct processes.

In this thesis, we argue that such inconsistency and contamination can complicate the design of applications, and we present fault-tolerant broadcast and multicast protocols that prevent inconsistency and contamination.

We begin by formally defining a hierarchy of different types of process inconsistency; these definitions are general, and hence are valid for *any* broadcast specification. Intuitively, contamination is the "spread" of inconsistency from faulty processes to correct processes. We formalize this concept, and show that only two forms of contamination arise from our hierarchy of types of inconsistency.

Atomic broadcast and *atomic multicast* are powerful communication abstractions that are central to many systems (*e.g.*, *Isis*, and IBM's *HAS*), and to Lamport's *state machine* approach to fault-tolerance. Using our general definitions of inconsistency and contamination, we derive necessary and sufficient conditions to prevent inconsistency and/or contamination when processes communicate using atomic broadcast. We also derive similar conditions for atomic multicast. Based on these conditions, we develop atomic broadcast protocols and atomic multicast protocols that prevent inconsistency and/or contamination.

In general, the prevention of inconsistency is a stronger requirement (and more difficult and more expensive to enforce) than the prevention of contamination. We characterize a class of problems for which the prevention of contamination is as good as the prevention of inconsistency. We show that an application that solves a problem in this class under the simplifying assumption that both inconsistency and contamination are prevented, remains correct even if it uses a (less expensive) broadcast protocol that only prevents contamination.

Biographical Sketch

Ajei Gopal was born in Madras, India. He attended several schools on his way to receiving a Doctorate from Cornell University—Mount St. Mary's in New Delhi, Windsor Grammar School in Windsor, England, the Indian Institute of Technology in Bombay, and the University of Arizona in Tucson. He has worked for Bellcore, and currently works for IBM.

His most notable achievement is jumping out of a plane with a malfunctioning parachute...and surviving!

To my family, especially to Mom and Dad.

Acknowledgements

I am deeply grateful to Sam Toueg, my friend and advisor, who has taught me to "do research" and to "write research." It has been a pleasure and a privilege to work with Sam, and this thesis is a result of our collaboration. I could not have asked for a better advisor.

Many thanks to Keith Marzullo and Joe Mitchell for serving on my committee. Thanks to David Shmoys for proxying for Joe at my defense.

Many people helped me in my stay at Cornell, and in my research. I thank my colleagues at Cornell and IBM who have influenced my thesis work, in particular Ken Birman, Navin Budhiraja, Tushar Chandra, Vassos Hadzilacos, Keith Marzullo, Gil Neiger, Prakash Panangaden, Fred Schneider, Pat Stephenson, Ray Strong and Mark Wood. I thank all my friends at Cornell, in particular Navin, Tushar and their housemates, and Samir Khuller and his housemates for providing me with a place to sleep when it became too late to go home, and my office-mates Steve Mitchell and Mark Wood for making our office such a pleasant place. I also thank Jan Batzer, Debbie Smith and the rest of the staff of the Cornell CS Department.

Thanks to IBM for awarding me a Fellowship that paid for the last two years of my PhD, and for allowing me to complete this thesis after coming on board. Thanks also to the NSF for indirectly funding me through Sam's grants.

I am very grateful to Gideon Frieder, Dean of the CIS Department at Syracuse University, for permitting me to work at SU; this saved me many, many miles of driving. Thanks also to Anand Rangachari for the coffees and the lunches.

Finally, I would like to thank all the members of my family. Minni's parents, Leela and Babu, for treating me like a son, and for their love and support. Oogles for rolling over four times in one day! Aasha for being the sister I always wanted. Inder, my first teacher and advisor, for always being there for me. Mom and Dad, for their love, understanding, confidence and constant encouragement; it is to them I owe everything. Minni for making it all worthwhile, now and forever.

Table of Contents

1 Introduction 1

1.1 Specifications of fault-tolerant broadcasts: Inconsistency and contamination 4

1.2 Implementations of fault-tolerant broadcasts: The consequences of layering 6

1.3 Organization of the thesis 8

2 An Execution Model for Application Protocols 10

2.1 Formal model 10

2.1.1 Process state 10

2.1.2 Broadcasting and delivering messages 10

2.1.3 Application protocols 11

2.1.4 Premature halting 11

2.1.5 Execution of an application protocol 12

2.1.6 History functions 13

2.1.7 Histories 13

2.1.8 Failure sets 14

2.1.9 Notation 14

2.2 Broadcast specifications 15

2.3 Modeling executions 16

2.4 Unsuccessful broadcasts 17

2.5 Successful deliveries 17

2.6 Causal precedence with broadcasts 17

2.7 Application specifications 19

3 Defining Inconsistency and Contamination 20

3.1 Crash behavior 21

3.1.1 Crash behavior in a broadcast history 21

3.1.2 Crash consistent behavior in an application history 23

3.1.3 Solving problems assuming crash behavior 24

3.2 Correctness conditions: Restrictions on faulty processes 27

3.2.1 An anomaly with crash behavior 27

3.2.2 Delivery correctness 29
3.2.3 Visible-broadcast/delivery correctness 31
3.2.4 Broadcast/delivery correctness 33
3.2.5 X-correctness 34
3.3 The correctness hierarchy 35
3.3.1 The independence and fifo-independence properties 36
3.3.2 Hierarchy theorems 38
3.4 Process consistency: Restrictions on faulty processes 40
3.5 Process contamination: Restrictions on correct processes 40
3.5.1 Contamination in an application history 42
3.6 Consistent and contamination-free histories 42

4 Solving Correct Restricted Problems **43**
4.1 The choice property of broadcast specifications 44
4.2 The SD-closed property 45
4.3 The maximal causal prefix 51
4.3.1 Notation 53
4.3.2 Defining the maximal causal prefix 53
4.4 The history hierarchy 56
4.5 Correct restricted problems 57

5 Inconsistency and Contamination with Atomic Broadcast **59**
5.1 Atomic broadcast 59
5.1.1 Notation 60
5.1.2 Consistency with atomic broadcast 60
5.1.3 Contamination with atomic broadcast 68
5.2 Timed atomic broadcasts 70
5.3 The independence property 71
5.4 The choice property 74

6 Extending the Formal Model **75**
6.1 Extended model 76
6.1.1 Process state 76
6.1.2 Sending and receive messages 76
6.1.3 Broadcast protocols 77
6.1.4 Premature halting 77
6.1.5 Execution of an application protocol using a broadcast protocol 78
6.2 Point-to-point message passing 79
6.3 Extended histories 80
6.4 Systems and process failures 81
6.5 Consistency and contamination in the extended model 82
6.6 Broadcast protocols implementing a broadcast specification 84

6.7 Solving a problem 84

7 Atomic Broadcast Protocols **86**
7.1 Preliminaries 86
7.1.1 Communication 87
7.1.2 Overview of results 88
7.2 Atomic broadcast 89
7.3 Atomic broadcast with no contamination 90
7.4 Atomic broadcast with no inconsistency 93
7.4.1 Simple protocol: latency $\Delta + \lambda$ 95
7.4.2 Optimal protocol: latency Δ 99
7.4.3 Ensuring BD-consistency 112

8 Atomic Multicast **114**
8.1 Preliminaries 114
8.2 Inconsistency and contamination with respect to pairwise atomic multicast 117
8.3 Overview of the pairwise atomic multicast protocols 120
8.4 Atomic multicast 122
8.5 Pairwise atomic multicast with no contamination 122
8.5.1 Protocol 1 124
8.5.2 Protocol 2 128
8.5.3 Lower bounds 139
8.6 Pairwise atomic multicast with no inconsistency 142

A Other Fault-tolerant Broadcasts **144**
A.1 Reliable broadcast 144
A.2 Causal broadcast 145
A.3 Causal atomic broadcast 146

Bibliography **148**

List of Figures

1.1 Executing an application protocol 1
1.2 Executing an application protocol—in most systems 3
1.3 Example of processes communicating via atomic broadcast 4
1.4 Example of inconsistency with atomic broadcast 5
1.5 Example of contamination with atomic broadcast 6

2.1 The execution of Π by process p 12

3.1 Illustrating the definition of crash behavior 22
3.2 A segment of p's history 23
3.3 Protocol that uses timed reliable broadcast with latency Δ 25
3.4 Portion of broadcast history B 28
3.5 Portion of broadcast history B' 29
3.6 Broadcast histories B and B' 32
3.7 The correctness hierarchy 35
3.8 The contamination hierarchy 41

4.1 Illustrating the proof of Theorem 4.5 49
4.2 Illustrating the maximal causal prefix 51
4.3 The history hierarchy 56

6.1 The execution of Π by process p 78

7.1 Atomic broadcast protocol 89
7.2 Atomic broadcast protocol that prevents BD-contamination 91
7.3 Atomic broadcast protocol that prevents VBD-inconsistency 96
7.4 Witness broadcast protocol 101
7.5 Atomic broadcast protocol that prevents VBD-inconsistency 107

8.1 Message delivery according to local atomic multicasts 116
8.2 Message delivery cycle in pairwise atomic multicast 117
8.3 Pairwise atomic multicast 120
8.4 Pairwise atomic multicast protocol 122
8.5 Pairwise atomic multicast protocol that prevents contamination . . . 126

8.6 Strong multicast protocol 130
8.7 Pairwise atomic multicast protocol that prevents contamination . . . 135
8.8 Illustration of the lower bound on fault tolerance 140
8.9 Pairwise atomic multicast protocol that prevents inconsistency . . . 143

Chapter 1

Introduction

An increasingly important paradigm for designing fault-tolerant applications for distributed systems is based on processes that communicate exclusively via fault-tolerant broadcasts and multicasts. Such a paradigm, illustrated in Figure 1.1, is supported by the IBM *HAS* project [Cri87], *Isis* [BJ87], the *Lazy Replication* scheme [LLS90] and *Psync* [PBS89].

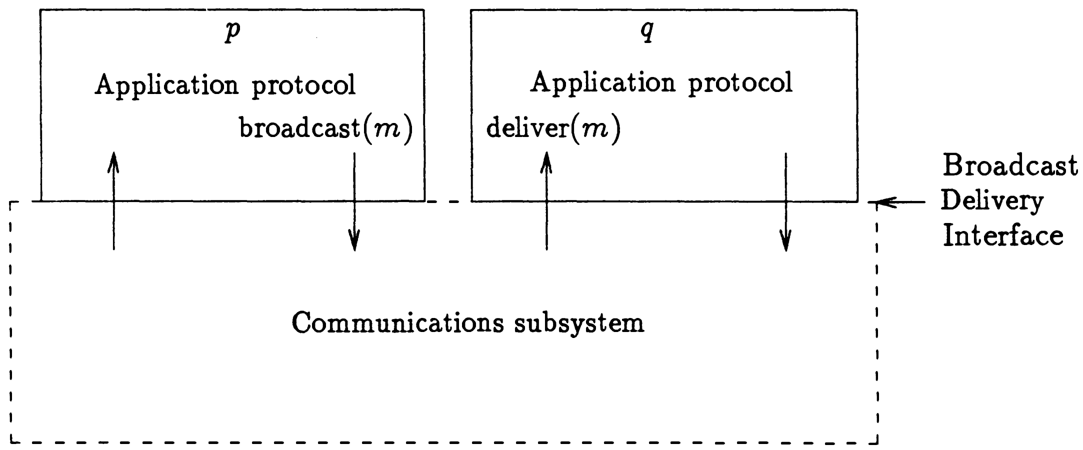


Figure 1.1: Executing an application protocol

In systems where processes are subject to *arbitrary* failures (*i.e.*, malicious or Byzantine failures, [LSP82]), a *faulty* process can arbitrarily change state to an "inconsistent" (*i.e.*, erroneous) state. This faulty process may later "spread" its inconsistency to the rest of the processes by broadcasting a message that reflects its erroneous state. By delivering this message and changing state accordingly, the *correct* processes incorporate the inconsistency of the faulty process into their own state, thereby becoming "contaminated." Clearly, having to tolerate such inconsistency and contamination complicates the design of fault-tolerant applications.

In this thesis, we concentrate on systems where processes are subject to "benign" failures, such as *crash* failures, *omission* failures [Had84,PT84] and *timing* failures [CASD85]. In such systems, faulty processes do not arbitrarily change state. Thus, the current state of every process is a function of the application being executed, the process's initial state and all the messages that the process has *delivered*. Furthermore, every process uses its current state to determine the messages that it must *broadcast* next.

Surprisingly, even in systems subject to benign failures, faulty processes may become inconsistent, and subsequently contaminate the correct processes. Consider the *specifications* of fault-tolerant broadcasts. Most broadcasts that are described in the literature specify the behavior of *correct* processes, but do not impose requirements on the behavior of *faulty* processes. [1] Such specifications allow a process that fails during a broadcast to reach an "inconsistent" state (for example by omitting the delivery of a message), and to continue execution from that state. This faulty process may later broadcast messages that "contaminate" the correct processes. Surprisingly, the usual specifications of *reliable broadcast* [LSP82], *causal broadcast* [BJ87,PBS89], *atomic broadcast* [Lam84,BJ87,CASD85] (and of the corresponding *multicasts*) that can be found in the literature permit both inconsistency and contamination.

Although the occurrence of inconsistency or contamination is not explicitly precluded by the specifications of most fault-tolerant broadcasts, can inconsistency and contamination actually occur in systems subject to benign failures, such as crash failures? As we show below, this is indeed the case. Hence, applications that are based on such broadcasts must be designed to overcome the problems associated with inconsistency and contamination, even in systems subject to benign failures.

Consider the *implementations* of fault-tolerant broadcasts. Such broadcasts are usually implemented by *broadcast protocols*. Thus, when a process executes an application protocol that is based on a fault-tolerant broadcast, that process must also execute a corresponding broadcast protocol. In Figure 1.2, we illustrate such a "layered" structure for a system where the broadcast protocol uses *send/receive* message-passing primitives across *point-to-point* communications links.

With such a structure, a benign failure at the send/receive interface (between

[1] In general, broadcast specifications that restrict the behavior of faulty processes cannot be implemented in systems where processes may fail arbitrarily. However, in the systems we consider in this thesis, it is possible to enforce restrictions on the behavior of faulty processes.

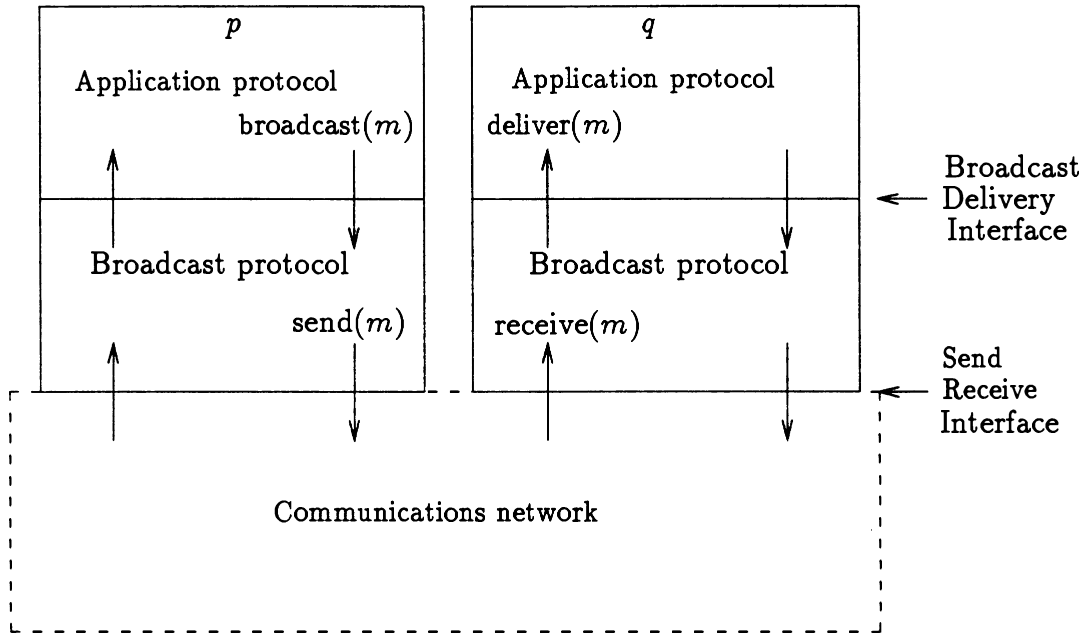


Figure 1.2: Executing an application protocol—in most systems

the broadcast protocol and the communications network) may not result in the same type of failure at the broadcast/delivery interface (between the application protocol and the broadcast protocol). For example, an omission to receive a message does not necessarily result in a corresponding omission to deliver a message. This undesirable behavior will be illustrated with *atomic broadcast.* Informally, atomic broadcast requires that all *correct* processes deliver the *same* messages in the *same* order. We will show that a well-known implementation of atomic broadcast can exacerbate the effect of a process's failure to receive a message, by causing that process to deliver messages out-of-order. Furthermore, we will describe another atomic broadcast protocol that allows the following non-intuitive behavior: a process may behave correctly until it crashes, and still deliver messages out-of-order. This surprising observation seems to contradict the definition of a crash failure.

Thus, we conclude that in general, inconsistency and contamination are possible even in systems subject to benign failures.

In summary, two problems may complicate the design of application protocols that rely on fault-tolerant broadcasts:

- The *specifications* of most fault-tolerant broadcasts allow faulty processes to become inconsistent and to contaminate correct processes.

- Some *implementations* of fault-tolerant broadcasts convert "benign" process failures at the send/receive interface into more "severe" failures at the broadcast/delivery interface.

In this thesis, we first give a *general* definition of process inconsistency and contamination; *i.e.*, we define these concepts with respect to *any* broadcast specification. We then use this general definition to derive specific definitions of inconsistency and contamination with respect to atomic broadcast and to atomic multicast. Finally, we show how to prevent inconsistency and/or contamination with respect to atomic broadcast and atomic multicast.

1.1 Specifications of fault-tolerant broadcasts: Inconsistency and contamination

Most broadcasts that are described in the literature specify the behavior of *correct* processes, but do not impose requirements on the behavior of *faulty* processes. Such specifications allow a process that fails during a broadcast to reach an "inconsistent" state, and to continue execution from that state. This faulty process may later "spread" its inconsistency to the rest of the processes by broadcasting a message that reflects its erroneous state. Such a message is "corrupted," and by delivering this message and changing state accordingly, the *correct* processes incorporate the inconsistency of the faulty process into their own state. Thus, the correct processes become "contaminated."

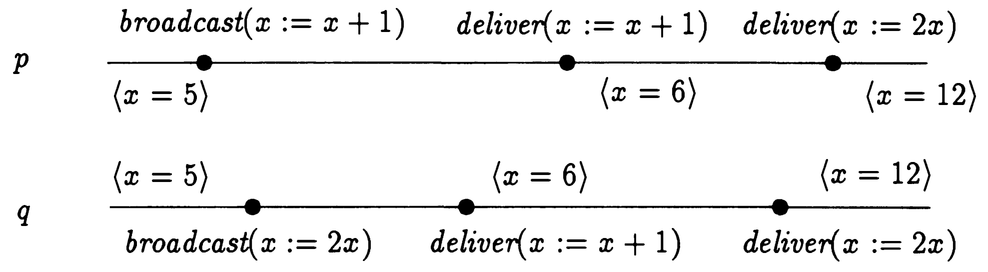


Figure 1.3: Example of processes communicating via atomic broadcast

We use *atomic broadcast* as an example to illustrate inconsistency and contamination. Informally, atomic broadcast requires that all *correct* processes deliver the *same* messages in the *same* order. Suppose a variable x is replicated at two correct processes p and q. Suppose that p broadcasts an instruction to increment x, and q broadcasts an instruction to double x (Figure 1.3). Atomic broadcast guarantees that these instructions are delivered in the same order everywhere; hence, all copies of x remain equal.

However, atomic broadcast does not specify the behavior of *faulty* processes. Thus, a faulty process may reach an inconsistent state in several ways: for example, by omitting to deliver a message m that is delivered by the correct processes, or by delivering an extra message m that is not delivered by the correct processes, or by delivering messages out-of-order. [2]

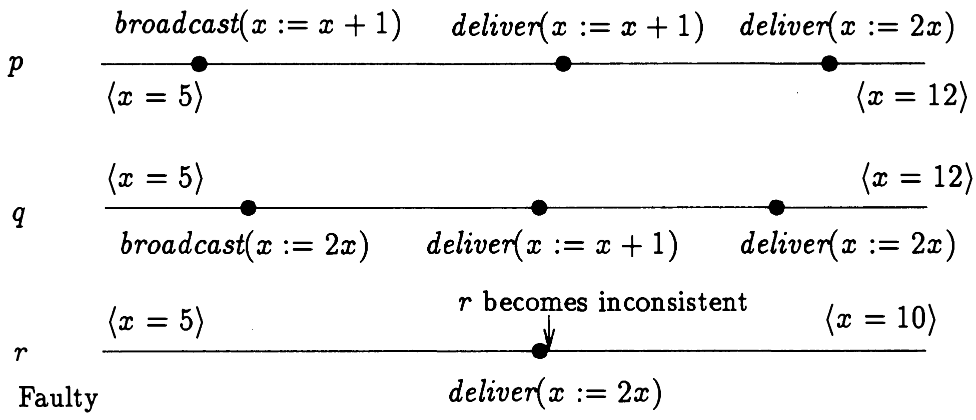


Figure 1.4: Example of inconsistency with atomic broadcast

An example of inconsistency with atomic broadcast is shown in Figure 1.4. The variable x is replicated at processes p, q and r. As in the previous example, p and q broadcast instructions to update x. However, r is faulty, and fails to deliver p's instruction to increment x, but delivers q's instruction to double x. By skipping the increment instruction, r becomes inconsistent and computes an incorrect value of x. Note that since r is faulty, and the specification of atomic broadcast only restricts the behavior of correct processes, the execution shown in Figure 1.4 *does* indeed satisfy the specification of atomic broadcast.

Once r is inconsistent, it can broadcast messages that are based on its inconsistent state and thus contaminate the correct processes. An example is shown in Figure 1.5. Process r uses its value of x to compute and broadcast the value of the replicated

[2] Chapter 5 formally describes the possible sources of inconsistency with respect to atomic broadcast.

variable y, which is supposed to be $3x$. Since r is inconsistent and has incorrectly computed x to be 10, r broadcasts $y := 30$ instead of the correct $y := 36$. When p and q deliver the message $y := 30$ and update their copies of y to be 30, they become contaminated.

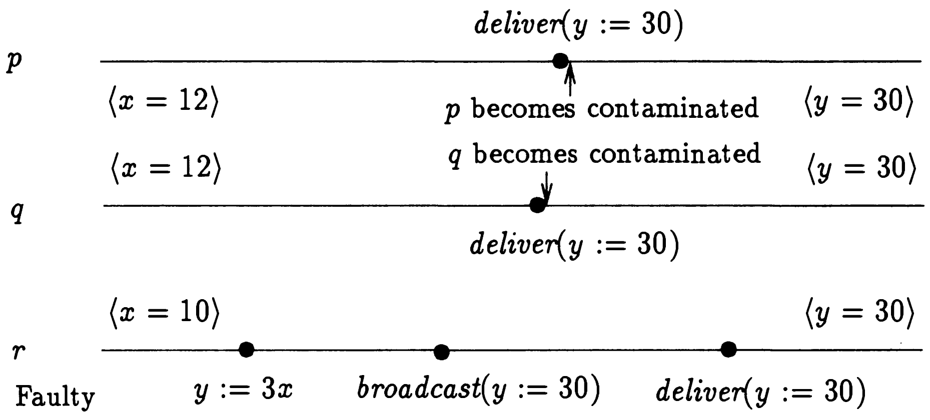


Figure 1.5: Example of contamination with atomic broadcast

Note that r becomes inconsistent by committing a "benign" failure—skipping the delivery of a single message. However, as a result of this undetected failure, r is allowed to subsequently broadcast the erroneous message $y := 30$ instead of the correct $y := 36$. At this point, r appears to commit a "Byzantine-like" failure, even though it actually fails benignly by omission. This clearly complicates the design of application protocols that rely on these broadcasts.

1.2 Implementations of fault-tolerant broadcasts: The consequences of layering

Fault-tolerant broadcasts are usually implemented by a *broadcast protocol* that uses lower-level communications primitives, such as point-to-point message *sends* and *receives* (Figure 1.2). Often, the *broadcast* and/or *delivery* of a message by a broadcast protocol requires the execution of many instructions, including several sends and receives. When a process p attempts to broadcast a message, a "thread" of execution is "forked" within the broadcast protocol layer. This thread executes (or appears to execute) concurrently with p's application protocol, and the broadcast is not complete until the thread terminates.

With such a layered structure, the failure of a process at the send/receive interface (between the broadcast protocol and the communications network) may not result in the same type of failure at the broadcast/delivery interface (between the application protocol and the broadcast protocol). We first show this undesirable behavior with Skeen's atomic broadcast protocol [BJ87]. With this protocol, a faulty process that fails to receive a point-to-point message (at the send/receive interface) may deliver messages out-of-order (at the broadcast/delivery interface). This broadcast protocol is sketched below.

When a process intends to broadcast a message m, it sends m to all processes. When a process receives m, it sends a *tentative sequence number* for m to all processes. Every process chooses m's *final sequence number* to be the maximum of all the tentative sequence numbers for m that it receives. Processes deliver messages in order of increasing final sequence numbers. Thus, if a faulty process fails to receive one of the tentative sequence numbers for m, it computes an incorrect final order number for m, and can deliver m out-of-order.

A coordinator-based atomic broadcast protocol illustrates a more surprising behavior: even if a faulty process behaves correctly until it crashes, it may still deliver messages out-of-order before it crashes. The protocol is as follows. When a process intends to broadcast a message m, it first sends m to a *coordinator*. The coordinator delivers messages in the order in which it receives them, and periodically informs the other processes of this message delivery order. Other processes deliver messages according to this order. When the coordinator crashes, another process takes over as coordinator.

Suppose a coordinator delivers m before m' and then crashes before informing any other process that m should be delivered before m'. The new coordinator cannot determine the order chosen by the faulty coordinator, and may decide that m' should be delivered before m. In this scenario, the faulty coordinator delivered messages out-of-order before it crashed.

The above example shows that a faulty process may become inconsistent before crashing, even though it executes correctly until it crashes. Furthermore, from the time that such a process becomes inconsistent to the time that it crashes, it may broadcast messages and thus contaminate the correct processes. Thus, even if processes can only fail by crashing, inconsistency and contamination can occur. [3]

[3] Clearly, the *prevention* of inconsistency and contamination is much easier with crash failures, than with omission or timing failures.

1.3 Organization of the thesis

In Chapter 2, we define a formal model of execution for an application protocol that is based on processes that communicate exclusively via a fault-tolerant broadcast. Informally, such a protocol relies upon the *specification* of this broadcast, and not on additional properties that may be provided by some particular *implementation* of the broadcast.

In Chapter 3, we give a general definition of inconsistency that applies to any fault-tolerant broadcast. In fact, we identify a hierarchy of three levels of process "consistency," and each one results in a different severity of inconsistency. In this chapter we also define the concept of contamination.

In Chapter 4, we characterize a class of problems for which the prevention of contamination is "as good" as the prevention of inconsistency. To solve a problem in this class, an application protocol can be designed with the simplifying assumption that it will use a broadcast protocol that prevents both inconsistency and contamination. The application protocol remains correct even if it uses a (less expensive) broadcast protocol that only prevents contamination.

In Chapter 5, we use our general definitions of inconsistency and contamination to derive specific definitions for the case of atomic broadcast.

In Chapter 6, we extend our model to include the execution of an underlying broadcast protocol. We define when a broadcast protocol implements a broadcast specification in a particular system. We also formally define when a process fails by *omission*, the failure mode tolerated by the broadcast protocols presented in Chapters 7 and 8.

In Chapter 7, we show how inconsistency and contamination can be prevented by focusing on atomic broadcast. We describe an atomic broadcast protocol that prevents contamination in systems where processes may fail by crashing, or by intermittently omitting to send or receive messages. This protocol has optimal message delivery time, and it only requires a small increase in message size over known atomic broadcast protocols that do not prevent contamination. Furthermore, this protocol does not require an upper bound on the number of processes that may fail. In Chapter 7, we also describe an atomic broadcast protocol that prevents inconsistency. However, this protocol requires that a majority of processes in the system remain correct. We show that this requirement is necessary, and hence the prevention of inconsistency is intrinsically harder than the prevention of contamination.

In Chapter 8, we concentrate on *atomic multicast*. Informally, an atomic multicast is an atomic broadcast in which messages are targeted to specific subsets of processes. We begin by defining a hierarchy of three natural types of atomic multicast. We then concentrate on *pairwise atomic multicast*, and define inconsistency and contamination with respect to such a multicast. We also present a pairwise atomic multicast protocol that prevent inconsistency and pairwise atomic multicast protocols that prevent contamination.

Finally, in Appendix A, we define inconsistency and contamination with *reliable broadcast*, *causal broadcast* and *causal atomic broadcast*.

Chapter 2

An Execution Model for Application Protocols

This chapter defines a formal model of execution for *application protocols*, which are protocols that assume that processes communicate exclusively via fault-tolerant broadcasts. The model introduced in this chapter is used to formally define inconsistency and contamination with respect to the specification of any fault-tolerant broadcast.

2.1 Formal model

Let $\mathcal{P}$ denote the set of processes in a distributed system. The processes communicate by broadcasting *messages*. They execute an *application protocol* which assumes that the processes communicate exclusively via *fault-tolerant broadcasts*.

2.1.1 Process state

Each process maintains a *local clock* taken from the set of positive integers, denoted $\mathcal{I}$. Each process has an *application state* (sometimes abbreviated to *state*) taken from the set $\mathcal{Q}$. The set of states includes the special state $\perp$, called the *premature halt state*.

2.1.2 Broadcasting and delivering messages

The processes communicate via a primitive to *broadcast messages*, and a primitive to *deliver* messages.

A *message* is a tuple of the form $m = (p, c, data)$, where p is the broadcaster of the message, c (the *timestamp*, denoted $ts(m)$) is the time on p's clock at which m was broadcast, and $data$ is the information that p wishes to broadcast. If p is the broadcaster of a message m, we write $p = bc(m)$. Let $\mathcal{M}$ denote the set of all messages, and let $\mathcal{M}^+$ denote the set of all sequences of messages.

If a process p invokes *broadcast* (m) at clock time c, we say that p *broadcasts* m at c. If the result of process p invoking the *deliver* primitive at clock time c is the sequence of messages $D = \langle m_1 \dots m_k \rangle$, then for each message m in D, we say that p *delivers* m at c. We write "p broadcasts (delivers) ϕ at c," to mean that p did not broadcast (deliver) any message at c.

2.1.3 Application protocols

Processes execute an *application protocol* (sometimes abbreviated to *protocol*) Π. Informally, Π specifies the messages to be broadcast by the processes, and the state transitions made by the processes. A protocol Π consists of two functions: the *message function*, denoted Π_m, and the *state transition function*, denoted Π_r.

The message function determines the message to be broadcast next; formally, $\Pi_m : \mathcal{P} \times \mathcal{I} \times \mathcal{Q} \times \mathcal{M}^+ \rightarrow \mathcal{M} \bigcup \phi$. If at time c, p is in state s and delivers the sequence of messages D, then p should broadcast message $\Pi_m(p, c, s, D)$.

The state transition function determines the next state of a process; formally, $\Pi_r : \mathcal{P} \times \mathcal{I} \times \mathcal{Q} \times \mathcal{M}^+ \rightarrow \mathcal{Q}$. If at time c, p is in state s and delivers the sequence of messages D, then p should enter state $\Pi_r(p, c, s, D)$.

2.1.4 Premature halting

Informally, a process may *prematurely halt* execution at any time; when it halts, it makes no further state transitions, does not deliver any more messages, and does not broadcast any more messages.

Such a premature halt is modeled by a state transition to the premature halt state $\perp$. Intuitively, a process can enter the state $\perp$ only after a failure. Thus, no process reaches $\perp$ as the result of a state transition specified by the protocol:

- $\forall p, \forall c, \forall s \neq \perp, \forall D: \ \Pi_r(p, c, s, D) \neq \perp$.

Once a process enters the halt state (*i.e.*, *halts*), it remains in this state:

- $\forall p, \forall c, \forall D: \ \Pi_r(p, c, \perp, D) = \perp$.

Furthermore, once a process halts, it does not broadcast messages.

- $\forall p, \forall c, \forall D: \ \Pi_m(p, c, \perp, D) = \phi$.

2.1.5 Execution of an application protocol

The execution of a protocol Π is illustrated in Figure 2.1. The state of each process is initialized to s_{init} and its clock to 1. At each clock tick, a process delivers a sequence of messages, broadcasts a message, changes state and increments its clock.

```
/*          Initialization          */
   s := s_init ? ⊥;  c := 1          /* p initializes its state and clock             */

/*          Main Loop               */
do forever
   s := s ? ⊥                        /* p's state may change to ⊥ due to a failure    */
   D ≡ deliver()                     /* ∀m ∈ D : p delivers m at c                    */
   s := s ? ⊥                        /* p's state may change to ⊥ due to a failure    */
   broadcast(Π_m(p, c, s, D))        /* p broadcasts a message at c                   */
   s := Π_r(p, c, s, D) ? ⊥          /* p changes its state according to Π or halts   */
   c := c + 1                        /* p increments its clock                        */
od
```

Figure 2.1: The execution of Π by process p

As mentioned above, a process may prematurely halt execution at any time as a result of a failure. This is modeled by the non-deterministic assignment statements used in Figure 2.1. The statement $s := s'\ ?\ \perp$ indicates that the state s non-deterministically changes either to s' (to model a state transition) or to $\perp$ (to model a halt). The statement $s := s\ ?\ \perp$ indicates that the state s non-deterministically changes to $\perp$ (to model a halt) or remains unchanged.

To simplify the presentation, we assume that a process may halt due to a failure either during initialization, or during execution at the following times:

- before delivering messages, or
- after delivering and before broadcasting, or
- after broadcasting a message.

This assumption does not, however, affect the generality of our results.

To simplify the presentation, we have assumed that Π is deterministic; it is straightforward to extend our model and results for non-deterministic application protocols.

2.1.6 History functions

The following functions describe the execution of an application protocol Π:

- The function $\sigma : \mathcal{P} \times \mathcal{I} \rightarrow \mathcal{Q}$ is the *history state function*—$\sigma(p,c)$ is process p's state when p's clock changes to c; *e.g.*, $\sigma(p,1) = s_{init}$.

- The function $\beta : \mathcal{P} \times \mathcal{I} \rightarrow (\mathcal{M} \bigcup \phi)$ is the *history broadcast function*—$\beta(p,c)$ is the message that p broadcasts at clock time c (*i.e.*, when p's clock equals c).

- The function $\delta : \mathcal{P} \times \mathcal{I} \rightarrow \mathcal{M}^+$ is the *history delivery function*—$\delta(p,c)$ is the sequence of messages that p delivers at clock time c.

For all p and c, if $\delta(p,c) = \langle \ldots m \ldots \rangle$, we write $m \in \delta(p,c)$. We say *p delivers m before m'* if $m \in \delta(p,c)$, $m' \in \delta(p,c')$ and either $c < c'$ or ($c = c'$ and $\delta(p,c) = \langle \ldots m \ldots m' \ldots \rangle$). The sequence of messages $\delta(p,1) \cdot \delta(p,2) \cdot \ldots \cdot \delta(p,c)$ is the *sequence of messages that p delivers by time c*. The sequence of messages $\delta(p,1) \cdot \delta(p,2) \cdot \ldots \cdot \delta(p,c) \cdot \ldots$ is the *sequence of messages that p eventually delivers.*

2.1.7 Histories

Let Π be an application protocol, and σ, β and δ be history state, broadcast and delivery functions respectively. The tuple $H = (\Pi, \sigma, \beta, \delta)$ is a *history of* Π. The tuple $A = (\sigma, \beta, \delta)$ is the *application history of H*. The tuple $B = (\beta, \delta)$ is the *broadcast history of H*; B is also the *broadcast history of A*.

A broadcast history $B = (\beta, \delta)$ is *well-formed* if:

- β is a history broadcast function and δ is a history delivery function.
- At all times c, every process p either broadcasts a message of the form $(p, c, —)$ or p does not broadcast a message.
  $\forall p, \forall c: \ \beta(p,c) = (p, c, —)$ or $\beta(p,c) = \phi$.
- At all times c, if a process p delivers a message m at time c, then m is timestamped less than c.
  $\forall p, \forall c: \ m \in \delta(p,c) \implies ts(m) < c$.

Let $\mathcal{B}$ denote the set of all well-formed broadcast histories.

An application history $A = (\sigma, \beta, \delta)$ is *well-formed* if:

- $B = (\beta, \delta)$ is a well-formed broadcast history.
- σ is a history state function.
- Every process is initialized to state s_{init} or halts before initialization.
  $\forall p: \ \sigma(p,1) = s_{init}$ or $\sigma(p,1) = \perp$.

- Once a process halts, it remains halted, and performs no further broadcasts or deliveries.
  $\sigma(p, c) = \perp \implies \sigma(p, c+1) = \perp$ and $\beta(p, c) = \phi$ and $\delta(p, c) = \phi$.

Let $\mathcal{A}$ denote the set of all well-formed application histories.

A history $H = (\Pi, \sigma, \beta, \delta)$ is *well-formed* if:

- $A = (\sigma, \beta, \delta)$ is a well-formed application history.
- Π is a protocol.
- At all times c, every process broadcasts messages according to Π, or halts before broadcasting.
  $\forall p, \forall c : [\beta(p, c) = \Pi_m(p, c, \sigma(p, c), \delta(p, c))]$ or $[\beta(p, c) = \phi \ \wedge \ \sigma(p, c+1) = \perp]$.
- Every process changes state according to Π, or halts.
  $\forall p, \forall c : \ [\sigma(p, c+1) = \Pi_r(p, c, \sigma(p, c), \delta(p, c))]$ or $[\sigma(p, c+1) = \perp]$.

Let $\mathcal{H}$ denote the set of all well-formed histories.

Let H be a well-formed history of Π. Intuitively, until a process p halts, p's state and the messages it broadcasts both depend on the protocol Π, p's initial state, and the messages that p delivers. Thus, a well-formed history models an execution where faulty processes cannot arbitrarily change state, or broadcast arbitrary messages.

2.1.8 Failure sets

A *failure set* is a subset of the set of processes and is denoted by the symbol F. If $p \in F$, we say p is *faulty*; otherwise, p is *correct*. We use $\overline{F}$ to denote the set of correct processes—*i.e.*, $\overline{F} = \mathcal{P} - F$.

2.1.9 Notation

Let $B = (\beta, \delta)$ be a broadcast history. *Silent*(B, p, c) is a predicate that holds if p does not broadcast any messages at or after time c in B—formally, $\forall c' \geq c : \beta(p, c') = \phi$. *Deaf*$(B, p, c)$ is a predicate that holds if p does not deliver any messages at or after time c in B—formally, $\forall c' \geq c : \delta(p, c') = \phi$.

Let $B' = (\beta', \delta')$ be another broadcast history, X be a subset of processes, and c be a clock time. For all processes $p \in X$, for all $c' \leq c$, if p broadcasts the same message in B' at c' as it does in B at c', we write $\beta'_X \overset{c}{=} \beta_X$. Formally:

$$\beta'_X \overset{c}{=} \beta_X \iff \forall p \in X, \ \forall c', \ 1 \leq c' \leq c : \ \beta'(p, c') = \beta(p, c)$$

If $\beta'_X \overset{c}{=} \beta_X$ and $c = \infty$, we omit c and write $\beta'_X = \beta_X$; similarly, if $X = \mathcal{P}$, we omit X and write $\beta' \overset{c}{=} \beta$. We define $\delta'_X \overset{c}{=} \delta_X$ analogously, and write $B'_X \overset{c}{=} B_X$ if and only if $\beta'_X \overset{c}{=} \beta_X \ \wedge \ \delta'_X \overset{c}{=} \delta_X$.

For all processes $p \in X$, for all $c' \leq c$, if p broadcasts the same message in B' at c' as it does in B at c', and p does not broadcast any message after time c in B', we write $\beta'_X \stackrel{c}{=} \beta_X|\phi$. For all processes $p \in X$, for all $c' \leq c$, if p delivers the same messages in B' at c' as it does in B at c', and p does not deliver any message after time c in B', we write $\delta'_X \stackrel{c}{=} \delta_X|\phi$. Formally:

$$\begin{aligned} \beta'_X \stackrel{c}{=} \beta_X|\phi \quad &\iff \quad \beta'_X \stackrel{c}{=} \beta_X \text{ and } \forall p \in X : \ \mathit{Silent}(B', p, c+1) \\ \delta'_X \stackrel{c}{=} \delta_X|\phi \quad &\iff \quad \delta'_X \stackrel{c}{=} \delta_X \text{ and } \forall p \in X : \ \mathit{Deaf}(B', p, c+1) \end{aligned}$$

We write $B'_X \stackrel{c}{=} B_X|\phi$ if and only if $\beta'_X \stackrel{c}{=} \beta_X|\phi$ and $\delta'_X \stackrel{c}{=} \delta_X|\phi$.

We write $\overline{X}$ to denote the set $\mathcal{P} - X$. When it is clear from the context, we write p to denote the singleton set $\{p\}$, and $\overline{p}$ to denote $\overline{\{p\}}$.

2.2 Broadcast specifications

A *broadcast specification* Σ_B is a predicate on broadcast histories and failure sets. If the predicate $\Sigma_B(B, F)$ is satisfied, we say that broadcast history B *satisfies broadcast specification* Σ_B *with respect to failure set* F.

Reliable broadcast is an example of a broadcast specification. Informally, such a broadcast requires that all correct processes deliver the same set of messages, including all messages broadcast by correct processes. More precisely, *reliable broadcast* is defined to be the conjunction of the following *validity*, *agreement*, and *integrity* properties:

- *Validity:* If a correct process broadcasts a message m, then it eventually delivers m.

- *Agreement:* If a correct process delivers a message m, then all correct processes eventually deliver m.

- *Integrity:* For any message m, any process delivers m at most once, and only if m was broadcast by some process.

Formally, *Reliable Broadcast* is a predicate on a broadcast history $B = (\beta, \delta)$ and a failure set F, defined as the following conjunction:

$$\mathit{Reliable\ broadcast}(B, F) \equiv \mathit{Validity}(B, F) \bigwedge \mathit{Agreement}(B, F) \bigwedge \mathit{Integrity}(B, F)$$

$$\begin{aligned} \mathit{Validity}(B, F) \quad &\equiv \quad \forall p \notin F, \forall c, \forall m \neq \phi : \ m \in \beta(p, c) \implies \exists c' : m \in \delta(p, c') \\ \mathit{Agreement}(B, F) \quad &\equiv \quad \forall p \notin F, \forall c, \forall m \neq \phi : \ m \in \delta(p, c) \implies \\ &\qquad\qquad \forall q \notin F, \exists c' : m \in \delta(q, c') \\ \mathit{Integrity}(B, F) \quad &\equiv \quad \forall p, \forall c, \forall m \neq \phi : \ m \in \delta(p, c) \implies \\ &\qquad\qquad \forall c' \neq c : m \notin \delta(p, c') \text{ and } \exists q, \exists c' : m \in \beta(q, c') \end{aligned}$$

We assume the following: Every broadcast specification requires both the integrity and the validity properties.

Informally, a *timed reliable broadcast* is a reliable broadcast with the additional requirement that message delivery time is bounded. More precisely, *timed reliable broadcast* is defined to be the conjunction of the validity, agreement and integrity properties and the following Δ-*timeliness* property.

- Δ-*Timeliness:* If *any* process p delivers a message m at time c on p's clock, then c is at most $ts(m) + \Delta$.

More formally, *Timed Reliable Broadcast* is a predicate on a broadcast history $B = (\beta, \delta)$ and a failure set F, defined as the following conjunction:

$$\begin{aligned} \textit{Timed Reliable Broadcast}(B,F) \equiv\ & \textit{Reliable Broadcast}(B,F)\ \wedge \\ & \Delta\text{-}\textit{Timeliness}(B,F) \end{aligned}$$

$$\Delta\text{-}\textit{Timeliness}(B,F) \ \equiv\ \forall p, \forall c, \forall m \in \delta(p,c) : c \leq ts(m) + \Delta$$

Note that any broadcast specification Σ_B can be augmented with the ∞-timeliness property without altering the semantics of the broadcast—*i.e.*, $\forall B, \forall F : \Sigma_B(B,F)$ if and only if $\Sigma_B(B,F)\ \wedge\ \infty\text{-}\textit{Timeliness}(B,F)$. Thus, without loss of generality, we assume that all broadcast specifications require the ∞-timeliness property.

A broadcast specification Σ_B that requires the Δ-timeliness property is said to have a *latency* of Δ. If Σ_B has a finite latency, it is a *timed broadcast*; otherwise it is an *untimed broadcast.* We assume that all broadcasts have a latency that is greater than 0.

2.3 Modeling executions

A history models the behavior of the processes in an execution of an application protocol. However, a single history may model more than one execution. This is illustrated below for a broadcast history.

Consider a system with two processes, p and q. Let B be a broadcast history in which p broadcasts m and subsequently delivers m. Suppose however that q never delivers m.

From the specification of reliable broadcast, *Reliable Broadcast*$(B, \{q\})$ is satisfied. This models an execution in which p correctly broadcasts and delivers m but q fails to deliver m. However, *Reliable Broadcast*$(B, \{p\})$ is also satisfied. Thus, B also models an execution in which p failed to broadcast m correctly.

Note that B does not model an execution in which both p and q are correct. This is because *Reliable Broadcast*(B, Φ) is not satisfied.

2.4 Unsuccessful broadcasts

Informally, if a process broadcasts a message in a broadcast history, it only means that the process *attempts* to broadcast the message. It does *not* imply that the process *successfully* broadcasts the message.

To explain this more formally, let $B = (\beta, \delta)$ be a well-formed broadcast history that satisfies the specifications of reliable broadcast when the processes in F are assumed to be faulty; that is, *Reliable Broadcast*(B, F). Clearly, a faulty process p (*i.e.*, $p \in F$) can broadcast a message m at some time c, even though no correct process delivers m; *i.e.*, $\beta(p, c) = m$, and for all processes $q \notin F$, for all times c', $m \notin \delta(q, c')$.

2.5 Successful deliveries

Intuitively, if a process delivers a message, then some process previously broadcast that message:

Lemma 2.1 *Let Σ_B be a broadcast specification, $B = (\beta, \delta) \in \mathcal{B}$ and F be such that $\Sigma_B(B, F)$. If some process q delivers a message m of the form $(p, c, —)$ in B, then p broadcasts m at time c in B. That is, $m = (p, c, —)$ and $m \in \delta(q, —)$ implies that $\beta(p, c) = m$.*

Proof: Suppose process q delivers $m = (p, c, —)$ in B; *i.e.*, $m = (p, c, —) \in \delta(q, —)$.

Since Σ_B requires the integrity property (by assumption) and $\Sigma_B(B, F)$ is satisfied, *Integrity*(B, F) is also satisfied. Since $m \in \delta(q, —)$, the integrity property requires that for some r and c', $\beta(r, c') = m$; *i.e.*, m was broadcast by some process r at some local time c' in B. Since B is well-formed and $\beta(r, c') = (p, c, —)$, $r \equiv p$ and $c' \equiv c$. Thus, $\beta(p, c) = (p, c, —)$; *i.e.*, p broadcasts m at time c in B. □

2.6 Causal precedence with broadcasts

Lamport defined *causal precedence* in systems in which processes communicate by sending and receiving point-to-point messages [Lam78]. His definition must be modified to define causal precedence in our model, where processes communicate via broadcasts. Informally, process p at time c *causally precedes* process q at time c' if p broadcasts a message at c, and q delivers that message at c', or if p equals q and $c \leq c'$.

Formally, we define causal precedence in broadcast history B as a relation $\rightarrow_B$ between tuples of the form (p,c), where p is a process, and c is a time.

Definition *Let B be a broadcast history. For all tuples (p,c) and (q,c'), (p,c) causally precedes (q,c') in B, denoted $(p,c) \rightarrow_B (q,c')$ if and only if:*

- *$p = q$ and $c \leq c'$, or*
- *$\beta(p,c) \neq \phi$ and $\beta(p,c) \in \delta(q,c')$, or*
- *$\exists(r,c^*):\ (p,c) \rightarrow_B (r,c^*)$ and $(r,c^*) \rightarrow_B (q,c')$.*

Note that the $\rightarrow_B$ relation is reflexive—*i.e.*, $(p,c) \rightarrow_B (p,c)$. Other than this trivial case, the $\rightarrow_B$ relation does not contain "cycles;" this is a consequence of the following lemma.

Lemma 2.2 *Let $B \in \mathcal{B}$. If $p \neq q$ and $(p,c) \rightarrow_B (q,c')$, then $c < c'$.*

Proof: Follows from the well-formedness of the broadcast history B. □

Lemma 2.3 *Let $B = (\beta,\delta) \in \mathcal{B}$. For all c', for all processes p, q, $p \neq q$, if c is the largest time such that $(p,c) \rightarrow_B (q,c')$, then some process $r \neq p$ delivers a message of the form $(p,c,—)$.*

Proof: Suppose for contradiction that the lemma is false. Let c' be the smallest time such that for some p, q, $p \neq q$, for some c:

Assumption 1: c is the largest time such that $(p,c) \rightarrow_B (q,c')$.

and furthermore:

Assumption 2: For all processes $r \neq p$, process r does not deliver a message of the form $(p,c,—)$.

Since $(p,c) \rightarrow_B (q,c')$ and $p \neq q$, there are two cases to consider. Either:

- for some $c \leq c*,\ c^+ \leq c' :\ \beta(p,c^*) \neq \phi$ and $\beta(p,c^*) \in \delta(q,c^+)$, or
- for some $(r,c^*),\ p \neq r,\ q \neq r :\ (p,c) \rightarrow_B (r,c^*)$ and $(r,c^*) \rightarrow_B (q,c')$.

The proof precedes by a case analysis.

Case: For some $c \leq c*,\ c^+ \leq c' : \beta(p,c^*) \neq \phi$ and $\beta(p,c^*) \in \delta(q,c^+)$: Since $\beta(p,c^*) \in \delta(q,c^+)$, $(p,c^*) \rightarrow_B (q,c^+)$.

Subcase: $c < c^*$: Since $(p,c^*) \rightarrow_B (q,c^+)$ and $c^+ \leq c'$, $(p,c^*) \rightarrow_B (q,c')$. Thus, the assumption that $c < c^*$ contradicts Assumption 1, that c is the largest time such that $(p,c) \rightarrow_B (q,c')$. Thus, c^* must equal c.

Subcase: $c^+ < c'$: Since c is the largest time such that $(p,c) \rightarrow_B (q,c')$, and $c^+ < c'$ and $(p,c) \rightarrow_B (q,c^+)$, c is also the largest time such that $(p,c) \rightarrow_B (q,c^+)$. From this and from Assumption 2, we conclude that c^+ contradicts the choice of c'

(as the smallest time such that Assumptions 1 and 2 are satisfied). Thus c^+ must equal c'.

Subcase: $c^* = c$ and $c^+ = c'$: Thus, $\beta(p,c) \neq \phi$ and $\beta(p,c) \in \delta(q,c')$. Since B is well-formed and $\beta(p,c) \neq \phi$, $\beta(p,c) = (p,c,—)$ and hence $(p,c,—) \in \delta(q,c')$. Thus, we have shown that q delivers a message of the form $(p,c,—)$, contradicting Assumption 2.

Case: For some (r,c^*), $p \neq r$, $q \neq r$: $(p,c) \rightarrow_B (r,c^*)$ and $(r,c^*) \rightarrow_B (q,c')$: Since $q \neq r$ and $(r,c^*) \rightarrow_B (q,c')$, Lemma 2.2 implies that $c^* < c'$. By an argument similar to the one used in the above "subcase" of $c^+ < c'$, we can reach a contradiction to the choice of c'. □

2.7 Application specifications

A *problem* is specified by an *application specification*, a predicate Σ_A on application histories and failure sets. If the predicate $\Sigma_A(A,F)$ is true, we say that application history A *satisfies specification* Σ_A *with respect to failure set* F.

When does a protocol solve a particular problem using a particular fault-tolerant broadcast? For example, what does the assertion "this protocol solves the consensus problem using reliable broadcast" mean? Intuitively, this means that for all executions of the protocol in which processes communicate via reliable broadcast, the specification of the consensus problem is satisfied. This also means that the protocol relies only on the properties required by the *specification* of reliable broadcast, and not on the additional properties provided by any particular *implementation* of reliable broadcast.

We write Π *solves a problem with application specification* Σ_A *using broadcast specification* Σ_B (or Π *solves* Σ_A *using* Σ_B) if the following is true. Let H be any history of Π, with broadcast history B and application history A. If F is any subset of processes, such that B satisfies Σ_B when the processes in F are assumed to be faulty, then A also satisfies Σ_A when the processes in F are assumed to be faulty. Formally:

Definition *Protocol* Π *solves problem* Σ_A *using* Σ_B *if:*

$$\forall H = (\Pi,\sigma,\beta,\delta) \in \mathcal{H} : \; [\forall F : \; \Sigma_B((\beta,\delta),F) \implies \Sigma_A((\sigma,\beta,\delta),F)]$$

Chapter 3

Defining Inconsistency and Contamination

Recall that an application protocol is executed in a "layered" manner; a process executes the application protocol and concurrently executes a broadcast protocol that uses point-to-point message-passing (Figure 1.2). The behavior of processes at the send/receive interface is usually specified by the system in which the processes execute, and cannot be controlled by the application protocol or the broadcast protocol.

Most broadcasts that are described in the literature specify the behavior of correct processes (at the broadcast/delivery interface), but do not impose any requirements on the behavior of faulty processes. However, the behavior of the faulty processes at the broadcast/delivery interface can often be additionally constrained by the broadcast protocol being executed. The designer of an application protocol can only take advantage of such additional restrictions on faulty processes if the *specification* of the broadcast used by the protocol is augmented to include these restrictions.

This chapter concentrates on *defining* general constraints on the behavior of processes at the broadcast/delivery interface with respect to the specification of *any* fault-tolerant broadcast. In Chapter 5, we use the general definitions presented in this chapter to derive corresponding specific constraints on the behavior of processes with respect to atomic broadcast.

In Section 3.1, we define crash behavior; informally, such behavior defines the broadcast/delivery behavior of processes when they communicate via a fault-tolerant broadcast in a system where processes may fail by *crashing*; *i.e.*, every process sends and receives correctly until it halts. Although crash behavior restricts the possible broadcast/delivery behavior of the faulty processes, it also permits some undesirably severe faulty behavior. For example, when processes communicate via atomic broadcast, a faulty process that exhibits crash behavior may deliver messages out-of-order before it halts.

In Section 3.2, we define a hierarchy of three increasingly stringent "correctness conditions" (with respect to any broadcast specification) that further restrict the behavior of faulty processes at the broadcast/delivery interface. In Section 3.3, we derive the relationship between crash behavior and these correctness conditions.

Informally, when a process violates one of the correctness conditions, we say it becomes *inconsistent*. In Section 3.4, we define a hierarchy of three forms of inconsistency with respect to any broadcast specification.

So far, we have discussed restricting the behavior of a faulty process at its own broadcast/delivery interface. However, the erroneous behavior of a faulty process can sometimes be manifested at the broadcast/delivery interface of a *correct* process; for example, a correct process may deliver a message that was broadcast by an inconsistent process. This "spread" of inconsistency from the faulty processes to the correct processes is called *contamination*, and is defined in Section 3.5.

3.1 Crash behavior

In this section, we define what it means for a process to exhibit crash behavior until a particular time in an execution. The formal definition uses the execution's broadcast history (which describes the broadcast/delivery behavior of the processes) and a failure set (which defines a subset of processes assumed to be faulty).

We then extend the above definition to describe when a process is *crash consistent* in an execution; informally, such a process exhibits crash behavior until it halts. The formal definition of crash consistency uses the execution's application history (which includes the broadcast history, and describes the states of the processes) and a failure set.

We say an application history is a *crash history* with respect to a failure set if it models an execution in which every process is crash consistent in that application history with respect to that failure set. Finally, we define what it means to solve a problem "assuming crash behavior."

3.1.1 Crash behavior in a broadcast history

Let Σ_B be a broadcast specification. Let B be a broadcast history, and F be a failure set such that B satisfies Σ_B with respect to F. Informally, for all processes p, for all times c, we say p exhibits crash behavior until time c in B with respect to

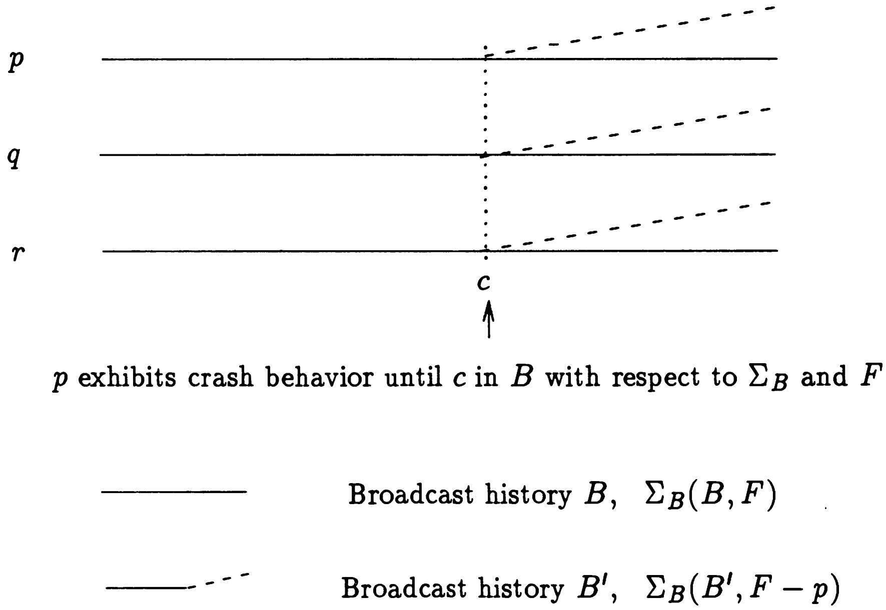


Figure 3.1: Illustrating the definition of crash behavior

Σ_B and F if the following intuitive condition is satisfied: the "global state" [1] of all the processes at time c in B can be reached in an execution (of a broadcast protocol that implements Σ_B) when the processes in $F - p$ are assumed to be faulty (*i.e.*, the processes in $\overline{F}$ *and* process p are assumed to be correct).

Definition *Let $B \in \mathcal{B}$ and F be such that $\Sigma_B(B, F)$. Process p exhibits crash behavior until time c in B with respect to Σ_B and F if there is a $B' \in \mathcal{B}$ such that $B' \stackrel{c}{=} B$ and $\Sigma_B(B', F - p)$. B' is called a crash extension of (B, p, c) with respect to Σ_B and F.*

If B' is a crash extension of (B, p, c) with respect to Σ_B and F, then B and B' are indistinguishable until time c (and hence reach the same "global state" at time c). Furthermore, B' satisfies Σ_B when the processes in $\overline{F}$ *and* process p are assumed to be correct.

We sometimes write "p exhibits crash behavior until c" instead of "p exhibits crash behavior until c in B with respect to Σ_B and F" when the broadcast history B, the broadcast specification Σ_B and the failure set F are clear from the context.

[1] Clearly, the global state at time c is given by the broadcast/delivery behavior of the processes until time c.

3.1.2 Crash consistent behavior in an application history

We now define what it means for a process to be *crash consistent* in an application history with respect to a broadcast specification and a failure set, and for an application history to be a *crash history* with respect to a broadcast specification and a failure set.

Let $A = (\sigma, \beta, \delta)$ and $B = (\beta, \delta)$ be an application and broadcast history respectively. $LB(B,p)$ is a function that denotes the time of p's last broadcast in B, and is defined as follows:

$$LB(B,p) \;=\; \begin{cases} \infty & \text{if } p \text{ broadcasts infinitely often} \\ min\{c \mid Silent(B,p,c+1)\} & \text{otherwise} \end{cases}$$

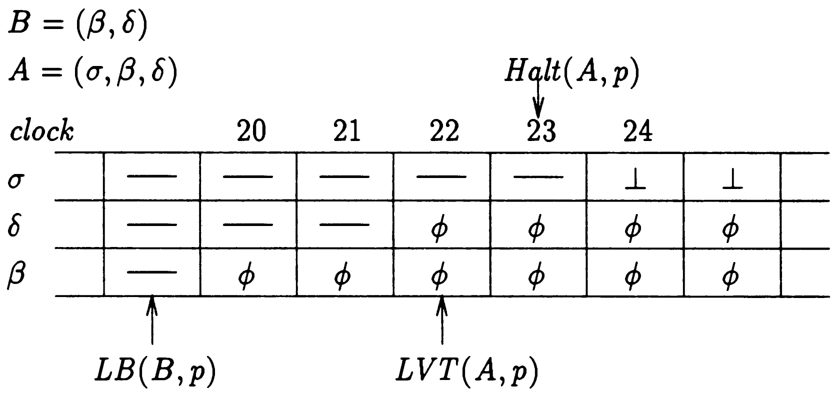


Figure 3.2: A segment of p's history

$Halt(A,p)$ is a function that denotes the time at which p halts, and is defined as follows:

$$Halt(A,p) \;=\; \begin{cases} \infty & \text{if } p \text{ does not halt} \\ min\{c \mid \sigma(p,c+1) = \perp\} & \text{otherwise} \end{cases}$$

Figure 3.2 illustrates these times in a sample history ($LVT(A,p)$ is defined below). The symbol "—" denotes a message or set of messages not equal to ϕ, or a state not equal to $\perp$.

Let p be a process that halts at time c in A—*i.e.*, $Halt(A,p) = c$. For p to be crash consistent in A, it is natural to require that p exhibit crash behavior until time c. However, if p halts before broadcasting at time c, it does not change its state to incorporate any messages that it delivers at time c. In other words, p's actions while its clock reads c cannot affect the state of any process, *including p's own state*. In such a case, it is reasonable to require that p only exhibit crash behavior until time $c - 1$.

The time $max(Halt(A,p) - 1, LB(B,p))$ is called p's *last visible time* in A and denoted $LVT(A,p)$. Note that if $Halt(A,p) = \infty$, then $LVT(A,p) = \infty$. Note also that if p broadcasts a message at some time c, and then halts at time c, then $LVT(A,p)$ is defined to be c.

Definition *Let $A \in \mathcal{A}$, $B = (\beta, \delta)$ be the broadcast history of A, and F be such that $\Sigma_B(B,F)$.*

Process p is crash consistent in A with respect to Σ_B and F if for all $c \leq LVT(A,p)$, p exhibits crash behavior until time c in B with respect to Σ_B and F.

A is a crash consistent history (crash history) with respect to Σ_B and F if for all p, p is crash consistent in A with respect to Σ_B and F.

3.1.3 Solving problems assuming crash behavior

In Chapter 2, we gave a formal definition of a protocol solving a problem using a broadcast specification. We extend that definition to formalize the meaning of statements such as "this protocol solves a particular problem using a particular fault-tolerant broadcast assuming crash consistent behavior." Formally, we say:

Definition *Protocol Π solves Σ_A using Σ_B assuming crash consistent behavior if:*

$$\begin{aligned} &\forall H = (\Pi, \sigma, \beta, \delta) \in \mathcal{H} : \\ &\quad [\forall F : \ \Sigma_B((\beta,\delta),F) \quad \textit{and} \quad (\sigma,\beta,\delta) \textit{ is a crash history with respect to } \Sigma_B \textit{ and } F \\ &\qquad\qquad\qquad \Longrightarrow \ \Sigma_A((\sigma,\beta,\delta),F)] \end{aligned}$$

To illustrate the definitions presented so far, we present a simple application protocol that solves the problem of *atomic updates* using timed reliable broadcast assuming crash consistent behavior. In the atomic updates problem, any process may *propose* an *update* to a replicated variable at any time, and processes *commit* updates as follows:

AU-1 : If a correct process proposes u, then it eventually commits u.

AU-2 : If a process commits u, then all correct processes eventually commit u.

AU-3 : If a process commits u before u', and any process p commits u', then p also commits u before u'.

A more formal description of the atomic update problem, as a predicate *Atomic Update*(A,F) on application histories A and failure sets F, is omitted for simplicity.

/* Process p commits updates at time c */

$U := \{u \mid p$ **delivered** $(—, c - \Delta, u)\}$

$L := sort(U)$ /* in lexicographic order */

commit each $u \in L$ in order

/* Process p proposes update u at time c */

broadcast (p, c, u) /* using timed reliable broadcast */

Figure 3.3: Protocol that uses timed reliable broadcast with latency Δ
Executed by process p at time c

Figure 3.3 describes an application protocol that solves the atomic update problem when it uses timed reliable broadcast with latency Δ and assumes crash consistent behavior. Although the protocol is defined operationally, it can also be described in terms of a protocol transition function and a protocol message function. Such a description is omitted for simplicity.

The protocol relies upon the following constraint on the behavior of the processes. If *any* process p commits an update u at time c, then for all correct processes q, the set of messages with timestamp $c - \Delta$ that p delivers by time c is identical to the set of messages with timestamp $c - \Delta$ that q delivers by time c.

Lemma 3.1 shows that the above constraint is indeed a consequence of crash consistent behavior with respect to timed reliable broadcast. The proof of correctness of the protocol in Figure 3.3 follows easily from Lemma 3.1, and hence is omitted.

Lemma 3.1 *Let Σ_B be the specification of timed reliable broadcast with latency Δ. Let H be a well-formed history of the protocol in Figure 3.3, with application history A and broadcast history B. Suppose F is a failure set such that A is a crash history with respect to Σ_B and F, and $\Sigma_B(B, F)$ is satisfied. For all processes p, if p commits an update at some time c in A, then for all correct processes q (i.e., $q \notin F$), the set of messages with timestamp $c - \Delta$ that p delivers by time c is identical to the set of messages with timestamp $c - \Delta$ that q delivers by time c.*

Proof: Suppose some process p commits an update at some time c in A. Thus, p changes state at time c, and hence $c \leq LVT(A, p)$.

Since A is a crash history and $c \leq LVT(A, p)$, p exhibits crash behavior until c in B with respect to Σ_B and F. By definition of crash behavior, there is a well-formed history B' such that $B' \stackrel{c}{=} B$ and $\Sigma_B(B', F - p)$.

Let q be any correct process (*i.e.*, $q \notin F$). Let M_p and M_q be the sets of messages with timestamp $c - \Delta$ that are delivered in B' by time c by p and q respectively. We first show that $M_p = M_q$.

Let m be any message in M_p; *i.e.*, p delivers m by time c in B' and $ts(m) = c - \Delta$. Since $\Sigma_B(B', F - p)$ is satisfied, and neither p nor q is in $F - p$, the agreement property required by Σ_B implies that q also delivers m in B'. Furthermore, the Δ-timeliness property required by Σ_B implies that q delivers m by time c in B'. Thus $m \in M_q$. Similarly, if $m \in M_q$, we can show that $m \in M_p$. Hence, $M_p = M_q$.

Since $B' \stackrel{c}{=} B$, until time c every process delivers the same sequence of messages at the same time in B and B'. Thus, M_p and M_q are also the sets of messages with timestamp $c - \Delta$ that are delivered in B by time c by p and q respectively. Since $M_p = M_q$, we conclude that the set of messages with timestamp $c - \Delta$ that p delivers by time c is identical to the set of messages with timestamp $c - \Delta$ that q delivers by time c. □

The protocol in Figure 3.3 does not solve the atomic update problem if processes are not crash consistent. Specifically, there is a well-formed history H of the protocol, with application history A and broadcast history B, and a failure set F, such that *Timed Reliable Broadcast*(B, F) is satisfied, A is not a crash history with respect to Σ_B and F, and *Atomic Update*(A, F) is not satisfied.

Consider the following execution of the protocol in a system with two processes p and q. Process q proposes u at time c by broadcasting $m = (q, c, u)$, and proposes u' at time $c' > c$ by broadcasting $m' = (q, c', u')$. Process q delivers m at some time after c, delivers m' at some time after c', and hence commits u at $c + \Delta$ and u' at time $c' + \Delta$. Process p does not deliver m, and delivers message m' at some time after c'; hence p does not commit u and commits u' at time $c' + \Delta$.

Let Σ_B be the specification of timed reliable broadcast with latency Δ. Let H be the history of the above execution; it is easy to show that H is well-formed. Let A and B be the application and broadcast histories of the history H, and let $F = \{p\}$. By construction, it is clear that $\Sigma_B(B, F)$ is satisfied.

Since p commits u' at time $c' + \Delta$, $c' + \Delta \leq LVT(A, p)$; since $c < c'$, $c + \Delta \leq LVT(A, p)$. We show below that A is not a crash history with respect to Σ_B and F by proving that p does not exhibit crash behavior until $c + \Delta$ in B with respect to Σ_B and F.

Consider any well-formed broadcast history B', such that $B' \stackrel{c+\Delta}{=} B$. In both B and B', q broadcasts m at time c, q delivers m by time $c + \Delta$ and p does not deliver m by time $c + \Delta$. Thus, the agreement and Δ-timeliness properties required by Σ_B imply that p and q cannot both be correct. Since q is correct (*i.e.*, $q \notin F$), $\Sigma_B(B', F - p)$ cannot be satisfied.

Thus, for all B' such that $B' \overset{c+\Delta}{=} B$, the predicate $\Sigma_B(B', F - p)$ is not satisfied. Hence, p does not exhibit crash behavior until $c + \Delta$ in B with respect to Σ_B and F, and A is not a crash history with respect to Σ_B and F.

Finally, we show that the application history A does not satisfy the specification of the atomic update problem with respect to the failure set F. From the description of the execution given earlier, it is clear that q commits u before u', and p commits u' and never commits u. Thus, the predicate *AU-3*(A, F) is not satisfied, and hence the predicate *Atomic Update*(A, F) is also not satisfied.

3.2 Correctness conditions: Restrictions on faulty processes

Although crash behavior does indeed restrict the broadcast/delivery behavior of the faulty processes, it also permits some unexpectedly severe faulty behavior. Intuitively, this is because even if a process has already failed by time c, its failure may not be detectable in the "global state" of the processes at time c.

In this section, we define a hierarchy of three increasingly stringent "correctness conditions" that proscribe the undesirable behavior permitted by crash behavior. Intuitively, these conditions are based on the following observation: the failure of a process p by time c may become obvious when p's behavior until time c is compared with the behavior of the correct processes in the entire execution, even if p's failure cannot be detected in the "global state" at time c.

3.2.1 An anomaly with crash behavior

When processes communicate via atomic broadcast, a "natural" assumption is that if a process p exhibits crash behavior until some time c, then p cannot deliver messages out-of-order. However, this assumption is incorrect, as we demonstrate below in an execution of a coordinator-based atomic broadcast protocol. We show that even though a faulty process p exhibits crash behavior until some time c, p's deliveries by time c are out-of-order; that is, p's failure is manifested by time c.

Consider the coordinator-based atomic broadcast protocol informally outlined below. When a process intends to broadcast a message m, it first sends m to a *coordinator*. The coordinator delivers messages (in any order it chooses), and periodically informs the other processes of this message delivery order; the other processes deliver messages according to this order. When the coordinator crashes, another process takes over as coordinator. The new coordinator must first order any messages that the old coordinator did not order, and then begin ordering new messages.

Such coordinator-based atomic broadcast protocols are frequently used in practice because they are message efficient in the absence of failures [BSS90]. However, such a protocol can lead to an execution in which a faulty process delivers a message m before a message m' even though the correct processes deliver m' before m.

Consider the following execution of the protocol where the coordinator p is the only faulty process: p delivers m before m' and then prematurely halts *before* informing any other process that m should be delivered before m'. Thus, the new coordinator, q, cannot know the delivery order chosen by p (or even if p actually delivered m or m'). Hence, q decides that m' should be delivered before m, and instructs the other processes to do the same. In this execution, the faulty coordinator delivers messages out-of-order.

Let B be the broadcast history of the above execution (partially illustrated in Figure 3.4), and let $F = \{p\}$. It is clear that B satisfies atomic broadcast with respect to F. We show below that p exhibits crash behavior until time c in B with respect to atomic broadcast and F.

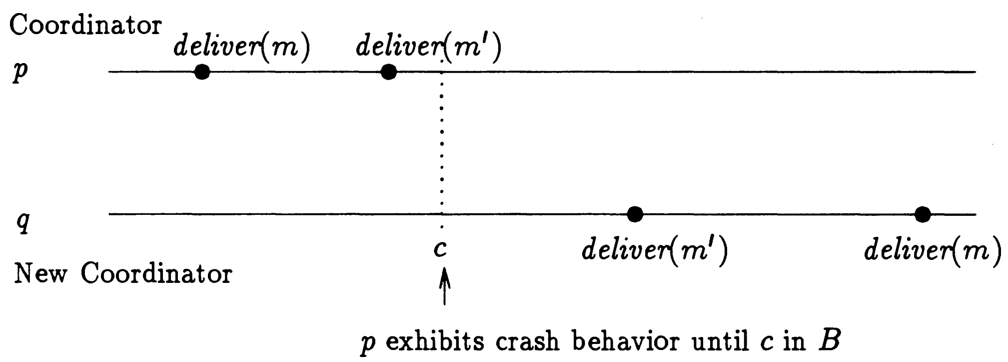


Figure 3.4: Portion of broadcast history B

Suppose B' is a well-formed broadcast history, such that $B' \stackrel{c}{=} B$, and every process other than p delivers m after time c, and then delivers m'. B' is partially illustrated in Figure 3.5. Clearly, B' satisfies atomic broadcast when there are no faulty processes. Thus, B' is a crash extension of (B, p, c) with respect to atomic broadcast and $F = \{p\}$, and hence p exhibits crash behavior until time c in B with respect to atomic broadcast and F.

The broadcast history B' used in the above argument actually models the follow-

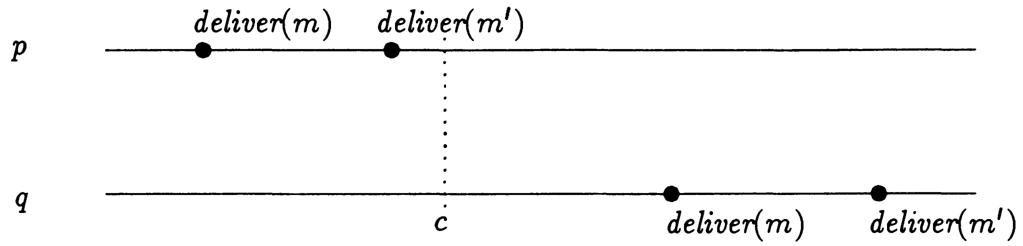


Figure 3.5: Portion of broadcast history B'

ing execution of the coordinator-based atomic broadcast protocol: process p does not halt in the execution described earlier, and instructs the correct processes to deliver m before m'. In such an execution, all the correct processes deliver m before m', and hence p does not deliver messages out-of-order by time c.

In summary, we have presented a broadcast history B that satisfies atomic broadcast with respect to a failure set F. We show that a faulty process $p \in F$ exhibits crash behavior until some time c in B with respect to atomic broadcast and F, even though the message sequence p delivers by time c is "clearly faulty." Intuitively, this shortcoming exists because crash behavior examines the broadcast/delivery behavior of the process only until time c, and does not consider any deliveries made after time c.

3.2.2 Delivery correctness

We propose delivery correctness (the weakest of the three correctness conditions we define) to preclude anomalies with crash behavior, such as the one discussed in the previous section.

Let B be a broadcast history that satisfies a broadcast specification Σ_B with respect to a failure set F. Informally, process p *delivers correctly until time c in B with respect to Σ_B and F* if the sequence of messages p delivers by time c is "consistent" with the sequence of messages that the correct processes *eventually* deliver in B. That is, when compared with the deliveries made by the correct processes during the entire history, p's deliveries until time c *could* have been made by a correct process.

As in the case of crash behavior, the formal definition of delivery correctness is given using another broadcast history B'. Intuitively, B' is restricted in such a way that if $\Sigma_B(B', F - p)$ is satisfied, we can conclude that p delivers correctly until time c in B with respect to Σ_B and F. That is, B' is used to "verify" that p delivers correctly in B.

We say p delivers correctly until c in B with respect to Σ_B and F if there is a well-formed broadcast history B' that satisfies Σ_B with respect to $(F-p)$ (*i.e.*, p remains correct in B'), and:

1. The correct processes broadcast and deliver the same messages at the same time in both B and B'—*i.e.*, $B'_{\overline{F}} = B_{\overline{F}}$.

2. Until time c, the faulty processes, except p, broadcast and deliver the same messages at the same time in both B and B'—*i.e.*, $B'_{F-p} \stackrel{c}{=} B_{F-p}$.

3. Until time c, process p delivers the same messages at the same time in both B and B'—*i.e.*, $\delta'_p \stackrel{c}{=} \delta_p$.

4. Process p's broadcasts in B' are a subset of its broadcasts in B—*i.e.*, $\forall c'$: $\beta'(p,c') = \beta(p,c')$ or ϕ.

   Intuitively, since we are determining if p delivers correctly in B, an unsuccessful broadcast by p in B need not be included in B'. In general, all of p's unsuccessful broadcasts in B cannot be omitted from B'.

   For example, suppose that p broadcasts m in B, and m is delivered by a faulty process $q \neq p$ by time c. Since $\delta'_{F-p} \stackrel{c}{=} \delta_{F-p}$, q also delivers m in B'. Since B' is well-formed, Lemma 2.1 implies that p broadcasts m in B'.

5. Suppose Σ_B is a timed broadcast with latency Δ. Until time $c-\Delta$, p broadcasts the same messages at the same time in both B and B'—*i.e.*, $\beta'_p \stackrel{c-\Delta}{=} \beta_p$.

   Since Σ_B is a timed broadcast with latency Δ, Σ_B requires both the validity and the Δ-timeliness properties (by assumption). Intuitively, stipulating that $\beta'_p \stackrel{c-\Delta}{=} \beta_p$ corresponds to "verifying" that p's deliveries by time c satisfy both these properties. This is explained below.

   If $\Sigma_B(B', F-p)$ is satisfied, then the validity and Δ-timeliness properties imply that p's deliveries by time c in B' must include all the messages that p broadcast by time $c-\Delta$ in B'. Since $\delta'_p \stackrel{c}{=} \delta_p$ (item 3) and $\beta'_p \stackrel{c-\Delta}{=} \beta_p$, we conclude that p's deliveries by time c in B include all the messages that p broadcast by time $c-\Delta$ in B. That is, p's deliveries by time c in B satisfy the validity and Δ-timeliness properties.

Definition *Let Σ_B be a broadcast specification with latency Δ* [2]*. Let $B = (\beta, \delta) \in \mathcal{B}$ and F be such that $\Sigma_B(B, F)$. Process p is delivery correct (*D*-correct) until time c in*

[2] Recall an untimed broadcast has an infinite latency.

B with respect to Σ_B and F if there is a $B' = (\beta', \delta') \in \mathcal{B}$ such that $\Sigma_B(B', F - p)$, and:

- $B'_{\overline{F}} = B_{\overline{F}}$ *and* $B'_{F-p} \stackrel{c}{=} B_{F-p}$
- $\delta'_p \stackrel{c}{=} \delta_p$ *and* $\beta'_p \stackrel{c-\Delta}{=} \beta_p$ *and* $\forall c' > c - \Delta : \beta'(p, c') = \beta(p, c')$ *or* ϕ.

B' is called a D*-extension of (B, p, c) with respect to Σ_B and F.*

Let B' be a D-extension of (B, p, c) with respect to Σ_B and F. If p is correct ($p \notin F$) then $B'_{\overline{F}} = B_{\overline{F}}$ and $B'_F \stackrel{c}{=} B_F$.

3.2.3 Visible-broadcast/delivery correctness

Suppose a process p broadcasts a message m before broadcasting a message m'. Intuitively, if the broadcast of m' is successful, it is desirable that the broadcast of m should also be successful. Indeed, it is reasonable to expect such behavior in practice. [3] However, a process may be "correct" by the definition of delivery correctness, may fail to successfully broadcast a message m, and may successfully broadcast a later message m'. This undesirable behavior, illustrated below, is prevented by *visible-broadcast/delivery correctness.*

Consider a system with two processes, p and q. Let B be a broadcast history (see Figure 3.6) in which by time c, p broadcasts message m followed by message m'. Suppose that p and q eventually deliver m', and neither p nor q delivers m. Clearly, B satisfies the specification of atomic broadcast when $F = \{p\}$—*i.e.*, process p is faulty. The broadcast history B' illustrated in Figure 3.6 is a D-extension of (B, p, c) with respect to Σ_B and F. Thus, p is D-correct until c in B with respect to Σ_B and F, even though it fails to successfully broadcast a message m but successfully broadcasts a later message m'.

Let B be a broadcast history that satisfies Σ_B with respect to failure set F. Informally, process p is *visible-broadcast/delivery correct until time c in B* if, whenever any message p broadcasts by time c is delivered by a process other than p (*i.e.*, "visible" to a process other than p), then all of p's previous broadcasts are successfully delivered. Formally, there is a broadcast history $B' \in \mathcal{B}$ that satisfies Σ_B with respect to $(F - p)$, where B' is a D-extension of (B, p, c) with the additional restrictions that:

1. Process p's broadcasts in B' are a *prefix* of p's broadcasts in B.

[3] Broadcast protocols sometimes enforce such behavior by "piggybacking" messages.

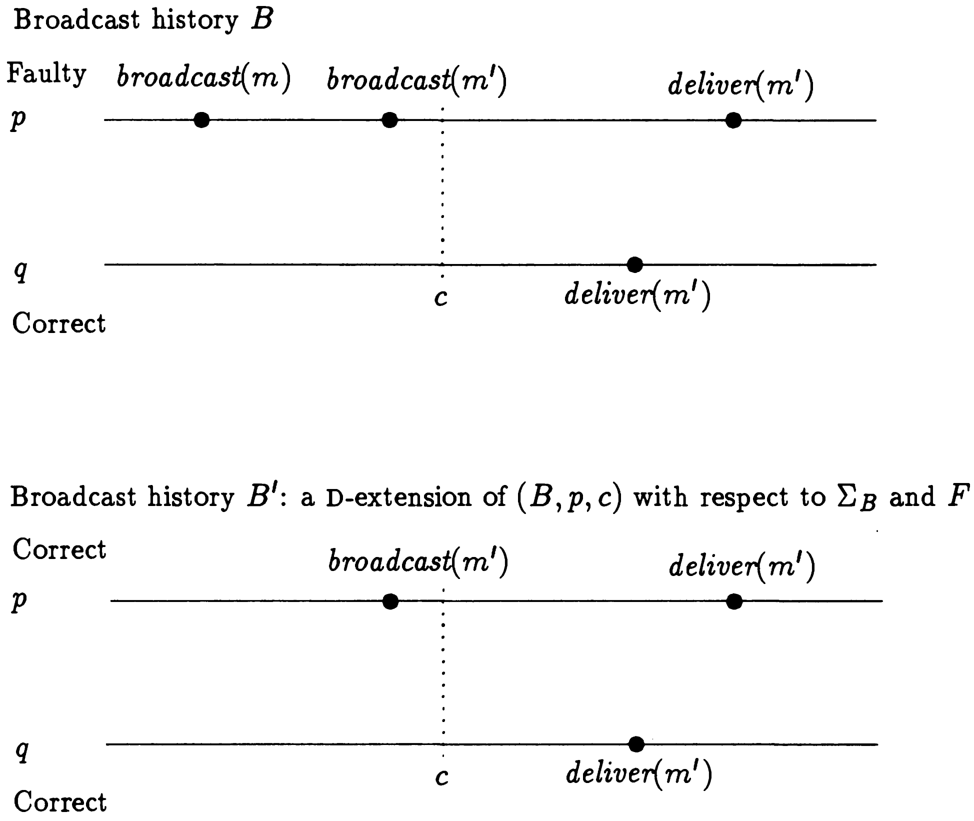


Figure 3.6: Broadcast histories B and B'

2. If, by time c, p broadcasts a message m that is delivered by any process $q \neq p$, then m is included in B'.

Let $Visible(B, p, c)$ denote the largest time $c' \leq c$, such that some process delivers a message that process p broadcasts at time c'. Formally, if $B = (\beta, \delta)$, then:

$$Visible(B, p, c) = max\{c' \mid c' \leq c,\ \beta(p, c') \neq \phi \text{ and } \exists q \neq p:\ \beta(p, c') \in \delta(q, -)\}$$

Definition *Let $B = (\beta, \delta) \in \mathcal{B}$ and F be such that $\Sigma_B(B, F)$. Process p is visible-broadcast/delivery correct (*VBD*-correct) until time c in B with respect to Σ_B and F if there is a $B' = (\beta', \delta') \in \mathcal{B}$ such that $\Sigma_B(B', F - p)$ and:*

- $B'_{\overline{F}} = B_{\overline{F}}$ *and* $B'_{F-p} \stackrel{c}{=} B_{F-p}$
- $\delta'_p \stackrel{c}{=} \delta_p$ *and* $\beta'_p \stackrel{c^*}{=} \beta_p|\phi$ [4], *where* $c^* \geq max(c - \Delta,\ Visible(B, p, c))$.

[4]Recall: $\beta'_p \stackrel{c^*}{=} \beta_p|\phi \iff \beta'_p \stackrel{c^*}{=} \beta_p$ and $Silent(B', p, c^* + 1)$.

B' is called a VBD-*extension of (B,p,c) with respect to Σ_B and F.*

Let B' be a VBD-extension of (B,p,c) with respect to Σ_B and F. The following lemma generalizes the observation that if p broadcasts m at any time c' in B, and m is delivered by a correct process q in B, then until time c', p broadcasts the same messages at the same time in B and B'.

Lemma 3.2 *Let $B = (\beta, \delta) \in \mathcal{B}$ and F be such that $\Sigma_B(B,F)$. Let $B' = (\beta', \delta')$ be a* VBD-*extension of (B,p,c) with respect to Σ_B and F. Let $q \neq p$ and c' be such that either $q \notin F$ or $c' \leq c$. If q delivers a message of the form $(p, c', —)$ in B, then $\exists c^+ \geq c'$ such that $\beta'_p \stackrel{c^+}{=} \beta_p|\phi$.*

Proof: Suppose $q \neq p$ delivers $m = (p, c', —)$ in B. Since $\Sigma_B(B,F)$ is satisfied, Lemma 2.1 implies that p broadcasts m at time c' in B—*i.e.*, $\beta(p,c') = m$.

Case: $c' \leq c$: By hypothesis, B' is a VBD-extension of (B,p,c), and hence $\beta'_p \stackrel{c^*}{=} \beta_p|\phi$ for some $c^* \geq \mathit{Visible}(B,p,c)$. Since p broadcasts m at $c' \leq c$ in B and m is delivered by process $q \neq p$ in B, $\mathit{Visible}(B,p,c) \geq c'$. Since $c^* \geq \mathit{Visible}(B,p,c)$, $c^* \geq c'$. Thus, (by choosing $c^+ \equiv c^*$) we conclude that there is a time $c^+ \geq c'$ such that $\beta'_p \stackrel{c^+}{=} \beta_p|\phi$.

Case: $q \notin F$: By hypothesis, B' is a VBD-extension of (B,p,c), and hence $\delta'_{\overline{F}} = \delta_{\overline{F}}$ and $\Sigma_B(B', F-p)$. Since q delivers m in B and $\delta'_{\overline{F}} = \delta_{\overline{F}}$ and $q \notin F$, q delivers m in B'. Since $\Sigma_B(B', F-p)$, Lemma 2.1 implies that $\beta'(p,c') = m$.

By hypothesis, $\beta'_p \stackrel{c^*}{=} \beta_p|\phi$ for some c^*; that is, $\beta'_p \stackrel{c^*}{=} \beta_p$ and $\mathit{Silent}(B',p,c^*+1)$. Since $\beta'(p,c') = m \neq \phi$ and $\mathit{Silent}(B',p,c^*+1)$, $c' \leq c^*$. Thus, (by choosing $c^+ \equiv c^*$) we conclude that there is a time $c^+ \geq c'$ such that $\beta'_p \stackrel{c^+}{=} \beta_p|\phi$. □

3.2.4 Broadcast/delivery correctness

Broadcast/delivery correctness, the most stringent correctness condition we define, is a natural strengthening of visible-broadcast/delivery correctness. Informally, process p *is broadcast/delivery correct until time c in B with respect to Σ_B and F* if p's broadcast/delivery behavior by time c is "consistent" with the broadcast/delivery behavior of the correct processes in the entire broadcast history B. Formally:

Definition *Let $B = (\beta, \delta) \in \mathcal{B}$ and F be such that $\Sigma_B(B,F)$. Process p is broadcast/delivery correct (*BD-*correct) until time c in B with respect to Σ_B and F if there is a $B' = (\beta', \delta') \in \mathcal{B}$ such that $\Sigma_B(B', F-p)$ and:*

- $B'_{\overline{F}} = B_{\overline{F}}$ *and* $B'_{F-p} \stackrel{c}{=} B_{F-p}$, *and*
- $B'_p \stackrel{c}{=} B_p$ *and* $\exists c^* \geq c:\ \beta'_p \stackrel{c^*}{=} \beta_p|\phi$.

B' is called a BD*-extension of (B,p,c) with respect to Σ_B and F.*

Note that if B' is a BD-extension of (B,p,c) with respect to Σ_B and F, then $B'_{\overline{F}} = B_{\overline{F}}$ and $B'_F \stackrel{c}{=} B_F$.

Let B be a broadcast history and F be such that $\Sigma_B(B,F)$. If p is a correct process ($p \notin F$) then p is always broadcast/delivery correct (and hence also visible-broadcast/delivery and delivery correct).

Lemma 3.3 *Let $B \in \mathcal{B}$ and F be such that $\Sigma_B(B,F)$. For all correct processes p ($p \notin F$), for all c, p is* BD*-correct until c in B with respect to Σ_B and F.*

Proof: Consider any correct process p and any time c. Clearly, B is a BD-extension of (B,p,c) with respect to Σ_B and F. Thus, p is BD-correct until c in B with respect to Σ_B and F. □

Although it is desirable to require that all processes are BD-correct, such a requirement may be unreasonable in practice. In most broadcast protocols, the broadcast of a message by a broadcast protocol requires the execution of many instructions, including several sends and receives. Thus, when a process p attempts to broadcast a message, a "thread" of execution is "forked" within the broadcast protocol layer. This thread executes (or appears to execute) concurrently with p's application protocol, and the broadcast is not complete until the thread terminates.

Thus in any execution, a process can broadcast a message at time c, update local variables, and then halt *before* the successful termination of the broadcast thread; *i.e.*, before the successful completion of the broadcast. In general, such a process would not be BD-correct after time c.

Informally, to ensure that processes are BD-correct, whenever a process p broadcasts a message m, p must be "prevented" from changing state until it is guaranteed that the correct processes will eventually deliver m. Such "blocking" protocols are often slow, and hence undesirable.

3.2.5 X-correctness

Where appropriate, we present our results in terms of "X-correctness," a "generic" correctness condition. Thus, any result stated in terms of X-correctness is valid for all three correctness conditions defined earlier.

3.3 The correctness hierarchy

This section examines the relationship between crash behavior, delivery correctness, visible-broadcast/delivery correctness and broadcast/delivery correctness.

Let $B \in \mathcal{B}$ and F be such that $\Sigma_B(B, F)$. Figure 3.7 illustrates the *correctness hierarchy* for process p at time c. For example, consider the arrow labeled "Fifo-independence property" and "Theorem 3.5" from the box marked "VBD-correct behavior" to the box marked "crash behavior." This indicates that Theorem 3.5 proves that if Σ_B has the fifo-independence property (defined later on), and if p is VBD-correct until c in B with respect to Σ_B and F, then p exhibits crash behavior until c in B with respect to Σ_B and F.

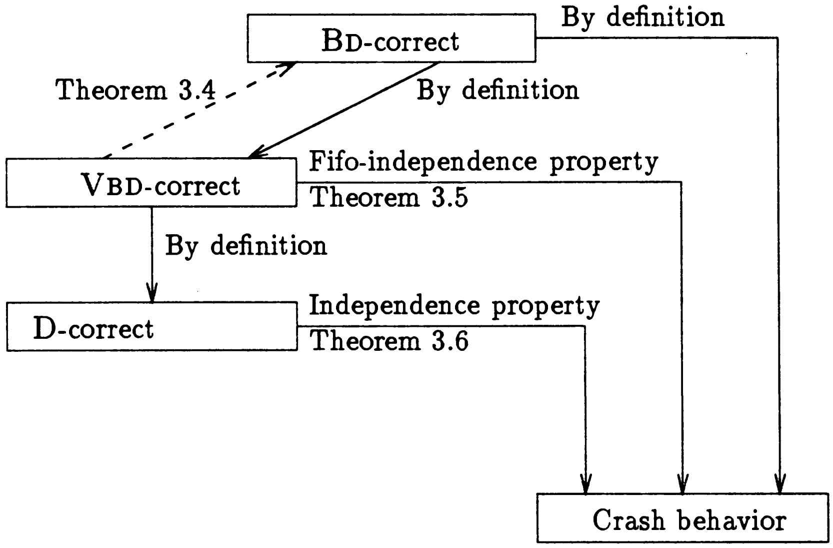


Figure 3.7: The correctness hierarchy

When it is clear from context which broadcast specification Σ_B, broadcast history B, and failure set F are being considered, we write "p is X-correct until c" when we mean "p is X-correct until c in B with respect to Σ_B and F."

Suppose p is BD-correct until c. From the definitions of the correctness conditions and of crash behavior, we conclude that p is both VBD-correct and D-correct until c, and p also exhibits crash behavior until c.

Suppose p is VBD-correct until c. Clearly, p is also D-correct until c. Theorem 3.4 shows that p is also BD-correct until c in certain scenarios; for example, if p broadcasts a message m at time c in B, and some correct process q delivers m in B.

This is indicated by the dashed line in Figure 3.7. Theorem 3.5 shows that p also exhibits crash behavior until c provided the broadcast specification Σ_B satisfies the *fifo-independence* property. This property is described below in Section 3.3.1.

Suppose p is D-correct until c. Theorem 3.6 shows that p also exhibits crash behavior until c provided Σ_B satisfies the *independence* property. This property is described below in Section 3.3.1.

3.3.1 The independence and fifo-independence properties

Intuitively, an untimed broadcast specification Σ_B has the *independence* property if any correct process may broadcast a message at any time, and the order in which any correct process delivers messages is "independent" of the order in which the messages were broadcast.

More formally, let B be a broadcast history and F be a failure set such that $\Sigma_B(B, F)$ is satisfied. Suppose a correct process p does not broadcast a message at time c in B. Then, for all $c' \geq c$, there is another broadcast history B', such that $\Sigma_B(B', F)$ and:

- All processes broadcast the same messages at the same time in B and B', except that p also broadcasts a *new* message $m = (p, c, —)$ at time c in B'.
- Until time c', the processes deliver the same messages at the same time in B and B'.

The first item above corresponds to the intuitive assertion that a correct process can broadcast a message at any time. This is because there are no restrictions on the correct process p, or the time c, or the message m, other than those inherent in the model—namely, at time c, p can broadcast at most one message, and that message must be of the form $(p, c, —)$.

We now show that the above informal definition corresponds to the intuitive assertion that the delivery order is independent of the broadcast order. In particular we show that a correct process q may deliver some message m' before m in the broadcast history B', even though a correct process p broadcasts m before m' in B'.

Suppose that p broadcasts some message m' after time c in B. The first bullet item above implies that p also broadcasts m' after time c in B'; thus p broadcasts m before m' in B'.

Suppose some correct process q delivers m' by time c' in B. The second bullet item above implies that q also delivers m' by time c' in B'. Suppose that q also delivers m in B'. [5] Clearly, the second bullet item above implies that q delivers m

[5] Since p and q are both correct, and $\Sigma_B(B', F)$ is satisfied, if Σ_B requires the agreement property, then q must deliver m in B'.

after time c' in B'; *i.e.*, after delivering m'.

Hence, in the broadcast history B', a correct process q delivers m' before m, even though a correct process p broadcasts m before m'. This corresponds to the intuitive assertion that the delivery order is independent of the broadcast order.

Examples of untimed broadcast specifications that have the independence property are reliable broadcast and atomic broadcast. (In Chapter 5, we prove that atomic broadcast does indeed have the independence property.)

Now suppose Σ_B is a timed broadcast. Intuitively, Σ_B has the independence property if the following condition is satisfied. When a process p broadcasts m and m', the delivery order is independent of the broadcast order provided that m and m' were broadcast less than Δ time apart.

The independence property is formally defined below as a "closure property" of broadcast histories that satisfy Σ_B. We use the following notation:

$$[\beta' = \beta \text{ except } \beta'(p,c) = m] \equiv$$
$$\beta'(p,c) = m \text{ and } \forall (q,c') \neq (p,c): \beta'(q,c') = \beta(q,c')$$

Definition *Broadcast specification Σ_B has the independence property if for all $B = (\beta, \delta) \in \mathcal{B}$, for all F such that $\Sigma_B(B,F)$, for all correct processes p ($p \notin F$), for all times c such that $\beta(p,c) = \phi$, for all messages m of the form $(p,c,—)$, and for all times c', $c \leq c' \leq c+\Delta-1$, there is a $B' = (\beta', \delta') \in \mathcal{B}$, such that $\Sigma_B(B',F)$, $\delta' \overset{c'}{=} \delta$ and $[\beta' = \beta$ except $\beta'(p,c) = m]$.*

Let B', c' and m be as in the above definition. If m is delivered in B' by some process q, then q delivers m *after* time c' in B'.

The fifo-independence property is weaker than the independence property. As in the case of the independence property, a broadcast specification has the fifo-independence property if any process may broadcast a message at any time. However, with fifo-independence, if a correct process p broadcasts m before m', then a correct process q cannot deliver m' before m. Clearly, any broadcast specification that has the independence property also has the fifo-independence property. Examples of broadcast specifications that have the fifo-independence property are causal broadcast and causal atomic broadcast.

Definition *Broadcast specification Σ_B has the fifo-independence property if for all $B = (\beta, \delta) \in \mathcal{B}$, for all F such that $\Sigma_B(B,F)$, for all correct processes p ($p \notin F$), for all times c such that $Silent(B,p,c)$, for all messages m of the form $(p,c,—)$, and for all times c', $c \leq c' \leq c+\Delta-1$, there is a $B' = (\beta', \delta') \in \mathcal{B}$, such that $\Sigma_B(B',F)$, $\delta' \overset{c'}{=} \delta$ and $[\beta' = \beta$ except $\beta'(p,c) = m]$.*

3.3.2 Hierarchy theorems

Theorem 3.4 defines when a VBD-correct process is BD-correct.

Theorem 3.4 *Let $B = (\beta, \delta) \in \mathcal{B}$ and F be such that $\Sigma_B(B, F)$. For all processes p, for all times c, if p is* VBD*-correct until c and some process $q \neq p$ delivers a message m of the form $(p, c, -)$, then p is* BD*-correct until c.*

Proof: Suppose p is VBD-correct until c in B. Let $B' = (\beta', \delta')$ be the VBD-extension of (B, p, c) given by the definition of p being VBD-correct until c in B. Thus, $B' \in \mathcal{B}$, $\Sigma_B(B', F-p)$, $B'_{\overline{F}} = B_{\overline{F}}$, $B'_{F-p} \stackrel{c}{=} B_{F-p}$ and $\delta'_p \stackrel{c}{=} \delta_p$.

Suppose $q \neq p$ delivers $m = (p, c, -)$ in B; Lemma 3.2 implies that there is a $c^* \geq c$ such that $\beta'_p \stackrel{c^*}{=} \beta_p|\phi$. Since $B'_{\overline{F}} = B_{\overline{F}}$ and $B'_{F-p} \stackrel{c}{=} B_{F-p}$ and $\delta'_p \stackrel{c}{=} \delta_p$ and $\beta'_p \stackrel{c^*}{=} \beta_p|\phi$, we conclude that B' is a BD-extension of (B, p, c), and that p is BD-correct until c in B. □

Suppose that the processes communicate using a fault-tolerant broadcast that has the fifo-independence property. Theorem 3.5 shows that VBD-correct behavior implies crash behavior; *i.e.*, VBD-correctness is stronger than crash behavior. Recall that a broadcast specification that has the independence property also has the fifo-independence property. Thus Theorem 3.5 is also valid for those broadcast specifications that have the independence property.

Theorem 3.5 *Suppose Σ_B has the fifo-independence property. Let $B = (\beta, \delta) \in \mathcal{B}$ and F be such that $\Sigma_B(B, F)$. If p is* VBD*-correct until time c, then p exhibits crash behavior at time c.*

Proof: Suppose p is VBD-correct until time c in B. To show that p exhibits crash behavior at time c in B, we prove the following claim:

Claim: If there is a $B^* = (\beta^*, \delta^*) \in \mathcal{B}$, such that:

- $\Sigma_B(B^*, F-p)$ and $\beta^*_{\overline{p}} \stackrel{c}{=} \beta_{\overline{p}}$ and $\delta^* \stackrel{c}{=} \delta$ and $\beta^*_p \stackrel{c^*}{=} \beta_p|\phi$ for some $c^* \geq c - \Delta$

then there is a $B^+ \in \mathcal{B}$ such that $B^+ \stackrel{c}{=} B$ and $\Sigma_B(B^+, F-p)$. In other words, B^+ is a crash extension of (B, p, c) and p exhibits crash behavior until c in B with respect to Σ_B and F.

Suppose the claim is true. It is straightforward to show that the VBD-extension of (B, p, c) (given by the definition of p being VBD-correct until c in B) satisfies the hypothesis of the above claim. Thus, p exhibits crash behavior until c in B with respect to Σ_B and F.

Proof of claim: Let B^* and c^* be given by the hypothesis of the claim. The proof is by induction on $k = c - c^*$.

Basis: Suppose $k \leq 0$—*i.e.*, $c^* \geq c$. Since $\beta^*_p \stackrel{c^*}{=} \beta_p|\phi$ (by the claim's hypothesis) and $c^* \geq c$, $\beta^*_p \stackrel{c}{=} \beta_p$. Since $\delta^* \stackrel{c}{=} \delta$ and $\beta^*_{\overline{p}} \stackrel{c}{=} \beta_{\overline{p}}$ (by the claim's hypothesis), $B^* \stackrel{c}{=} B$. By the claim's hypothesis $\Sigma_B(B^*, F-p)$ and $B^* \in \mathcal{B}$, and hence we conclude that choosing $B^+ \equiv B^*$ proves the claim.

Induction Hypothesis: Assume the claim holds for $c - c^* < k$ where $k \geq 1$.

Induction: Suppose $c - c^* = k$. We show that there is a $B' = (\beta', \delta')$, such that

$$B' \in \mathcal{B} \text{ and } \Sigma_B(B', F-p) \text{ and } \beta'_{\overline{p}} \stackrel{c}{=} \beta_{\overline{p}} \text{ and } \delta' \stackrel{c}{=} \delta \text{ and } \beta'_p \stackrel{c^*+1}{=} \beta_p|\phi \tag{3.1}$$

Since $c - (c^* + 1) < k$, the claim follows from the induction hypothesis.

There are two cases to consider.

Case: $\beta(p, c^*+1) = \phi$: Since $\beta^*_p \stackrel{c^*}{=} \beta_p|\phi$ and $\beta(p, c^*+1) = \phi$, $\beta^*_p \stackrel{c^*+1}{=} \beta_p|\phi$. By choosing $B' \equiv B^*$, we satisfy equation 3.1, thus completing the proof.

Case: $\beta(p, c^*+1) \neq \phi$: B' is constructed as follows. Since $\beta^*_p \stackrel{c^*}{=} \beta_p|\phi$, p does not broadcast any messages after time c^* in B^*—*i.e.*, *Silent*(B^*, p, c^*+1). Since $c^*+1 \leq c$ (by induction) and $c \leq c^* + \Delta$ (by the claim's hypothesis), $c^* + 1 \leq c \leq c^* + \Delta$; *i.e.*, $(c^*+1) \leq c \leq (c^*+1) + \Delta - 1$.

Since B is well-formed and $\beta(p, c^*+1) \neq \phi$, $\beta(p, c^*+1)$ is of the form $(p, c^* + 1, —)$. By the theorem's hypothesis, Σ_B has the fifo-independence property. Since $\Sigma_B(B^*, F-p)$ and *Silent*(B^*, p, c^*+1), and $(c^*+1) \leq c \leq (c^*+1) + \Delta - 1$, the definition of fifo-independence implies that there is a broadcast history $B' \in \mathcal{B}$ such that $\Sigma_B(B', F-p)$, and $\delta' \stackrel{c}{=} \delta^*$ and $[\beta' = \beta^*$ *except* $\beta'(p, c^*+1) = \beta(p, c^*+1)]$.

The proof is complete if we show that B' satisfies equation 3.1. By choice of B', $B' \in \mathcal{B}$ and $\Sigma_B(B', F-p)$. Since $\delta' \stackrel{c}{=} \delta^*$ and $\delta^* \stackrel{c}{=} \delta$ and $[\beta' = \beta^*$ *except* $\beta'(p, c^* + 1) = \beta(p, c^*+1)]$ and $\beta^*_{\overline{p}} \stackrel{c}{=} \beta_{\overline{p}}$, it is easy to show that $\delta' \stackrel{c}{=} \delta$ and $\beta'_{\overline{p}} \stackrel{c}{=} \beta_{\overline{p}}$. Furthermore, since $\beta^*_p \stackrel{c^*}{=} \beta_p|\phi$ (by the claim's hypothesis), we can also show that $\beta'_p \stackrel{c^*}{=} \beta_p$, $\beta'(p, c^*+1) = \beta(p, c^*+1)]$ and *Silent*(B', p, c^*+2); that is $\beta'_p \stackrel{c^*+1}{=} \beta_p|\phi$.

We have shown that B' satisfies equation 3.1, thus completing the proof. □

Suppose the processes communicate by a broadcast that satisfies the independence property. Theorem 3.6 shows that D-correct behavior implies crash behavior; *i.e.*, D-correctness is stronger than crash behavior.

Theorem 3.6 *Suppose Σ_B satisfies the independence property. Let $B \in \mathcal{B}$ and F be such that $\Sigma_B(B, F)$. If p is* D*-correct until time c, then p exhibits crash behavior at time c.*

Proof: Similar to Theorem 3.5. □

3.4 Process consistency: Restrictions on faulty processes

Intuitively, a process p becomes *inconsistent* when its *state* reflects p's violation of a correctness condition. Thus, we formally define inconsistency using *application* histories, which contain both state information and broadcast/delivery information.

Let $A = (\sigma, \beta, \delta)$ and $B = (\beta, \delta)$ be an application and broadcast history respectively. Let p be a process that halts at time c in A—*i.e.*, $Halt(A, p) = c$. For p to be X-consistent in A, it is natural to require that p be X-correct until time c. However, as in the case of crash behavior, if p halts before broadcasting at time c, it is reasonable to require that p be X-correct only until time $c - 1$. This leads to the following definition:

Definition *Let $A \in \mathcal{A}$, $B = (\beta, \delta)$ be the broadcast history of A, and F be such that $\Sigma_B(B, F)$. Process p is* X*-consistent in A with respect to Σ_B and F if and only if for all $c \leq LVT(A, p)$, p is* X*-correct until time c in B with respect to Σ_B and F.*

3.5 Process contamination: Restrictions on correct processes

Intuitively, contamination is the "spread" of "incorrectness" from faulty processes to correct processes. Informally, a correct process p is X-contaminated at time c if the causal past of the tuple (p, c) includes a process that is not X-correct. That is, a correct process p is X-contaminated at time c in B with respect to Σ_B and F if and only if there is a process q and a time c' such that $(q, c') \rightarrow_B (p, c)$ and q is not X-correct until c' in B with respect to Σ_B and F. Note that a faulty process can never be contaminated.

Definition *Let $B \in \mathcal{B}$, F be such that $\Sigma_B(B, F)$, and p be a correct process. Let $B \in \mathcal{B}$ and F be such that $\Sigma_B(B, F)$. Suppose a process p is correct.*
Process p becomes X*-contaminated at time c in B with respect to Σ_B and F by delivering a message m if and only if:*

- *there is a process q and a time c', such that $(q, c') \rightarrow_B (bc(m), ts(m))$ and q is not* X*-correct until c' in B with respect to Σ_B and F, and*
- *for all messages m' that p delivers before m, for all processes q and times c', such that $(q, c') \rightarrow_B (bc(m), ts(m))$, q is* X*-correct until c' in B with respect to Σ_B and F.*

Process p is X*-contaminated at time c in B with respect to Σ_B and F if and only if p becomes* X*-contaminated at time $c' \leq c$ in B with respect to Σ_B and F by delivering a message m.*

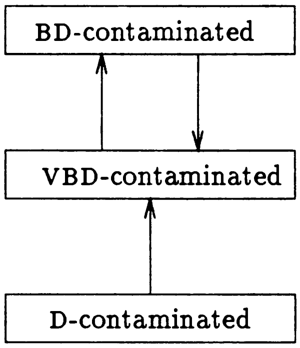


Figure 3.8: The contamination hierarchy

Recall the X-correctness denotes a "generic" correctness condition. Thus, the above definition leads to three forms of contamination—D-contamination, VBD-contamination and BD-contamination—corresponding to our three definitions of correctness—D-correctness, VBD-correctness and BD-correctness. Figure 3.8 illustrates the relationships between these three forms of contamination for any correct process p at any time c in any broadcast history B with respect to any broadcast specification Σ_B and any failure set F. This is called the *contamination hierarchy.*

Theorem 3.7 *Let $B = (\beta, \delta) \in \mathcal{B}$ and F be such that $\Sigma_B(B, F)$. For all correct processes p, for all times c:*

- *If p is* D*-contaminated at c, then p is* VBD*-contaminated at c.*
- *p is* VBD*-contaminated at c if and only if p is* BD*-contaminated at c.*

Proof: Suppose a correct process p is D-contaminated at some time c in B. From the definition of D-contaminated, there is a process q that is not D-correct until time c', and $(q, c') \rightarrow_B (p, c)$. Since q is not D-correct until c', q is not VBD-correct until c'. Hence, p is also VBD-contaminated at time c in B.

Suppose a correct process p is VBD-contaminated at some time c in B. From an argument similar to the one above, p is BD-contaminated at time c in B.

Suppose a correct process p is BD-contaminated at some time c in B. Thus, there is a faulty process $q \neq p$ that is not BD-correct until some time c', and $(q, c') \rightarrow_B (p, c)$.

Let c'' be the largest time such that $(q, c'') \rightarrow_B (p, c)$. Since $q \neq p$, Lemma 2.3 implies that some process $r \neq q$ delivers a message of the form $(q, c'', —)$. By choice of c'', $c'' \geq c'$ and hence q is not BD-correct until c'' in B. Theorem 3.4 implies that q is not VBD-correct until time c'', and hence p is VBD-contaminated at time c in B. □

3.5.1 Contamination in an application history

We now define X-contamination-free processes in application histories.

Definition *Let $A \in \mathcal{A}$, $B = (\beta, \delta)$ be the broadcast history of A, and F be such that $\Sigma_B(B, F)$. Process p is X-contamination-free in A with respect to Σ_B and F if and only if for all c, p is not X-contaminated at time c in B with respect to Σ_B and F.*

Note that a faulty process is, by definition, X-contamination-free.

3.6 Consistent and contamination-free histories

This section defines X-*consistent* and X-*contamination-free* histories, in terms of X-consistent and X-contamination-free processes.

Definition *Let $A \in \mathcal{A}$, $B = (\beta, \delta)$ be the broadcast history of A, and F be such that $\Sigma_B(B, F)$. A is X-consistent (X-contamination-free) with respect to Σ_B and F if and only if for all processes p, p is X-consistent (X-contamination-free) in A with respect to Σ_B and F.*

If A is not X-consistent, we say A is X-*inconsistent.*

Finally, we use the following definition:

Definition *Protocol Π solves Σ_A using Σ_B assuming X-consistent behavior if and only if:*

$$\forall H = (\Pi, \sigma, \beta, \delta) \in \mathcal{H} :$$
$$[\forall F : \Sigma_B((\beta, \delta), F) \quad \textit{and} \quad (\sigma, \beta, \delta) \textit{ is X-consistent with respect to } \Sigma_B \textit{ and } F \implies \Sigma_A((\sigma, \beta, \delta), F)]$$

The word "X-consistent" can be replaced with "X-contamination-free" to yield an analogous definition for X-contamination-free behavior.

Chapter 4

Solving Correct Restricted Problems

In Chapter 3, we defined when a faulty process is inconsistent and when a correct process is contaminated. We showed that preventing inconsistency also prevents contamination; however, preventing contamination does not prevent inconsistency.

In this chapter, we characterize a class of problems, called *correct restricted* problems (*cr-problems*), which are specified by imposing restrictions on the behavior of correct processes, but not on the behavior of faulty processes. We prove the following "substitution theorem" which shows that for cr-problems, the prevention of contamination is "as good" as the prevention of inconsistency. To solve a cr-problem, an application protocol can be designed with the simplifying assumption that it will use a broadcast protocol that prevents both inconsistency and contamination. The application protocol remains correct even if it uses a broadcast protocol that only prevents contamination. Since the prevention of contamination is often less expensive than the prevention of both inconsistency and contamination, such a substitution often improves the performance of the application.

The above "substitution theorem" is valid for broadcast specifications that have the *choice* property. Intuitively, if processes communicate via such a broadcast, then any process may choose to stop broadcasting messages at any time, or any *faulty* process may choose to stop delivering messages at any time. Most broadcasts considered in the literature, such as reliable broadcast, causal broadcast and atomic broadcast have this property.

The "substitution theorem" is valid for any form of inconsistency that is due to a "violation" of a "correctness condition," providing the latter satisfies the SD-*closed* property. We formally define this property, and prove that D-correctness, VBD-correctness and BD-correctness all have the SD-closed property. Thus, our results are valid for the three types of inconsistency introduced in Chapter 3.

Some of the proofs presented in this chapter are given using the *maximal causal prefix* of a history. Let A be a well-formed application history, and F be a failure set. Since all communication between processes is by message broadcasts and deliveries, only a portion of the history is "visible" to (*i.e.*, lies in the causal past of) the correct processes. The application history that is derived in the natural manner from this visible portion of A is called the *maximal causal prefix* of A with respect to F.

Intuitively, we show that if A is contamination-free with respect to Σ_B and F, then the maximal causal prefix of A with respect to F is consistent with respect to Σ_B and F. This result is used to prove the main result of this chapter, the "substitution" theorem: If an application protocol solves a cr-problem Σ_A using a broadcast that satisfies Σ_B and assumes consistent behavior, then the protocol solves Σ_A using Σ_B assuming contamination-free behavior.

4.1 The choice property of broadcast specifications

The results presented in this chapter are valid for those broadcast specifications that have the *choice* property.

Let Σ_B be any broadcast specification. Informally, Σ_B has the *choice* property if for all $B \in \mathcal{B}$, for all F such that $\Sigma_B(B, F)$:

- For all processes p and all times c, p unilaterally decides whether to broadcast a message at c.
  More formally, let p be any process and c be any time such that p broadcasts a message m at c in B. If B' is the broadcast history that is identical to B, except that p does not broadcast m, and no process delivers m, then B' is well-formed and $\Sigma_B(B', F)$ is satisfied.
- For all *faulty* processes p and all times c, p unilaterally decides to stop delivering messages after any time c.
  More formally, for all faulty processes p ($p \in F$) and times c, if B' is the broadcast history that is identical to B, except that p does not deliver any messages after time c in B', then B' is well-formed and $\Sigma_B(B', F)$ is satisfied.

The formal definition of the choice property uses the *Subtract* and *Deafen* operations, defined below using the broadcast histories $B = (\beta, \delta)$ and $B' = (\beta', \delta')$.

- $B' = \mathit{Subtract}(B, M)$: If M is a set of messages, B' is created by removing from B the broadcast and deliveries of all messages in the set M.
  Formally, $B' = \mathit{Subtract}(B, M)$ if and only if:

- $\delta'(p,c) = \{m \mid m \in \delta(p,c) \text{ and } m \notin M\}$ and
- $\beta(p,c) \notin M \implies \beta'(p,c) = \beta(p,c)$, and $\beta(p,c) \in M \implies \beta'(p,c) = \phi$

- $B' = \mathit{Deafen}(B, X)$: Suppose X is a set of tuples of the form (p,c), such that $(p,c) \in X \implies (p, c' \neq c) \notin X$. B' is identical to B, except that for all $(p,c) \in X$, p does not deliver any messages after c in B'.
  Formally, $B' = \mathit{Deafen}(B, X)$ if and only if for all p:
  - $(p, —) \notin X \implies B'_p = B_p$ and
  - $\exists c$, $(p,c) \in X \implies \beta'_p = \beta_p$ and $\delta'_p \stackrel{c}{=} \delta_p|\phi$

Definition *Broadcast specification Σ_B has the choice property if for all $B = (\beta, \delta) \in \mathcal{B}$, for all F such that $\Sigma_B(B, F)$:*
- *For all m, if $B' = \mathit{Subtract}(B, \{m\})$, then $B' \in \mathcal{B}$ and $\Sigma_B(B', F)$, and*
- *For all $p \in F$, for all c, if $B' = \mathit{Deafen}(B, \{(p,c)\})$, then $B' \in \mathcal{B}$ and $\Sigma_B(B', F)$.*

Lemmas 4.1 and 4.2 are generalizations of the above definition; the proofs are straightforward and hence omitted.

Lemma 4.1 *Suppose Σ_B has the choice property. Let $B \in \mathcal{B}$ and F be such that $\Sigma_B(B, F)$. Let M be any set of messages. If $B' = \mathit{Subtract}(B, M)$, then $B' \in \mathcal{B}$ and $\Sigma_B(B', F)$.*

Lemma 4.2 *Suppose Σ_B has the choice property. Let $B \in \mathcal{B}$ and F be such that $\Sigma_B(B, F)$. Let X be a set of tuples of the form (process, time), such that if (p,c) is in X, then p is faulty, and $(p, c' \neq c)$ is not in X; i.e., $X \subseteq \mathcal{P} \times \mathcal{I}$, such that if $(p,c) \in X$ then $p \in F$, and $(p, c' \neq c) \notin X$.*
If $B' = \mathit{Deafen}(B, X)$ then $B' \in \mathcal{B}$ and $\Sigma_B(B', F)$.

Assumption *All broadcast specifications we consider have the choice property.*

Most broadcasts considered in the literature, such as reliable broadcast, causal broadcast, atomic broadcast *etc.*, have the choice property. In Chapter 5, we prove that atomic broadcast has this property.

4.2 The SD-closed property

We will show that for cr-problems, the prevention of inconsistency is "as good as" the prevention of contamination. This result is valid for the inconsistency that results from the "violation" of *any* correctness condition that has the SD-*closed* property. We formally define this property, and prove that D-correctness, VBD-correctness and BD-correctness all have the SD-closed property. Thus, the "substitution theorem" is valid for the three types of inconsistency defined in Chapter 3, and the corresponding forms of contamination.

Suppose Σ_B has the choice property. Let $B = (\beta, \delta) \in \mathcal{B}$ and F be such that $\Sigma_B(B, F)$. Let process p and time c be such that p is X-correct until time c in B with respect to Σ_B and F.

- Informally, X-correctness is *subtract closed (*S*-closed) with respect to* Σ_B if the following condition holds: for all faulty processes r and times c', if B' is the broadcast history derived from B by "subtracting" all the broadcasts r makes after c', then p remains X-correct until c in B'.
  Formally, for all $r \in F$, for all c', if $M = \{\beta(r, c'') \mid c'' \geq c'\}$, then p is X-correct until c in *Subtract*(B, M) with respect to Σ_B and F.

- Informally, X-correctness is *deafen closed (*D*-closed) with respect to* Σ_B if the following condition holds. Suppose that either p is not faulty, or p broadcasts some message at or after time c in B. For all faulty processes r, for all times c' at or after r's last broadcast in B, if B' is a broadcast history derived from B by "deafening" r at c', then p remains X-correct until c in B'.
  Formally, if either $p \notin F$ or $c \leq LB(B, p)$, then for all $r \in F$, for all $c' \geq LB(B, r)$, p is X-correct until c in *Deafen*$(B, \{(r, c')\})$ with respect to Σ_B and F.

  Suppose however that p is faulty and p does not broadcast any message in B at or after time c. In this case, we can choose $r = p$ and $c' < c$ (in the above definition of B'), to give a broadcast history B' in which p is "deafened" at some time before c. In such a case, it is easy to construct a scenario in which p is BD-correct until time c in B with respect to Σ_B and F, but p is not even D-correct until time c in B' with respect to Σ_B and F.

Definition *Suppose* Σ_B *has the choice property.* X*-correctness is subtract-deafen closed (*SD*-closed) with respect to* Σ_B *if and only if* X*-correctness is both* S*-closed and* D*-closed with respect to* Σ_B.

Suppose X-correctness is SD-closed with respect to Σ_B. Let p be a process that is X-correct until some time c in a broadcast history B with respect to Σ_B and some failure set F. Informally, Lemma 4.3 shows that the broadcast history B', obtained by subtracting a suffix of messages broadcast by each one of a subset of faulty processes, is such that p is X-correct until c in B'.

Note that Lemma 4.3 is true even if X-correctness is only S-closed, and not SD-closed. Thus, this lemma is a generalization of the definition of a S-closed correctness condition.

Lemma 4.3 *Suppose* X*-correctness is* SD*-closed with respect to* Σ_B. *Let* $B = (\beta, \delta) \in \mathcal{B}$ *and* F *be such that* $\Sigma_B(B, F)$. *Let* p *be a process that is* X*-correct until some time* c *in* B *with respect to* Σ_B *and* F. *Let* $X \subseteq \mathcal{P} \times \mathcal{I}$, *such that if* $(q, c') \in X$ *then* $q \in F$, *and* $(q, c'' \neq c') \notin X$.

If $M = \{\beta(q, c'') \mid (q, c') \in X$ *and* $c'' \geq c'\}$, *then* p *is* X*-correct until* c *in* $Subtract(B, M)$ *with respect to* Σ_B *and* F.

Proof: Let $M = \{\beta(q, c'') \mid (q, c') \in X \text{ and } c'' \geq c'\}$. The proof that p is X-correct until c in $Subtract(B, M)$ with respect to Σ_B and F is by induction on the cardinality of the set X.

Basis: Suppose $|X| = 0$. Thus, $M = \phi$ and $B = Subtract(B, M)$. Since p is X-correct until c in B (by the lemma's hypothesis), the lemma is trivially true.

Induction Hypothesis: Suppose the lemma is true for all $|X| < k$, for some $k \geq 1$.

Induction: Suppose $|X| = k$. Let (r, c') be any tuple in X, $M' = \{\beta(r, c'') \mid c'' \geq c'\}$, and B' be the broadcast history $Subtract(B, M')$. By the lemma's hypothesis, X-correctness is SD-closed, $\Sigma_B(B, F)$, and $r \in F$. By the definition of X-correctness being SD-closed, p is X-correct until time c in B'. Since (by assumption) Σ_B has the choice property, Lemma 4.1 implies that $B' \in \mathcal{B}$ and $\Sigma_B(B', F)$.

Let $X' = X - \{(r, c')\}$, and $M'' = \{\beta(q, c^*) \mid (q, c'') \in X' \text{ and } c^* \geq c''\}$. Since $B' \in \mathcal{B}$, $\Sigma_B(B', F)$, and p is X-correct until c in B', we conclude from the induction hypothesis that p is X-correct until c in $Subtract(B', M'')$.

Since $M = M' \bigcup M''$ and $B' = Subtract(B, M')$, it is easy to show that $Subtract(B, M) = Subtract(B', M'')$. Since p is X-correct until c in the broadcast history $Subtract(B', M'')$, we conclude that p is X-correct until c in $Subtract(B, M)$ with respect to Σ_B and F. □

Lemma 4.4 generalizes the definition of a D-closed correctness condition (just as the previous lemma generalized the definition of S-closed correctness condition).

Lemma 4.4 *Suppose* X*-correctness is* SD*-closed with respect to* Σ_B. *Let* $B \in \mathcal{B}$ *and* F *be such that* $\Sigma_B(B, F)$. *Let* p *be a process that is* X*-correct until some time* c *in* B. *Let* $X \subseteq \mathcal{P} \times \mathcal{I}$, *such that if* $(q, c') \in X$ *then* $q \in F$, $c' \geq LB(B, q)$ *and* $(q, c'' \neq c') \notin X$.

If either $p \notin F$ *or* $c \leq LB(B, p)$, *then* p *is* X*-correct until* c *in* $Deafen(B, X)$ *with respect to* Σ_B *and* F.

Proof: Suppose that either $p \notin F$ or $c \leq LB(B,p)$. The proof that p is X-correct until c in *Deafen*(B,X) with respect to Σ_B and F is by induction on the cardinality of the set X.

Basis: Suppose $|X| = 0$. Thus, $B = \mathit{Deafen}(B,M)$, and the lemma is trivially true.

Induction Hypothesis: Suppose the lemma is true for all $|X| < k$, for some $k \geq 1$.

Induction: Suppose $|X| = k$. Let (r,c') be any tuple in X, and B' be the broadcast history *Deafen*$(B,\{(r,c')\})$. By the lemma's hypothesis, X-correctness is SD-closed, either $p \notin F$ or $c \leq LB(B,p)$, $\Sigma_B(B,F)$, $r \in F$ and $c' \geq LB(B,r)$. By the definition of X-correctness being SD-closed, p is X-correct until time c in B'. Since (by assumption) Σ_B has the choice property, Lemma 4.2 implies that $B' \in \mathcal{B}$ and $\Sigma_B(B',F)$.

Since the *Deafen* operation does not affect any broadcasts (*i.e.*, $\beta' = \beta$), for all q, $LB(B,q) = LB(B',q)$. Therefore, we conclude that either $p \notin F$ or $c \leq LB(B',p)$ (since $p \notin F$ or $c \leq LB(B,p)$) and that for all tuples $(q,c'') \in X : c'' \geq LB(B',q)$ (since $c'' \geq LB(B,q)$).

Let $X' = X - \{(r,c')\}$. Since $B' \in \mathcal{B}$, $\Sigma_B(B',F)$, and p is X-correct until c in B', we conclude from the induction hypothesis that p is X-correct until c in *Deafen*(B',X').

Since $X = X' \bigcup \{(r,c')\}$ and $B' = \mathit{Deafen}(B,\{(r,c')\})$, it is easy to show that *Deafen*$(B,X) =$ *Deafen*(B',X'). Since p is X-correct until c in *Deafen*(B',X'), we conclude that p is X-correct until c in *Deafen*(B,X) with respect to Σ_B and F. □

We now show that VBD-correctness is SD-closed with respect to any broadcast specification that has the choice property. The proofs that D-correctness and BD-correctness are also SD-closed are similar to the one presented below, and hence are omitted.

Theorem 4.5 *Suppose Σ_B has the choice property.* VBD*-correctness is* SD*-closed with respect to Σ_B.*

Proof: Let $B = (\beta,\delta) \in \mathcal{B}$ and F be such that $\Sigma_B(B,F)$. Suppose a process p is VBD-correct until some time c in B. Let $B^* = (\beta^*,\delta^*)$ be a VBD-extension of (B,p,c). Thus, $B^* \in \mathcal{B}$, $\Sigma_B(B^*,F-p)$, and:

$$\begin{array}{l} \bullet\ B^*_{\overline{F}} = B_{\overline{F}} \text{ and } B^*_{F-p} \stackrel{c}{=} B_{F-p} \\ \bullet\ \delta^*_p \stackrel{c}{=} \delta_p \text{ and } \beta^*_p \stackrel{c^*}{=} \beta_p|\phi \quad \text{for some } c^* \geq max(c-\Delta,\ \mathit{Visible}(B,p,c)) \end{array} \tag{4.1}$$

We first show that VBD-correctness is S-closed with respect to Σ_B (Claim 1), and then show that VBD-correctness is D-closed with respect to Σ_B (Claim 2).

Claim 1: For all $r \in F$, for all c', if $M = \{\beta(r, c'') \mid c'' \geq c'\}$, then p is X-correct until c in $Subtract(B, M)$ with respect to Σ_B and F.

Proof of claim: Let $B' = (\beta', \delta')$ be the broadcast history $Subtract(B, M)$. Since Σ_B has the choice property, Lemma 4.1 implies that $B' \in \mathcal{B}$ and $\Sigma_B(B', F)$ is satisfied. Let $B'' = (\beta'', \delta'') = Subtract(B^*, M)$. Since $\Sigma_B(B^*, F - p)$, Lemma 4.1 implies that $B'' \in \mathcal{B}$ and $\Sigma_B(B'', F - p)$ is satisfied.

It is straightforward to combine equation 4.1 with the expressions for B' in terms of B, and for B'' in terms of B^* (given by the *Subtract* operation), to show that B'' is a VBD-extension of (B', p, c) with respect to Σ_B and F. Since B' is the broadcast history $Subtract(B, F)$, we conclude that p is VBD-correct until c in $Subtract(B, F)$, thus proving the claim.

$$\begin{tikzcd}[column sep=huge, row sep=large]
B \arrow[r, "\text{VBD-extension}"] \arrow[d, "Subtract"'] & B^* \arrow[d, "Subtract"] \\
B' \arrow[r, dashed, "\text{VBD-extension}"'] & B''
\end{tikzcd}$$

Figure 4.1: Illustrating the proof of Theorem 4.5

Figure 4.1 illustrates the structure of the proof; *e.g.*, , the arrow from B to B^* indicates that B^* is a VBD-extension of (B, p, c) with respect to Σ_B and F.

The proof that B'' is a VBD-extension of (B', p, c) with respect to Σ_B and F is a case analysis. Since the proofs of the cases are similar, to avoid repetition, we only present the most interesting case: when $p = r$, we show that $\beta''_p \overset{c^*}{\doteq} \beta'_p|\phi$ for some $c^* \geq max(c - \Delta,\ Visible(B', p, c))$.

Suppose $p = r$. Since B' is obtained from B by removing the broadcasts made by p at or after c', and also by removing the corresponding deliveries, it is easy to show that $Visible(B, p, c) \geq Visible(B', p, c)$.

Note that $\beta^*_p \overset{c^*}{\doteq} \beta_p|\phi$ for some $c^* \geq max(c - \Delta,\ Visible(B, p, c))$ (by equation 4.1), and $\beta'_p \overset{c'}{\doteq} \beta_p|\phi$ and $\beta''_p \overset{c'}{\doteq} \beta^*_p|\phi$ (by assumption, since $p = r$).

- If $c^* \leq c'$, then $\beta'_p \overset{c^*}{\doteq} \beta_p$ and $\beta''_p \overset{c^*}{\doteq} \beta^*_p$, and hence we get $\beta''_p \overset{c^*}{\doteq} \beta'_p|\phi$.
- If $c^* > c'$, then $\beta^*_p \overset{c'}{\doteq} \beta_p$; since for all $c'' > c'$, $\beta''(p, c'') = \phi$ and $\beta'(p, c'') = \phi$, we get $\beta''_p = \beta'_p$.

Since $Visible(B, p, c) \geq Visible(B', p, c)$, and $c^* \geq max(c - \Delta,\ Visible(B', p, c))$, we conclude that $\beta''_p \overset{c^*}{\doteq} \beta'_p|\phi$ for $c^* \geq max(c - \Delta,\ Visible(B', p, c))$. •

Claim 2: If either $p \notin F$ or $c \leq LB(B,p)$, then for all $r \in F$, for all $c' \geq LB(B,r)$, p is VBD-correct until time c in $Deafen(B, \{(r,c')\})$.

Proof of claim: Suppose $p \notin F$ or $c \leq LB(B,p)$. Let r be any faulty process, and let $c' \geq LB(B,r)$. Let $B' = (\beta', \delta')$ be the broadcast history $Deafen(B, \{(r,c')\})$. Since Σ_B has the choice property, Lemma 4.2 implies that $B' \in \mathcal{B}$ and $\Sigma_B(B', F)$ is satisfied.

The proof is a case analysis.

Case: $p = r$: Since $r \in F$, $p \in F$ and $c' \geq LB(B,p)$. Since $c \leq LB(B,p)$, $c \leq c'$. $B' = Deafen(B, \{(p,c')\})$ implies that $\beta' = \beta$, $\delta'_{\overline{p}} = \delta_{\overline{p}}$, and $\delta'_p \stackrel{c'}{\doteq} \delta_p|\phi$; furthermore, $c \leq c'$ implies that $\delta'_p \stackrel{c}{\doteq} \delta_p$. From this and from equation 4.1, we conclude that B^* is a VBD-extension of (B', p, c) with respect to Σ_B and F. Since $B' = Deafen(B, \{(p,c')\})$ and $p = r$, p is VBD-correct until c in $Deafen(B, \{(r,c')\})$.

Case: $p \neq r$: Let $B'' = (\beta'', \delta'') = Deafen(B^*, \{(r,c')\})$. Since Σ_B has the choice property, and $\Sigma_B(B^*, F-p)$, Lemma 4.2 implies that $B'' \in \mathcal{B}$ and $\Sigma_B(B'', F-p)$ is satisfied.

From the definition of *Deafen*:

- $B''_{\overline{r}} = B^*_{\overline{r}}$ and $B'_{\overline{r}} = B_{\overline{r}}$.
- $\beta''_r = \beta^*_r$ and $\beta'_r = \beta_r$ and $\delta''_r \stackrel{c'}{\doteq} \delta^*{}_r|\phi$ and $\delta'_r \stackrel{c'}{\doteq} \delta_r|\phi$.

It is straightforward to combine the above with the equation 4.1, to show that B'' is a VBD-extension of (B', p, c) with respect to Σ_B and F. Since $B' = Deafen(B, \{(r,c')\})$, p is VBD-correct until c in $Deafen(B, \{(r,c')\})$, thus proving the claim.

The proof that B'' is a VBD-extension of (B', p, c) with respect to Σ_B and F is further divided into several parts. For brevity, we only present the most interesting part: we show that $\delta''_r \stackrel{c}{\doteq} \delta'_r$.

Note that $\delta^*_r \stackrel{c}{\doteq} \delta_r$ (equation 4.1, since $r \in F$); thus, the definition of *Deafen* implies that $\delta''_r \stackrel{c'}{\doteq} \delta^*{}_r|\phi$ and $\delta'_r \stackrel{c'}{\doteq} \delta_r|\phi$.

- If $c \leq c'$, then $\delta''_r \stackrel{c}{\doteq} \delta^*{}_r$ and $\delta'_r \stackrel{c}{\doteq} \delta_r$, and we conclude that $\delta''_r \stackrel{c}{\doteq} \delta'_r$.
- If $c > c'$, then $\delta^*_r \stackrel{c'}{\doteq} \delta_r$; since for all $c'' > c'$, $\delta''(r, c'') = \phi$ and $\delta'(r, c'') = \phi$, we get $\delta''_r = \delta'_r$. Thus, we conclude that $\delta''_r \stackrel{c}{\doteq} \delta'_r$.

□

4.3 The maximal causal prefix

Let H be a well-formed history, and F be a failure set. Since all communication between processes is by message broadcasts and deliveries, only a portion of the history is "visible" to (*i.e.*, lies in the causal past of) the correct processes (*i.e.*, the processes not in F). The history that is derived in the natural manner from this visible portion of H is called the *maximal causal prefix* of H with respect to F and denoted $\text{MCP}(H, F)$ (see Figure 4.2).

History H and failure set $F = \{r, s\}$. The maximal causal prefix of H with respect to F is derived from the part of H to the left and above the dashed line.

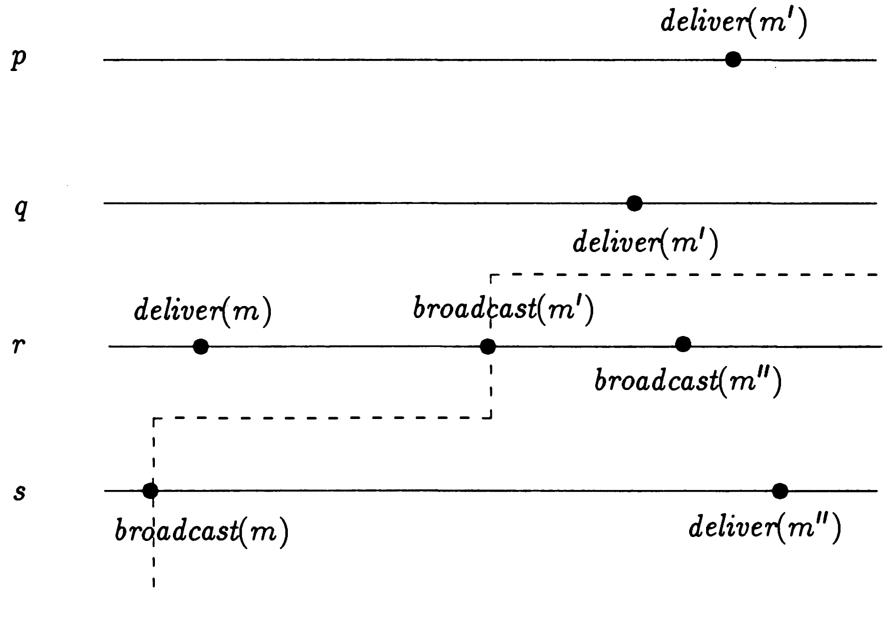


Figure 4.2: Illustrating the maximal causal prefix

Let H be a well-formed history, B be the broadcast history of H, and F be a failure set such that $\Sigma_B(B, F)$ is satisfied. Suppose $H^m = \text{MCP}(H, F)$, and B^m is the broadcast history of H^m. We show that $\Sigma_B(B^m, F)$ is also satisfied (Lemma 4.9). Furthermore, if p is X-correct until some time c in B with respect to Σ_B and F, and the tuple (p, c) lies in the causal past of any correct process, then p is X-correct until c in B^m with respect to Σ_B and F. This result is the basis of the main result of this chapter, proven in Section 4.5.

For all processes p, let $\Gamma_{p(B,F)}$ denote the greatest time such that $(p, \Gamma_{p(B,F)})$

lies in the causal past of the correct processes in B. For convenience, the formal definition of $\Gamma_p(B,F)$ uses a function PVC (for "past visible to correct processes").

Definition *Let $B = (\beta, \delta)$ be a broadcast history, and F be a failure set. For all processes p:*

$$PVC_p(B,F) \equiv \{c \mid \exists q \notin F,\ \exists c' :\ (p,c) \rightarrow_B (q,c')\}$$
$$\Gamma_p(B,F) \equiv \begin{cases} max(PVC_p(B,F)) & \textit{if } PVC_p(B,F) \textit{ is finite and nonempty} \\ 0 & \textit{if } PVC_p(B,F) \textit{ is empty} \\ \infty & \textit{otherwise} \end{cases}$$

Recall that for all broadcast histories B, for all processes p, for all times c, $(p,c) \rightarrow_B (p,c)$. Therefore, for all B, for all failure sets F, for all correct processes p ($p \notin F$), $|PVC_p(B,F)| = \infty$ and $\Gamma_p(B,F) = \infty$.

To simplify the notation, we write Γ_p to denote $\Gamma_p(B,F)$, Γ_p^m to denote $\Gamma_p(B^m,F)$, and so on.

The following lemmas are straightforward consequences of the above definition.

Lemma 4.6 *Let $B = (\beta, \delta)$ be a broadcast history and F be a failure set.*

- *For all p, if p is correct ($p \notin F$), then $\Gamma_p = \infty$.*
- *For all p, if $\Gamma_p \neq \infty$, then p is faulty ($p \in F$) and p broadcasts a message at time Γ_p in B (i.e., $\beta(p, \Gamma_p) \neq \phi$).*
- *For all p, if p is faulty ($p \in F$) and $\Gamma_p = \infty$, then for all c, there is a correct process q and a time c' such that $(p,c) \rightarrow_B (q,c')$. Furthermore, p broadcasts an infinite number of messages in B (i.e., $LB(B,p) = \infty$).*

Lemma 4.7 *Let B be a broadcast history and F be a failure set. For all p, if $\Gamma_p \neq \infty$, then:*

- *For all q such that $\Gamma_q \neq \infty$, $(p, \Gamma_p + 1) \not\rightarrow_B (q, \Gamma_q)$.*
- *For all q such that $\Gamma_q = \infty$, for all c, $(p, \Gamma_p + 1) \not\rightarrow_B (q, c)$.*

Proof: Suppose p is a process such that $\Gamma_p \neq \infty$. We prove the first part of the lemma below. The proof of the second part of the lemma is similar, and hence omitted.

Suppose for contradiction that the first part of the lemma is false—*i.e.*, $\exists q$ such that $\Gamma_q \neq \infty$, and $(p, \Gamma_p + 1) \rightarrow_B (q, \Gamma_q)$. By definition of Γ_q, there is a correct process r and time c, such that $(q, \Gamma_q) \rightarrow_B (r, c)$. Therefore, $(p, \Gamma_p + 1) \rightarrow_B (r, c)$, contradicting the definition of Γ_p. $\square$

4.3.1 Notation

Let $A = (\sigma, \beta, \delta)$, $B = (\beta, \delta)$, $A' = (\sigma', \beta', \delta')$ and $B' = (\beta', \delta')$. Let $X \subseteq \mathcal{P}$ and let c be any time. If for all processes $p \in X$, for all $c' \leq c$, p is in the same state in A' at c' as it is in A at c', we write $\sigma'_X \stackrel{c}{=} \sigma_X$. Formally: $\sigma'_X \stackrel{c}{=} \sigma_X \iff \forall p \in X,\ (\forall c',\ 1 \leq c' \leq c : \ \sigma'(p, c') = \sigma(p, c))$.

If $\sigma'_X \stackrel{c}{=} \sigma_X$ and the processes in X are in the halt state at time $c+1$ in A', we write $\sigma'_X \stackrel{c}{=} \sigma_X|\bot$. Formally: $\sigma'_X \stackrel{c}{=} \sigma_X|\bot \iff \sigma'_X \stackrel{c}{=} \sigma_X$ and $(\forall p \in X, \forall c' > c : \sigma(p, c') = \bot)$.

We write $A'_X \stackrel{c}{=} A_X$ if and only if $B'_X \stackrel{c}{=} B_X \ \wedge\ \sigma'_X \stackrel{c}{=} \sigma_X$. We write $A'_X \stackrel{c}{=} A_X|\phi$ if and only if $B'_X \stackrel{c}{=} B_X|\phi$ and $\sigma'_X \stackrel{c}{=} \sigma_X|\bot$.

4.3.2 Defining the maximal causal prefix

The maximal causal prefix of a history is formally defined below.

Definition *Let $H = (\Pi, \sigma, \beta, \delta)$ and $H^m = (\Pi, \sigma^m, \beta^m, \delta^m)$ be histories with application and broadcast histories of A and B, and A^m and B^m respectively. Let F be a failure set.*

B^m is the maximal causal prefix of B with respect to F, denoted $\textsc{mcp}(B, F)$, *if for all p, $\Gamma_p = \infty \implies B^m_p = B_p$ and $\Gamma_p \neq \infty \implies B^m_p \stackrel{\Gamma_p}{=} B_p|\phi$.*

A^m is the maximal causal prefix of A with respect to F, denoted $\textsc{mcp}(A, F)$, *if for all p, $\Gamma_p = \infty \implies A^m_p = A_p$ and $\Gamma_p \neq \infty \implies A^m_p \stackrel{\Gamma_p}{=} A_p|\phi$.*

H^m is the maximal causal prefix of H with respect to F, denoted $\textsc{mcp}(H, F)$, *if $A^m =$* $\textsc{mcp}(A, F)$.

The following lemma is a consequence of the above definition of maximal causal prefix.

Lemma 4.8 *Let H be a well-formed history, and A and B be the application and broadcast history of H respectively. Let F be a failure set. Let $H^m =$* $\textsc{mcp}(H, F)$ *and A^m and B^m be the application and broadcast history of H^m respectively.*

- *H^m, A^m and B^m are well-formed.*
- *$A^m_{\overline{F}} = A_{\overline{F}}$.*
- *For all p, if $\Gamma_p \neq \infty$, then $Halt(A^m, p) = \Gamma^m_p = \Gamma_p$.*

Proof: The well-formedness of H^m, A^m and B^m follow directly from the definition of well-formed histories.

Let p be any correct process (*i.e.*, $p \in F$). Thus $\Gamma_p = \infty$, and by the definition of maximal causal prefix, $A^m_p = A_p$. This proves the second part of the lemma.

The third part of the lemma follows from the definitions of Γ_p and Γ_p^m, and the definition of the maximal causal prefix. □

Let $B \in \mathcal{B}$ and F be such that $\Sigma_B(B, F)$. We show below that B^m, the maximal causal prefix of B with respect to F, can be derived from B by the following *Subtract* and *Deafen* operations:

- For all (faulty) processes p with $\Gamma_p \neq \infty$, the *Subtract* operation removes all the broadcasts p makes after time Γ_p, and the deliveries of these messages by the other processes.
- For all (faulty) processes p with $\Gamma_p \neq \infty$, the *Deafen* operation removes all of p's deliveries after time Γ_p.

We show that B^m is a well-formed broadcast history, and that $\Sigma_B(B^m, F)$ is satisfied.

Lemma 4.9 *Let $B = (\beta, \delta) \in \mathcal{B}$ and F be such that $\Sigma_B(B, F)$. Let $M = \{\beta(p, c) \mid \Gamma_p \neq \infty$ and $c > \Gamma_p\}$ and $X = \{(p, \Gamma_p) \mid \Gamma_p \neq \infty\}$. If $B^+ = Subtract(B, M)$ and $B^* = Deafen(B^+, X)$, then:*

- *$B^* \in \mathcal{B}$, $\Sigma_B(B^*, F)$, and B^* is the maximal causal prefix of B with respect to F (i.e., $B^* = \text{MCP}(B, F)$).*
- *For all $q \in F$: $\Gamma_q = LB(B^+, q)$.*

Proof: Let $B^+ = (\beta^+, \delta^+)$ and $B^* = (\beta^*, \delta^*)$ be as in the statement of the lemma. Consider the first part of the lemma. To prove that $B^* = \text{MCP}(B, F)$, we show that for all q, $\Gamma_q = \infty \implies B_q^* = B_q$ and $\Gamma_q \neq \infty \implies B_q^* \stackrel{\Gamma_q}{=} B_q|\phi$.

Let q be any process. The proof is a case analysis.

Case: $\Gamma_q = \infty$: Since $\Gamma_q = \infty$ and the *Subtract* operation only affects the broadcasts of all processes r with $\Gamma_r \neq \infty$, $\beta_q^+ = \beta_q$. For all $m \in M$, since m was broadcast by some process p after time Γ_p and $\Gamma_p \neq \infty$, Lemma 4.7 implies that q never delivers m in B. Thus, $\delta_q^+ = \delta_q$, and we get that $B_q^+ = B_q$. Since the *Deafen* operation only affects all processes p with $\Gamma_p \neq \infty$, q is unaffected and we get $B_q^* = B_q^+$. Thus, we conclude that $B_q^* = B_q$.

Case: $\Gamma_q \neq \infty$: Since $\Gamma_q \neq \infty$, the *Subtract* operation removes all broadcasts by q after time Γ_q, and hence $\beta_q^+ \stackrel{\Gamma_q}{=} \beta_q|\phi$. For all $m \in M$, since m was broadcast by some process p after time Γ_p and $\Gamma_p \neq \infty$, Lemma 4.7 implies that q does not deliver m at or before Γ_q in B. Thus, $\delta_q^+ \stackrel{\Gamma_q}{=} \delta_q$. Since the *Deafen* operation does not affect broadcasts, and removes all deliveries by q after time Γ_q, $\beta_q^* = \beta_q^+$ and $\delta_q^* \stackrel{\Gamma_q}{=} {\delta^+}_q|\phi$. Combining the above, we conclude that $B_q^* \stackrel{\Gamma_q}{=} B_q|\phi$.

Thus, we have shown that $B^* = \text{MCP}(B, F)$. We now show that $B^* \in \mathcal{B}$ and $\Sigma_B(B^*, F)$.

By choice of M, for all q such that a message of the form $(q, \text{—}, \text{—}) \in M$, $\Gamma_q \neq \infty$, and hence (by Lemma 4.6) q is faulty (*i.e.*, $q \in F$). Since $\Sigma_B(B, F)$ and $B^+ = \mathit{Subtract}(B, M)$, Lemma 4.1 implies that $B^+ \in \mathcal{B}$ and $\Sigma_B(B^+, F)$. By choice of X, for all tuples $(q, \text{—}) \in X$, $\Gamma_q \neq \infty$, and hence (by Lemma 4.6) q is faulty. Since $\Sigma_B(B^+, F)$ and $B^* = \mathit{Deafen}(B^+, X)$, Lemma 4.2 implies that $B^* \in \mathcal{B}$ and $\Sigma_B(B^*, F)$.

Thus, we have shown that $B^* = \text{MCP}(B, F)$, $B^* \in \mathcal{B}$ and $\Sigma_B(B^*, F)$, completing the proof of the first part of the lemma.

Let q be any process in F. To prove the second part of the lemma, we show $\Gamma_q = LB(B^+, q)$.

Case: $\Gamma_q = \infty$: By Lemma 4.6, $LB(B, q) = \infty$; *i.e.*, q broadcasts an infinite number of messages in B. Since $B_q^+ = B_q$, $LB(B^+, q) = \infty$.

Case: $\Gamma_q \neq \infty$: By Lemma 4.6, $\beta(q, \Gamma_q) \neq \phi$. Since $\beta_q^+ \stackrel{\Gamma_q}{=} \beta_q|\phi$, for all $c > \Gamma_q : \beta^+(q, c) = \phi$. That is, $LB(B^+, q) = \Gamma_q$. □

The following lemma relates the SD-closed property and maximal causal prefixes.

Lemma 4.10 *Suppose* X*-correctness is* SD*-closed with respect to* Σ_B*. Let* $B \in \mathcal{B}$ *and* F *be such that* $\Sigma_B(B, F)$*. For all processes* p*, for all times* $c \leq \Gamma_p$*, if* p *is* X*-correct until some time* c *in* B *with respect to* Σ_B *and* F*, then* p *is* X*-correct until time* c *in* $\text{MCP}(B, F)$ *with respect to* Σ_B *and* F*.*

Proof: Suppose a process p is X-correct until some time $c \leq \Gamma_p$ in B. Let $M = \{\beta(q, c') \mid \Gamma_q \neq \infty \text{ and } c' > \Gamma_q\}$ (as in the hypothesis of Lemma 4.9). Thus, for all messages $(q, \text{—}, \text{—}) \in M$, $\Gamma_q \neq \infty$, and (from Lemma 4.6) $q \in F$.

Let $B^+ = \mathit{Subtract}(B, M)$. Lemma 4.1 and Lemma 4.3 imply that $B^+ \in \mathcal{B}$ and $\Sigma_B(B^+ F)$ is satisfied, and that p is X-correct until c in B^+ with respect to Σ_B and F.

Lemma 4.9 implies that if $p \in F$, then $\Gamma_p = LB(B^+, p)$. Since $c \leq \Gamma_p$, we conclude that either $p \notin F$ or $c \leq LB(B^+, p)$.

Let $X = \{(q, \Gamma_q) \mid \Gamma_q \neq \infty\}$ (as in the hypothesis of Lemma 4.9). Thus, for all tuples $(q, \text{—}) \in X$, $\Gamma_q \neq \infty$, and (from Lemma 4.6) $q \in F$.

Let $B^* = \mathit{Deafen}(B^+, X)$. Since $B^+ \in \mathcal{B}$, $\Sigma_B(B^+ F)$, and p is X-correct until c in B^+, and either $p \notin F$ or $c \leq LB(B^+, p)$, Lemma 4.4 implies that p is X-correct until c in B^* with respect to Σ_B and F.

By construction of B^*, from Lemma 4.9 we conclude that $B^* = \text{MCP}(B, F)$. Since p is X-correct until c in B^* with respect to Σ_B and F, we have shown that p is X-correct until c in $\text{MCP}(B, F)$. □

4.4 The history hierarchy

Let Σ_B be a broadcast specification. Let A be a well-formed application history, B be the broadcast history of A, and F be such that $\Sigma_B(B, F)$ is true.

Suppose A is BD-consistent with respect to Σ_B and F. From the correctness hierarchy (Chapter 3), A is VBD-consistent with respect to Σ_B and F. This is illustrated in Figure 4.3 by the arrow from the box marked "BD-consistent" to the box marked "VBD-consistent." From the contamination hierarchy (Chapter 3), A is BD-contamination-free with respect to Σ_B and F. This is illustrated in Figure 4.3 by the arrow from the box marked "BD-consistent" to the box marked "BD-contamination-free."

Suppose A is BD-contamination-free with respect to Σ_B and F. Lemma 4.11 (given below) proves that the maximal causal prefix of A with respect to F is BD-consistent with respect to Σ_B and F.

Figure 4.3 illustrates the rest of the *history hierarchy*.

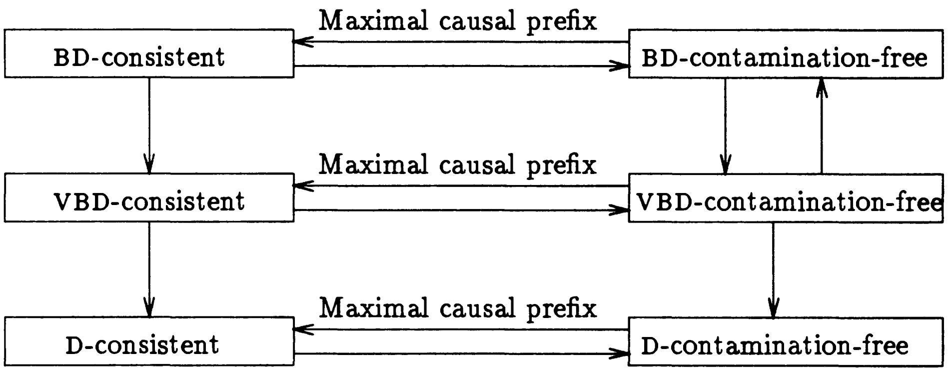


Figure 4.3: The history hierarchy

Lemma 4.11 *Suppose X-correctness is SD-closed with respect to Σ_B. Let $A \in \mathcal{A}$, $B = (\beta, \delta)$ be the broadcast history of A, and F be such that $\Sigma_B(B, F)$. If A is X-contamination-free with respect to Σ_B and F, then $\text{MCP}(A, F)$ is X-consistent with respect to Σ_B and F.*

Proof: Suppose A is X-contamination-free with respect to Σ_B and F. Let $A^m = \text{MCP}(A, F)$, and let $B^m = (\beta^m, \delta^m)$ be the broadcast history of A^m. Since A is well-formed, Lemma 4.8 implies that A^m and B^m are well-formed. Since $\Sigma_B(B, F)$, Lemma 4.9 implies that $\Sigma_B(B^m, F)$ is also true.

Let p be any process and c be any time $\leq LVT(A^m, p)$. To show that A^m is X-consistent with respect to Σ_B and F, we must show that p is X-correct until c in B^m with respect to Σ_B and F.

Note that $c \leq \Gamma_p$. This is clearly the case if $\Gamma_p = \infty$. If $\Gamma_p \neq \infty$, then Lemma 4.8 implies that $Halt(A^m, p) = \Gamma_p$. Since $LVT(A^m, p) \leq Halt(A^m, p)$ and $c \leq LVT(A^m, p)$, $c \leq \Gamma_p$.

Claim: Process p is X-correct until c in B with respect to Σ_B and F.

Proof of claim: The proof is a case analysis.

Case: $\Gamma_p \neq \infty$: Since $c \leq \Gamma_p$, to prove the claim it is enough to show that p is X-correct until Γ_p in B with respect to Σ_B and F. By definition of Γ_p, for some process $q \notin F$ and some time c', $(p, \Gamma_p) \rightarrow_B (q, c')$. Since A is X-contamination-free with respect to Σ_B and F, p is X-correct until Γ_p in B.

Case: $\Gamma_p = \infty$: Since $\Gamma_p = \infty$, Lemma 4.6 implies that for some process $q \notin F$, and some time c', $(p, c) \rightarrow (q, c')$. As above, since A is X-contamination-free with respect to Σ_B and F, p is X-correct until c in B. •

Since X-correctness is SD-closed with respect to Σ_B, $\Sigma_B(B, F)$ and p is X-correct until time c in B with respect to Σ_B and F, we conclude from Lemma 4.10 that p is X-correct until time c in B^m. □

4.5 Correct restricted problems

We now characterize a class of problems, called *correct restricted* problems (*cr-problems*), for which the prevention of contamination is "as good" as the prevention of inconsistency. Intuitively, the specifications of such problems ignore the behavior of faulty processes, and concentrate only on the behavior of correct processes. Formally:

Definition Σ_A *is a correct restricted specification (cr-specification) if and only if:*

$$\forall A \in \mathcal{A},\ \forall A' \in \mathcal{A} :\ \forall F,\ [\Sigma_A(A, F) \text{ and } A'_{\overline{F}} = A_{\overline{F}} \implies \Sigma_A(A', F)]$$

We now prove the main result of this chapter. Suppose X-correctness is SD-closed with respect to Σ_B, a broadcast specification with the choice property. If an application protocol Π solves a cr-specification Σ_A using Σ_B assuming X-consistent behavior, then Π solves Σ_A using Σ_B assuming X-contamination-free behavior.

In other words, Π can be designed with the simplifying assumption that both inconsistency and contamination will be prevented. However, Π will solve Σ_A using Σ_B even if only contamination is prevented. Since the prevention of contamination is often less expensive than the prevention of both inconsistency and contamination, such a "substitution" often improves the performance of the application.

Theorem 4.12 *Suppose* X*-correctness is* SD*-closed with respect to* Σ_B. *If* Π *solves a cr-specification* Σ_A *with* Σ_B *assuming* X*-consistent behavior, then* Π *solves* Σ_A *with* Σ_B *assuming* X*-contamination-free behavior.*

Proof: Suppose Π solves Σ_A with Σ_B assuming X-consistent behavior. Let $H = (\Pi, \sigma, \beta, \delta)$ be a well-formed history, $A = (\sigma, \beta, \delta)$, $B = (\beta, \delta)$ and let F be any failure set such that $\Sigma_B(B, F)$. Suppose A is X-contamination-free with respect to Σ_B and F. To prove the lemma, we show that $\Sigma_A(A, F)$ is satisfied.

Let $H^m = (\Pi, \sigma^m, \beta^m, \delta^m) = \text{MCP}(H, F)$, $A^m = (\sigma^m, \beta^m, \delta^m)$ and $B^m = (\beta^m, \delta^m)$. Lemma 4.8 implies that H^m is well-formed. Lemma 4.9 implies that $\Sigma_B(B^m, F)$ is satisfied. Since X-correctness is SD-closed with respect to Σ_B, and $\Sigma_B(B, F)$ and A is X-contamination-free with respect to Σ_B and F, Lemma 4.11 implies that A^m is X-consistent with respect to Σ_B and F.

Since H^m is a well-formed history, and $\Sigma_B(B^m, F)$ is satisfied, and A^m is X-consistent with respect to Σ_B and F, and Π solves Σ_A using Σ_B assuming X-consistent behavior, $\Sigma_A(A^m, F)$ is satisfied. Lemma 4.8 implies that $A^m_{\overline{F}} = A_{\overline{F}}$. Since Σ_A is a cr-specification, $\Sigma_A(A, F)$ is also satisfied. □

Recall that the three correctness conditions we defined in Chapter 3 have the SD-closed property. Recall also that a VBD-contamination-free history is equivalent to a BD-contamination-free history. Therefore, if Π is designed with the strong assumption of BD-consistent behavior, then Π works correctly when executed under conditions that prevent VBD-contamination.

Corollary 4.13 *Suppose* X*-correctness is* SD*-closed with respect to* Σ_B. *If* Π *solves a cr-specification* Σ_A *with* Σ_B *assuming* BD*-consistent behavior, then* Π *solves* Σ_A *with* Σ_B *assuming* VBD*-contamination-free behavior.*

Chapter 5

Inconsistency and Contamination with Atomic Broadcast

In this chapter, we present definitions of inconsistency and contamination with *atomic broadcast* and *timed atomic broadcast.* These definitions are derived from the general definitions of inconsistency and contamination given in Chapter 3.

5.1 Atomic broadcast

Informally, with atomic broadcast, any process may broadcast a message at any time, such that:

- All correct processes deliver the same messages in the same order.
- Every correct process delivers all the messages broadcast by all correct processes.

Formally, atomic broadcast is defined to be the conjunction of the validity, agreement and integrity properties (defined in Chapter 2 and repeated below) and the *total order* property defined below. [1]

- *Validity:* If a correct process broadcasts a message m, then it eventually delivers m.

- *Agreement:* If a correct process delivers a message m, then all correct processes eventually deliver m.

- *Integrity:* For any message m, any process delivers m at most once, and only if m was broadcast by some process.

[1] Recall that such properties are more formally expressed as predicates of broadcast histories and failure sets.

- *Total order:* If correct processes p and q deliver messages m and m', then p delivers m before m' if and only if q delivers m before m'.

Atomic broadcast is one the most powerful fault-tolerant broadcasts used for communication. It is the basis for Lamport's *state machine* approach to fault tolerance [Sch86]. It is also used in the *Isis* system [BJ87] and the IBM *Highly Available System* [Cri87].

5.1.1 Notation

Suppose $B = (\beta, \delta)$ is a broadcast history. $\text{BCAST}_p^c(B)$ denotes the set of messages that process p broadcasts by time c in B; formally, $\text{BCAST}_p^c(B) = \bigcup_{c' \leq c} \beta(p, c')$. $\overrightarrow{\text{DLVD}}_p^c(B)$ denotes the *sequence* of messages that p delivers by time c (defined in Chapter 2). $\text{DLVD}_p^c(B)$ denotes the *set* of messages that p delivers by time c; $\text{DLVD}_p^c(B) = \{m \mid m \in \overrightarrow{\text{DLVD}}_p^c(B)\}$. The superscript c is omitted when referring to all the messages broadcast or delivered during a complete history.

Lemma 5.1 *Let $B = (\beta, \delta) \in \mathcal{B}$ and F be a failure set, such that B satisfies atomic broadcast with respect to F.*

- *All correct processes deliver the same sequence of messages:*
  $\forall p \notin F, \forall q \notin F : \overrightarrow{\text{DLVD}}_p(B) = \overrightarrow{\text{DLVD}}_q(B)$.
- *If a correct process broadcasts a message m, then all correct processes eventually deliver m:*
  $\forall p \notin F, \forall q \notin F : \text{BCAST}_p(B) \subseteq \text{DLVD}_q(B)$.

Proof: Directly from the definition of atomic broadcast. □

Suppose B is a broadcast history and F is a failure set such that B satisfies atomic broadcast with respect to F. $\overrightarrow{\text{DLVD}}_{\overline{F}}(B)$ and $\text{DLVD}_{\overline{F}}(B)$ respectively denote the sequence of messages and the set of messages that the correct processes deliver during B.

5.1.2 Consistency with atomic broadcast

We show that delivery consistency is ensured with atomic broadcast, if the following condition is satisfied (Lemma 5.4): the sequence of messages delivered by *every* process (correct or faulty) is a *prefix* of the sequence of messages delivered by the correct processes during the entire execution. This "prefix" property, in conjunction with fifo message delivery order, is enough to ensure visible-broadcast/delivery consistency (Lemma 5.7).

However, the "prefix" property is not enough to ensure broadcast/delivery consistency. As was mentioned in Chapter 3, broadcast/delivery consistency can be ensured only if the following (informal) condition holds: a process may only broadcast a message after all of its previous broadcasts are successfully completed. In other words, broadcast/deliver consistency can only be ensured with atomic broadcast using a "blocking" protocol.

We begin by defining D-correctness, VBD-correctness and BD-correctness with respect to atomic broadcast. We show that these correctness conditions lead to the "prefix" definition of consistency that was mentioned earlier.

Delivery correctness and consistency

The following lemma describes when a process is D-correct with respect to atomic broadcast.

Lemma 5.2 *Let $B = (\beta, \delta) \in \mathcal{B}$ and F be such that B satisfies atomic broadcast with respect to F. Process p is D-correct until c with respect to atomic broadcast and F if and only if:*

1. *The sequence of messages p delivers by time c is a prefix of the sequence of messages the correct processes eventually deliver: $\overrightarrow{\text{DLVD}}^c_p(B) \preceq \overrightarrow{\text{DLVD}}_{\overline{F}}(B)$.*
2. *If p broadcasts a message m that some process q delivers by time c, then all the correct processes eventually deliver m: if $p \equiv bc(m)$ and $m \in \text{DLVD}^c_q(B)$, then $m \in \text{DLVD}_{\overline{F}}(B)$.*

The first condition (labeled 1) in Lemma 5.2 is the "prefix" property that was already mentioned. The second condition (labeled 2) is more subtle. Suppose a broadcast history B satisfies atomic broadcast with respect to a failure set F. Suppose that a process p broadcasts a message m, a faulty process q delivers m by some time c and no correct process delivers m. In such a scenario (which would not be permitted by the second condition), p cannot be D-correct until c in B with respect to atomic broadcast and F; in particular, there is no D-extension of (B, p, c).

To see this, suppose for contradiction that $B' = (\beta', \delta')$ is a D-extension of (B, p, c) with respect to atomic broadcast and F. Since, by definition, $\delta'_{\overline{F}} = \delta_{\overline{F}}$ and $\delta'_F \stackrel{c}{=} \delta_F$, the faulty process q delivers m by time c in B', and no correct process delivers m in B'.

- If p broadcasts m in B', then it is clear that B' does not satisfy atomic broadcast with respect to $F - p$; *i.e.*, p and the processes in $\overline{F}$ cannot all be correct.
- Suppose p does not broadcast m in B'. Since m was broadcast by p in B, m is of the form $(p, c, —)$. Thus, by the well-formedness of B', no process broadcasts

m in B'. Since q delivers m, we conclude that B' does not satisfy the integrity property.

Thus, we have shown that B' does not satisfy atomic broadcast with respect to $F-p$, contradicting the assumption that B' is a D-extension of (B,p,c) with respect to atomic broadcast and F.

Note that the second condition in Lemma 5.2 is not required if the specification of atomic broadcast is weakened to include only the validity, agreement and total order properties.

We now present the proof of Lemma 5.2.

Proof of the "only if" assertion of Lemma 5.2: Suppose p is D-correct until c in B with respect to atomic broadcast and F. If p is correct, then the lemma is trivially true.

Suppose that p is faulty ($p \in F$). Let $B' = (\beta', \delta')$ be a D-extension of (B,p,c) with respect to atomic broadcast and F given by the definition of D-correctness; thus $\delta'_{\overline{F}} = \delta_{\overline{F}}$, and $\delta'_F \stackrel{c}{=} \delta_F$, and B' satisfies atomic broadcast with respect to $F-p$.

1. Let q be any correct process ($q \notin F$). Since $\delta'_{\overline{F}} = \delta_{\overline{F}}$ and $q \notin F$, $\overrightarrow{\text{DLVD}}_q(B') = \overrightarrow{\text{DLVD}}_q(B)$. Since B' satisfies atomic broadcast with respect to $F-p$, Lemma 5.1 implies that $\overrightarrow{\text{DLVD}}_p(B') = \overrightarrow{\text{DLVD}}_q(B')$; hence $\overrightarrow{\text{DLVD}}^c_p(B') \preceq \overrightarrow{\text{DLVD}}_q(B')$. Since $\delta'_p \stackrel{c}{=} \delta_p$, $\overrightarrow{\text{DLVD}}^c_p(B) = \overrightarrow{\text{DLVD}}^c_p(B')$. Thus, we conclude that $\overrightarrow{\text{DLVD}}^c_p(B) \preceq \overrightarrow{\text{DLVD}}_q(B)$. Since q is correct, Lemma 5.1 implies that $\overrightarrow{\text{DLVD}}^c_p(B) \preceq \overrightarrow{\text{DLVD}}_{\overline{F}}(B)$.

2. Suppose that for some process q, and for some message m that was broadcast by p, $m \in \text{DLVD}^c_q(B)$. We show that $m \in \text{DLVD}_{\overline{F}}(B)$.

   Suppose q is correct ($q \notin F$). Then, by Lemma 5.1, $m \in \text{DLVD}_{\overline{F}}(B)$.

   Suppose q is faulty ($q \in F$). Since $\delta'_F \stackrel{c}{=} \delta_F$ and $m \in \text{DLVD}^c_q(B)$, $m \in \text{DLVD}^c_q(B')$. Hence, by Lemma 2.1, p broadcasts m in B'. Since B' satisfies atomic broadcast with respect to $F-p$, and $m \in \text{BCAST}_p(B')$, Lemma 5.1 implies that $m \in \text{DLVD}_{\overline{F}}(B')$. Since $\delta'_{\overline{F}} = \delta_{\overline{F}}$, we conclude that $m \in \text{DLVD}_{\overline{F}}(B)$.

This completes the proof of the first part of the lemma. •

Proof of the "if" assertion of Lemma 5.2: If p is correct, then the lemma is trivially true. Hence, suppose that p is faulty ($p \in F$).

Suppose the following conditions are satisfied:

- $\overrightarrow{\text{DLVD}}^c_p(B) \preceq \overrightarrow{\text{DLVD}}_{\overline{F}}(B)$.
- For all processes q, for all messages m, if p broadcasts m and $m \in \text{DLVD}^c_q(B)$, then $m \in \text{DLVD}_{\overline{F}}(B)$.

We prove by construction that p is D-correct until c in B with respect to atomic broadcast and F—in particular, we construct a broadcast history $B' = (\beta', \delta')$ such that B' is a D-extension of (B, p, c) with respect to atomic broadcast and F. B' is derived from B as follows:

- The correct processes broadcast and deliver the same messages at the same time in both B and B'. Formally: $B'_{\overline{F}} = B_{\overline{F}}$.
- The faulty processes, not including p, broadcast the same messages at the same time in both B and B'. Formally: $\beta'_{F-p} = \beta_{F-p}$.
- Until time c, the faulty processes, not including p, deliver the same messages at the same time in both B and B'. After time c in B', the faulty processes, not including p, do not deliver any messages. Formally: $\delta'_{F-p} \stackrel{c}{=} \delta_{F-p}|\phi$.
- Process p broadcasts in B' exactly the subset of its broadcasts that were delivered by the correct processes in B.
  Thus: $\forall c' : \beta'(p, c') = \begin{cases} \beta(p,c) & \text{if } \beta(p,c) \in \text{DLVD}_{\overline{F}}(B) \\ \phi & \text{otherwise} \end{cases}$
- Until time c, process p delivers the same messages at the same time in both B and B'; *i.e.*, $\delta'_p \stackrel{c}{=} \delta_p$. Process p's deliveries after time c in B' are obtained by the procedure described below.

  Recall that by assumption $\overrightarrow{\text{DLVD}}^c_p(B) \preceq \overrightarrow{\text{DLVD}}_{\overline{F}}(B)$. Therefore, there is some sequence of messages $S = \langle m_1, m_2, \ldots m_i, m_{i+1} \ldots \rangle$ such that:
  $$[\overrightarrow{\text{DLVD}}^c_p(B) \cdot S = \overrightarrow{\text{DLVD}}_{\overline{F}}(B)].$$
  Informally, for all messages $m \in S$, p delivers m after time $ts(m)$ in B'; furthermore, p delivers all the messages in S in sequence order. Formally, for all $m_i \in S$, p delivers message m_i at time c_i in B', as defined below:

  - Process p delivers m_1 at time $ts(m_1) + 1$, or at time $c + 1$, whichever is later; *i.e.*, $c_1 = max(ts(m_1) + 1, c + 1)$.
  - For all m_i, $i > 1$, in S, p delivers m_i at time $ts(m_i) + 1$, or at time c_{i-1}, whichever is later; *i.e.*, $c_i = max(ts(m_i) + 1, c_{i-1})$.

Claim: $B'_F = B_F$, $B'_{F-p} \stackrel{c}{=} B_{F-p}$, $\delta'_p \stackrel{c}{=} \delta_p$, ${\beta'}_p \stackrel{c-\Delta}{=} \beta_p$ and $\forall c' > c - \Delta : \beta'(p, c') = \beta(p, c')$ or ϕ.

Proof of claim: Directly from the construction, $B'_F = B_F$, $B'_{F-p} \stackrel{c}{=} B_{F-p}$, $\delta'_p \stackrel{c}{=} \delta_p$ and $\forall c' : \beta'(p, c') = \beta(p, c')$ or ϕ. Since $\Delta = \infty$ for an untimed broadcast, it follows trivially that $\beta'_p \stackrel{c-\Delta}{=} \beta_p$. This proves the claim. •

Claim: B' is well-formed.

Proof of claim: We show below that B' satisfies the three properties required of a well-formed history (as defined in Chapter 2).

1. It is clear that β' is a history broadcast function and δ' is a history delivery function.

2. Suppose for some process q, time c', and message $m \neq \phi$, $m \in \delta'(q, c')$. We prove that $ts(m) < c'$.

   Suppose $q \neq p$. Since $\delta'_{\overline{F}} = \delta_{\overline{F}}$ and $\delta'_{F-p} \stackrel{c}{=} \delta_{F-p}|\phi$, and $m \in \delta'(q, c')$, either $q \notin F$, or $c' \leq c$. In either case, $m \in \delta(q, c')$. Since B is well-formed, $ts(m) < c'$.

   Suppose $q = p$ and $c' \leq c$. Since $\delta'_p \stackrel{c}{=} \delta_p$ (by construction), $m \in \delta(q, c')$. Since B is well-formed, $ts(m) < c'$.

   Suppose $q = p$ and $c' > c$. From the construction of B', if $m \in \delta(p, c')$ then $c' \geq ts(m) + 1$; *i.e.*, $ts(m) < c'$.

3. Consider any message $m \neq \phi$ broadcast by any process q at any time c' in B'. We prove that m is of the form $(q, c', —)$.

   From the construction, if $\beta'(q, c') \neq \phi$, then $\beta'(q, c') = \beta(q, c')$. Since B is well-formed and $m \neq \phi$, $m = (q, c', —)$.

Thus, B' is well-formed. •

Claim: B' satisfies atomic broadcast with respect to $F - p$.

Proof of claim: We show that B' satisfies each of the properties of atomic broadcast when the processes in $F - p$ are assumed to be faulty.
Agreement and total order: To prove that B' satisfies both the agreement and the total order properties, we show that all processes not in $(F - p)$ deliver the same sequence of messages in B' (*i.e.*, $\forall q \notin F - p : \overrightarrow{\text{DLVD}}_q(B') = \overrightarrow{\text{DLVD}}_{\overline{F}}(B)$).

Let q be any process $\notin F - p$. Suppose $q \neq p$; thus $q \notin F$, and by construction, $\overrightarrow{\text{DLVD}}_q(B') = \overrightarrow{\text{DLVD}}_q(B) = \overrightarrow{\text{DLVD}}_{\overline{F}}(B)$. Suppose $q = p$; by construction of B', $\overrightarrow{\text{DLVD}}_q(B') = \overrightarrow{\text{DLVD}}_{\overline{F}}(B)$.

Validity: Let q be any process $\notin F - p$. To prove that B' satisfies the validity property, we show that q delivers all of its own broadcasts in B'; *i.e.*, $\text{BCAST}_q(B') \subseteq \text{DLVD}_q(B')$.

Suppose $q \neq p$; thus $q \notin F$. Since B satisfies atomic broadcast when the processes in F are assumed to be faulty, $\text{BCAST}_q(B) \subseteq \text{DLVD}_q(B)$. By construction, $\text{BCAST}_q(B') = \text{BCAST}_q(B)$, and $\text{DLVD}_q(B') = \text{DLVD}_q(B)$. Thus, $\text{BCAST}_q(B') \subseteq \text{DLVD}_q(B')$.

Suppose $q = p$. By construction, $\text{BCAST}_p(B') \subseteq \text{DLVD}_{\overline{F}}(B)$. Since $\overrightarrow{\text{DLVD}}_p(B') = \overrightarrow{\text{DLVD}}_{\overline{F}}(B)$, $\text{BCAST}_p(B') \subseteq \text{DLVD}_p(B')$.

Integrity: Let q be any process. To prove that B' satisfies the integrity property, we must show that q does not deliver any message more than once. This is straightforward, and hence the proof is omitted.

To prove that B' satisfies the integrity property, we must also show that if q delivers a message, then that message is actually broadcast by some process. The proof is a case analysis and we only show one case below.

Case: Process $q \in F - p$ delivers $m = (p, —, —)$ by time c in B': Since $\delta'_{F-p} \overset{c}{=} \delta_{F-p}$, q also delivers m by time c in B (*i.e.*, $m \in \text{DLVD}^c_q(B)$). We show below that p also broadcasts m in B'.

Since q delivers m in B, and B satisfies the integrity property, p broadcasts m in B. Since $m \in \text{DLVD}^c_q(B)$, by hypothesis, $m \in \text{DLVD}_{\overline{F}}(B)$. By construction, since p broadcasts m in B and $m \in \text{DLVD}_{\overline{F}}(B)$, p also broadcasts m in B'. •

Thus, B' is a D-extension of (B, p, c) with respect to atomic broadcast and F, and hence we have proven that p is D-correct until c in B with respect to atomic broadcast and F. □

Lemma 5.2 above gave necessary and sufficient conditions for a process to be D-correct with respect to atomic broadcast. A weaker but more "elegant" formulation is given in Lemma 5.3. It is stated using the following *uniform agreement* property, which informally requires that *all* processes, correct or faulty, satisfy the agreement property:

- *Uniform agreement:* If *any* process p delivers a message m, then all correct processes eventually deliver m.

Uniform agreement is also called *uniformity* in the literature [NT90,GT89].

Lemma 5.3 *Let $B = (\beta, \delta) \in \mathcal{B}$ and F be such that B satisfies atomic broadcast with respect to F. Suppose B satisfies the uniform agreement property with respect to F.*

Process p is D-correct until c with respect to atomic broadcast and F if and only if $\overrightarrow{\text{DLVD}}^c_p(B) \preceq \overrightarrow{\text{DLVD}}_{\overline{F}}(B)$.

Proof: From Lemma 5.2. □

Having stated what it means for a process to be D-correct, we can now prove a sufficient condition for an application history to be delivery consistent.

Lemma 5.4 *Let $A \in \mathcal{A}$, B be the broadcast history of A, and F be such that B satisfies atomic broadcast with respect to F. If for all processes p, for all times c, $\overrightarrow{\mathrm{DLVD}}^c_p(B) \preceq \overrightarrow{\mathrm{DLVD}}_{\overline{F}}(B)$, then A is a D-consistent history with respect to atomic broadcast and F.*

Proof: Suppose for all processes p, for all times c, $\overrightarrow{\mathrm{DLVD}}^c_p(B) \preceq \overrightarrow{\mathrm{DLVD}}_{\overline{F}}(B)$. Clearly, this implies that any message delivered by any process (correct or faulty) at any time, is also eventually delivered by all the correct processes; *i.e.*, B satisfies the uniform agreement property when the processes in F are assumed to be faulty.

To complete the proof, we show that for all processes p, for all $c \leq LVT(A,p)$, p is D-correct until time c in B with respect to atomic broadcast and F.

Let p be any process. If p is correct, then it is trivial to show that for all c, p is D-correct until time c in B with respect to atomic broadcast and F.

Suppose p is faulty ($p \in F$). Let c be any time $\leq LVT(A,p)$. Since $\overrightarrow{\mathrm{DLVD}}^c_p(B) \preceq \overrightarrow{\mathrm{DLVD}}_{\overline{F}}(B)$, and B satisfies the uniform agreement property with respect to F, Lemma 5.3 implies that p is D-correct until c in B with respect to atomic broadcast and F. □

Note that the converse of the lemma is false. That is, if A is a D-consistent history with respect to atomic broadcast and F, then there may be a process p and a time c such that $\overrightarrow{\mathrm{DLVD}}^c_p(B) \not\preceq \overrightarrow{\mathrm{DLVD}}_{\overline{F}}(B)$. This is because a D-consistent history permits the following scenario:

- A faulty process delivers a message m at some time c, and halts *before* broadcasting a message at time c, or changing state at time c.
- No correct process ever delivers m.

Thus, the sequence of messages that process p delivers by time c is not a prefix of the messages delivered by the correct processes during the entire execution. Note however that p's "erroneous" delivery at time c is not reflected in p's state, nor in the states of the other processes in the system.

Visible-broadcast/delivery correctness and consistency

The following lemma describes when a process is VBD-correct with respect to atomic broadcast. The proof is similar to that of Lemma 5.2, and hence is omitted.

Lemma 5.5 *Let $B = (\beta, \delta) \in \mathcal{B}$ and F be such that B satisfies atomic broadcast with respect to F. Process p is* VBD*-correct until c with respect to atomic broadcast and F if and only if:*

- *The sequence of messages p delivers by time c is a prefix of the sequence of messages the correct processes deliver during the entire execution:* $\overrightarrow{\mathrm{DLVD}}_p^c(B) \preceq \overrightarrow{\mathrm{DLVD}}_{\overline{F}}(B)$.
- *If a faulty process q (i.e., $q \in F$) delivers a message m that p broadcasts at $c' \leq c$, then a correct process eventually delivers m.*
- *If a correct process q (i.e., $q \notin F$) delivers a message m that p broadcasts at any time, then q eventually delivers all the messages that p broadcasts before m.*

However, a weaker, but more convenient form of the above lemma is given below using the following *fifo order* property:

- *FIFO Order:* If any process broadcasts a message m before it broadcasts a message m', and a correct process p delivers m', then p delivers m before m'.

Consider an execution in which processes communicate via atomic broadcast, such that the uniform agreement and fifo order properties are also satisfied. Lemma 5.6 states that a process p is VBD-correct until time c if and only if the sequence of messages p delivers by time c is a prefix of the sequence of messages that the correct processes eventually deliver.

Lemma 5.6 *Let $B = (\beta, \delta) \in \mathcal{B}$ and F be such that B satisfies atomic broadcast with respect to F. Suppose B satisfies the uniform agreement and fifo order properties with respect to F.*

Process p is VBD*-correct until c with respect to atomic broadcast and F if and only if* $\overrightarrow{\mathrm{DLVD}}_p^c(B) \preceq \overrightarrow{\mathrm{DLVD}}_{\overline{F}}(B)$.

Proof: Omitted for brevity. □

Lemma 5.7 *Let $A \in \mathcal{A}$, B be the broadcast history of A, and F be such that B satisfies atomic broadcast with respect to F. If B satisfies the fifo order property with respect to F, and for all processes p, for all times c, $\overrightarrow{\mathrm{DLVD}}_p^c(B) \preceq \overrightarrow{\mathrm{DLVD}}_{\overline{F}}(B)$, then A is a* VBD*-consistent history with respect to atomic broadcast and F.*

Proof: From Lemma 5.5. □

Since it is easy to enforce the fifo order property (*e.g.*, using message sequence numbers), Lemmas 5.4 and 5.7 show that providing visible-broadcast/delivery consistency is only slightly more expensive than providing delivery consistency.

Broadcast/delivery correctness and consistency

Finally, we state the conditions when a process is BD-correct with respect to atomic broadcast.

Lemma 5.8 *Let $B = (\beta, \delta) \in \mathcal{B}$ and F be such that B satisfies atomic broadcast with respect to F. Process p is* BD*-correct until c with respect to atomic broadcast if and only if:*

- $\overrightarrow{\text{DLVD}}^c_p(B) \preceq \overrightarrow{\text{DLVD}}_{\overline{F}}(B)$.
- $\text{BCAST}^c_p(B) \subseteq \text{DLVD}_{\overline{F}}(B)$.
- *If a correct process q (i.e., $q \notin F$) delivers a message m that p broadcasts, then for all messages m' that p broadcasts before m, q eventually delivers m'.*

The following lemma is weaker, but more "elegant."

Lemma 5.9 *Let $B = (\beta, \delta) \in \mathcal{B}$ and F be such that B satisfies atomic broadcast with respect to F. Suppose B satisfies the fifo order property with respect to F. Process p is* BD*-correct until c with respect to atomic broadcast if and only if:*

- $\overrightarrow{\text{DLVD}}^c_p(B) \preceq \overrightarrow{\text{DLVD}}_{\overline{F}}(B)$.
- $\text{BCAST}^c_p(B) \subseteq \text{DLVD}_{\overline{F}}(B)$.

Finally, we state a sufficient condition for a BD-consistent application history:

Lemma 5.10 *Let $A \in \mathcal{A}$, B be the broadcast history of A, and F be such that B satisfies atomic broadcast with respect to F. If B satisfies the fifo order property with respect to F, and for all processes p, for all times c, $\overrightarrow{\text{DLVD}}^c_p(B) \preceq \overrightarrow{\text{DLVD}}_{\overline{F}}(B)$, and $\text{BCAST}^c_p(B) \subseteq \text{DLVD}_{\overline{F}}(B)$, then A is a* BD*-consistent history with respect to atomic broadcast and F.*

5.1.3 Contamination with atomic broadcast

In this section, we study contamination with respect to atomic broadcasts. Recall that contamination is the spread of "incorrectness" from the faulty processes to the correct processes. We defined three forms of contamination in Chapter 3, one corresponding to each of the three correctness conditions we proposed. We showed that BD-contamination and VBD-contamination are equivalent. We also showed that a process that is D-contaminated is also BD-contaminated; however, if a process is BD-contaminated, then it is not necessarily D-contaminated. In other words, if BD-contamination is prevented, then VBD-contamination and D-contamination are also prevented. Hence, we concentrate on the prevention of BD-contamination.

In Theorem 5.11, we show when a process is BD-contamination-free with respect to atomic broadcast.

Theorem 5.11 *Let $B = (\beta, \delta) \in \mathcal{B}$ and F be such that B satisfies atomic broadcast with respect to F. Suppose B satisfies fifo order with respect to F. A correct process p is* BD*-contamination-free until c with respect to atomic broadcast and F if and only if:*

- *if p delivers any message m at or before time c, then $\overrightarrow{\text{DLVD}}_{bc(m)}^{ts(m)}(B) \preceq \overrightarrow{\text{DLVD}}_p(B)$.*

Proof of the "only if" assertion of Theorem 5.11: Suppose a correct process p is BD-contamination-free until c with respect to atomic broadcast and F. Suppose that p delivers $m = (q, c', —)$ by time c. Suppose for contradiction that $\overrightarrow{\text{DLVD}}_q^{c'}(B) \not\preceq \overrightarrow{\text{DLVD}}_p(B)$.

Lemma 5.8 implies that if q is BD-correct until c', then $\overrightarrow{\text{DLVD}}_q^{c'}(B) \preceq \overrightarrow{\text{DLVD}}_p(B)$. Since $\overrightarrow{\text{DLVD}}_q^{c'}(B) \not\preceq \overrightarrow{\text{DLVD}}_p(B)$, q is not BD-correct at c'. Thus, by delivering m by time c, p is BD-contaminated by time c, a contradiction. •

Proof of the "if" assertion of Theorem 5.11:

Suppose whenever a correct process p delivers any message m^* at or before some time c, $\overrightarrow{\text{DLVD}}_{bc(m^*)}^{ts(m^*)}(B) \preceq \overrightarrow{\text{DLVD}}_p(B)$.

Suppose for contradiction that p becomes BD-contaminated by delivering some message m at some time $c' \le c$. Let $q = bc(m)$. By hypothesis, $\overrightarrow{\text{DLVD}}_q^{ts(m)}(B) \preceq \overrightarrow{\text{DLVD}}_p(B)$. The proof continues as a case analysis.

Case: q is not BD-correct until $ts(m)$: Since p is correct and delivers m, the fifo order property implies that for all messages m' that q broadcasts before time $ts(m)$, p delivers m' before m. Thus, $\text{BCAST}_q^{ts(m)}(B) \subseteq \text{DLVD}_p(B)$.

Since p is correct, Lemma 5.1 implies that $\overrightarrow{\text{DLVD}}_p(B) = \overrightarrow{\text{DLVD}}_{\overline{F}}(B)$. Since $\text{BCAST}_q^{ts(m)}(B) \subseteq \text{DLVD}_p(B)$ and $\overrightarrow{\text{DLVD}}_q^{ts(m)}(B) \preceq \overrightarrow{\text{DLVD}}_p(B)$, we conclude that $\text{BCAST}_q^{ts(m)}(B) \subseteq \text{DLVD}_{\overline{F}}(B)$, and $\overrightarrow{\text{DLVD}}_q^{ts(m)}(B) \preceq \overrightarrow{\text{DLVD}}_{\overline{F}}(B)$. Furthermore, since B satisfies the fifo order property when the processes in F are assumed to be faulty, Lemma 5.9 implies that q is BD-correct until $ts(m)$, a contradiction.

Case: q is BD-correct until $ts(m)$ and $\exists (r, c^*) \rightarrow_B (q, ts(m))$, and r is not BD-correct until c^*: Thus, q delivers some message m' at some time $\le ts(m)$ (*i.e.*, $m' \in \overrightarrow{\text{DLVD}}_q^{ts(m)}(B)$), and $(r, c^*) \rightarrow_B (bc(m'), ts(m'))$. Since by hypothesis, $\overrightarrow{\text{DLVD}}_q^{ts(m)}(B) \preceq \overrightarrow{\text{DLVD}}_p(B)$, p also delivers m' in B.

By the well-formedness of B, if any process delivers m, then it delivers m after time $ts(m)$; in particular, $m \notin \overrightarrow{\text{DLVD}}_q^{ts(m)}(B)$. Since p delivers m and m', and $m' \in \overrightarrow{\text{DLVD}}_q^{ts(m)}(B)$, and $\overrightarrow{\text{DLVD}}_q^{ts(m)}(B) \preceq \overrightarrow{\text{DLVD}}_p(B)$, we conclude that p delivers m' before m.

Therefore, by choice of m', p becomes BD-contaminated by delivering m' (before delivering m). This contradicts the choice of m, and completes the proof. □

5.2 Timed atomic broadcasts

Atomic broadcast does not place any restriction on the time a correct process may take to deliver a message. However, by augmenting the specifications of atomic broadcast with the following Δ-timeliness property, we can bound the time required for message delivery.

- Δ-*Timeliness:* If *any* process p delivers a message m at time c on p's clock, then c is at most $ts(m) + \Delta$.

The constant Δ is called the *latency* of the broadcast. We assume that the latency of any broadcast must be greater than 0.

Processes that communicate via timed atomic broadcasts are subject to two sources of incorrectness that are not present with atomic broadcast.

Suppose a correct process delivers a message m, and p does not deliver m by time $ts(m) + \Delta$. If p never delivers m, then p "violates" the agreement property. Furthermore, if p delivers m after $ts(m) + \Delta$, then p "violates" the Δ-timeliness property.

Now suppose that p broadcasts a message m, and does not deliver m by time $ts(m) + \Delta$. In this case, if p never delivers m, then p "violates" the validity property. As before, if p delivers m after $ts(m) + \Delta$, then p "violates" the Δ-timeliness property.

From this, it is straightforward to derive a sufficient condition for an application history to be VBD-consistent with respect to timed atomic broadcast.

Lemma 5.12 *Let $A \in \mathcal{A}$, B be the broadcast history of A, and F be such that B satisfies timed atomic broadcast with respect to F. Suppose B satisfies the fifo order property with respect to F. If for all processes p, for all times c:*

- $\overrightarrow{\mathrm{DLVD}}^c_p(B) \preceq \overrightarrow{\mathrm{DLVD}}_{\overline{F}}(B)$ *(as with atomic broadcast), and*
- *p is not "late" at c in B; that is:*
  - *p delivers its own messages ($\mathrm{BCAST}^{c-\Delta}_p(B) \subseteq \mathrm{DLVD}^c_p(B)$), and*
  - *p delivers messages delivered by the correct processes ($\forall m \in \mathrm{DLVD}_{\overline{F}}(B) : ts(m) \leq c - \Delta \implies m \in \mathrm{DLVD}^c_p(B)$).*

then A is a VBD-consistent history with respect to timed atomic broadcast and F.

It is also straightforward to derive necessary and sufficient conditions for a history to be BD-contamination-free with respect to timed atomic broadcast.

Lemma 5.13 *Let $B = (\beta, \delta) \in \mathcal{B}$ and F be such that B satisfies timed atomic broadcast with respect to F. Suppose B satisfies the fifo order property with respect to F. A correct process p is* BD*-contamination-free until c if and only if:*

- *if p delivers a message m at or before time c, then*
  - $\overrightarrow{\text{DLVD}}^{ts(m)}_{bc(m)}(B) \preceq \overrightarrow{\text{DLVD}}_p(B)$ *(as before), and*
  - *For all $c' \leq ts(m)$, $bc(m)$ does not appear to p to be "late" at c'; that is:*
    - $\text{BCAST}^{c'-\Delta}_{bc(m)}(B) \subseteq \text{DLVD}^{c'}_{bc(m)}(B)$.
    - $\forall m \in \text{DLVD}^{c'}_p(B):\ ts(m) \leq c' - \Delta \implies m \in \text{DLVD}^{c'}_{bc(m)}(B)$.

5.3 The independence property

In Chapter 3, we defined the independence and fifo independence properties. We showed that if a broadcast specification Σ_B has the independence property, then a process p that is D-correct until some time c with respect to Σ_B and a failure set F also exhibits crash behavior until c. We also showed that if Σ_B only has the weaker fifo independence property, then p exhibits crash behavior until c if it is VBD-correct until c.

In this section we show that timed atomic broadcast has the independence property; thus atomic broadcast also has the independence property. Thus, the results presented in Chapter 3 imply that with (timed) atomic broadcast, crash behavior, D-correctness, VBD-correctness and BD-correctness form a hierarchy of increasingly stringent restrictions on the behavior of faulty processes during execution.

For convenience, we repeat the definition of the independence property:

Definition *Broadcast specification Σ_B has the independence property if for all $B = (\beta, \delta) \in \mathcal{B}$, for all F such that $\Sigma_B(B, F)$, for all correct processes p ($p \notin F$), for all times c such that $\beta(p, c) = \phi$, for all messages m of the form $(p, c, —)$, and for all times c', $c \leq c' \leq c + \Delta - 1$, there is a $B' = (\beta', \delta') \in \mathcal{B}$, such that $\Sigma_B(B', F)$, $\delta' \stackrel{c'}{=} \delta$ and $[\beta' = \beta$ except $\beta'(p, c) = m]$.*

Theorem 5.14 *Timed atomic broadcast has the independence property.*

Proof: Let $B = (\beta, \delta) \in \mathcal{B}$ and F be such that B satisfies atomic broadcast with respect to F. Let p be a correct process, and c be a time such that $\beta(p, c) = \phi$. Let m be any message of the form $(p, c, —)$ and let c' be any time, $c \leq c' \leq c + \Delta - 1$. To show that atomic broadcast has the independence property, we construct a $B' = (\beta', \delta') \in \mathcal{B}$, such that $\Sigma_B(B', F)$ where $\delta' \stackrel{c'}{=} \delta$ and $(\beta' = \beta$ except $\beta'(p, c) = m)$.

The construction of B' is as follows:

1. All processes broadcast the same messages at the same time in B and B', except that p also broadcasts the message m at time c in B'. Formally: $\beta' = \beta$ except $\beta'(p,c) = m$.

2. The faulty processes deliver the same messages at the same time in B and B'. Until time c', the correct processes deliver the same messages at the same time in B and B'. Formally: $\delta'_F = \delta_F$ and $\delta'_{\overline{F}} \stackrel{c'}{=} \delta_{\overline{F}}$.

3. After time c', the correct processes deliver messages as follows:

Let x be a correct process that has delivered the most messages by time c'; *i.e.*, $x \notin F$ and $\forall q \notin F$: $\overrightarrow{\text{DLVD}}_q^{c'}(B) \preceq \overrightarrow{\text{DLVD}}_x^{c'}(B)$. Thus, for all correct processes q ($q \notin F$), $[\overrightarrow{\text{DLVD}}_q^{c'}(B) \cdot S_q = \overrightarrow{\text{DLVD}}_x^{c'}(B)]$, for some sequence of messages S_q. Note that S_x is the empty sequence.

Let q be any correct process ($q \notin F$). Informally, at time $c'+1$, q first "catches up" with the messages that process x has delivered by time c', and then delivers m. After delivering m, q delivers the same messages in B' that x delivers in B. Formally:

- $\delta'(q, c'+1) = S_q \cdot \langle m \rangle \cdot \delta(x, c'+1)$
- $\forall c'' > c'+1 : \delta'(q, c'') = \delta(x, c'')$.

Claim: B' is well-formed.

Proof of claim: We show below that B' satisfies the three properties required of a well-formed history (as defined in Chapter 2).

1. It is clear that β' is a history broadcast function and δ' is a history delivery function.

2. Suppose for some process q, time c^*, and message $m^* \neq \phi$, $m^* \in \delta'(q, c^*)$. We prove that $ts(m^*) < c^*$.

Suppose that q is faulty ($q \in F$), or that $c^* \leq c'$. Since $\delta'_F = \delta_F$ and $\delta'_{\overline{F}} \stackrel{c'}{=} \delta_{\overline{F}}$, $\delta'(q, c^*) = \delta(q, c^*)$. Since B is well-formed (by hypothesis), $ts(m^*) < c^*$.

Suppose that q is correct, $c^* > c'$, and $m^* \neq m$. From the construction, process x delivered m^* at c^* in B. Since B is well-formed (by hypothesis), $ts(m^*) < c^*$.

Suppose that q is correct, $c^* > c'$, and $m^* = m$. From the construction, q delivers m at time $c'+1$; *i.e.*, $c^* = c'+1$. Since $ts(m) = c$, and $c \leq c'$, and $c^* = c'+1$, $ts(m) < c^*$.

3. Consider any message $m^* \neq \phi$ broadcast by any process q at any time c^* in B'. From the construction, and the well-formedness of B, it is straightforward to show that m^* is of the form $(q, c^*, —)$.

•

Claim: $\delta' \stackrel{c'}{=} \delta$ and ($\beta' = \beta$ except $\beta'(p,c) = m$).

Proof of claim: Directly from the construction. •

Claim: B' satisfies timed atomic broadcast with respect to F.

Proof of claim: We show that B' satisfies each of the properties of atomic broadcast when the processes in F are assumed to be faulty.

Agreement and total order: To prove that B' satisfies both the agreement and the total order properties, we show that any correct process delivers the same sequence of messages in B' as any other correct process.

Let q and r be correct processes (*i.e.*, $q \notin F, r \notin F$). From the construction, $\overrightarrow{\text{DLVD}}_q(B') = \overrightarrow{\text{DLVD}}_x^{c'}(B) \cdot \langle m \rangle \cdot \delta(x, c'+1)\ldots$, and $\overrightarrow{\text{DLVD}}_r(B') = \overrightarrow{\text{DLVD}}_x^{c'}(B) \cdot \langle m \rangle \cdot \delta(x, c'+1)\ldots$. Thus, $\overrightarrow{\text{DLVD}}_q(B') = \overrightarrow{\text{DLVD}}_q(B')$.

Validity: Let q be any correct process ($q \notin F$). To prove that B' satisfies the validity property, we show that q delivers all of its own broadcasts in B'; *i.e.*, $\text{BCAST}_q(B') \subseteq \text{DLVD}_q(B')$.

Since B satisfies atomic broadcast with respect to F, $\text{BCAST}_q(B) \subseteq \text{DLVD}_q(B)$. Note also that by construction, $\text{DLVD}_q(B') = \text{DLVD}_q(B) \bigcup \{m\}$.

If $q \neq p$, then by construction, $\text{BCAST}_q(B') = \text{BCAST}_q(B)$, and hence $\text{BCAST}_q(B') \subseteq \text{DLVD}_q(B')$.

If $q = p$, then by construction, $\text{BCAST}_q(B') = \text{BCAST}_q(B) \bigcup \{m\}$, and hence $\text{BCAST}_q(B') \subseteq \text{DLVD}_q(B')$.

Integrity: It is straightforward to show that the integrity property is satisfied.

Δ-Timeliness: Let q be any process, and suppose q delivers some message m^* at time c^* in B'. To prove that B' satisfies the Δ-timeliness property, we show that $c^* \leq ts(m^*) + \Delta$.

Suppose q is faulty. Since $\delta'_q = \delta_q$, q delivers m^* at time c^* in B. Since B satisfies timed atomic broadcast (*i.e.*, B satisfies Δ-timeliness), $c^* \leq ts(m^*) + \Delta$.

Suppose q is correct, and $m^* \neq m$. From the construction, at least one of process q and process r delivers m^* at time c^* in B. Since B satisfies Δ-timeliness, $c^* \leq ts(m^*) + \Delta$.

Suppose q is correct, and $m^* = m$. By construction, m is delivered at time $c'+1$, where $c \leq c' < c + \Delta - 1$. Since $ts(m) = c$, and $c^* = c'+1$, $c^* \leq ts(m) + \Delta$.

□

Corollary 5.15 *Atomic broadcast has the independence property.*

Proof: From Theorem 5.14. □

5.4 The choice property

In Chapter 4, we defined the choice property. We proved that when solving a certain class of problems using broadcasts that satisfy the choice property, the prevention of contamination is "as good as" the prevention of inconsistency.

For convenience, the definition of choice property is repeated below:

Definition *Broadcast specification Σ_B has the choice property if for all $B = (\beta, \delta) \in \mathcal{B}$, for all F such that $\Sigma_B(B, F)$:*

- *For all m, if $B' = Subtract(B, \{m\})$, then $B' \in \mathcal{B}$ and $\Sigma_B(B', F)$, and*
- *For all $p \in F$, for all c, if $B' = Deafen(B, \{(p, c)\})$, then $B' \in \mathcal{B}$ and $\Sigma_B(B', F)$.*

Atomic broadcast has the choice property. To prove this, we must show that when processes communicate via atomic broadcast:

- Any process p can unilaterally decide whether or not to broadcast a message at any time c.
- Any *faulty* process p can unilaterally decide whether or not to stop delivering messages after any time c.

Since timed atomic broadcast and atomic broadcast place no restriction on when a process can broadcast a message, or on the behavior of faulty processes, (other than the integrity and Δ-timeliness properties), it is straightforward to show that both the broadcasts have the choice property. In the interest of brevity, we omit the proofs.

Theorem 5.16 *Timed atomic broadcast and atomic broadcast have the choice property.*

Chapter 6

Extending the Formal Model

Few distributed systems provide fault-tolerant broadcasts as "built-in" communications primitives. Hence fault-tolerant broadcasts are implemented by protocols that use the low-level communications mechanisms available in the system—such protocols are called *broadcast protocols.*

This chapter extends the model introduced in Chapter 2 to describe the execution of protocols based on fault-tolerant broadcasts in systems where processes communicate via point-to-point messages. We only consider systems where processes have perfectly synchronized clocks and link delays are bounded. We assume that processes may only fail by *omission*; *i.e.*, by prematurely halting, or by intermittently omitting to send and/or receive messages.

We define what it means for a broadcast protocol Θ to *implement* a broadcast specification Σ_B in a system S and prevent inconsistency and/or contamination. Informally, such a protocol only permits broadcast histories that satisfy Σ_B and satisfy consistent and/or contamination-free behavior with respect to the processes that *actually* failed during execution; *i.e.*, when the processes that are assumed faulty are exactly those processes that are actually faulty. To prevent a faulty process from becoming inconsistent (*e.g.*, by delivering an incorrect message m), the broadcast protocol is permitted to "force" such a process to halt. However, if the protocol halts a process, then we require that the process must already have failed (for example, by omitting to send or receive a point-to-point message).

Finally, we show that if protocol Π solves a problem using a broadcast specification assuming consistent or contamination-free behavior, and broadcast protocol Θ implements the broadcast in a system S, ensuring consistency or contamination-freedom (as appropriate), then Π solves the problem using Θ in S.

6.1 Extended model

In this section, we extend the formal model of computation presented in Section 2.1. Where possible, we will *not* repeat any information already presented in that section.

Let $\mathcal{P}$ denote the set of processes in a distributed system. The processes communicate by message passing along bidirectional *communication links* (sometimes abbreviated to *links*). The processes execute an *application protocol* which assumes that the processes communicate exclusively via a *fault-tolerant broadcast.* The processes concurrently execute a *broadcast protocol*, a protocol that implements the broadcast when processes communicate via point-to-point message passing.

Each process maintains a *local clock* taken from the set of positive integers, denoted $\mathcal{I}$. In general, a process's clock time may be different from the *real time.* Let $\mathcal{R}$ denote the set of real times.

6.1.1 Process state

The *state* of each process consists of a *application state* and a *communication state.* The application state is taken from the set $\mathcal{Q}_\Pi$, and informally corresponds to the part of the process state that is "associated" with the application protocol that the processes are executing. The communication state is taken from the set $\mathcal{Q}_\Theta$, and informally corresponds to the part of the processes state that is "associated" with the broadcast protocol that the processes are executing. Both the set of application states and the set of communication states include the special state $\perp$, called the *premature halt state.*

6.1.2 Sending and receive messages

The processes communicate by *sending* and *receiving* point-to-point *messages.* A *point-to-point message* is a tuple of the form $(p, q, c, data)$, where p is the sender of the message (denoted $snd(m)$), q is the intended recipient of the message (denoted $rpt(m)$), c (the *timestamp*) is the time on p's clock at which the message was broadcast (denoted $ts(m)$), and *data* is a sequence of bits. Let $\mathcal{S}$ denote the set of all point-to-point messages, and let $\mathcal{S}^+ = 2^{\mathcal{S}}$ denote the set of all subsets of $\mathcal{S}$.

If a process p invokes *send* (S) at clock time c, where S is a set of point-to-point messages, then for each $m \in S$, we say that p *sent* m at c. If the result of process p invoking the *receive* primitive at time c is the set of point-to-point messages S, then for each message $m \in S$, we say that p *receives* m at c. This is further described in Section 6.2.

6.1.3 Broadcast protocols

Processes execute a *broadcast protocol*, Θ, which specifies the messages to be delivered by the processes, the point-to-point messages to be sent by the processes and the communication state transitions made by the processes. A broadcast protocol consists of four functions: the *delivery function*, denoted Θ_d, the *sending function*, denoted Θ_s, the *communication state transition function*, denoted Θ_r and the *halt-decision function*, denoted Θ_h.

The delivery function determines the next messages to be delivered; formally, $\Theta_d : \mathcal{P} \times \mathcal{I} \times \mathcal{Q}_\Theta \times \mathcal{S}^+ \rightarrow \mathcal{M}^+$. If at time c, p is in communication state s and receives the set of point-to-point messages S, then p should deliver the sequence of messages $\Theta_d(p, c, s, S)$.

The sending function determines the next messages to be sent; formally, $\Theta_s : \mathcal{P} \times \mathcal{I} \times \mathcal{Q}_\Theta \times \mathcal{S}^+ \times (\mathcal{M} \bigcup \phi) \rightarrow \mathcal{S}^+$. If at time c, p is in communication state s, receives the set of point-to-point messages S, and wishes to broadcast the message m, then p should send the set of point-to-point messages $\Theta_s(p, c, s, S, m)$.

The communication state transition function determines the next communication state; formally, $\Theta_r : \mathcal{P} \times \mathcal{I} \times \mathcal{Q}_\Theta \times \mathcal{S}^+ \times (\mathcal{M} \bigcup \phi) \rightarrow \mathcal{Q}_\Theta$. If at time c, p is in communication state s, receives the set of point-to-point messages S, and broadcasts the message m, then p should enter communication state $\Theta_r(p, c, s, S, m)$.

6.1.4 Premature halting

The set of possible communication states includes the special state $\perp$, the (premature) halt state. Recall that Chapter 2 stipulated that a protocol Π cannot halt a process by changing its state to $\perp$ from a state other than $\perp$. In contrast, a broadcast protocol's halt-decision function Θ_h *can* halt a process. Formally: $\Theta_h : \mathcal{P} \times \mathcal{I} \times \mathcal{Q}_\Theta \times \mathcal{S}^+ \rightarrow \{OK, \perp\}$. If at time c, p is in communication state s and receives the set of point-to-point messages S, then p should prematurely halt if $\Theta_h(p, c, s, S)$ is $\perp$.

Once a process halts, it remains halted:

- $\forall p, \forall c, \forall S, \forall m : \;\; \Theta_r(p, c, \perp, S, m) = \perp$.

Once a process halts, it does not subsequently send any point-to-point messages:

- $\forall p, \forall c, \forall S, \forall m : \;\; \Theta_s(p, c, \perp, S, m) = \phi$.

Once a process halts, it does not subsequently deliver any messages:

- $\forall p, \forall c, \forall S : \;\; \Theta_d(p, c, \perp, S) = \phi$.

6.1.5 Execution of an application protocol using a broadcast protocol

```
/*        Initialization          */
  (s_Π, s_Θ) := (s_Π,init, s_Θ,init) ? (⊥, ⊥)
  c := 1

/*        Main Loop               */
do forever
  (s_Π, s_Θ) := (s_Π, s_Θ) ? (⊥, ⊥)
  S ≡ receive()                       /* Θ receives messages from other processes */
  if Θ_h(p, c, s_Θ, S) = ⊥
     then (s_Π, s_Θ) := (⊥, ⊥)        /* Θ decides if p should halt            */
  D ≡ deliver() ≡ Θ_d(p, c, s_Θ, S)   /* Θ "tells" Π to deliver messages        */
  (s_Π, s_Θ) := (s_Π, s_Θ) ? (⊥, ⊥)
  m := Π_m(p, c, s_Π, D))
  broadcast m                         /* Π "tells" Θ to broadcast a message     */
  (s_Π, s_Θ) := (s_Π, s_Θ) ? (⊥, ⊥)
  send (Θ_s(p, c, s_Θ, S, m))         /* Θ sends messages to other processes    */
  (s_Π, s_Θ) := (Π_r(p, c, s_Π, D),  Θ_r(p, c, s_Θ, S, m)) ? (⊥, ⊥)
  c := c + 1
od
```

Figure 6.1: The execution of Π by process p

The execution of a protocol Π using broadcast protocol Θ is illustrated in Figure 6.1 (compare with Figure 2.1 in Chapter 2). Each process is initialized to application state $s_{\Pi,init}$, communication state $s_{\Theta,init}$ and to a clock value of 1. At each clock tick, the broadcast protocol layer receives point-to-point messages from other processes, and determines whether or not the process must halt. The broadcast protocol layer then instructs the application protocol layer to deliver a sequence of messages. The application protocol layer subsequently informs the broadcast protocol layer if a message is to be broadcast. Then, the broadcast protocol layer sends a set of point-to-point messages to other processes. Finally, the process changes state and increments its clock.

A process may also non-deterministically halt during execution as the result of a failure; such a halt is denoted $(s_\Pi, s_\Theta) := (s_\Pi, s_\Theta) \ ? \ (\bot, \bot)$. As indicated in Figure 6.1, we stipulate that such a failure may only occur at certain points during execution. This assumption does not, however, affect the generality of our results.

6.2 Point-to-point message passing

We assume that point-to-point message passing is reliable. Informally, a point-to-point message sent by a correct process reaches its destination without corruption or duplication. Furthermore, if any process receives a point-to-point message, then that message was actually sent by some process. We also assume that a correct process always receives a point-to-point message sent by another correct process.

Let the *link delay* λ be an upper bound on the *time* required for a point-to-point message sent by a correct process to be received by another correct process.

To model reliable point-to-point message passing, we assume that each process p has an *input* and an *output* buffer for point-to-point messages, and that both buffers are initially empty. Let $IB_{p,c}$ denote the contents of p's input buffer immediately *after* p's clock changes to c, and let $OB_{p,c}$ denote the contents of p's output buffer immediately *before* p's clock changes to $c+1$. We assume that for all processes p, $IB_{p,1}$ and $OB_{p,0}$ are ϕ.

- *Sending:* Suppose process p invokes *send* (S) at time c: If $m = (p, q, —, —) \in S$, then m should be added to $OB_{p,c}$; *i.e.*, $m \in OB_{p,c}$. However, if p fails during its invocation of the *send* primitive, only a subset of S may be added to $OB_{p,c}$. If $m \in S$ and m is added to $OB_{p,c}$, then we say p *actually sends* m at time c.
- *Reliable transmission:* Suppose point-to-point message $m = (p, q, —, —)$ is added $OB_{p,c}$ (at time c); *i.e.*, $m \notin OB_{p,c-1}$ and $m \in OB_{p,c}$: At some time $c', c < c' \leq c + \lambda$, m is removed from $OB_{p,c}$ and added to $IB_{q,c'}$ as a single, indivisible action; *i.e.*, $m \in IB_{q,c'}$ and $m \notin OB_{p,c'}$.
- *Receiving:* Suppose process p invokes the *receive* primitive at time c: This should result in the following indivisible actions—the *receive* primitive returns S, the set of point-to-point messages in $IB_{p,c}$, and resets the input buffer to ϕ; *i.e.*, $IB_{p,c}$ is returned, and $\forall m : m \in IB_{p,c}, \implies m \notin IB_{p,c+1}$.

  If S is the result of p invoking *receive* () at time c, for all $m \in S$, we say p *receives* m at time c.

  If p fails during the receive primitive, the set of messages returned by the receive primitive may only be a subset of $IB_{p,c}$. Furthermore, for all messages $m \in IB_{p,c}$, if m is not returned by the receive primitive, then p never receives m; *i.e.*, after the *receive* primitive, the input buffer is reset to ϕ.

6.3 Extended histories

The following functions describe the execution of an application protocol Π using a broadcast protocol Θ.

- The functions σ, β and δ, which were defined in Chapter 2.
- The function $\sigma_\Theta : \mathcal{P} \times \mathcal{I} \rightarrow \mathcal{Q}_\Theta$ is called the *history communication state function*—$\sigma_\Theta(p, c)$ is process p's communication state when its clock changes to c.
- The function $\mu : \mathcal{P} \times \mathcal{I} \rightarrow \mathcal{S}^+$ is called the *history sending function*—$\mu(p, c)$ is the set of point-to-point messages that p actually sent (added to its output buffer) at clock time c; *i.e.*, $\mu(p, c) = OB_{p,c} - OB_{p,c-1}$.
- The function $\rho : \mathcal{P} \times \mathcal{I} \rightarrow \mathcal{S}^+$ is called the *history receiving function*—$\rho(p, c)$ is the set of point-to-point messages that p actually receives at clock time c.
- The function $\tau : \mathcal{P} \times \mathcal{R} \rightarrow \mathcal{I}$ is called the *history real time function*—$\tau(p, t)$ is p's clock time at real time t.

Let Π be an application protocol, Θ be a broadcast protocol, σ, β and δ be history state, broadcast and delivery functions respectively, and σ_Θ, μ and ρ be history communication state, sending and receiving functions respectively. The tuple $E = ((\Pi, \sigma, \beta, \delta), (\Theta, \sigma_\Theta, \mu, \rho, \tau))$ is an *extended history of protocol* Π *using broadcast protocol* Θ (or the *history of* Π *using* Θ). The tuple $H = (\Pi, \sigma, \beta, \delta)$ is the *protocol history* of E. The tuple $A = (\sigma, \beta, \delta)$ is the application history of E (also of H). The tuple $C = ((\beta, \delta), (\Theta, \sigma_\Theta, \mu, \rho, \tau))$ is the *communication history of E*. The tuple $B = (\beta, \delta)$ is the broadcast history of E (also of H and of C). If $C = ((\beta, \delta), (\Theta, \sigma_\Theta, \mu, \rho, \tau))$, then C is a *communication history using* Θ (C *uses* Θ).

A communication history $C = ((\beta, \delta), (\Theta, \sigma_\Theta, \mu, \rho, \tau))$ is *well-formed* if:

- Θ is a broadcast protocol, σ_Θ is a history communication state function, μ is a history sending function, and ρ is a history receiving function.
- $B = (\beta, \delta)$ is a well-formed broadcast history.
- Every process initializes its communication state to $s_{\Theta,init}$ or halts before initialization.
  $\forall p : \ \sigma_\Theta(p, 1) = s_{\Theta,init}$ or $\sigma_\Theta(p, 1) = \perp$.
- Once p halts, it remains halted, and sends and receives no further point-to-point messages, and broadcasts and delivers no further messages:
  $\forall p, \forall c : \ \sigma_\Theta(p, c) = \perp \implies \sigma_\Theta(p, c + 1) = \perp$ and
  $\mu(p, c) = \rho(p, c) = \beta(p, c) = \delta(p, c) = \phi.$

- Once p is "forced" to halt, it remains halted, and sends no further point-to-point messages, and broadcasts and delivers no further messages:
  $\forall p, \forall c:\ \Theta_h(p, c, \sigma_\Theta(p, c), \rho(p, c)) = \bot \implies$
  $\sigma_\Theta(p, c+1) = \bot$ and $\mu(p, c) = \beta(p, c) = \delta(p, c) = \phi$.
- Every process changes communication state according to Θ, or halts:
  $\forall p, \forall c:\ \sigma_\Theta(p, c+1) = \Theta_r(p, c, \sigma_\Theta(p, c), \rho(p, c), \beta(p, c))$ or $\sigma_\Theta(p, c+1) = \bot$.
- Every process delivers messages according to Θ, or halts:
  $\forall p, \forall c:\ \delta(p, c) = \Theta_d(p, c, \sigma_\Theta(p, c), \rho(p, c))$ or $(\delta(p, c) = \phi, \sigma_\Theta(p, c+1) = \bot)$.
- At all times c, if a process sends or receives a point-to-point message, then the point-to-point message is of the form $(p, —, c, —)$.
  $\forall p, \forall c, \forall m \neq \phi:\ \ m \in \mu(p, c)$ or $m \in \rho(p, c) \implies m = (p, —, c, —)$.
- No process sends "extra" point-to-point messages:
  $\forall p, \forall c: \mu(p, c) \subseteq \Theta_s(p, c, \sigma_\Theta(p, c), \rho(p, c), \beta(p, c))$.
- Every process receives any point-to-point message m at most once:
  $\forall p, \forall c:\ \forall m \in \rho(p, c) \implies \forall c' \neq c : m \notin \rho(p, c')$.
- If any process receives a point-to-point message, then the message was sent:
  $\forall p, \forall c:\ \forall m \in \rho(p, c) \implies \exists q, \exists c' : m \in \mu(q, c')$.
- Message receipt does not take "zero time," and takes no longer than λ time:
  $\forall p, \forall c : \forall m \in \rho(p, c) \implies 0 < c - ts(m) \leq \lambda$.
- All processes have perfectly synchronized clocks:
  $\forall p, \forall q, \forall t:\ \tau(p, t) = \tau(q, t)$

Note that a well-formed communication history corresponds to the communication history of an execution in which processes are not subject to *Byzantine* (*i.e.*, *arbitrary*) failures [LSP82], message passing is reliable, and clocks are perfectly synchronized.

An extended history $E = ((\Pi, \sigma, \beta, \delta),\ (\Theta, \sigma_\Theta, \mu, \rho, \tau))$ is *well-formed* if:

- $C = ((\beta, \delta), (\Theta, \sigma_\Theta, \mu, \rho, \tau))$ is well-formed.
- $H = (\Pi, \sigma, \beta, \delta)$ is well-formed.
- When a process halts, both the communication state and the application state are set to $\bot$:
  $\forall p, \forall c:\ \ \sigma_\Theta(p, c) = \bot \iff \sigma(p, c) = \bot$.

6.4 Systems and process failures

A *system* S is a set of well-formed communication histories. We say a communication history C *is a (communication) history of system* S, if $C \in S$. An extended history E is a *history of* S (denoted $E \in S$) if C, the communication history of E, is such that $C \in S$. A broadcast history B *is a (broadcast) history of* S (denoted $B \in S$) if B is the broadcast history of a communication history C and $C \in S$. A protocol history H is a *protocol history of* S, (denoted $H \in S$) if B, the broadcast history of H, is such that $B \in S$.

Suppose $C = ((\beta, \delta), (\Theta, \sigma_\Theta, \mu, \rho, \tau))$ is a well-formed communication history. We first define what it means for a process p to fail to send and receive messages at time c:

- *p sends correctly at c in C* if $\mu(p, c) = \Theta_s(p, c, \sigma_\Theta(p, c), \rho(p, c), \beta(p, c))$.
- *p receives correctly at c in C* if $\forall q, \forall m : \ m \in \mu(q, c - \lambda) \ \bigwedge \ p = rpt(m) \implies \exists c' \leq c : m \in \rho(p, c')$

The above definitions lead to the following definitions:

- A process p is *correct at c in C* if p does not halt by time c ($\sigma_\Theta(p, c+1) \neq \perp$), and for all $c' \leq c$, p sends and receives messages correctly at time c' in C.
- A process p *fails by omission by time c in C* if p halts by time c, or there is a time $c' \leq c$, such that p fails to send or receive messages correctly at time c' in C.

Note that p is not correct at c in C if and only if p fails by omission by c in C. Since we are assuming that all systems are sets of well-formed histories, the above implies that we only consider systems in which processes have perfectly synchronized clocks, and may fail by omission. In particular, we do not consider systems that are subject to *timing failures* [CASD85].

Let $\mathcal{F}_C^c$ denote the set of processes that are not correct at time c in C ($\mathcal{F}_C^c \equiv \{p \mid p$ is not correct at c in $C\}$), and let $\mathcal{F}_C$ denote the set $\bigcup_{c>0} \mathcal{F}_C^c$.

If E is an extended history, and C is E's communication history, then for all times c, $\mathcal{F}_E^c \equiv \mathcal{F}_C^c$, and hence $\mathcal{F}_E \equiv \mathcal{F}_C$.

The *maximum number of faulty processes in a system S*, denoted f, is equal to $max\{|\mathcal{F}_C| \mid C \in S\}$.

6.5 Consistency and contamination in the extended model

In this section, we define consistency and contamination-freedom in communication histories.

Let $C = ((\beta, \delta), (\Theta, \sigma_\Theta, \mu, \rho, \tau))$ be any communication history, and let $B = (\beta, \delta)$. For the rest of this section we will be using this C and B.

Definition *Process p is* X*-contamination-free in C with respect to Σ_B and S if for all c, p is not* X*-contaminated at time c in B with respect to Σ_B and $\mathcal{F}_C$.*

We define $Halt(C,p)$ to be the time at which p halts in C, and $LVT(C,p)$ to be the time of p's last "visible action" in C. Our definitions are analogous to those in Chapter 3 of $Halt(A,p)$ and $LVT(A,p)$ for an application history A.

$$\begin{aligned} Halt(C,p) &= \begin{cases} \infty & \text{if } p \text{ does not halt} \\ min\{c \mid \sigma_\Theta(p,c+1) = \bot)\} & \text{otherwise} \end{cases} \\ LVT(C,p) &= max\,(Halt(C,p)-1,\ LB(B,p)) \end{aligned}$$

Thus, we define a consistent process as follows:

Definition *Process p is* X*-consistent in C with respect to Σ_B and S if for all $c \leq LVT(C,p)$, p is* X*-correct until time c in B with respect to Σ_B and $\mathcal{F}_C$.*

We now define consistent and contamination-free communication histories.

Definition *C is* X*-consistent (*X*-contamination-free) with respect to Σ_B and S if and only if for all p, p is* X*-consistent (*X*-contamination-free) in C with respect to Σ_B and S.*

Recall that $C = ((\beta,\delta),(\Theta,\sigma_\Theta,\mu,\rho,\tau))$ and $B = (\beta,\delta)$. Consider any extended history E, such that C is E's communication history. Suppose $E = ((\Pi,\sigma,\beta,\delta),\ (\Theta,\sigma_\Theta,\mu,\rho,\tau))$. Let $A = (\sigma,\beta,\delta)$. Lemmas 6.1 and 6.2 compare X-consistent and X-contamination-freedom in the two histories C and A.

Lemma 6.1 *Process p is* X*-consistent in C with respect to Σ_B and S if and only if p is* X*-consistent in A with respect to Σ_B and $\mathcal{F}_C$.*

Proof: If p is X-consistent in C with respect to Σ_B and S, then for all $c \leq LVT(C,p)$, p is X-correct until time c in B with respect to Σ_B and $\mathcal{F}_C$. If p is X-consistent in A with respect to Σ_B and $\mathcal{F}_C$, then for all $c \leq LVT(A,p)$, p is X-correct until time c in B with respect to Σ_B and $\mathcal{F}_C$. Thus, the lemma follows if we show that $LVT(A,p)$ equals $LVT(C,p)$.

Since E is well-formed, $\forall p, \forall c : \sigma(p,c) = \bot \iff \sigma_\Theta(p,c) = \bot$. Thus, $Halt(A,p) = Halt(C,p)$, and hence $LVT(A,p) = LVT(C,p)$. □

Lemma 6.2 *Process p is* X*-contamination-free in C with respect to Σ_B and S if and only if p is* X*-contamination-free in A with respect to Σ_B and $\mathcal{F}_C$.*

The following lemma follows from definitions and the above lemmas:

Lemma 6.3 *C is* X*-consistent (*X*-contamination-free) with respect to Σ_B and S if and only if A is* X*-consistent (*X*-contamination-free) with respect to Σ_B and $Failed(C,\lambda)$.*

6.6 Broadcast protocols implementing a broadcast specification

So far, we have not excluded trivial broadcast protocols, such as one which forces all processes, correct *and* faulty, to halt. To preclude such protocols, we first define what it means for a broadcast protocol Θ to *respect correctness* in a communication history C. Informally, such a broadcast protocol does not "force" a correct process to halt. In other words, if Θ "forces" a process p to halt at time c, then p failed by the time the halt-decision function was evaluated at time c; *i.e.*, p failed by time $c-1$, or p failed to receive messages at time c.

Definition *Broadcast protocol* Θ *respects correctness in communication history* $C = ((\beta, \delta), (\Theta, \sigma_\Theta, \mu, \rho, \tau))$ *if for all* p *and* c *such that* $\Theta_h(p, c, \sigma_\Theta(p, c), \rho(p, c)) = \perp$, *either* $p \in \mathcal{F}_C^{c-1}$, *or* p *does not receive correctly at* c *in* C.

The following definitions state what it means for a broadcast protocol to implement a broadcast specification in a system.

Definition *Broadcast protocol* Θ *implements broadcast specification* Σ_B *in system* S *if, for all* $C = ((\beta, \delta), (\Theta, \sigma_\Theta, \mu, \rho, \tau)) \in S$, $\Sigma_B((\beta, \delta), \mathcal{F}_C)$ *and* Θ *respects correctness in* C.

Definition *Broadcast protocol* Θ *implements broadcast specification* Σ_B *and ensures* X*-consistent (*X*-contamination-free) behavior in* S *if*

- Θ *implements* Σ_B *in* S, *and*
- *for all communication histories* $C \in S$ *such that* C *uses* Θ, C *is* X*-consistent (*X*-contamination-free) with respect to* Σ_B *and* S.

6.7 Solving a problem

We have already defined what it means for an application protocol to solve a problem using a broadcast specification (Chapter 2). We now define what it means for an application protocol to solve a problem using a particular broadcast protocol in a system.

Definition *Application protocol* Π *solves problem* Σ_A *using broadcast protocol* Θ *in system* S *if and only if* $\forall E = ((\Pi, \sigma, \beta, \delta), (\Theta, \sigma_\Theta, \mu, \rho, \tau)) \in S$, $\Sigma_A((\sigma, \beta, \delta), \mathcal{F}_E)$.

Finally, we prove the following theorem that relates the two concepts mentioned above: solving a problem using a broadcast specification, and solving a problem using a broadcast protocol in a system.

Theorem 6.4 *If an application protocol* Π *solves a problem* Σ_A *using a broadcast specification* Σ_B *assuming* X*-consistent (*X*-contamination-free) behavior, and a broadcast protocol* Θ *implements* Σ_B *and ensures* X*-consistent (*X*-contamination-free) behavior in some system* S*, then* Π *solves* Σ_A *using* Θ *in* S.

Proof: Suppose Π solves Σ_A using Σ_B assuming X-consistent (X-contamination-free) behavior, and Θ implements Σ_B and ensures X-consistent (X-contamination-free) behavior in S.

Suppose $E = ((\Pi, \sigma, \beta, \delta), (\Theta, \sigma_\Theta, \mu, \rho, \tau))$ is any extended histories of Π using Θ in system S. Let C, H, A and B be E's communication, protocol, application and broadcast histories respectively. To prove the theorem, we must show that $\Sigma_A(A, \mathcal{F}_E)$ is satisfied.

Since $E \in S$, $C \in S$. Since Θ implements Σ_B, $\Sigma_B(B, \mathcal{F}_C)$. Since Θ also ensures X-consistent (X-contamination-free) behavior, C is X-consistent (X-contamination-free) with respect to Σ_B and S. By Lemma 6.3, A is X-consistent (X-contamination-free) with respect to Σ_B and $\mathcal{F}_C$.

Since $E \in S$, $H \in S$. Since Π solves Σ_A using Σ_B assuming X-consistent (X-contamination-free) behavior, for all failure sets F, if $\Sigma_B(B, F)$ and A is X-consistent (X-contamination-free) with respect to Σ_B and F, then $\Sigma_A(A, F)$ is satisfied.

Since $\Sigma_B(B, \mathcal{F}_C)$ is satisfied, and A is X-consistent (X-contamination-free) with respect to Σ_B and $\mathcal{F}_C$, $\Sigma_A(A, \mathcal{F}_C)$ is satisfied. Recall $\mathcal{F}_E \equiv \mathcal{F}_C$, and hence $\Sigma_A(A, \mathcal{F}_E)$ is satisfied.

We have shown that Π solves Σ_A using Θ in S, thus proving the theorem. □

Chapter 7

Atomic Broadcast Protocols

In Chapter 5, we derived definitions of inconsistency and contamination with respect to atomic broadcast. In this chapter, we present atomic broadcast protocols that prevent inconsistency and/or contamination in systems where processes are subject to omission failures.

7.1 Preliminaries

When informally describing a protocol, we "follow" the broadcast of a particular message m, by describing all the actions taken by the broadcaster of m (to broadcast m), and the actions taken by any other process to deliver m.

A broadcast protocol is formally described using the four functions provided by a broadcast protocol—the delivery function, the sending function, the transition function and the halt-decision function (See Chapter 6). However, for simplicity, we avoid such a formal description.

We describe a broadcast protocol operationally (in pseudo-code), by giving the actions taken by any process p at any instant of clock time c. In general, at time c, p receives all incoming point-to-point messages, determines whether or not to halt, [1] delivers messages, and then sends point-to-point messages. To make the protocols more readable, we choose to follow a more "natural" presentation; for example, by giving the pseudo-code required to broadcast a message before giving the pseudo-code required to deliver a message. However, the actual sequence of operations is as outlined above and formally defined in Figure 6.1 in Chapter 6.

To prove that a broadcast protocol implements broadcast specification Σ_B in a

[1]Recall that a broadcast protocol is permitted to force a faulty process to halt before it becomes inconsistent.

system S, we prove that for all primitive histories C of the protocol in system S, the corresponding broadcast history B satisfies Σ_B with respect to the processes that are *actually* faulty; *i.e.*, $\Sigma_B(B, \mathcal{F}_C)$ is satisfied. In this context, a *faulty* process p is one that is in $\mathcal{F}_C$, and a *correct* process q is one that is not in $\mathcal{F}_C$.

7.1.1 Communication

Recall that in the extended model (Chapter 6), we assumed that processes communicate via reliable message passing (Section 6.2). We write "p *sends* m to q at time c" when we mean that p sends the point-to-point message (p, q, m, c) at time c. We write "p *sends* m to all at time c" when we mean that for all processes q, p sends the point-to-point messages (p, q, m, c).

Without loss of generality, we assume that the message sends and receives obey the following fifo-links property:

- *Fifo-links:* If p *sends* m to q before p *sends* m' to q, and q *receives* m' at time c', then q *receives* m at time $c \leq c'$.
  If p *sends* m and m' to q at time c, and q *receives* m' at time c', then q also *receives* m at time c'.

Using "piggybacking" techniques and/or sequence numbers, it is straightforward to implement send and receive primitives with the above fifo-links property.

To simplify the presentation, our atomic broadcast protocols use (as a subroutine) a fault-tolerant broadcast called a *basic broadcast*. Formally, a basic broadcast is a timed reliable broadcast that also satisfies the following *uniform fifo order* property:

- *Uniform Fifo Order:* If *any* process p broadcasts a message m before it broadcasts a message m', and *any* process q delivers m', then q delivers m before m'.

Note that p and q may both be faulty in the above definition.

Thus, a basic broadcast is one that satisfies the validity, agreement, integrity, Δ-timeliness and uniform fifo order properties.

It is straightforward to implement a basic broadcast with a latency of $\Delta = (f+1)\lambda$ in a system subject to omission failures, where λ is the link delay and f is the maximum number of faulty processes in the system.

To avoid possible confusion (since we use more than one type of broadcast in our protocols), we "tag" each broadcast and delivery with the broadcast type— for example, we write *broadcast* (*atomic*, m) to mean that m is broadcast using an atomic broadcast protocol.

When informally describing an atomic broadcast protocol, or in a proof of correctness, we sometimes write "p atomically broadcasts (delivers) m" instead of "p broadcasts (delivers) m using an atomic broadcast;" similarly we sometimes write "p basic broadcasts (delivers) m" instead of "p broadcasts (delivers) m using a basic broadcast."

7.1.2 Overview of results

We first present a simple atomic broadcast protocol. This protocol has a *latency* of Δ; that is, if a process atomically delivers a message m at time c in any execution of the protocol, then $c - ts(m) \leq \Delta$.

We present an atomic broadcast protocol that ensures BD-contamination-free behavior, and has a latency of Δ (which is known to be optimal). [2] We also show how to reduce the message complexity of this protocol. Indeed, similar message reduction techniques can be used to reduce the message complexity of all the subsequent protocols.

Recall that the prevention of BD-contamination is a more stringent restriction than the prevention of other forms of contamination. Therefore, our protocols prevent D-contamination, VBD-contamination and BD-contamination with respect to atomic broadcast.

We then consider the problem of inconsistency, and present a protocol that ensures VBD-consistent behavior with respect to atomic broadcast. This protocol has a latency of $\Delta + \lambda$. Based on this protocol, we develop another protocol that prevents VBD-inconsistency with respect to atomic broadcast, and has optimal latency of Δ.

Clearly, the above protocols also ensure D-consistent behavior with respect to atomic broadcast. However, they do not ensure BD-consistent behavior with respect to atomic broadcast. Indeed, we prove that BD-consistent behavior with respect to atomic broadcast cannot be enforced by a "non-blocking" protocol in a system subject to omission failures.

Since we concentrate on BD-contamination and on VBD-consistency, we use "contamination" to mean BD-contamination, and we use "consistency" to mean VBD-consistency in this chapter.

[2] Although the protocol has a latency of Δ, it does not prevent contamination with respect to *timed* atomic broadcast. The latency only serves as a measure of efficiency of the protocol.

7.2 Atomic broadcast

The simple atomic broadcast protocol shown in Figure 7.1 is well-known [Lam84].

The protocol in Figure 7.1 is informally described below, by "following" the broadcast of a message m by a process p. To atomically broadcast a message $m = (p, c, data)$ at time c, process p *broadcasts* $(basic, m)$. If by time $c + \Delta$, a process q *delivers* $(basic, m)$, then q atomically delivers m at time $c + \Delta$ (and at no other time). Furthermore, if q atomically delivers another message m' at time $c + \Delta$, then q atomically delivers m before m' if and only if m precedes m' in *lexicographic order*—*i.e.*, an order that corresponds to the timestamp of messages, with ties broken using process names and message contents. Such an ordering of messages is more commonly known as *timestamp order*.

The protocol also ensures the additional property that all processes which atomically deliver a message m do so at the same instant of local time; this is called the *simultaneity* property:

- *Simultaneity:* If processes p and q deliver message m, then they both deliver m at the same local time.

/* p atomically broadcasts a message $m = (p, c, data)$ at time c */
  To broadcast $(atomic, m = (p, c, data))$ p **broadcasts** $(basic, m)$

/* p delivers basic broadcast messages at c */
  $M_p^c \equiv \{m \mid ts(m) = c - \Delta,\ p$ **delivered** $(basic, m)\}$

/* p atomically delivers messages at c */
  $D \equiv$ **sort** M_p^c in timestamp order
  p **delivers** $(atomic, D)$

Figure 7.1: Atomic broadcast protocol
Executed by process p at time c

Formally, the protocol is described in pseudo-code in Figure 7.1 by giving the actions taken by any process p at any instant of clock time c. If p wishes to atomically broadcast a message m, p *broadcasts* $(basic, m)$. Let M_p^c denote the set of messages timestamped $c - \Delta$ that p has delivered using the basic broadcast. Process p sorts this set of messages in timestamp order (*i.e.*, lexicographic order), and atomically delivers this sorted sequence of messages.

The properties provided by the basic broadcast imply that the protocol guarantees the validity, agreement, integrity and total order properties, and hence the protocol is an atomic broadcast protocol. It is also straightforward to prove that the protocol satisfies the Δ-timeliness property (*i.e.*, has a latency of Δ), and guarantees message delivery in timestamp order.

Theorem 7.1 *The protocol in Figure 7.1 is an atomic broadcast protocol that tolerates omission failures, and has a latency of* Δ.

7.3 Atomic broadcast with no contamination

We now present an atomic broadcast protocol that ensures BD-contamination-freedom (Figure 7.2). We then describe an optimization to reduce the message complexity of the protocol.

To atomically broadcast $m = (p, c, data)$ at c, process p *broadcasts* $(basic, b)$, where b is the tuple $b = (p, c, (data, \overrightarrow{\text{DLVD}}^c_p))$, and $\overrightarrow{\text{DLVD}}^c_p$ is the sequence of messages that p has atomically delivered by c. If process q *delivers* $(basic, b)$ by time $c + \Delta$, then q atomically delivers $(p, c, data)$ at $c + \Delta$ if and only if $\overrightarrow{\text{DLVD}}^c_q = \overrightarrow{\text{DLVD}}^c_p$. In other words, q atomically delivers m only if p and q atomically deliver the same sequence of messages by time $ts(m)$.

We show that this protocol is an atomic broadcast protocol—*i.e.*, it satisfies the validity (Lemma 7.5), agreement, total order (Lemma 7.4), and integrity properties. Since the protocol also satisfies the fifo order (Lemma 7.8) property, and the delivery of message $m' = (q, c', —)$ by p is conditional upon whether $\overrightarrow{\text{DLVD}}^{c'}_q \preceq \overrightarrow{\text{DLVD}}_p$ (Lemma 7.7), Theorem 5.11 in Chapter 5 implies that the protocol prevents BD-contamination with respect to atomic broadcast.

Lemma 7.2 *If any process atomically delivers* m*, then it atomically delivers* m *at time* $ts(m) + \Delta$.

Lemma 7.3 *If a correct process atomically delivers* m*, then all correct processes atomically deliver* m.

Proof: Suppose for contradiction that the lemma is false. Let c be the earliest time such that some correct process q atomically delivers a message m at time $c + \Delta$, m was broadcast at time c, and another correct process r does not atomically deliver m. Thus, q and r atomically delivered the same set of messages by time c, and Lemma 7.2 implies that $\overrightarrow{\text{DLVD}}^c_r = \overrightarrow{\text{DLVD}}^c_q$.

/* p atomically broadcasts a message $m = (p, c, data)$ at time c */

To broadcast $(atomic, m = (p, c, data))$

p **broadcasts** $(basic, b = (p, c, (data, \overrightarrow{\text{DLVD}}_p^c)))$

/* p delivers basic broadcast messages at c */

$M_p^c \equiv \{b \mid ts(b) = c - \Delta,\ p\ \textbf{delivered}\ (basic, b)\}$

/* p atomically delivers messages at c */

$T \equiv \{(q, c-\Delta, data') \mid (q, c-\Delta, (data', \overrightarrow{\text{DLVD}}_q^{c-\Delta}) \in M_p^c,$
$\overrightarrow{\text{DLVD}}_p^{c-\Delta} = \overrightarrow{\text{DLVD}}_q^{c-\Delta}\}$

$D \equiv$ **sort** T in timestamp order

p **delivers** $(atomic, D)$

Figure 7.2: Atomic broadcast protocol that prevents BD-contamination
Executed by p at c

Suppose p is the broadcaster of m (*i.e.*, $p = bc(m)$). Since q atomically delivered m, q *delivered* $(basic, b)$, $b = (p, c, (m, \overrightarrow{\text{DLVD}}_p^c))$ by time $c + \Delta$, and $\overrightarrow{\text{DLVD}}_p^c = \overrightarrow{\text{DLVD}}_q^c$. Since q and r are correct, the properties of *broadcast* $(basic, b)$ and *deliver* $(basic, b)$ ensure that r also *delivered* $(basic, b)$ by time $c + \Delta$. Since r did not atomically deliver m, $\overrightarrow{\text{DLVD}}_p^c \neq \overrightarrow{\text{DLVD}}_r^c$. Thus, $\overrightarrow{\text{DLVD}}_r^c \neq \overrightarrow{\text{DLVD}}_q^c$, a contradiction. □

Lemma 7.4 *For all correct processes p and q, and all times c,* $\overrightarrow{\text{DLVD}}_p^c = \overrightarrow{\text{DLVD}}_q^c$.

Proof: From Lemmas 7.2 and 7.3. □

Lemma 7.5 *If the broadcaster of a message m is correct, then all correct processes atomically deliver m.*

Proof: It is clear from the protocol that if a correct process p atomically broadcasts m, then p atomically delivers m. The lemma follows from Lemma 7.3. □

Lemma 7.6 is a consequence of the fact that any process p can only atomically deliver m at time $ts(m) + \Delta$, and at no other time.

Lemma 7.6 *If p and q atomically deliver messages m and m', then p atomically delivers m before m' if and only if q atomically delivers m before m'.*

Lemma 7.7 *For all p, q and times c, if* $\overrightarrow{\text{DLVD}}_q^c = \overrightarrow{\text{DLVD}}_p^c$ *then* $\overrightarrow{\text{DLVD}}_q^c \preceq \overrightarrow{\text{DLVD}}_p$.

Lemma 7.8 *If p atomically broadcasts m before m', and any process q atomically delivers m', then q atomically delivers m before m'.*

Proof: The proof is by contradiction. Suppose for contradiction that p atomically broadcasts m before m' (*i.e.*, $ts(m) < ts(m')$), and q atomically delivers m', but does not atomically deliver m by the time it atomically delivers m'. Let c denote $ts(m)$ and c' denote $ts(m')$.

Since q atomically delivers m' at $c' + \Delta$, $\overrightarrow{\text{DLVD}}_p^{c'} = \overrightarrow{\text{DLVD}}_q^{c'}$. Since $c \leq c'$, Lemma 7.2 implies that $\overrightarrow{\text{DLVD}}_p^c = \overrightarrow{\text{DLVD}}_q^c$.

Since q atomically delivers m', q *delivers* $(basic, m')$ at or before time $c' + \Delta$. Hence, by the fifo order and Δ-timeliness properties satisfied by basic broadcast and delivery, q *delivers* $(basic, m)$ by time $c + \Delta$. Since q did not atomically deliver m, we conclude that $\overrightarrow{\text{DLVD}}_p^c \neq \overrightarrow{\text{DLVD}}_q^c$. This contradiction completes the proof of the lemma. □

Theorem 7.9 *The protocol in Figure 7.2 is an atomic broadcast protocol that prevents the contamination of correct processes, tolerates omission failures and has a latency of Δ.*

Proof: It is clear that the integrity property is satisfied. Hence Lemmas 7.4, 7.5, and 7.6 imply that the protocol is an atomic broadcast protocol. Lemma 7.8 implies that the fifo order property is satisfied. The protocol implies that if p atomically delivers $m = (q, c, —)$, then $\overrightarrow{\text{DLVD}}_q^c = \overrightarrow{\text{DLVD}}_p^c$, and hence (by Lemma 7.7) $\overrightarrow{\text{DLVD}}_q^c \preceq \overrightarrow{\text{DLVD}}_p$. Thus, Theorem 5.11 in Chapter 5 implies that the protocol prevents BD-contamination with respect to atomic broadcast.

The latency of Δ follows directly from the protocol. □

The simple protocol is expensive in terms of message size. However, the fifo order and simultaneity properties allow us to reduce the size of messages. We now describe an improved version of the above protocol in which the broadcaster p of a message m at time c uses an efficient encoding of $\overrightarrow{\text{DLVD}}_p^c$, as explained below.

For all processes p and all times c, let $Msgs_p^c$ be a vector that contains the following information about the number of messages atomically delivered by p: for all processes

q let $Msgs_p^c(q) = k$ if and only if, by time c, p has atomically delivered k messages broadcast by q.

Suppose for processes p, q and r and time c, $Msgs_p^c(q) = Msgs_r^c(q) = k$. The fifo order of messages delivery implies that at time c, both p and r have atomically delivered the first k messages that q atomically broadcast (and no other messages broadcast by q). Furthermore, the simultaneity property implies that they atomically delivered these messages at the same times. Thus, $Msgs_p^c = Msgs_r^c$ if and only if $\overrightarrow{\text{DLVD}}_p^c = \overrightarrow{\text{DLVD}}_r^c$. Hence, the vector $Msgs_p^c$ can replace the message sequence $\overrightarrow{\text{DLVD}}_p^c$ in the simple protocol (Figure 7.2).

If $c' < c$, then $Msgs_p^c$ equals $Msgs_p^{c'}$ plus an "increment" vector $Inc_p^{c,c'}$, where $Inc_p^{c,c'}(q)$ is the number of messages originated by q that p atomically delivered after time c' and by time c. With the improved protocol, to atomically broadcast m at time c, process p *broadcasts* $(basic, b)$, where b is the tuple $(p, c, (data, Inc_p^{c,c'}))$, and c' is the time of p's previous atomic broadcast. Thus, each message that p atomically delivers between c and c' can increase the size of $Inc_p^{c,c'}$ by at most one bit. This is in contrast to the *broadcast* $(basic, (p, c, (data, \overrightarrow{\text{DLVD}}_p^c)))$ required by the protocol in Figure 7.2, where the broadcast message includes the sequence of all the messages that p atomically delivered by time c.

7.4 Atomic broadcast with no inconsistency

In this section, we present broadcast protocols that implement atomic broadcast and ensure VBD-consistent behavior.

In order to prevent contamination, each process q need only perform a "local" check (*e.g.*, comparing $\overrightarrow{\text{DLVD}}_p^{ts(m)}$ with $\overrightarrow{\text{DLVD}}_q^{ts(m)}$) prior to atomically delivering a message m atomically broadcast by p. Such a local check is not sufficient to prevent inconsistency. This is illustrated by the following example.

Let p be a correct process, q be a faulty process, and c be a time such that $\overrightarrow{\text{DLVD}}_p^c = \overrightarrow{\text{DLVD}}_q^c$. Suppose p atomically broadcasts a message m at c, and subsequently atomically delivers some message m' before atomically delivering m. Suppose that q fails to atomically deliver m'. Although $\overrightarrow{\text{DLVD}}_p^c = \overrightarrow{\text{DLVD}}_q^c$, q cannot atomically deliver m; if it did, it would become inconsistent because it would not have atomically delivered m' before m.

Informally, to ensure consistency, a process p atomically delivers m only if it agrees with the correct processes on the sequence of messages that should be atomically delivered before m. Our protocol only works if no more than half the processes in the system can be faulty. We show that the requirement $n > 2f$ is necessary for any protocol that prevents inconsistency with respect to atomic broadcast.

Theorem 7.10 *Any atomic broadcast protocol that ensures* D*-consistency in a system subject to omission failures requires* $n > 2f$.

Proof: Suppose for contradiction that the theorem is false, and that there is an atomic broadcast protocol that ensures D-consistency in a system with f faulty processes and $n = 2f$ processes. Let A and B be any two sets of processes, such that $|A| = |B| = f$, and $A \bigcap B = \phi$. Let p be any process in set A.

Suppose p atomically broadcasts a message m at some time c. We consider two possible executions of the protocol. In both these executions, we assume that no message is received in less than λ time; thus either a message is received exactly λ time after it was sent, or it is never received. Furthermore, we assume that if a message is supposed to be sent by a process in A to another process in A, then the message is actually sent and is received. Similarly, if a message is supposed to be sent by a process in B to another process in B, then the message is actually sent and is received.

Scenario 1: Suppose the processes in A are correct, and the processes in B are faulty. Suppose that whenever a process a in A sends a message to a process b in B, b fails to receive the message. Suppose also that whenever a process b in B sends a message to a process a in A, b fails to send the message. In this scenario, it is clear that the processes in A must eventually atomically deliver m, and the processes in B never atomically deliver m.

Scenario 2: Suppose that the processes in A are faulty, and the processes in B are correct. Suppose that whenever a in A sends a message to b in B, a fails to send the message; whenever b in B sends a message to a in A, a fails to receive the message.

The two scenarios are indistinguishable to the processes in A and B. Hence, as in Scenario 1, the processes in A atomically deliver m and the processes in B never atomically deliver m. However, in this execution (unlike the execution described in Scenario 1), the processes in A are faulty. It is clear that by atomically delivering m (a message that the correct processes never deliver), a process a in A becomes D-inconsistent with respect to atomic broadcast and the failure set of A. Thus, this execution of the atomic broadcast protocol contradicts the assumption that the protocol ensures D-consistency. □

We now present a simple atomic broadcast protocol that prevents inconsistency and has a latency of $\Delta + \lambda$ (Figure 7.3). We then present a protocol that has a latency of Δ which is optimal (Figure 7.5).

7.4.1 Simple protocol: latency $\Delta + \lambda$

Informally, our simple atomic broadcast protocol that prevents inconsistency works as follows. To atomically broadcast $m = (p, c, —)$ at time c, p *broadcasts* $(basic, m)$ at time c.

If a process q *delivers* $(basic, m)$ by time $c + \Delta$, then q sends the message $(q, m, AllM_q^{c+\Delta})$ to all processes at time $c + \Delta$, where $AllM_q^{c+\Delta}$ denotes the set of all messages m', timestamped c or less, such that q *delivered* $(basic, m')$.

At time $c + \Delta + \lambda$, if a process q receives at least $n - f$ messages of the form $(r, m, AllM_r^{c+\Delta})$, with $AllM_r^{c+\Delta} = AllM_q^{c+\Delta}$, then q *delivers* $(atomic, m)$. In this scenario, all the correct processes agree with q that m should be atomically delivered, and also agree with q on the sequence of messages that should be atomically delivered before m.

However, if q does not receive at least $n - f$ messages of the form described above, then q halts. In this scenario, q is faulty and if it were to atomically deliver m, it would become inconsistent.

The protocol is formally described in Figure 7.3. We show that the protocol is an atomic broadcast protocol—*i.e.* it satisfies the validity, agreement, total order and integrity properties. We also show that the protocol respects correctness (*i.e.*, does not halt a correct process), and ensures VBD-consistency. To do the latter, we show that the protocol ensures both the uniform agreement and the fifo order properties, and that the messages atomically delivered by any process are always a prefix of the messages atomically delivered by the correct processes. Thus, Lemma 5.7 in Chapter 5 implies that the protocol prevents VBD-inconsistency.

Lemma 7.11 *If any process atomically delivers a message m, then it does so at time $ts(m) + \Delta$.*

Lemma 7.12 *If no correct process halts before time c, then for all correct processes q and r, $AllM_q^c = AllM_r^c$.*

Proof: From the properties of the basic broadcast, and the observation that if a correct process halts at time c, it does so after receiving messages and delivering basic broadcasts. □

Lemma 7.13 *Suppose m is atomically broadcast, and no correct process halts before time $ts(m) + \Delta + \lambda$. If any correct process delivers $(basic, m)$, then for every correct process q, $|R_{q,m}| \geq n - f$.*

/* p atomically broadcasts a message $m = (p, c, data)$ at time c */
To broadcast $(atomic, m = (p, c, data))$ p **broadcasts** $(basic, m)$

/* p delivers basic broadcast messages at c */
$M_p^c \equiv \{m \mid ts(m) = c - \Delta,\ p\ \textbf{delivered}\ (basic, m)\}$
$AllM_p^c \ \equiv\ \bigcup_{c' \leq c} M_p^{c'}$

/* p sends messages at c */
For all $m \in M_p^c$ **then** p **sends** $(p, m, AllM_p^c)$ to all

/* p receives all messages at c */
$R_{p,m} \equiv \{q \mid p\ \textbf{received}\ (q, m, AllM_q^{c-\lambda}),\ m \in M_p^{c-\lambda},\ AllM_p^{c-\lambda} = AllM_q^{c-\lambda}\}$

/* p checks whether it should halt */
If $\exists m \in M_p^{c-\lambda}$ such that $|R_{p,m}| < n - f$ **then** p **halts**

/* p atomically delivers messages at c */
$D \equiv$ **sort** $M_p^{c-\lambda}$ in timestamp order
p **delivers** $(atomic, D)$

Figure 7.3: Atomic broadcast protocol that prevents VBD-inconsistency
Executed by process p at time c

Proof: Suppose some correct process *delivers* $(basic, m)$. From the properties of the basic broadcast, all correct processes *deliver* $(basic, m)$ by time c, where c denotes the time $ts(m) + \Delta$.

Let q be any correct process and let A denote the set of messages $AllM_q^c$.

Since every correct process *delivers* $(basic, m)$ by time c, the protocol and Lemma 7.12 imply that every correct process sends the message $(m, —, A)$ to all processes at time c.

Since (by hypothesis) no correct process halts before time $c + \lambda$, any correct process that halts, does so after delivering basic broadcast messages at time $c + \lambda$. Since there are at least $n - f$ correct processes, q receives at least $n - f$ messages of the form $(m, —, A)$ by time $c + \lambda$, and hence $|R_{q,m}| \geq n - f$. $\square$

Lemma 7.14 *A correct process never halts.*

Proof: Suppose for contradiction that the lemma is false. Let c be the earliest time such that some correct process q halts at time $c + \lambda$.

The protocol in Figure 7.3 implies that there is a message m, with $ts(m) = c - \Delta$, such that q *delivers* (*basic*, m) by time c, and $|R_{q,m}| < n - f$.

Since q is correct and *delivers* (*basic*, m), Lemma 7.13 implies that $|R_{q,m}| \geq n - f$. This contradiction completes the proof. □

Lemma 7.15 *If a correct process atomically delivers message m, then all correct processes atomically deliver m.*

Proof: If a correct process atomically delivers m, then it *delivers* (*basic*, m); thus, all correct processes *deliver* (*basic*, m). From the protocol, if any process basic-delivers a message m, then it either atomically delivers m at time $ts(m) + \Delta + \lambda$, or it halts by time $ts(m) + \Delta + \lambda$, before atomically delivering any messages at time $ts(m) + \Delta + \lambda$. Lemma 7.14 states that no correct process halts. Thus, we conclude that all correct processes atomically deliver m at time $ts(m) + \Delta$. □

Lemma 7.16 *For all correct processes p and q, and all times c, $\overrightarrow{\text{DLVD}}_p^c = \overrightarrow{\text{DLVD}}_q^c$.*

Proof: Follows from Lemmas 7.15 and 7.14. □

Lemma 7.17 *If q delivers (basic, m), and q does not atomically deliver m, then q halts at or before $ts(m) + \Delta + \lambda$.*

Proof: Directly from the protocol. □

Lemma 7.18 *If a correct process atomically broadcasts message m, then all correct processes atomically deliver m.*

Proof: Suppose a correct process p atomically broadcasts a message m at time c. From the protocol, p *broadcasts* (*basic*, m) at time c. The properties of the basic broadcast imply that all correct processes *deliver* (*basic*, m).

Let q be any correct process. Since q is correct, and hence cannot halt (Lemma 7.14), Lemma 7.17 implies that q atomically delivers m. By Lemma 7.15 all correct processes atomically deliver m. □

Lemma 7.19 *If p atomically broadcasts m before m', and any process q atomically delivers m', then q atomically delivers m before m'.*

Proof: The proof is by contradiction. Suppose for contradiction that p atomically broadcasts m before m' (*i.e.*, $ts(m) < ts(m')$), and q atomically delivers m', but does not atomically deliver m by the time it atomically delivers m'. Let c denote $ts(m)+\Delta$, and c' denote $ts(m') + \Delta$. From the protocol, q atomically delivers m' at $c' + \lambda$.

Since p atomically broadcasts m before m', p *broadcasts* $(basic, m)$ before p *broadcasts* $(basic, m')$. Since q atomically delivers m', q *delivers* $(basic, m')$ at or before time c'. The fifo and Δ-timeliness properties satisfied by the basic broadcast imply that q *delivers* $(basic, m)$ by time c.

Since q did not atomically deliver m, Lemma 7.17 implies that q halts at or before $c+\lambda$. Hence q does not atomically deliver any messages at or after time $c+\lambda$ and, in particular, q does not atomically deliver m' at $c' + \lambda$. This contradiction completes the proof. □

Lemma 7.20 *If any process atomically delivers m, then all correct processes atomically deliver m.*

Proof: Suppose some process q atomically delivers m. From the protocol, $|R_{q,m}| \geq n - f$. Since $n > 2f$, there is at least one correct process r in $R_{q,m}$.

Since $r \in R_{q,m}$, r *delivered* $(basic, m)$. The agreement property of the basic broadcast implies that all correct processes also *deliver* $(basic, m)$. Since correct processes do not halt (Lemma 7.14), Lemma 7.17 implies that all correct processes atomically deliver m. □

Lemma 7.21 *For all q, for all c, for all correct r, $\overrightarrow{\text{DLVD}}_q^c \preceq \overrightarrow{\text{DLVD}}_r$.*

Proof: The proof is by contradiction. Suppose q is any process, r is a correct process, and c is any time. Since the protocol ensures the simultaneity property, $\overrightarrow{\text{DLVD}}_q^c \not\preceq \overrightarrow{\text{DLVD}}_r$ only if q skips a message; *i.e.*, r atomically delivers a message m before a message m', and q atomically delivers m', but does not atomically deliver m. Assume for contradiction that this is the case.

Let c denote $ts(m) + \Delta$ and c' denote $ts(m') + \Delta$. Since r atomically delivers m before m', $c \leq c'$.

Since $c \leq c'$ and q did not atomically deliver m and atomically delivered m', Lemma 7.17 implies that q does not *deliver* $(basic, m)$; hence $m \notin \mathit{AllM}_q^{c'}$.

Since r atomically delivers m before m', r *delivers* (*basic*, m), and hence $m \in AllM_r^{c'}$. Since correct processes do not halt, Lemma 7.12 implies that for all correct processes s, $m \in AllM_s^{c'}$. Thus at least $n - f$ processes do not agree with q's value of $AllM_q^{c'}$.

Since $|R_{q,m'}|$ only consists of (a subset of) the processes that agree with q's value of $AllM_q^{c'}$, and since $n > 2f$, $|R_{q,m'}|$ is less than $n - f$. Hence, q halts before atomically delivering m'. This is a contradiction. □

Theorem 7.22 *Suppose $n > 2f$. The protocol in Figure 7.3 is an atomic broadcast protocol that ensures* VBD*-inconsistent behavior, tolerates omission failures and has a latency of $\Delta + \lambda$.*

Proof: It is clear that the integrity property is satisfied. Lemmas 7.18 and 7.16 imply that the validity, agreement, and total order properties are satisfied. Lemma 7.14 shows that the protocol respects correctness. Lemmas 7.19 and 7.20 imply that the fifo order and uniform agreement properties are satisfied. Since Lemma 7.21 shows that for all q, for all c, for all correct processes r, $\overrightarrow{\mathrm{DLVD}}_q^c \preceq \overrightarrow{\mathrm{DLVD}}_r$, Lemma 5.6 in Chapter 5 implies that that no process becomes VBD-inconsistent. This completes the proof. □

7.4.2 Optimal protocol: latency Δ

The simple protocol described above consists of two parts. In the first part, a process p *broadcasts* (*basic*, m) to all processes. The second part of the protocol begins at time $ts(m) + \Delta$. At that time, every process q is supposed to send an "acknowledgement" which includes information about all the messages that q intends to atomically deliver before m. These acknowledgements are received by the other processes by time $ts(m) + \Delta + \lambda$. Based on the acknowledgements it receives, a process decides whether to atomically deliver m or to halt.

This section describes an optimization of the protocol given in Figure 7.3. The new protocol is optimal in that it permits a process to decide whether to atomically deliver a message m at time $ts(m) + \Delta$; that is, unlike the simple protocol which has a latency of $\Delta + \lambda$, the new protocol has an optimal latency of Δ.

As in the simple protocol, the optimal protocol also consists of two parts, taking Δ and λ time respectively. The difference is that in the broadcast of a message m, the second part of the optimal protocol begins at time $ts(m) + \Delta - \lambda$, not at time $ts(m) + \Delta$ (as in the simple protocol). Thus, the two parts of the protocol are "pipelined" and the entire protocol takes at most Δ time to terminate.

The optimal protocol assumes that f, the maximum number of faulty processes in the system, is greater than 1. If this is not the case, then there are other (simpler) atomic broadcast protocols that ensure consistency.

Suppose p wants to atomically broadcast $m = (p, c, data)$ at time c. The first part of the protocol, called a *witness broadcast*, is used to "disseminate" the message m to all processes. Furthermore, each process q determines a set of *witnesses* to the broadcast of m. Informally, a process r is a witness to the broadcast of m, if r "believes" that p correctly disseminated the message m. At the end of the first part of the protocol, all the correct processes agree on whether the cardinality of the witness set for m is less than $n - f$.

In the second part of the protocol, a process uses its witness sets to determine if it should continue with message delivery or if it should halt before becoming inconsistent. (Compare this with the *AllM* sets used in the simple protocol). The second part of the protocol is explained in more detail later on in the text.

Witness broadcast

Any process p may *witness-broadcast* message $m = (p, c, data)$ at any time c; this is denoted p *broadcasts* $(witness, m)$ at time c.

Informally, a witness broadcast is a basic broadcast, with the addition that every process q maintains a monotone *witness* set for each message m that it witness-delivers. Process q's witness set for a message m at time c is the set of processes that, according to q's knowledge at time c, certify that the broadcaster of the message m correctly broadcast m at time $ts(m)$.

Let $Wit^c_{p,m}$ denotes p's witness set for message m at time c, immediately before p's clock changes to $c + 1$. Informally, for any message m, at time $ts(m) + \Delta$, the correct processes agree on whether the cardinality of the witness set for m is at least $n - f$. Furthermore, at time $ts(m) + \Delta$, if *any* process (correct or faulty) determines that the cardinality of the witness set is at least $n - f$, then all correct processes witness-deliver message m by time $ts(m) + \Delta - \lambda$.

For simplicity of presentation, we assume the following. Suppose a process p crashes at time c. For all messages m, for all $c' \geq c$, p's witness set for m at time c' is identical to its witness set for m immediately before it crashed.

Informally, to witness-broadcast a message $m = (p, c, data)$ at time c, p sends (p, m) to all processes. Process p also immediately adds itself to $Wit^c_{p,m}$. When p first receives a message of the form (r, m), it adds r to its witness set for m; note that it does not relay (r, m).

/* p witness-broadcasts a message $m = (p, c, data)$ at time c */

To broadcast $(witness, m = (p, c, data))$

p **sends** (p, m) to all

$Wit^{c}_{p,m} = \{p\}$

/* p witness-delivers its own broadcast */

For all m : p witness-broadcast m at $c - 1$, p **delivers** $(witness, m)$

/* p receives all messages at c */

$A^{c}_{p} \equiv \{(q, m) \mid p$ **received** $(q, m)\}$

$R^{c}_{p} \equiv \{m \mid (-, m) \in A^{c}_{p}\}$

/* p updates its witness sets for messages it broadcast */

For all $(q, m) \in A^{c}_{p}$

if $c \leq ts(m) + \Delta$ **and** $p = bc(m)$ **then** $Wit^{c}_{p,m} = Wit^{c}_{p,m} \cup \{q\}$

/* p witness-delivers messages broadcast by other processes */

For all $m \in R^{c}_{p}$ **if** $c \leq ts(m) + \Delta$ **and** $Wit^{c-1}_{p,m}$ undefined **then**

/* Note that $p \neq bc(m)$ */

p **delivers** $(witness, m)$

$Wit^{c}_{p,m} = \{bc(m)\}$

if $c \leq ts(m) + \lambda$ **then**

$Wit^{c}_{p,m} = Wit^{c}_{p,m} \cup \{p\}$

p **sends** (p, m) to all

/* p relays messages for messages broadcast by other processes */

For all $(q, m) \in A^{c}_{p}$

if $c \leq ts(m) + \Delta$, $p \neq bc(m)$ **and** p has not sent (q, m) before **then**

$Wit^{c}_{p,m} = Wit^{c}_{p,m} \cup \{q\}$

p **sends** (q, m) to all

Figure 7.4: Witness broadcast protocol
Executed by process p at time c

When a process $q \neq p$ first receives a message of the form (r, m) by time $c+\Delta$, it adds p and r to its witness sets for m, and relays (r, m) to all processes. Furthermore, if $q \neq p$ first receives a message of the form (r, m) at some time $c' \leq c+\lambda$, then at time c', q adds itself to its own witness set, and sends (q, m) to all processes.

The protocol is more formally described in Figure 7.4, and satisfies the following lemmas. For simplicity, when the set $Wit^{c}_{p,m}$ is undefined, we say that $Wit^{c}_{p,m} = \phi$.

Lemmas 7.23, 7.24 and 7.25 follow directly from the protocol.

Lemma 7.23 *For all messages m, for all processes $p \neq bc(m)$, for all time c, p delivers $(witness, m)$ at time c if and only if the broadcaster of m is first added to p's witness set at time c; i.e., $bc(m) \in Wit^{c}_{p,m}$ and $bc(m) \notin Wit^{c-1}_{p,m}$.*

Lemma 7.24 *For all messages m, for all processes p, for all time c, the first member of p's witness set for m is the broadcaster of m; i.e., if $Wit^{c}_{p,m} \neq \phi$, then $bc(m) \in Wit^{c}_{p,m}$.*

Lemma 7.25 *For all processes p, for all messages m, for all time c, the set $Wit^{c}_{p,m}$ satisfies the following conditions:*

- *Suppose $p = bc(m)$. Process p joins its own witness set for m when it witness-broadcasts m; i.e., $p = bc(m)$ and p broadcasts $(witness, m)$ at c if and only if $p \in Wit^{c}_{p,m}$ and $p \notin Wit^{c-1}_{p,m}$.*
- *Suppose $p \neq bc(m)$. If p witness-delivers m at time $c \leq ts(m)+\lambda$, then p joins its witness set for m at time c; i.e., $p \neq bc(m)$ delivers $(witness, m)$ at $c \leq ts(m)+\lambda$ if and only if $p \in Wit^{c}_{p,m}$ and $p \notin Wit^{c-1}_{p,m}$.*

Lemma 7.26 *For all messages m, for all correct processes $p \neq bc(m)$, for all correct processes q, such that neither p nor q process halts by $ts(m)+\Delta$, for all time c, the following condition holds. If some process r is in p's witness set for m at time $c \leq ts(m)+\Delta-\lambda$, then r will be in q's witness set for m by time $c+\lambda$; i.e., for all r, if $c \leq ts(m)+\Delta-\lambda$ and $r \in Wit^{c}_{p,m}$, then $r \in Wit^{c+\lambda}_{q,m}$.*

Proof: Suppose p and q are correct processes. For some message m suppose that c is the smallest time by $ts(m)+\Delta-\lambda$, such that for some process r, $r \in Wit^{c}_{p,m}$; thus, $r \notin Wit^{c-1}_{p,m}$. The proof is a case analysis, and we will only prove one of the cases $(r = bc(m))$; the other case $(r \neq bc(m))$ is very similar to the one shown.

Case: $r = bc(m)$: In this case, p first received a message of the form $(—, m)$ at time c. Thus, p relays this message to all processes. Since p and q are correct, and the link delay is bounded by λ, q receives the relay by time $c+\lambda$. Since $c+\lambda \leq ts(m)+\Delta$, we conclude that $r \in Wit^{c+\lambda}_{q,m}$.

□

Lemma 7.27 below is proved by a "pigeonhole" argument. This technique will be used in the proof of several results in this chapter and in Chapter 8.

Lemma 7.27 *If a process p witness-broadcast a message m, and some process r first receives (p,m) at time $c > ts(m)+i\lambda$ for some integer i, then at least $i+1$ processes sent the message (p,m) by time $ts(m)+i\lambda$.*

Proof: The proof is a "pigeonhole" argument, and is by induction.

For all processes q, if q receives m, let c_q denote the time when q first receives the message m; otherwise let $c_q = \infty$.

Basis: Suppose $i = 0$. The claim asserts that at least 1 process has sent (p,m) by time $ts(m)$. Since p witness-broadcast m at time $ts(m)$, and r received (p,m), we conclude from the protocol that p sent (p,m) at time $ts(m)$.

Induction Hypothesis: Suppose the claim is true for $i = i^*$, for some $i^* \geq 0$.

Induction: Suppose $i = i^* + 1$. Since $c > ts(m) + i\lambda$, process r first receives (p,m) after time $ts(m)+i\lambda$. Let q be the first process that receives m after time $ts(m)+i\lambda$; *i.e.*, $c_q > ts(m) + i\lambda$, and for all processes s, either $c_s \leq ts(m) + i\lambda$ or $c_s \geq c_q$.

Suppose q first receives (p,m) from some process s. From the protocol, when s first receives (p,m) (*i.e.*, at time c_s), it sends (p,m) to all processes, and s sends (p,m) at no other time.

Since message receipt does not take 0 time, $c_s < c_q$, and thus $c_s \leq ts(m) + i\lambda$. Message link delay is bounded by λ, and hence $c_s + \lambda \geq c_q$. Since $c_q > ts(m) + i\lambda$, $c_s > ts(m) + (i-1)\lambda$.

Thus, by the induction hypothesis, at least i processes sent the message (p,m) by time $ts(m) + (i-1)\lambda$. Since s sent (p,m) after time $ts(m) + (i-1)\lambda$ (and by time $ts(m) + i\lambda$), and s does not send (p,m) more than once, we conclude that at least $i+1$ processes, not including r have sent (p,m) by time $ts(m) + i\lambda$. □

Lemma 7.28 *Suppose some process r first enters a process p's witness set for a message m at some time $c > ts(m) + k\lambda$, for some integer k. Then, at least k processes, not including the broadcaster of m, have r in their witness sets by time $ts(m) + k\lambda$.*

That is, if $r \in Wit^{c}_{p,m}$ and $r \notin Wit^{c-1}_{p,m}$ and $c > ts(m) + k\lambda$, then $|\{q \mid q \neq bc(m), r \in Wit^{ts(m)+k\lambda}_{q,m}\}| \geq k$.

Proof: Suppose some process r first enters a process p's witness set for a message m at some time $c > ts(m) + k\lambda$, for some integer k. The proof is a case analysis. If $r = bc(m)$, then the lemma follows from Lemma 7.27. If $r \neq bc(m)$, then the proof is by a "pigeonhole" argument, similar to the one in the proof of Lemma 7.27. □

Lemma 7.29 *Suppose a process p sends a point-to-point message to another process q at some time c, and that q receives this message. For all messages m' that p did not witness-broadcast, for all $c' \leq c$, $Wit_{p,m'}^{c'} \subseteq Wit_{q,m'}^{c'+\lambda}$.*

Proof: Suppose p, q and c are as in the statement of the lemma. Let m' be any message that p did not witness-broadcast. Let c' be the smallest time such that for some process r, $r \in Wit_{p,m'}^{c'}$. The proof is complete if we show that $r \in Wit_{q,m'}^{c'+\lambda}$.

If r is the broadcaster of m', then p sends a message of the form $(—, m')$ to all processes at time c'. If r did not broadcast m', then p sends a message of the form (r, m') at time c'. In either case, since $c' \leq c$, and point-to-point message passing is assumed to satisfy the fifo-link property, q receives the message p sent at time c' by the time it receives the message sent at time c. Since link delay is bounded by λ, we conclude that $r \in Wit_{q,m'}^{c'+\lambda}$. □

Lemma 7.30 *If a correct process p witness-broadcasts m at time c, and no correct process halts before time $c + 2\lambda$, then for all correct processes q, $|Wit_{q,m}^{c+2\lambda}| \geq n - f$.*

Proof: Suppose a correct process p witness-broadcasts m at time c, and no correct process halts before time $c + 2\lambda$.

Let q be any correct process. From Lemma 7.23 and the monotonicity of the witness sets, $p \in Wit_{p,m}^{c+\lambda}$ and $q \in Wit_{p,m}^{c+\lambda}$. Since q does not halt before time $c + 2\lambda$, Lemma 7.26 implies that for all correct processes r, $r \in Wit_{q,m}^{c+2\lambda}$. Since there are $n - f$ correct processes, $|Wit_{q,m}^{c+2\lambda}| \geq n - f$. □

Lemma 7.31 *Suppose that no correct process halts before time $ts(m) + \Delta$. If p and q are correct, then $|Wit_{p,m}^{ts(m)+\Delta}| \geq n - f$ if and only if $|Wit_{q,m}^{ts(m)+\Delta}| \geq n - f$.*

Proof: The proof is a case analysis, where c denotes $ts(m)$.

Case: $bc(m)$ is correct: From Lemma 7.30, since the broadcaster of message m is correct, for all correct processes p, $|Wit_{p,m}^{c+2\lambda}| \geq n - f$. Since $\Delta = (f+1)\lambda$, and $f > 1$ (by assumption), $\Delta \geq 3\lambda$. Thus, the lemma follows because no correct process halts before time $ts(m) + \Delta$, and the witness sets are monotone.

Case: $bc(m)$ is faulty: Let p be any correct process such that some process $s \in Wit^{c+\Delta}_{p,m}$. Let q be any correct process. The lemma follows if we show that $s \in Wit^{c+\Delta}_{q,m}$.

If $s \in Wit^{c+\Delta-\lambda}_{p,m}$, then from Lemma 7.26, $s \in Wit^{c+\Delta}_{q,m}$. Hence, suppose $s \notin Wit^{c+\Delta-\lambda}_{p,m}$.

Since $s \in Wit^{c+\Delta}_{p,m}$, and $\Delta = (f+1)\lambda$, Lemma 7.28 shows that $|\{r \mid r \neq bc(m), s \in Wit^{c+\Delta-\lambda}_{r,m}\}| \geq f$. Since $bc(m)$ is faulty, there is at least one correct process r, such that $s \in Wit^{c+\Delta-\lambda}_{r,m}$. By Lemma 7.26, $s \in Wit^{c+\Delta}_{q,m}$. This completes the proof. □

Lemma 7.32 *If $p \neq bc(m)$ is correct, $c \leq \Delta - \lambda$, and no correct process halts before time $c + \lambda$, then for all correct processes q, $Wit^{c}_{p,m} \subseteq Wit^{c+\lambda}_{q,m}$.*

Proof: Follows from Lemma 7.26. □

Lemma 7.33 *Suppose a faulty process witness-broadcasts m. Let s be such that for all correct processes p, $s \notin Wit^{ts(m)+\Delta-2\lambda}_{p,m}$. Let q be any faulty process, $q \neq bc(m)$. If either $s \in Wit^{ts(m)+\Delta}_{q,m}$, or $s \in Wit^{ts(m)+\Delta}_{p,m}$ for some correct p, then $s \in Wit^{ts(m)+\Delta-2\lambda}_{q,m}$.*

Proof: Let s be such that for all correct processes p, $s \notin Wit^{c+\Delta-2\lambda}_{p,m}$, where $c = ts(m)$. Let q be any faulty process, $q \neq bc(m)$. The proof is a case analysis.

Case: $s \in Wit^{c+\Delta}_{q,m}$: Let c^* denote the smallest time such that $s \in Wit^{c^*}_{q,m}$. Suppose for contradiction that $c^* > \Delta - 2\lambda$.

Since $s \notin Wit^{c+\Delta-2\lambda}_{q,m}$ and $\Delta = (f+1)\lambda$, Lemma 7.28 implies that the set $X \equiv \{r \mid r \neq bc(m), s \in Wit^{c+\Delta-2\lambda}_{r,m}\}$ is such that $|X| \geq f - 1$. Since (by hypothesis) for all correct processes p, $s \notin Wit^{c+\Delta-2\lambda}_{p,m}$, $p \notin X$. Thus, only faulty processes are in the set X. Since $bc(m)$ is faulty, there are only $f-1$ faulty processes not including $bc(m)$. Since $|X| \geq f-1$, we conclude that for all faulty processes $r \neq bc(m)$, $s \in Wit^{c+\Delta-2\lambda}_{r,m}$; in particular $s \in Wit^{c+\Delta-2\lambda}_{q,m}$. This is a contradiction.

Case: $s \in Wit^{c+\Delta}_{p,m}$ for some correct p: Since $s \notin Wit^{c+\Delta-2\lambda}_{p,m}$ and $\Delta = (f+1)\lambda$, Lemma 7.28 implies that $|\{r \mid r \neq bc(m), s \in Wit^{c+\Delta-2\lambda}_{r,m}\}| \geq f-1$. Since $bc(m)$ is faulty, there are only $f-1$ faulty processes not including $bc(m)$; thus, for all faulty processes $r \neq bc(m)$, $s \in Wit^{c+\Delta-2\lambda}_{r,m}$. Since q is a faulty process, $s \in Wit^{c+\Delta-2\lambda}_{q,m}$. □

Lemma 7.34 *If for some p, $|Wit^{ts(m)+\Delta}_{p,m}| \geq n - f$, and no correct process halts by time $ts(m) + 2\lambda$, then for all correct processes q, q delivers $(witness, m)$ by time $ts(m) + 2\lambda \leq ts(m) + \Delta - \lambda$.*

Proof: If $bc(m)$ is correct, then the lemma is trivially true. Suppose $bc(m)$ is faulty. Lemma 7.25 implies that at least one correct process *delivers* $(witness, m)$ by time $ts(m) + \lambda$, and Lemma 7.26 implies that all correct processes witness-deliver m by time $ts(m) + 2\lambda$.

Since $f > 1$ and $\Delta = (f+1)\lambda$, $2\lambda \leq \Delta - \lambda$. Thus, all correct processes *deliver* $(witness, m)$ by time $ts(m) + \Delta - \lambda$. □

Atomic broadcast protocol to prevent inconsistency

Informally, to broadcast m at time c, p *broadcasts* $(witness, m)$ at time c.

At time $c + 2\lambda$, suppose p believes that there are fewer than $n - f$ witnesses for m; *i.e.*, $|Wit_{p,m}^{c+2\lambda}| < n - f$. In this scenario, p failed to properly broadcast the message m to at least one correct process, and hence to avoid becoming inconsistent, p halts.

At time $c + \Delta - \lambda$, if a process q has already *delivered* $(witness, m)$, then q sends the message $(m, q, Wit_{p,m}^{c+\Delta-\lambda})$ to all processes.

At time $c + \Delta$, if a process q has already *delivered* $(witness, m)$, then:

- $R_{q,m}^{c+\Delta}$ denotes the set of processes that appear correct to q. That is, $r \in R_{q,m}^{c+\Delta}$ if q received r's message $(m, r, Wit_{r,m}^{c+\Delta-\lambda})$, and r has correctly relayed its witness sets to q.
- If $r \in R_{q,m}^{c+\Delta}$, and if r believes that there are fewer than $n - f$ witnesses to the broadcast of m, then $r \in A_{q,m}^{c+\Delta}$; otherwise $r \in B_{q,m}^{c+\Delta}$. Note that if $r \in B_{q,m}^{c+\Delta}$, then, intuitively, q "knows" that r "believes" that p correctly broadcast m.

After computing the R, A and B sets, q uses these sets to determine whether to halt:

- If $q = p$ and $|B_{q,m}^{c+\Delta}| < n - f$ then q halts. In this scenario, at least one correct process believes that q failed to broadcast m successfully. Thus, q is indeed faulty and, to avoid becoming inconsistent, q halts.
- If $p \in R_{q,m}^{c+\Delta}$, $|Wit_{q,m}^{c+\Delta}| < n - f$ and $|A_{q,m}^{c+\Delta}| < n - f$ then q halts. Since $p \in R_{q,m}^{c+\Delta}$, p did not halt at time $c + 2\lambda$, and hence there are at least $n - f$ witnesses to the broadcast of m. However, q does not believe that there are $n - f$ such witnesses. Since $|A_{q,m}^{c+\Delta}| < n - f$, fewer than $n - f$ processes agree with q on this (*i.e.*, fewer than $n - f$ processes agree with q that there are fewer than $n - f$ witnesses). Hence q erroneously believes that p is faulty, and to avoid becoming inconsistent, q halts.
- If $|Wit_{q,m}^{c+\Delta}| \geq n - f$ and $|R_{q,m}^{c+\Delta}| < n - f$ then q halts. Process q determines that there are at least $n - f$ witnesses to the broadcast of m, but also determines that it failed to correctly relay this information to the other processes. Thus, q is faulty and halts.

/* p atomically broadcasts a message $m = (p, c, data)$ at time c */

To **broadcast** $(atomic, m = (p, c, data))$ p **broadcasts** $(witness, m)$

/* p delivers basic broadcast messages at c */

$M_p^c \equiv \{m \mid c = ts(m) + \Delta,\ p\ \textbf{delivered}\ (witness, m)\}$

$L_p^c \equiv \{m \mid c = ts(m) + \Delta - \lambda,\ p\ \textbf{delivered}\ (witness, m)\}$

/* p sends messages at c */

$E \equiv$ **sort** L_p^c in timestamp order

for each m in E in order **send** $(m, p, Wit_{p,m}^c)$ to all

/* p receives all messages at c */

$R_{p,m}^c \equiv \{q \mid p\ \textbf{received}\ (m, q, Wit_{q,m}^{c-\lambda}),\ ts(m) = c - \Delta$ and
(either $p = bc(m)$ or $Wit_{p,m}^{c-2\lambda} \subseteq Wit_{q,m}^{c-\lambda})\}$

$A_{p,m}^c \equiv \{q \mid q \in R_{p,m}^c$ and $|Wit_{q,m}^{c-\lambda}| < n - f\}$

$B_{p,m}^c \equiv R_{p,m}^c - A_{p,m}^c$

Figure 7.5: Atomic broadcast protocol that prevents VBD-inconsistency
Executed by process p at time c

Finally, if q does not halt, then q atomically delivers m at time $c + \Delta$ if and only if:

- q determines there are at least $n - f$ witnesses to the broadcast of m, and
- q has atomically delivered all the messages that p atomically broadcast before it broadcast m.

Figure 7.5 formally describes our atomic broadcast protocol that prevents inconsistency and has optimal latency.

Lemma 7.35 *Suppose m is an atomically broadcast message and let c be $ts(m) + \Delta$. Suppose no correct process halts before time c. If $|Wit_{p,m}^c| \geq n - f$ for some process p, then for all correct processes q, q sends $(m, q, Wit_{q,m}^{c-\lambda})$ to all at time $c - \lambda$.*

Proof: Suppose $|Wit_{p,m}^c| \geq n - f$ for some process p. Let q be any correct process. Lemma 7.34 states that q *delivers* $(witness, m)$ by time $c - \lambda$. Hence (from the protocol) q sends $(m, q, Wit_{q,m}^{c-\lambda})$. □

/* p checks whether it should halt */

If $\exists m,\ c = ts(m) + 2\lambda,\ p = bc(m)$ and $|\,Wit^c_{p,m}| < n - f$ **then halt**

If $\exists m \in M^c_p,\ p = bc(m)$ and $|B^c_{p,m}| < n - f$ **then halt**

If $\exists m \in M^c_p, bc(m) \in R^c_{p,m},\ |\,Wit^c_{p,m}| < n - f$ and $|A^c_{p,m}| < n - f$ **then halt**

If $\exists m \in M^c_p, |\,Wit^c_{p,m}| \geq n - f$ and $|R^c_{p,m}| < n - f$ **then halt**

/* p atomically delivers messages at c */

$X \equiv \{m \mid m \in M^{c-\lambda}_p, |\,Wit^c_{p,m}| \geq n - f,$
$bc(m)$ broadcast m' before m, and p **delivered** $(atomic, m')\}$.

$D \equiv$ **sort** X in timestamp order

p **delivers** $(atomic, D)$

Figure 7.5: Atomic broadcast protocol that prevents VBD-inconsistency (continued) Executed by process p at time c

Lemma 7.36 *A correct process never halts.*

Proof: Suppose for contradiction that the lemma is false. Let c be the earliest clock time such that some correct process p halts at clock time c.

Case: $\exists m,\ ts(m) = c - 2\lambda,\ p = bc(m)$ and $|\,Wit^c_{p,m}| < n - f$: Since p is correct, and *broadcasts* $(witness, m)$ at time $c - 2\lambda$, and no correct process halts before time c, Lemma 7.30 states that $|\,Wit^c_{p,m}| \geq n - f$. This is a contradiction.

Case: $\exists m \in M^c_p,\ p = bc(m)$ and $|B^c_{p,m}| < n - f$: Let q be any correct process. Since p is correct, Lemma 7.30 implies that $|\,Wit^{c-\lambda}_{q,m}| \geq n - f$. Lemma 7.35 shows that q sends $(m, q,\ Wit^{c-\lambda}_{q,m})$ to all processes at time $c - \lambda$. Since both p and q are correct, $q \in B^c_{p,m}$.

Since there are at least $n - f$ correct processes, $|B^c_{p,m}| \geq n - f$, a contradiction.

Case: $\exists m \in M^c_p, bc(m) \in R^c_{p,m},\ |\,Wit^c_{p,m}| < n - f$ and $|A^c_{p,m}| < n - f$: Let $q = bc(m)$. Since q sends $(m, q,\ Wit^{c-\lambda}_{q,m})$, and since $\Delta - \lambda \geq 2\lambda$, q did not halt at time $ts(m) + 2\lambda$, and hence $|\,Wit^{ts(m)+2\lambda}_{q,m}| \geq n - f$. Since the witness sets are monotone, $|\,Wit^c_{q,m}| \geq n - f$.

Let r be any correct process. Since no correct process halts before time c, Lemmas 7.34 and 7.35 imply that r *delivers* (*witness*, m) and sends $(m, r, Wit^{c-\lambda}_{r,m})$ to all processes by time $c - \lambda$. Since $|Wit^{c}_{p,m}| < n - f$ and p is correct, Lemma 7.31 states that for all correct processes r, $|Wit^{c}_{r,m}| < n - f$.

Since both p and r are correct, process p receives $(m, r, Wit^{c-\lambda}_{r,m})$ (from r) by time c, with $|Wit^{c-\lambda}_{r,m}| < n - f$. Furthermore, Lemma 7.32 implies that $r \in R^{c}_{p,m}$, and also $r \in A^{c}_{p,m}$.

Since there are at least $n - f$ correct processes, $|A^{c}_{p,m}| \geq n - f$, a contradiction.

Case: $\exists m \in M^{c}_{p}, |Wit^{c}_{p,m}| \geq n - f$ and $|R^{c}_{p,m}| < n - f$: Since $|Wit^{c}_{p,m}| \geq n - f$, from Lemma 7.35, each correct process r sends $(m, r, Wit^{c-\lambda}_{r,m})$. By Lemmas 7.32 and 7.26, $|R^{c}_{p,m}| \geq n - f$. This is a contradiction.

□

Lemma 7.37 *If a correct process atomically broadcasts m then, all correct processes eventually atomically deliver m.*

Proof: Suppose for contradiction that the lemma is false. Let m be the first message in timestamp order that is atomically broadcast by a correct process, and not atomically delivered by a correct process p.

By choice of m, p atomically delivers all messages m' that $bc(m)$ had atomically broadcast before m. Since $bc(m)$ is correct, $|Wit^{ts(m)+\Delta}_{p,m}| \geq n - f$. Since no correct process halts, p *delivers* (*atomic*, m) at time $ts(m) + \Delta$. This is a contradiction. □

Lemma 7.38 *If a faulty process atomically broadcasts m, and some process p atomically delivers m, then all correct processes atomically deliver m.*

Proof: Suppose for contradiction that the lemma is false. Let m be the first message in timestamp order that is atomically broadcast by a faulty process, such that some process p *delivers* (*atomic*, m), and a correct process q does not *deliver* (*atomic*, m). Let c denote $ts(m) + \Delta$.

Since p *delivers* (*atomic*, m), the protocol shows that p atomically delivers all the messages m' that $bc(m)$ broadcast before m. Thus, by choice of m, all messages m' that $bc(m)$ broadcast before m were atomically delivered by both p and q.

Since p atomically delivers m, $|Wit^{c}_{p,m}| \geq n - f$. Lemma 7.36 states that no correct process halts. Lemmas 7.34 implies that all correct processes witness-deliver m by time $\Delta - \lambda$. Since q is correct, $m \in M^{c-\lambda}_{q}$. Since q does not atomically deliver m, $|Wit^{c}_{q,m}| < n - f$.

The proof continues as a case analysis: *Case:* p is correct: If p and q are correct and $|Wit^c_{p,m}| \geq n-f$, Lemma 7.31 implies that $|Wit^c_{q,m}| \geq n-f$. This is a contradiction.

Case: p is faulty and $p = bc(m)$: Let r be any correct process. Since $|Wit^c_{q,m}| < n-f$, from Lemma 7.26, $|Wit^{c-\lambda}_{r,m}| < n-f$. Hence, r cannot be included in $B^c_{p,m}$.

Since there are at least $n-f$ correct processes, at least $n-f$ processes cannot be included in $B^c_{p,m}$. Since $n > 2f$, $|B^c_{p,m}| < n-f$.

Since $p = bc(m)$ and $|B^c_{p,m}| < n-f$, p halts before atomically delivering m, a contradiction.

Case: p is faulty and $p \neq bc(m)$: We claim below that $Wit^c_{p,m} \subseteq Wit^c_{q,m}$. If the claim is true, since $|Wit^c_{p,m}| \geq n-f$, $|Wit^c_{q,m}| \geq n-f$. This contradiction then completes the proof of the lemma.

Claim: $Wit^c_{p,m} \subseteq Wit^c_{q,m}$.

Proof of claim: Suppose for contradiction that the claim is false. Let s be a process, such that $s \in Wit^c_{p,m}$, and $s \notin Wit^c_{q,m}$. Thus, for all correct processes r, $s \notin Wit^{c-\lambda}_{r,m}$.

Since p and $bc(m)$ are faulty, and $p \neq bc(m)$, from the above and from Lemma 7.33, $s \in Wit^{c-2\lambda}_{p,m}$.

Let r be any correct process. Since $p \neq bc(m)$, $s \in Wit^{c-2\lambda}_{p,m}$ and $s \notin Wit^{c-\lambda}_{r,m}$, r cannot be included in p's $R^c_{p,m}$ set.

Thus, no correct process can be included in $R^c_{p,m}$, and hence $|R^c_{p,m}| < n-f$. Thus, p halts before atomically delivering m, a contradiction. □

Lemma 7.39 *If p atomically broadcasts m before m', and q atomically delivers m', then q atomically delivers m before m'.*

Proof: Directly from the protocol. □

Lemma 7.40 *If p atomically broadcasts m, then p either atomically delivers m or halts at time $\leq ts(m) + \Delta$.*

Proof: Directly from the protocol. □

Lemma 7.41 *For all processes p, for all messages m, if p atomically delivers m, then by $ts(m) + \Delta$, p receives a message sent by some correct processes at time $ts(m) + \Delta - \lambda$.*

Proof: Suppose a process p atomically delivers a message m. Let c denote $ts(m) + \Delta$. Thus, from the protocol $|\,Wit^{c}_{p,m}| \geq n - f$ and $|R^{c}_{p,m}| \geq n - f$. Since $n > 2f$, there is some correct process $q \in R^{c}_{p,m}$. Thus, from the protocol, by time c process p receives the message $(m, q, Wit^{c-\lambda}_{q,m})$ sent by q at time $c - \lambda$. □

Lemma 7.42 *For all processes p, for all correct processes q, for all times c, $\overrightarrow{\text{DLVD}}^{c}_{p} \preceq \overrightarrow{\text{DLVD}}_{q}$.*

Proof: The proof is by contradiction. Consider any process p. Let m be the earliest message (first message in timestamp order) such that p does not atomically deliver m, and some correct process q atomically delivers m. Suppose for contradiction that q atomically delivers m before some message m' and that p atomically delivers m'. Let c denote $ts(m) + \Delta$ and c' denote $ts(m') + \Delta$. Since q atomically delivers m before m', $c \leq c'$.

From the protocol and the choice of m, all messages broadcast by $bc(m)$ before m are atomically delivered by both q and p.

Since p does not atomically deliver m, and atomically delivers m' at $c' \geq c$, Lemma 7.40 implies that $p \neq bc(m)$.

Since p atomically delivered m', by Lemma 7.41, p received a message that some correct process r sent at time $c' - \lambda$. Since $c \leq c'$, Lemma 7.29 implies that $Wit^{c-\lambda}_{r,m} \subseteq Wit^{c}_{p,m}$.

We claim below that $|\,Wit^{c}_{p,m}| \geq n - f$. Suppose the claim is true. Then, from the protocol, either p halts and does not atomically deliver any messages at time c or later, or p atomically delivers m. Since p does not atomically deliver m, and $c \leq c'$, p halts before atomically delivering m'. This contradiction completes the proof.

Claim: $|\,Wit^{c}_{p,m}| \geq n - f$.

Proof of claim: We showed above that $Wit^{c-\lambda}_{r,m} \subseteq Wit^{c}_{p,m}$. If $|\,Wit^{c-\lambda}_{r,m}| \geq n - f$, then $|\,Wit^{c}_{p,m}| \geq n - f$.

Hence, suppose that $|\,Wit^{c-\lambda}_{r,m}| < n - f$. Since r is correct, Lemma 7.30 implies that $bc(m)$ is faulty.

Consider any process s, $s \in Wit^{c}_{r,m}$, $s \notin Wit^{c-\lambda}_{r,m}$. Lemma 7.26 implies that for all correct processes u, $s \notin Wit^{c-2\lambda}_{u,m}$. Since p is faulty, $bc(m)$ is faulty, and $p \neq bc(m)$, Lemma 7.33 implies that $s \in Wit^{c-2\lambda}_{p,m}$. Thus, $(\,Wit^{c}_{r,m} - Wit^{c-\lambda}_{r,m}) \subseteq Wit^{c-2\lambda}_{p,m}$.

From the above and $Wit^{c-\lambda}_{r,m} \subseteq Wit^{c}_{p,m}$, we conclude that $Wit^{c}_{r,m} \subseteq Wit^{c}_{p,m}$. Since r atomically delivers m, $|\,Wit^{c}_{r,m}| \geq n - f$, and hence $|\,Wit^{c}_{p,m}| \geq n - f$. □

Theorem 7.43 *Suppose $n > 2f$, and $f > 1$. The protocol in Figure 7.5 is an atomic broadcast protocol that ensures* VBD*-inconsistent behavior, tolerates omission failures and has a latency of Δ.*

7.4.3 Ensuring BD-consistency

Finally, we show that no atomic broadcast protocol can ensure BD-consistency in a system subject to omission failures.

Recall that in Chapter 6, we defined a broadcast protocol to be "non-blocking." Informally, we assumed that a process is not permitted to initiate a broadcast, and "suspend" its application protocol until the broadcast terminates. The impossibility result below is only valid for such "non-blocking" protocols.

Theorem 7.44 *No atomic broadcast protocol can ensure* BD*-consistency in a system subject to omission failures.*

Proof: Suppose for contradiction Θ is an atomic broadcast protocol that ensures BD-consistency in a system S that is subject to omission failures.

We consider three possible executions of the protocol. In all these executions, we assume that no message is received in less than λ time; thus either a message is received exactly λ time after it was sent, or it is never received.

Scenario 1: Consider an execution of the protocol in S, in which a process p attempts to broadcast a message m at time c. Suppose that no other messages are broadcast.

Suppose further that all processes are correct. Thus, p correctly broadcasts m, and changes state at time c. Furthermore, by the properties of atomic broadcast, all processes eventually atomically deliver m.

Scenario 2: Consider a second execution of the protocol in S, in which p attempts to broadcast m at time c. Suppose that no other messages are broadcast.

Suppose further that p is faulty, that all processes other than p are correct and that p fails to send any messages to any process at time c. Since Scenarios 1 and 2 are indistinguishable to p until time c, p changes state at time c (as it does in Scenario 1).

Now suppose that p halts at time $c + 1$. Since p atomically broadcasts m at time c, and p changes state at time c, and Θ ensures BD-consistency with respect to atomic broadcast, p is BD-correct at time c. Therefore, all processes other than p eventually atomically deliver the message m. Intuitively, this is impossible because they cannot "guess" the contents of the message m. This is formalized below.

Scenario 3: Consider a third execution of the protocol in S, in which p atomically broadcasts a message $m' \neq m$ at time c. Suppose that no other messages are broadcast. Suppose further that p is faulty and all processes other than p are correct. Suppose that p fails to send any messages to any process at time c, and then halts at time $c+1$.

Since Scenarios 2 and 3 are indistinguishable to any process $q \neq p$, q must eventually atomically deliver m (as in Scenario 2). However, no process broadcasts m in this scenario, and hence the execution violates the specifications of atomic broadcast. This contradicts the assumption that Θ is an atomic broadcast protocol, and completes the proof. $\square$

Chapter 8

Atomic Multicast

In this chapter, we concentrate on *atomic multicasts*; these are atomic broadcasts where the intended recipients of any broadcast message are a subset of the processes in the system, called a *group* of processes.

We begin by defining a natural hierarchy of three types of atomic multicast. Informally, all three of these atomic multicasts require that messages targeted to a particular group are delivered in the same order by processes belonging to that group. However, the multicasts differ in the requirements placed on the relative order of delivery of messages multicast to different groups.

We concentrate on *pairwise atomic multicast*, which requires that if two groups g and g' overlap, then all messages that are multicast either to group g or to group g' are delivered in the same order by the processes that belong to both groups g and g'. We define inconsistency and contamination with such a multicast. We also describe pairwise atomic multicast protocols that prevent contamination and/or ensure consistency in systems that are subject to omission failures.

8.1 Preliminaries

In some applications, the system is configured as a collection of (possibly overlapping) groups, each consisting of a subset of processes. A *multicast* is a broadcast that is targeted exclusively to the members of some particular group. We assume that groups are static, and that each process knows the groups in the system, the groups to which it belongs and the members of each of the groups. Let $belongs(p)$ denote the set of all the groups to which a process p belongs.

Formally, a *group* g is a *name* of a subset of the processes in the system. When there is no ambiguity, we identify the name of a group with the set of processes that belong to that group. Thus, we say p *is in* g ($p \in g$), when process p belongs to the group named g.

Since each message is multicast to a particular group, we assume a multicast message is a tuple of the form $m = (p, c, g, data)$, where p is the multicaster of the message, c is the timestamp of the message, g is the group to which m is targeted, and $data$ is the information that p wishes to multicast; we write $group(m) = g$. Without loss of generality, we assume that if p multicasts a message m to group g, then p is in g.

We consider three types of atomic multicasts, *local, pairwise*, and *global*, each one of which satisfies the following properties:

- *Validity:* If a correct process multicasts a message m, then it eventually delivers m.
- *Agreement:* If a correct process delivers a message m, then all correct processes in $group(m)$ eventually deliver m.
- *Integrity:* For any message m, a correct process p delivers m at most once, and only if p is in $group(m)$, and m was multicast by some process.

However, the three atomic multicasts require different message ordering properties. These ordering properties form a hierarchy, each one corresponding to an increasing degree of interaction between groups.

A *local atomic multicast* is a multicast that satisfies the above validity, agreement and integrity properties as well as the following message ordering property:

- *Local total order:* If correct processes p and q both deliver messages m and m', and $group(m) = group(m')$, then p delivers m before m' if and only if q delivers m before m'.

Thus, only messages multicast to the same group have to be ordered with respect to each other. Inconsistency and contamination are also defined on a per group basis in the obvious manner.

If applications span several groups or interact with each other, the local total order property (which was based on completely independent groups) is insufficient. For example, consider a system with two groups $g = \{p, q, r\}$ and $g' = \{p, q, s\}$, where all four processes (p, q, r, and s) are correct (Figure 8.1). Suppose messages m'_1 and m'_2 are multicast to g' and message m to g. To satisfy *local total order*, processes p and q must both deliver m'_1 and m'_2 in the same order. However, p is allowed to deliver m before m'_1 even though q delivers m after m'_1.

We write $m < m'$ if and only if some *correct* process delivers m before it delivers m'. Thus, the above example shows that $m < m'_1$ and $m'_1 < m$. This may be undesirable in certain applications, and is prevented by *pairwise atomic multicast*, a

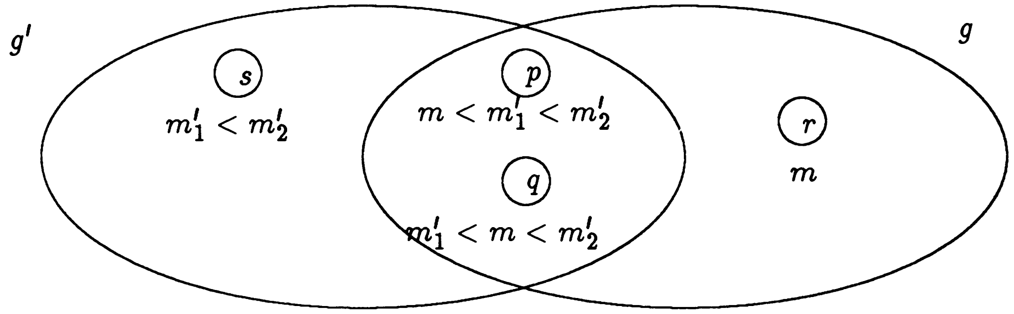


Figure 8.1: Message delivery according to local atomic multicasts

multicast that satisfies the validity, agreement, and integrity properties, as well as the following property:

- *Pairwise total order:* If correct processes p and q both deliver messages m and m', then p delivers m before m' if and only if q delivers m before m'.

However, pairwise atomic multicast permits "global cycles" in message delivery order. For example, consider a system with three groups, $g = \{p, q\}$, $g' = \{q, r\}$ and $g'' = \{r, p\}$ (Figure 8.2), where processes p, q and r are correct. Note that the intersection of any two groups in the system consists of exactly one process. The messages m, m' and m'' are multicast to groups g, g' and g'' respectively. Pairwise total order allows process p to deliver m'' before m, q to deliver m before m', and r to deliver m' before m'', leading to a global cycle in message delivery order.

Global atomic multicast, the strongest type of atomic multicast that we consider, precludes such cycles, and reflects the intuition of atomic delivery being a "single, indivisible" action. A global atomic multicast satisfies validity, agreement, and integrity properties, as well as the following property:

- *Global total order:* The relation $<$ between messages is acyclic.

Note that the specifications of each of the atomic multicasts discussed above can be augmented with the Δ-timeliness property resulting in three types of timed atomic multicasts.

We focus on visible-broadcast/delivery inconsistency and contamination with respect to pairwise atomic multicast. Thus, for the rest of this chapter, we will write "inconsistency" when we mean VBD-inconsistency and "contamination" when we mean VBD-contamination. [1]

[1]Recall that in Chapter 3 we showed that the prevention of visible-

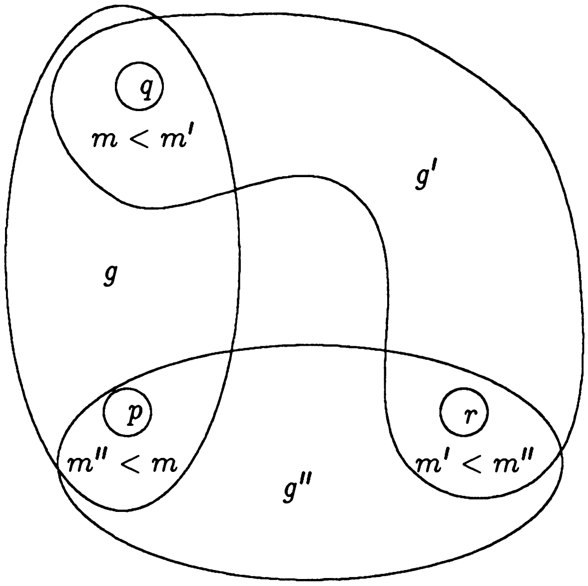


Figure 8.2: Message delivery cycle in pairwise atomic multicast

8.2 Inconsistency and contamination with respect to pairwise atomic multicast

Recall that in Chapter 5 we derived necessary and sufficient conditions for the prevention of inconsistency and contamination with respect to atomic broadcast from the general definitions of inconsistency and contamination presented in Chapter 3. However, these definitions were given for fault-tolerant broadcasts, and do not include any information about process groups. Hence, in general, they are not suitable for fault-tolerant multicasts.

For example, with local atomic multicast, inconsistency and contamination is defined on a per groups basis; thus, a process may be simultaneously inconsistent with respect to a group g, and consistent with respect to another group g'. This cannot be expressed in the formalism given in Chapter 3.

In contrast, with pairwise atomic multicast, a process is either inconsistent, or is "consistent with all groups." Thus, inconsistency and contamination with respect to pairwise atomic multicast can be expressed with minor changes to formal definitions

broadcast/delivery contamination with respect to any fault-tolerant broadcast is equivalent to the prevention of broadcast/delivery contamination with respect to that broadcast.

given in Chapter 3. In this section, we assume such changes have been made. In particular, we assume that broadcast histories are extended to include annotations about the groups to which messages are multicast, and that the set of well-formed broadcast histories $\mathcal{B}$ is appropriately modified. However, a formal extension of the material in Chapter 3 to include multicasts is beyond the scope of this thesis, and is left for future work.

Informally, a process p is not VBD-correct until some time c with respect to pairwise atomic multicast if p and some correct process q both belong to some group g, and they disagree on the sequence of messages delivered in g. In addition, p is not VBD-correct if there are two groups g and g', and a correct process q, such that both p and q belong to g and g', and they disagree on the interleaving of the messages delivered in g and g'.

For all broadcast histories B, for all subsets of groups G, $\overrightarrow{\text{DLVD}}_p^c(B)$ *restricted to* G, denoted $\overrightarrow{\text{DLVD}}_p^c(B)|G$, is the sub-sequence of messages that process p delivers by time c in any group g in G. We write $\overrightarrow{\text{DLVD}}_p^c(B)|g$ as an abbreviation for $\overrightarrow{\text{DLVD}}_p^c(B)|\{g\}$. When the broadcast history B is obvious from the context, we omit the "(B)" from the above notation. For any two processes p and q, $G_{p,q}$ is the set of all groups to which both p and q belong; formally, $G_{p,q} = \{g | p \in g, q \in g\}$.

We now define visible-broadcast/delivery correctness with respect to pairwise atomic multicast.

Definition *Let $B = (\beta, \delta) \in \mathcal{B}$ and F be such that B satisfies pairwise atomic multicast with respect to F. Process p is* VBD*-correct until c with respect to pairwise atomic multicast and F if and only if:*

- *For all correct processes q, the sequence of messages p delivers by time c restricted to $G_{p,q}$ is a prefix of the sequence of messages q delivers during the entire execution also restricted to $G_{p,q}$: $\overrightarrow{\text{DLVD}}_p^c(B)|G_{p,q} \preceq \overrightarrow{\text{DLVD}}_q(B)|G_{p,q}$.*
- *If a faulty process q (i.e., $q \in F$) delivers a message m that p multicasts at $c' \leq c$, then a correct process in $group(m)$ eventually delivers m.*
- *If a correct process q (i.e., $q \notin F$) delivers a message m that p multicasts to group g, then for all messages m' that p multicasts to g before multicasting m, q eventually delivers m'.*

Note that the above definition is a generalization of Lemma 5.5, which derives necessary and sufficient conditions for a process p to be VBD-correct with respect to atomic broadcast. The difference between the above definition and Lemma 5.5 arises from the following. With atomic broadcast, the correct processes all deliver the same sequence of messages; however, with pairwise atomic multicast, the correct processes may deliver different sequences of messages, depending on the groups to which they belong. However, if there is only one group of processes in the system, and that group includes all the processes in the system, then the above definition is identical to Lemma 5.5.

In Lemma 8.1, we derive sufficient conditions for visible-broadcast/delivery consistency. In Lemma 8.2, we derive necessary and sufficient conditions to ensure visible-broadcast/delivery contamination-free histories. The lemmas are stated using the following fifo order property:

- *FIFO order:* Suppose a process multicasts a message m to a group g before it multicasts a message m' to g. If a correct process p delivers m', then p delivers m before m'.

We also use the following property:

- *Uniform agreement:* If any process delivers a message m, then all correct processes in $group(m)$ eventually deliver m.

Lemma 8.1 *Let $A \in \mathcal{A}$, B be the broadcast history of A, and F be such that B satisfies pairwise atomic multicast with respect to F. If B satisfies the fifo order property with respect to F, and for all processes p, for all times c, for all correct processes q, $\overrightarrow{\text{DLVD}}^c_p(B)|G_{p,q} \preceq \overrightarrow{\text{DLVD}}_q(B)|G_{p,q}$, then A is a* VBD*-consistent history with respect to pairwise atomic multicast and F.*

Proof: Suppose that B satisfies the fifo order property, and that for all processes p, for all times c, for all correct processes q, $\overrightarrow{\text{DLVD}}^c_p(B)|G_{p,q} \preceq \overrightarrow{\text{DLVD}}_q(B)|G_{p,q}$. Clearly, this implies that any message delivered by any process (correct or faulty) in any group at any time, is also eventually delivered by all the correct processes in that group; *i.e.*, B satisfies the uniform agreement property with respect to F.

To complete the proof, we show that for all processes p, for all $c \leq LVT(A,p)$, p is VBD-correct until time c in B with respect to pairwise atomic broadcast and F.

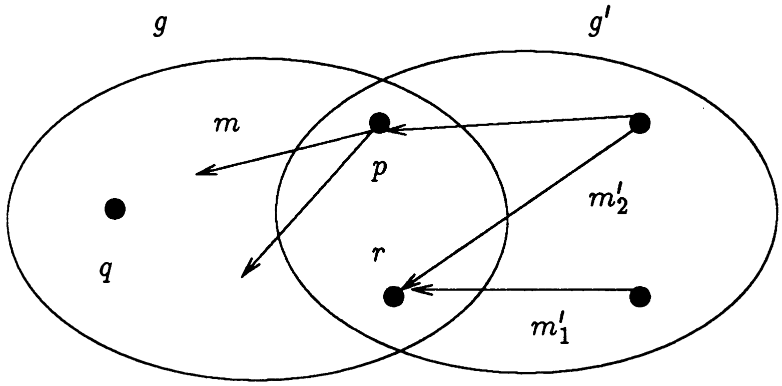
Let p be any process. If p is correct, then it is trivial to show that for all c, p is VBD-correct until time c in B with respect to pairwise atomic multicast and F.

Suppose p is faulty ($p \in F$). Let c be any time $\leq LVT(A,p)$. Since for all correct processes q, $\overrightarrow{\text{DLVD}}^c_p(B)|G_{p,q} \preceq \overrightarrow{\text{DLVD}}_q(B)|G_{p,q}$, and B satisfies both the uniform agreement and the fifo order properties with respect to F, the definition of VBD-correctness given above implies that p is VBD-correct until c in B with respect to pairwise atomic multicast and F. □

The following lemma derives necessary and sufficient conditions to ensure VBD-contamination-freedom with respect to pairwise atomic multicast. The proof is similar to the proof of Theorem 5.11.

Lemma 8.2 *Let $B = (\beta, \delta) \in \mathcal{B}$ and F be such that B satisfies pairwise atomic multicast with respect to F. Suppose B satisfies fifo order with respect to F. A correct process p is* VBD*-contamination-free until c with respect to pairwise atomic multicast and F if and only if:*

- *if p delivers any message m at or before time c, then:*
  for all correct processes q, $\overrightarrow{\mathrm{DLVD}}^{ts(m)}_{bc(m)}(B)|G_{q,bc(m)} \preceq \overrightarrow{\mathrm{DLVD}}_q(B)|G_{q,bc(m)}$.



$ts(m'_1) < ts(m'_2)$

Figure 8.3: Pairwise atomic multicast

The following example illustrates inconsistency and contamination with respect to pairwise atomic multicast. Consider a system with two groups, g and g', such that $g = \{p, q, r\}$ and $g \cap g' = \{p, r\}$ (Figure 8.3). Suppose messages m'_1 and m'_2 are atomically multicast to group g'. Suppose, r atomically delivers m'_1 followed by m'_2. Suppose that p is faulty; it atomically delivers m'_2 by some time c, but fails to atomically deliver m'_1. Thus p is inconsistent at time c with respect to pairwise atomic multicast. If the faulty process p atomically multicasts m to group g at time c (*i.e.*, after atomically delivering m'_2), and process q delivers m at some time c', then q is contaminated at time c' with respect to pairwise atomic multicast.

8.3 Overview of the pairwise atomic multicast protocols

For any group g, let n_g denote the number of processes in group g, and f_g be an upper bound on the maximum number of processes in group g that may fail. Let

$\Delta_g = (f_g + 1)\lambda$, where λ is the maximum link delay. For any two intersecting groups g and g', let $n_{g,g'}$ denote the number of processes in the intersection of g and g', and $f_{g,g'}$ be an upper bound on the maximum number of processes in the intersection of g and g' that may fail. We assume that for all groups g, $f_g \geq 1$; that is, at least one process may fail in every group.

Our multicast protocols have the desirable properties informally given below:

- *Group locality:* The processing of a multicast targeted to a particular group does not involve any process that does not belong to that group. That is, the processing of a multicast does not "spill over" to other groups.

- *Time locality:* The latency of a multicast to group g is proportional to the size of g. Thus, a multicast to a smaller group takes less time than a multicast to a larger group.

We present two different pairwise atomic multicast protocols that prevent the contamination of correct processes:

- Protocol 1 (Section 8.5.1): This protocol requires $n_{g,g'} > 2f_{g,g'}$ for any two intersecting groups g and g'. It has a latency of $\Delta_g + \lambda$ for every group g.
- Protocol 2 (Section 8.5.2): This protocol requires $n_g > 2f_g$ for every group g, and requires $n_{g,g'} > f_{g,g'}$ for any two intersecting groups g and g'. It has a latency of $2\Delta_g$, for every group g.

Our protocol that prevents inconsistency (Section 8.6) is derived from Protocol 1; it requires $n_{g,g'} > 2f_{g,g'}$ for any two intersecting groups g and g'. It has a latency of $\Delta_g + \lambda$ for every group g.

As in the case of atomic broadcast (Chapter 7), we present our protocols modularly in terms of a communications abstraction called a *basic multicast*. Formally, a basic multicast is a timed reliable multicast (*i.e.*, a multicast that satisfies the validity, agreement, integrity and Δ_g-timeliness properties for each group g) that also satisfies the following *uniform fifo order* property:

- *Uniform fifo order:* Suppose a process multicasts a message m to a group g before it multicasts a message m' to g. If any process p delivers m', then p delivers m before m'.

It is straightforward to derive a basic multicast protocol with a latency of Δ_g for message delivery in any group g.

8.4 Atomic multicast

Our simple atomic multicast protocol is very similar to our simple atomic broadcast protocol (Figure 8.4). The difference is that if a process q basic-delivers a message m in a group g, then q atomically delivers the message at time $ts(m) + \Delta_g$. Clearly, q must also belong to g.

/* p atomically multicasts a message $m = (p, c, g, data)$ at time c to group g */
  To multicast (*pairwise atomic*, $m = (p, c, g, data)$) p **multicasts** (*basic*, m)

/* p delivers basic multicast messages at c */
  For all groups $g \in belongs(p)$
    $M^c_{p,g} \equiv \{m \mid c = ts(m) + \Delta_g, p$ **delivered** $(basic, m)\}$

/* p atomically delivers messages at c */
  $M \equiv \bigcup_{g \in belongs(p)} M^c_{p,g}$
  $D \equiv$ **sort** M in timestamp order
  p **delivers** (*pairwise atomic*, D)

Figure 8.4: Pairwise atomic multicast protocol
Executed by process p at time c

Theorem 8.3 *The protocol in Figure 8.4 is a pairwise atomic multicast protocol that tolerates omission failures, and has a latency of Δ_g in each group g.*

8.5 Pairwise atomic multicast with no contamination

Recall that the protocol to prevent contamination with atomic broadcast was based on a "local" check for "mutual inconsistency." This local check is not sufficient to prevent contamination with pairwise atomic multicast, as illustrated by the following example.

Consider the system illustrated in Figure 8.3. Recall that the figure depicted a system with two groups, g and g', such that $g = \{p, q, r\}$ and $g \bigcap g' = \{p, r\}$. To make the example more readable, we will use the words "broadcast" and "deliver" when we mean "atomically broadcast" and "atomically deliver."

Suppose messages m'_1 and m'_2 are multicast to group g'. Suppose, r delivers m'_1 followed by m'_2. Suppose that p is faulty; it delivers m'_2 by some time c, but fails to deliver m'_1. Note that p is inconsistent at time c with respect to pairwise atomic multicast. Finally, suppose that the faulty process p multicasts m to group g at time c (*i.e.*, after delivering m'_2).

Since p and r disagree on the sequence of messages delivered to group g' by time c, to avoid becoming contaminated, r cannot deliver the message m.

Suppose that q is correct. When q receives m from p, it cannot "notice" p's inconsistency: q does not belong to group g', so, by the group locality property, q is not even aware of the message m'_1 that p failed to deliver. Thus if q applies a local consistency check to p, it will decide to deliver m. In such a case, q will disagree with r on the delivery of m. This violates the requirement that all correct processes in group g agree on the messages delivered in g.

Thus, to decide whether or not to deliver m, q *cannot* make a purely local decision (as was the case with atomic broadcast), but must rely on "help" from the processes in $g \bigcap g'$. Consequently, it is clear that preventing contamination with respect to pairwise atomic multicast is more difficult than preventing contamination with atomic broadcast.

Suppose q "consults" with the processes in $g \bigcap g'$ (in this case processes p and r) to determine whether p was inconsistent at the time it multicast m, and q delivers m only if p was consistent. Thus, q must decide if the correct processes in group g' delivered m'_1 before m'_2, or if they only delivered m'_2; that is, q must decide whether p is faulty or whether r is faulty. If r is correct, the agreement property of atomic multicast implies that q must not deliver m. If p is correct, the validity property of atomic multicast implies that q must deliver m.

To determine whether or not to deliver m, q cannot "consult" with processes in group g' that are not also in group g (by the group locality property). Thus, it appears that q cannot decide which of p and r is faulty unless more than half the processes in the intersection of groups g and g' are correct. Surprisingly, this is not the case.

We present two approaches by which q could determine which of p or r is faulty. The first approach, (Protocol 1, Section 8.5.1), requires a majority of correct processes in the intersection of any two groups g and g' (*i.e.*, if $g \bigcap g' \neq \phi$ then $n_{g,g'} > 2f_{g,g'}$).

The second approach by which q could decide which of p or r is faulty does not require a majority of correct processes in $g \bigcap g'$. This is explained below.

Suppose that m_1' and m_2' had been multicast using a pairwise atomic multicast that guaranteed both the simultaneity (Chapter 7) and the uniform agreement (defined earlier in this chapter) properties.

With such a multicast, a faulty process cannot deliver "extra" messages. In particular, if a faulty process delivers a message in a particular group, then all the correct processes in that group will also deliver that message. Furthermore, if a correct and a faulty process both deliver any message, then they both deliver the message at the same local time. Thus, a faulty process cannot deliver messages out-of-order. Indeed, the *only way* in which a faulty process may disagree with the correct processes is by failing to deliver a message that is delivered by the correct processes.

With such a multicast, it is clear that the scenario illustrated in Figure 8.3 can arise only if p is faulty. In particular, since r delivers m_1' at some time c', the uniformity and simultaneity properties imply that all correct processes in g' also deliver m_1' at time c'. Thus, since p does not deliver m_1' at time c', and $p \in g'$, p cannot be correct.

Protocol 2 (Section 8.5.2) is based on the approach discussed above. It requires that if any two groups intersect, then there is at least one correct process in the intersection of the two groups. It also requires that a majority of processes in each group are correct; we show that this requirement is necessary for any pairwise atomic multicast protocol that prevents the contamination of correct processes.

8.5.1 Protocol 1

To atomically multicast a message $m = (p, c, g, data)$ to group g at time c, process p *multicasts* $(basic, b)$ to g, where b is the tuple $b = (p, c, g, (data, \overrightarrow{\text{DLVD}}_p^c))$, and $\overrightarrow{\text{DLVD}}_p^c$ is the sequence of all the messages that p has atomically delivered by time c in all the groups to which p belongs.

If process q *delivers* $(basic, b)$, by time $c + \Delta_g$, then for each group g' to which both p and q belong, q does the following. If $\overrightarrow{\text{DLVD}}_q^c|g' = \overrightarrow{\text{DLVD}}_p^c|g'$, then at time $c + \Delta_g$, q sends (q, b, g', ACK) to all processes in g.

At time $c + \Delta_g + \lambda$, q checks to see if, for *every* group g' to which p belongs, a majority of processes in the intersection of g and g' agree with p on the sequence of messages atomically delivered in g'. If so, then q atomically delivers $(p, c, g, data)$; otherwise, q ignores the message.

In response to a process p's multicast of a message m, the protocol requires that a process q send up to $|G_{p,q}|$ messages of the form $(q, —, —, ACK)$. It is straightforward to modify the protocol so that q sends at most one such message. Furthermore, techniques similar to those proposed in Section 7.3 can be used to further reduce the message complexity of this protocol. For clarity, however, we present the less efficient protocol informally described above (Figure 8.5).

To simplify the presentation, if $b = (p, c, g, (data, —))$ and $m = (p, c, g, data)$, we write $ts(b)$ to mean $ts(m)$ (*i.e.*, c) and $bc(b)$ to mean $bc(m)$ (*i.e.*, p).

Lemma 8.4 *If any process atomically delivers a message m, then it atomically delivers m at time $ts(m) + \Delta_{group(m)} + \lambda$.*

Lemma 8.5 *If p and q atomically deliver messages m and m', then p atomically delivers m before m' if and only if q atomically delivers m before m'.*

Lemma 8.6 *If a correct process atomically delivers m, then all correct processes in $group(m)$ eventually atomically deliver m.*

Proof: Suppose for contradiction that m is the earliest message (*i.e.*, the smallest message in lexicographic order) such that some correct process p atomically delivers m, and some correct process q in $group(m)$ does not atomically deliver m. Suppose that $m = (x, c, g, data)$ and let b denote $(x, c, g, (data, \overrightarrow{\mathrm{DLVD}}^c_x))$.

Since p atomically delivers m, $b \in B_p^{c+\Delta_g}$, and by the properties of the basic multicast, for all correct processes $r \in g$, $b \in B_r^{c+\Delta_g}$; in particular, since q is correct and $q \in g$, $b \in B_q^{c+\Delta_g}$.

Since q does not atomically deliver m at time $c + \Delta_g + \lambda$, x belongs to some group g^*, such that $|acks_{q,b,g^*}| < n_{g,g^*} - f_{g,g^*}$.

Since p atomically delivers m, for all groups g' to which x belongs, $|acks_{p,b,g'}| \geq n_{g,g'} - f_{g,g'}$; in particular, $|acks_{p,b,g^*}| \geq n_{g,g^*} - f_{g,g^*}$. Since $n_{g,g^*} > 2f_{g,g^*}$ (by assumption), at least one correct process r sent a message (r, b, g^*, ACK); *i.e.*, there is at least one correct process r in $g \bigcap g^*$ such that $\overrightarrow{\mathrm{DLVD}}^c_x|g^* = \overrightarrow{\mathrm{DLVD}}^c_r|g^*$.

Let s be any correct process in $g \bigcap g^*$. By Lemma 8.4 and by choice of m (as the earliest message on which the correct processes disagree), $\overrightarrow{\mathrm{DLVD}}^c_r|g^* = \overrightarrow{\mathrm{DLVD}}^c_s|g^*$. Since s is correct, $b \in B_s^{c+\Delta_g}$ (as we showed above), and hence s sends (s, b, g^*, ACK) at time $c + \Delta_g$.

Since process q is correct, q receives at least $n_{g,g^*} - f_{g,g^*}$ messages of the form $(—, b, g^*, ACK)$; hence $|acks_{q,b,g^*}| \geq n_{g,g^*} - f_{g,g^*}$. This is a contradiction to the earlier assertion that $|acks_{q,b,g^*}| < n_{g,g^*} - f_{g,g^*}$. □

Lemma 8.7 *For all correct processes p and q, and all times c, $\overrightarrow{\mathrm{DLVD}}^c_p|G_{p,q} = \overrightarrow{\mathrm{DLVD}}^c_q|G_{p,q}$.*

Proof: From Lemmas 8.4 and 8.6. □

/* p atomically multicasts a message $m = (p, c, g, data)$ at time c to group g */

To **multicast** (*pairwise atomic*, $m = (p, c, g, data)$)

p **multicasts** (*basic*, $b = (p, c, g, (data, \overrightarrow{\text{DLVD}}_p^c)))$

/* p delivers basic multicast messages at c */

$B_p^c \equiv \{b \mid ts(b) = c - \Delta_{group(b)},\ p$ **delivered** (*basic*, $b)\}$

/* p sends messages at c */

For all $b \in B_p^c$ **for all** $g \in G_{p,bc(b)}$

if $\overrightarrow{\text{DLVD}}_{bc(b)}^{ts(b)}|g = \overrightarrow{\text{DLVD}}_p^{ts(b)}|g$ **then** p **sends** (p, b, g, ACK) to all in $group(b)$

/* p receives all messages at c */

For all $b \in B_p^{c-\lambda}$ **for all** groups $g \in belongs(bc(b))$

$ack_{p,b,g} \equiv \{q \mid p$ **received** $(q, b, g, ACK)\}$

/* p atomically delivers messages at c */

$S = \phi$

for all $b \in B_p^{c-\lambda}$, $b = (q, c - \lambda - \Delta_{g^*}, g^*, (data, \text{—}))$

if for all groups $g \in belongs(bc(b))$: $|ack_{p,b,g}| \geq n_{g,g^*} - f_{g,g^*}$ **then**

$S = S \bigcup \{(q, c - \lambda - \Delta_{g^*}, g^*, data)\}$

$D \equiv$ **sort** S in timestamp order

p **delivers** (*pairwise atomic*, D)

Figure 8.5: Pairwise atomic multicast protocol that prevents contamination
Executed by process p at time c

Lemma 8.8 *If the multicaster of a message m is correct, then all correct processes in $group(m)$ eventually atomically deliver m.*

Proof: Suppose a correct process p atomically multicasts a message m at time c to group g. Thus process p basic-multicasts the message $b = (p, c, g, (data, \overrightarrow{\text{DLVD}}_p^c))$ to group g at time c.

Let q be any correct process in g. The validity and agreement properties satisfied by the basic multicast imply that q basic-delivers b by time $c + \Delta_g$. Therefore, to prove the lemma it is enough to show that by time $c + \Delta_g + \lambda$, for all groups $g' \in belongs(p)$, q receives at least $n_{g,g'} - f_{g,g'}$ messages of the form $(—, b, g', ACK)$.

Let g' be any group such that $g' \in belongs(p)$. Let r be any correct process in $g \bigcap g'$. The validity property of the basic multicast implies that $b \in B_r^{c+\Delta_g}$. Lemma 8.6 implies that $\overrightarrow{\text{DLVD}}_r^c|g' = \overrightarrow{\text{DLVD}}_p^c|g'$. Thus, r sends a message of the form (r, b, g', ACK) to all processes in group g. Therefore, q receives at least $n_{g,g'} - f_{g,g'}$ messages of the form $(—, b, g', ACK)$. □

Lemma 8.9 *If a correct process p atomically delivers a message m, then for all correct processes q, $\overrightarrow{\text{DLVD}}_{bc(m)}^{ts(m)}|G_{q,bc(m)} = \overrightarrow{\text{DLVD}}_q^{ts(m)}|G_{q,bc(m)}$.*

Proof: Suppose for contradiction that m is the earliest message, such that a correct process p atomically delivers m, and $\overrightarrow{\text{DLVD}}_{bc(m)}^{ts(m)}|G_{q,bc(m)} \neq \overrightarrow{\text{DLVD}}_q^{ts(m)}|G_{q,bc(m)}$ for some correct process q. Let $m = (x, c, g, data)$ and $b = (x, c, g, (data, \overrightarrow{\text{DLVD}}_x^c))$. Lemma 8.5 implies that there is a group $g' \in G_{q,x}$ such that $\overrightarrow{\text{DLVD}}_x^c|g' \neq \overrightarrow{\text{DLVD}}_q^c|g'$.

Since $g' \in G_{q,x}$, and x belongs to both groups g and g', $g \bigcap g' \neq \phi$.

Let r be any correct process in $g \bigcap g'$. Since q and r are correct, and both belong to group g', Lemma 8.7 implies that $\overrightarrow{\text{DLVD}}_r^c|g' = \overrightarrow{\text{DLVD}}_q^c|g'$; thus, $\overrightarrow{\text{DLVD}}_x^c|g' \neq \overrightarrow{\text{DLVD}}_r^c|g'$. Thus, r does not send a message of the form $(—, b, g, ACK)$. Since $n_{g,g'} > 2f_{g,g'}$, p receives fewer than $n_{g,g'} - f_{g,g'}$ messages of the form $(—, b, g, ACK)$. Hence p does not atomically deliver m. This is a contradiction. □

Corollary 8.10 *If a correct process p atomically delivers a message m, then for all correct processes q, $\overrightarrow{\text{DLVD}}_{bc(m)}^{ts(m)}|G_{q,bc(m)} \preceq \overrightarrow{\text{DLVD}}_q|G_{q,bc(m)}$.*

Lemma 8.11 *If p atomically multicasts a message m to a group g before atomically multicasting m' to g, and any correct process q atomically delivers m', then q atomically delivers m before m'.*

Proof: The proof is by contradiction. Suppose for contradiction that p atomically multicasts m to group g before m' to g (*i.e.*, $ts(m) < ts(m')$), and a correct process

q atomically delivers m', but does not atomically deliver m by the time it atomically delivers m'. Let c denote $ts(m)$ and c' denote $ts(m')$.

Since q atomically delivers m', q *delivers* $(basic, m')$ by time $c' + \Delta_g$. Hence, by the fifo order and Δ-timeliness properties satisfied by basic multicast and delivery, q *delivers* $(basic, m)$ by time $c + \Delta_g$. Since q did not atomically deliver m, for some group g^* such that p belongs to g^*, for all correct processes r in $g \bigcap g^*$, $\overrightarrow{\text{DLVD}}_r^c|g^* \neq \overrightarrow{\text{DLVD}}_p^c|g^*$.

Since q atomically delivers m' at $c' + \Delta_g + \lambda$, for all groups g' such that p belongs to g', for all correct processes r in $g \bigcap g'$, $\overrightarrow{\text{DLVD}}_r^{c'}|g' = \overrightarrow{\text{DLVD}}_p^{c'}|g'$. In particular, for all correct processes $r \in g \bigcap g^*$, $\overrightarrow{\text{DLVD}}_r^{c'}|g^* = \overrightarrow{\text{DLVD}}_p^{c'}|g^*$. Since $c \leq c'$, this contradicts the choice of the group g^*, and completes the proof. □

Theorem 8.12 *Suppose for all groups g and g', if $g \bigcap g' \neq \phi$, then $n_{g,g'} > 2f_{g,g'}$. The protocol in Figure 8.5 is a pairwise atomic multicast protocol that prevents the contamination of correct processes, tolerates omission failures and has a latency of $\Delta_g + \lambda$ in each group g.*

Proof: It is clear that the integrity property is satisfied. Hence Lemmas 8.7, 8.8, and 8.5 imply that the protocol is a pairwise atomic multicast protocol. Lemma 8.2, together with Corollary 8.10 and Lemma 8.11, implies that the protocol prevents contamination with respect to pairwise atomic multicast.

The latency of $\Delta_g + \lambda$ follows directly from the protocol. □

8.5.2 Protocol 2

The previous protocol requires that more than half the processes in the intersection of any two groups are correct. This seems necessary to identify the faulty processes in the intersection of the two groups. However, this is not necessary. To prevent contamination, it is sufficient for each group in the system to have a majority of correct processes. This is the case in the protocol in Figure 8.7, which requires that $n_g > 2f_g$ for all groups g. As we mentioned in the introduction to this section, the protocol ensures both the simultaneity (Lemma 8.23) and uniform agreement (Lemma 8.27) properties.

The protocol uses (as subroutines) the following multicasts:

- *Uniform multicast*: This satisfies the validity, uniform agreement, integrity and Δ_g-timeliness properties. We do not present a uniform multicast protocol;

however, such a protocol is easily derived from our atomic broadcast protocol that prevents inconsistency and has a latency of Δ (Chapter 5).

- *Strong uniform multicast*: This is a uniform multicast that also satisfies the property stated informally below, and formalized in the next section:
  Suppose any process delivers a message m. Any process p that is "aware" of the multicast of m either delivers m or halts by time $ts(m) + \Delta_{group(m)}$.

Strong uniform multicast

A strong uniform multicast (called a *strong multicast* for short) is a uniform multicast that satisfies the following properties. For all groups g, for all processes p in g, for all time c, the processes in g determine whether or not to *allow* p to multicast a message at time c as follows:

- If a correct process q allows p to multicast a message to g at time c, then all correct processes also allow p to multicast a message to g at time c.
- If p multicasts a message m to group g at time c, then p allows itself to multicast a message to group g at time c.
- If any process delivers a message $m = (p, c, g, —)$, then for all processes q in g, either q delivers m by time $c + \Delta_g$, or q halts by time $c + \Delta_g$, or q did not allow p to multicast a message to group g at time c.

In our strong-multicast protocol (Figure 8.6), the variable $allow_q(p, c, g)$ denotes whether q allows p to strong-multicast a message to group g at time c. We assume:

- $allow_q(p, c, g)$ is either defined by q at time c before any messages are sent or multicast, or remains undefined.
- $allow_q(p, c, g)$ is defined to be *true* if and only if q allows p to strong-multicast a message to group g at time c.
- For all groups g, for all processes p in g, for correct processes q and r in g, for all c, $allow_q(p, c, g) = true$ if and only if $allow_r(p, c, g) = true$.
- If p strong-multicasts a message m at c to g, $allow_p(p, c, g) = true$.

We also assume that at least one process in each group may be faulty (*i.e.*, $\forall g, f_g \geq 1$).

To *multicast* $(strong, m)$ to group g at time c, p sends m to all processes in g.

Suppose a process q allows p to strong-multicast a message to g at time c; *i.e.*, $allow_q(p, c, g)$ is true. When q first receives m, q sends m to all processes in g.

At time $c + \Delta_g - \lambda$, q sends a message to all processes in g, indicating whether it has received a message m multicast by process p at time c (an "ack") or not (a "nak").

At time $c + \Delta_g$, q first determines whether it is faulty and, if so, it halts.

- If q does not receive at least $n_g - f_g$ of the messages sent at time $c + \Delta_g - \lambda$, then q has failed to receive messages from the correct processes; hence q halts.
- Suppose that either $q = p$ or q receives m by time $c + \Delta_g - 2\lambda$. If q does not receive at least $n_g - f_g$ "ack" messages, then q has failed to send m properly; hence q halts.

At time $c + \Delta_g$, if q has not halted, then q strong-delivers m.

/* p **multicasts** ($strong, m = (p, c, g, data)$) at time c to group g */
/* For all processes q, $allow_q(p, c, g)$ is a boolean that is known by time c */

If $allow_p(p, c, g)$ **then at** time c p **sends** m to all in g

/* Executed by any process q */
If $allow_q(p, c, g)$ **then**
  when $q \neq p$ first **receives** a message of the form $m = (p, c, g, —)$:
    q **sends** m to all in g

  At time $c + \Delta_g - \lambda$
    if q has **received** $m = (p, c, g, —)$
      then q **sends** $(q, (p, c, g), ACK)$ to all in g
    else q **sends** $(q, (p, c, g), NAK)$ to all in g

  At time $c + \Delta_g$
    $nak_q(p, c, g) := \{r | q$ **received** $(r, (p, c, g), NAK)\}$
    $ack_q(p, c, g) := \{r | q$ **received** $(r, (p, c, g), ACK)\}$
    if $|ack_q(p, c, g)| + |nak_q(p, c, g)| < n_g - f_g$ **then** q **halts**
    if ($q = p$ or q **received** $(p, c, g, —)$ by $c + \Delta_g - 2\lambda$)
      and $|ack_q(p, c, g)| < n_g - f_g$ **then** q **halts**
    if q **received** $m = (p, c, g, —)$ by $c + \Delta_g$ **then** q **delivers** ($strong, m$)

Figure 8.6: Strong multicast protocol

Lemma 8.13 *If any process q strong-delivers a message of the form $(p, c, g, —)$, then $allow_q(p, c, g)$ is true.*

Lemma 8.14 *For all groups g, for all p and q in g, for all c, if $allow_q(p, c, g)$ is true and $allow_r(p, c, g)$ is not true for some correct process r in g, then q halts by time $c + \Delta_g$ before delivering any messages at time $c + \Delta_g$.*

Proof: Suppose for some group g, for processes p and q in g, for some c, $allow_q(p, c, g)$ is true and $allow_r(p, c, g)$ is not true for some correct process r in g. In other words, q allows p to strong-multicast a message at time c, but some correct process r in g does not.

Let s be any correct process in g. Since $allow_r(p, c, g)$ is not true, and r is correct, $allow_s(p, c, g)$ is not true (by assumption). Therefore, at time $c + \Delta_g - \lambda$, s does not send a message of the form $(s, (p, c, g), —)$, and hence $s \notin ack_q(p, c, g)$ and $s \notin nak_q(p, c, g)$.

Since $n_g > 2f_g$, process q receives at most f_g "acks" and "naks," and hence we conclude that $|ack_q(p, c, g)| + |nak_q(p, c, g)| < n_g - f_g$. Therefore, q halts by time $c + \Delta_g$.

Since q determines whether or not to halt itself at any time *before* delivering any messages at that time (see Figure 6.1 of Chapter 6), we conclude that q halts by time $c + \Delta_g$, before delivering any messages at time $c + \Delta_g$. □

Lemma 8.15 *A correct process never halts.*

Proof: Assume for contradiction that q is the first correct process to halt. Thus, from the protocol, there is some group g to which q belongs, some process p in g, and some time c such that $allow_q(p, c, g)$ is true, and q halts at time $c + \Delta_g$.

Case: $|ack_q(p, c, g)| + |nak_q(p, c, g)| < n_g - f_g$: Recall that by assumption, all correct processes in group g agree on whether p is allowed to multicast a message to group g at time c. Since q is correct and $allow_q(p, c, g)$ is true, for all correct processes r in g, $allow_r(p, c, g)$ is true.

Since q is the first correct process that halts (at time $c + \Delta_g$), no correct process halts by time $c + \Delta_g - \lambda$. Thus, all correct processes in group g send a message of the form $(—, (p, c, g), —)$. Since q is correct, and $n_g > 2f_g$, q receives at least $n_g - f_g$ such messages; *i.e.*, $|ack_q(p, c, g)| + |nak_q(p, c, g)| \geq n_g - f_g$. This is a contradiction.

Case: ($q = p$ or q received $m = (p, c, g, —)$ by $c + \Delta_g - 2\lambda$) and $|ack_q(p, c, g)| < n_g - f_g$: Suppose $q = p$, or q received m by $c + \Delta_g - 2\lambda$. In either case, (since

$f_g \geq 1$ and hence $\Delta_g - 2\lambda \geq 0$) q sends the message m by time $c + \Delta_g - 2\lambda$ to all processes in g.

Let r be any correct process in group g. Since q is correct, and $allow_q(p, c, g)$ is true, $allow_r(p, c, g)$ is true. Thus, r receives m by time $c + \Delta_g - \lambda$, and sends the message $(r, (p, c, g), ACK)$. Furthermore, this message is received by q by time $c + \Delta_g$.

Since there are at least $n_g - f_g$ correct processes in group g, q receives at least $n_g - f_g$ messages of the form $(—, (p, c, g), ACK)$; *i.e.*, $|ack_q(p, c, g)| \geq n_g - f_g$. This is a contradiction. □

Lemma 8.16 *If a correct process q receives $m = (p, c, g, —)$, and $allow_q(p, c, g)$ is true, then all correct processes strong-deliver m by $c + \Delta_g$.*

Proof: Suppose a correct process q receives $m = (p, c, g, —)$, and $allow_q(p, c, g)$ is true. By assumption, for all correct processes r in g, $allow_r(p, c, g)$ is also true. By a standard "pigeonhole" argument, it is easy to show that all correct processes receive m by time $c + \Delta_g$. Since correct processes cannot halt (Lemma 8.15), we conclude that all correct processes strong-deliver m by time $c + \Delta_g$. □

Lemma 8.17 *If a correct process strong-multicasts m at time c to g, then:*

- *All correct processes in g receive and relay m by time $c + \lambda$.*
- *All correct processes in g strong-deliver m by time $c + \Delta_g$.*

Lemma 8.18 *The protocol in Figure 8.6 satisfies the validity, agreement, Δ_g-timeliness, and integrity properties.*

Lemma 8.19 *If any process strong-delivers a message $m = (p, c, g, —)$, then for all correct processes q in g, $allow_q(p, c, g)$ is true.*

Proof: Suppose a process p strong-multicasts a message m at some time c to a group g, and a process r strong-delivers m. Thus, $allow_r(p, c, g)$ is true. Since r strong-delivers m, it does not halt before delivering messages at time $c + \Delta_g$. Hence, Lemma 8.14 implies that for all correct process q in g, $allow_q(p, c, g)$ is true. □

Lemma 8.20 *If any process strong-delivers a message $m = (p, c, g, —)$, then all correct processes in g strong-deliver m at time $c + \Delta_g$.*

Proof: Suppose p strong-multicasts message m at time c to group g, and a process q strong-delivers m. First note that $allow_q(p,c,g)$ is true and that by Lemma 8.19, for all correct processes r in g, $allow_r(p,c,g)$ is true.

If p is correct, then the lemma follows from Lemma 8.17. If q is correct, then the lemma follows from Lemma 8.16. Hence suppose p and q are faulty. The proof is a case analysis.

Case: $q = p$ or q receives m by time $c + \Delta_g - 2\lambda$: Since q strong-delivers m and does not halt, $|ack_q(p,c,g)| \geq n_g - f_g$. Since $n_g > 2f_g$, some correct process $r \in ack_q(p,c,g)$; *i.e.*, r receives and relays m by time $c + \Delta_g - \lambda$. Since $allow_r(p,c,g)$ is true, Lemma 8.16 implies that all correct processes in g strong-deliver m by time $c + \Delta_g$.

Case: $q \neq p$ and q receives m after time $c + \Delta_g - 2\lambda$: By a standard "pigeonhole" argument, at least $f_g - 1$ processes, not including p nor q, receive and relay m by time $c + \Delta_g - 2\lambda$. Since p and q are faulty, at least one of those $f_g - 1$ processes is correct. Since for all correct processes r in g, $allow_r(p,c,g)$ is true, Lemma 8.16 implies that all correct processes in g strong-deliver m by time $c + \Delta_g$. □

Lemma 8.21 *Suppose a process strong-delivers a message $m = (p,c,g,—)$. If any process q in g does not strong-deliver m at time $c + \Delta_g$, and $allow_q(p,c,g)$ is true, then q halts by time $c + \Delta_g$, before delivering any messages at time $c + \Delta_g$.*

Proof: Suppose p strong-multicasts m at time c in g, and some process strong-delivers m; by Lemmas 8.19 and 8.20, for all correct processes r in g, $allow_r(p,c,g)$ is true, r receives m at time $c + \Delta_g$ and strong-delivers m by time $c + \Delta_g$.

Suppose for contradiction that some process q in g does not strong-deliver m at time $c + \Delta_g$, $allow_q(p,c,g)$ is true and that q does not halt before delivering messages at $c + \Delta_g$. Thus, q does not receive m and $|ack_q(p,c,g)| + |nak_q(p,c,g)| \geq n_g - f_g$. Furthermore, since q does not strong-deliver m, and all correct processes strong-deliver m, q is faulty.

Since q is faulty and does not receive m, and the correct processes in g receive m, we conclude from the "pigeonhole" principle that at least one correct process in g receives m by time $c + \Delta_g - 2\lambda$. Thus, by time $c + \Delta_g - \lambda$, all correct processes in g send m to all processes in g.

Let r be any correct process in g. From the protocol, since $allow_r(p,c,g)$ is true, r sends a message of the form $(r,(p,c,g),ACK)$ to all processes in g at time $c + \Delta_g - \lambda$. By the fifo-links property (Chapter 5), if q receives this message, q receives r's relay of message m by time $c + \Delta_g$.

Therefore, q does not receive any message of the form $(—, (p, c, g), —)$ from any correct process. Since $n_g > f_g$, we conclude that $|ack_q(p, c, g)| + |nak_q(p, c, g)| < n_g - f_g$. This is a contradiction. □

Theorem 8.22 *Suppose for all groups g, $n_g > 2f_g$ and $f_g \geq 1$. The protocol illustrated in Figure 8.6 is a strong multicast protocol that tolerates omission failures and has a latency of Δ_g in each group g.*

Proof: From Lemmas 8.18, 8.20 and 8.21. □

With the strong multicast protocol presented above, for all groups g, for all processes p in g, p may strong-multicast more than one message to g at any instant of time using "piggybacking" techniques. Thus, if p wishes to strong-multicast $m = (p, c, g, data)$ and $m' = (p, c, g, data')$ at the same time, it actually strong-multicasts the message $m^* = (p, c, g, (data, data'))$. If a process q strong-delivers m^* at time c', for convenience, we say that q strong-delivers both m and m' at time c'. This "transparent piggybacking" technique is assumed in the our pairwise atomic multicast protocol that prevents contamination (Figure 8.7).

Pairwise atomic multicast protocol to prevent contamination

The pairwise atomic multicast protocol to prevent contamination, shown in Figure 8.7, is given using uniform multicast and strong multicast as subroutines.

To atomically multicast $m = (p, c, g, data)$ to group g at time c, process p *multicasts* $(uniform, b)$ to g, where b is the tuple $b = (p, c, g, (data, \overrightarrow{\text{DLVD}}_p^c))$, and $\overrightarrow{\text{DLVD}}_p^c$ is the sequence of messages that p has atomically delivered by time c in all the groups to which p belongs.

If a process q *delivers* $(uniform, b)$ in g by time $c + \Delta_g$, then q does the following at time $c + \Delta_g$. If q atomically delivers a message m^* by time c, $p \in group(m^*)$, and p does not atomically deliver m^* by time c (*i.e.*, there is a message m^* that p failed to atomically deliver), then q multicasts the message $(q, c + \Delta_g, g, (b, NAK))$ to group g using the strong multicast. Furthermore, for all processes r in g, q allows r to multicast a message $(r, c + \Delta_g, g, (b, NAK))$; *i.e.*, q sets $allow_q(r, c + \Delta_g, g)$ to true.

At time $c + 2\Delta_g$, if q does not *deliver* $(strong, (—, c + \Delta_g, g, (b, NAK)))$, then q atomically delivers m; otherwise q ignores m.

Lemma 8.23 *If any process atomically delivers a message m, then it atomically delivers m at time $ts(m) + 2\Delta_{group(m)}$.*

Lemma 8.24 *If for some group g, $allow_q(-, c+\Delta_g, g)$ is true for some process q in g at time $c+\Delta_g$, then q has uniform-delivered a message $b = (-, c, g, (-, -))$ by time $c+\Delta_g$; i.e., $b \in B_{q,g}^{c+\Delta_g}$.*

Proof: Directly from the protocol. □

/* p atomically multicasts a message $m = (p, c, g, data)$ at time c to group g */
  To multicast $(pairwise\ atomic, m = (p, c, g, data))$
    p **multicasts** $(uniform, b = (p, c, (data, \overrightarrow{\text{DLVD}}_p^c)))$

/* p delivers uniform multicast messages at c */
  For all g : $B_{p,g}^c \equiv \{b \mid ts(m) = c - \Delta_g,\ p$ **delivered** $(uniform, b)\}$

/* p multicasts messages at time c using the strong multicast*/
  For all g **for all** $b \in B_{p,g}^c$
    for all $q \in g$ $allow_p(q, c, g) := true$
    if $\exists m^* \in \overrightarrow{\text{DLVD}}_p^{c-\Delta_g}$ and $bc(b) \in group(m^*)$ and $m^* \notin \overrightarrow{\text{DLVD}}_{bc(b)}^{c-\Delta_g}$ **then**
      p **multicasts** $(strong, (p, c, g, (b, NAK)))$

/* p atomically delivers messages at c */
  $S = \phi$
  for all g **for all** $b \in B_{p,g}^{c-\Delta_g}$, $b = (q, c - 2\Delta_g, g, (data, \overrightarrow{\text{DLVD}}_q^{c-2\Delta_g}))$
    if p did not **deliver** $(strong, (-, c - \Delta_g, g, (b, NAK)))$
      and p atomically delivered q's previous multicasts to g
        then $S = S \bigcup \{(q, c - 2\Delta_g, g, data)\}$
  $D \equiv$ **sort** S in timestamp order
  p **delivers** $(pairwise\ atomic, D)$

Figure 8.7: Pairwise atomic multicast protocol that prevents contamination
Executed by process p at time c

Lemmas 8.25 and 8.26 show that the assumptions about the "*allow*" variables made for the strong multicast protocol are indeed satisfied.

Lemma 8.25 *For all groups g, for all $p \in g$, for all c, for all processes q in g, if $allow_q(p, c, g)$ is true at time c, then:*

- *for all processes s in g, $allow_q(s, c, g)$ is true at time c, and*
- *if q is correct, then for all correct processes r in g, $allow_r(p, c, g)$ is true at time c.*

Proof: Suppose for some g, for some p in g, for some c, $allow_q(p, c, g)$ is true for some process q in g. Lemma 8.24 implies that there is a message $b = (—, c - \Delta_g, g, (—, —)) \in B^c_{q,g}$.

Let s be any process in group g. Since $b \in B^c_{q,g}$, q also sets $allow_q(s, c, g)$ to true at time c.

Suppose q is correct. Let r be any correct process in group g. By the agreement property of the uniform multicast, $b \in B^c_{r,g}$. Hence, from the protocol, r also sets $allow_r(p, c, g)$ to true at time c.

□

Lemma 8.26 *If any process p strong-multicasts a message $(p, c, g, (—, NAK))$, then $allow_p(p, c, g) = true$.*

Proof: Directly from the protocol. □

Lemma 8.27 *If any process atomically delivers m, then all correct processes in $group(m)$ atomically deliver m at time $ts(m) + 2\Delta_g$.*

Proof: Suppose a process q in some group g atomically delivers a message $m = (p, c - \Delta_g, g, data)$ at time $c + \Delta_g$. Thus by time c, q uniform-delivers the message $b = (p, c - \Delta_g, g, (\overrightarrow{\text{DLVD}}_p^{c-\Delta_g}, data))$. Hence, from the protocol, for all processes s in g, $allow_q(s, c, g)$ is true.

Since q uniform-delivers b, the uniform agreement property satisfied by uniform multicast implies that for all correct processes r in g, r also uniform-delivers b by time c.

For all processes s in g, since $allow_q(s, c, g)$ is true, and q does not halt before atomically delivering m at time $ts(m) + \Delta_g$, Lemma 8.14 implies that some correct process allows s to strong-multicast a message at time c to group g. Thus, from Lemma 8.25, for all correct processes r in g, for all processes s in g, r allows s to strong-multicast a message at time c to group g; *i.e.*, $allow_r(s, c, g)$ is true.

Since q atomically delivers m, q does not *deliver* $(strong, (—, c, g, (b, NAK)))$. Since q does not halt before delivering messages at time $c + \Delta_g$, and for all s in g,

$allow_q(s, c, g)$ is true, Lemma 8.21 implies that for all correct processes r in g, r does not *deliver* $(strong, (—, c, g, (b, NAK)))$.

Let r be any correct process in g. Since r uniform-delivers b by time c, and r does not *deliver* $(strong, (—, c, g, (b, NAK)))$, we conclude that r atomically delivers m at time $c + \Delta_g$. □

Lemma 8.28 *For all correct processes p and q, and all times c, $\overrightarrow{\text{DLVD}}_p^c | G_{p,q} = \overrightarrow{\text{DLVD}}_q^c | G_{p,q}$.*

Proof: Follows from Lemmas 8.23 and 8.27 □

Lemma 8.29 *If the multicaster of a message m is correct, then all correct processes in $group(m)$ atomically deliver m at $ts(m) + 2\Delta_g$.*

Proof: Suppose for contradiction that a correct process p atomically multicasts a message $m = (p, c, g, data)$ at time c to group g, and a correct process $q \in g$ does not atomically deliver m.

Since p atomically multicasts m, and p is correct (and hence does not halt), p also uniform-multicasts the message $b = (p, c, g, (data, \overrightarrow{\text{DLVD}}_p^c))$ at time c. Since p and q are correct, the validity and agreement property of the uniform multicast implies that q uniform-delivers b by time $c + \Delta_g$; *i.e.*, $b \in B_{q,g}^{c+\Delta_g}$. Since q does not atomically deliver m, q *delivers* $(strong, (r, c + \Delta_g, g, (b, NAK)))$ for some process r in g. Therefore, there is a message m^* such that r atomically delivers m^* by time c, p does not atomically deliver m^* by time c, and p belongs to $group(m^*)$.

Since p is correct, and r atomically delivers m^* by time c, Lemmas 8.23 and 8.27 imply that p also atomically delivers m^* by time c. This is a contradiction. □

Lemma 8.30 *If p and q atomically deliver messages m and m', then p atomically delivers m before m' if and only if q atomically delivers m before m'.*

Lemma 8.31 *If a correct process p atomically delivers a message m, then for all correct processes q, $\overrightarrow{\text{DLVD}}_{bc(m)}^{ts(m)} | G_{q,bc(m)} = \overrightarrow{\text{DLVD}}_q^{ts(m)} | G_{q,bc(m)}$.*

Proof: Suppose for contradiction that m is the earliest message such that a correct process p atomically delivers m, and $\overrightarrow{\text{DLVD}}_{bc(m)}^{ts(m)} | G_{q,bc(m)} \neq \overrightarrow{\text{DLVD}}_q^{ts(m)} | G_{q,bc(m)}$ for some correct process q. Let $m = (x, c, g, data)$ and $b = (x, c, g, (data, \overrightarrow{\text{DLVD}}_x^c))$. Lemma 8.30 implies that there is a group $g' \in G_{q,x}$ such that $\overrightarrow{\text{DLVD}}_x^c | g' \neq \overrightarrow{\text{DLVD}}_q^c | g'$.

In particular, Lemmas 8.23 and 8.27 imply that there is a message m^*, such that $group(m^*) = g'$, $m^* \in \overrightarrow{\text{DLVD}}^c_q$, $x \in g'$, and $m^* \notin \overrightarrow{\text{DLVD}}^c_x$.

Since p atomically delivers m, p also uniform-delivers b; *i.e.*, $b \in B^{c+\Delta_g}_{p,g}$. The agreement property satisfied by uniform multicast implies that for all correct processes r in g, $b \in B^{c+\Delta_g}_{r,g}$, and hence for all processes s in g, $allow_r(s, c+\Delta_g, g)$ is true.

Since $g' \in G_{q,x}$, and x belongs to both groups g and g', $g \bigcap g' \neq \phi$. By assumption $n_{g,g'} > f_{g,g'}$ and hence there is at least one correct process in $g \bigcap g'$.

Let r be any correct process in $g \bigcap g'$. Since q and r are correct, and both belong to group g', $\overrightarrow{\text{DLVD}}^c_r|g' = \overrightarrow{\text{DLVD}}^c_q|g'$; thus the message $m^* \in \overrightarrow{\text{DLVD}}^c_r$.

Since r is a correct process in g, $b \in B^{c+\Delta_g}_{r,g}$, and for all processes s in g, $allow_r(s, c+\Delta_g, g)$ is true (as we showed above). Since there is a message $m^* \in \overrightarrow{\text{DLVD}}^c_r$, $x \in group(m^*)$, and $m^* \notin \overrightarrow{\text{DLVD}}^c_x$, process r strong-multicasts the message $(r, c+\Delta_g, g, (b, NAK))$.

Since p and r are both correct, and $allow_r(r, c+\Delta_g, g)$ is true, Lemma 8.25 states that $allow_p(r, c+\Delta_g, g)$ is true, and hence process p strong-delivers the message $(r, c+\Delta_g, g, (b, NAK))$. Therefore, p does not atomically deliver the message m, a contradiction. □

Corollary 8.32 *If a correct process p atomically delivers a message m, then for all correct processes q,* $\overrightarrow{\text{DLVD}}^{ts(m)}_{bc(m)}|G_{q,bc(m)} \preceq \overrightarrow{\text{DLVD}}_q|G_{q,bc(m)}$.

Lemma 8.33 *If p atomically multicasts a message m to a group g before atomically multicasting m' to g, and any correct process q atomically delivers m', then q atomically delivers m before m'.*

Proof: Let p, g, m and m' be as in the statement of the lemma. Suppose a correct process q in g atomically delivers m'. From the protocol, q also atomically delivers all of p's previous multicasts to g by the time it delivers m'. In particular, q atomically delivers m before m'. □

Theorem 8.34 *Suppose that for all groups g and g', if* $g \bigcap g' \neq \phi$, *then* $n_{g,g'} > f_{g,g'}$, *and that for all groups g,* $n_g > 2f_g$ *and* $f_g \geq 1$. *The protocol in Figure 8.7 is a pairwise atomic multicast protocol that prevents the contamination of correct processes, tolerates omission failures and has a latency of* $2\Delta_g$ *for each group g.*

Proof: Lemmas 8.28 and 8.29 imply that the protocol is a pairwise atomic multicast protocol. From Corollary 8.32 and 8.33, we conclude that the protocol prevents the contamination of correct processes. □

8.5.3 Lower bounds

Recall that we require that the processing of a multicast targeted to a particular group does not involve any process that does not belong to that group. We formalize this as follows.

A multicast protocol has the *group locality* property in a system S, if there is a constant L such that, for all executions of the protocol in system S, the following conditions hold. Let Q_p denote the sequence of messages, ordered in timestamp order, that are multicast to any group to which p belongs.

- If Q_p is empty, then p does not send or receive any messages during the execution.
- Suppose $Q_p = \langle m_1, m_2, \ldots, m_j, \ldots \rangle$. Then:
  - p does not send or receive any messages until time $ts(m_1)$.
  - if $|Q_p| = k$, then p does not send or receive any messages after time $ts(m_k) + L$.
  - for all i, $1 \leq i < |Q_p|$, for all c, $ts(m_i) + L < c < ts(m_{i+1})$, p does not send or receive any messages at c.

Recall that in Chapter 7, we presented an atomic broadcast protocol that prevented contamination, and required $n > f$. We show that all pairwise atomic multicast protocols that prevent contamination and have the group locality property in a system subject to omission failures require $n_g > 2f_g$ in every group g.

Theorem 8.35 *In systems subject to omission failures, any pairwise atomic multicast that prevents the contamination of correct processes and has the group locality property requires $n_g > 2f_g$ for all groups g.*

Proof: The proof is by contradiction. Suppose there is a pairwise atomic multicast protocol that prevents contamination in a system subject to omission failures even if for some group g, $n_g \leq 2f_g$.

Consider the system shown in Figure 8.8 with two groups g and g', where $g \subset g'$ and $n_g = 2f_g$. Suppose g is partitioned into two disjoint subsets, A and B, $|A| = |B| = f_g$. Suppose $p \in A$ and $q \in B$. Suppose r is some correct process that is in group g', but not in group g.

The proof considers four different possible executions of the hypothetical pairwise atomic multicast protocol. In all the executions, we assume that no message is received in less than λ time; thus either a message is received exactly λ time after it was sent, or it is never received. We also assume that no process halts.

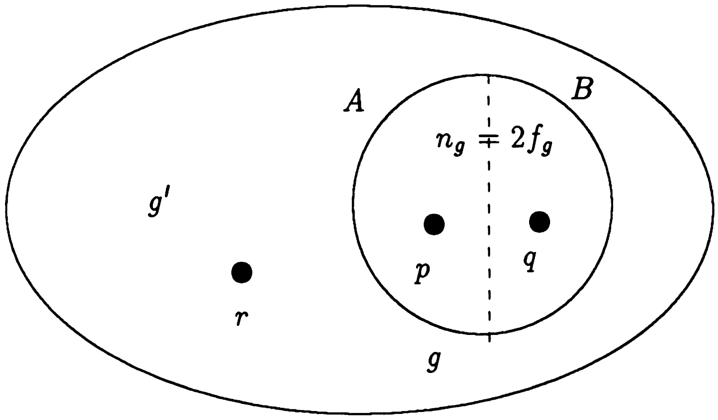


Figure 8.8: Illustration of the lower bound on fault tolerance

In the first two executions (Scenarios 1 and 2), we assume that if a message is supposed to be sent by a process in A to another process in A, then the message is actually sent and is received. Similarly, if a message is supposed to be sent by a process in B to another process in B, then the message is actually sent and is received.

Both Scenarios 1 and 2 are executions of the protocol in which no messages are atomically multicast until time c. Suppose that at time c, p atomically multicasts m_p to group g and that q atomically multicasts m_q to group g. Suppose that no other messages are multicast.

First note that there are no multicasts to group g'. Hence, the group locality property implies that *any* process in g' that is not also in g (for example, process r) does not send or receive any message during the entire execution.

Scenario 1: Suppose that the processes in A are faulty and those in B are correct. Suppose that whenever a process $a \in A$ sends a message to a process $b \in B$, a performs a send omission failure, and whenever b sends a message to a, a performs a receive omission failure.

Since the processes in B are correct, it is clear that the processes in B must atomically deliver m_q. Let c_B denote the latest time at some process in B delivers message m_q.

Since there is no communication between the processes in A and B, it is clear that the processes in A do not atomically deliver m_q.

Scenario 2: Now suppose that the processes in A are correct and those in B are faulty. Suppose that whenever a process $a \in A$ sends a message to a process $b \in B$, b

performs a receive omission failure, and whenever b sends a message to a, b performs a send omission failure.

Since the processes in A are correct, it is clear that the processes in A must atomically deliver m_p. Let c_A denote the latest time at which some process in A delivers message m_p.

Since there is no communication between the processes in A and B, it is clear that the processes in B do not atomically deliver m_p.

Scenarios 1 and 2 are indistinguishable to processes p and q. To see this, recall that the group locality property implies that r neither sends nor receives any messages in both scenarios. Thus, p and q are only permitted to communicate with the other processes in group g. Since, there are f_g processes in each of the two sets A and B, and there is no communication between processes in A and B, we conclude that Scenarios 1 and 2 are indistinguishable to processes p and q. Therefore, by time $max(c_A, c_B)$ in both scenarios, p atomically delivers m_p and does not atomically deliver m_q, and q atomically delivers m_q and does not atomically deliver m_p.

Now consider two new scenarios as follows. Scenarios 3 and 4 are identical to Scenarios 1 and 2 respectively until time $c^* = max(c + L + \lambda, c_A, c_B) + 1$. However, in both Scenarios 3 and 4 p atomically multicasts a message m^* to group g' at time $c^* + 1$, and no further messages are multicast. Furthermore from time $c^* + 1$ onward, no process performs a send or receive omission failure.

Since c_A and c_B are both less than c^* ($c^* > max(c_A, c_B)$), and Scenarios 1 and 2 are identical to Scenarios 3 and 4 respectively until time c^*, the processes in A deliver m_p by time c^* and omit m_q, and the processes in B deliver m_q by time c^* and omit m_p in the latter two scenarios.

In Scenarios 3 and 4, since the first message multicast to group g' is timestamped $c^* + 1$, the group locality property implies that r does not send or receive any message until time $c^* + 1$. Since $c^* > c + L + \lambda$, r does receive any message that a process in g sent by time $c + L$. The group locality property also implies that no process in g sends or receives a message between $c + L$ and $c^* + 1$. Since there are no omission failures from time $c^* + 1$ onward, we conclude that process r cannot distinguish between Scenarios 3 and 4.

Since q is correct in Scenario 3, and p delivered m_p and not m_q, and q delivered m_q and not m_p, it is clear that p is inconsistent at time $c^* + 1$. Thus, by atomically delivering the message m^*, the correct process r would become contaminated; hence r does not atomically deliver m^*.

Since Scenarios 3 and 4 are indistinguishable to r, r does not deliver m^* in Scenario 4. However, in Scenario 4, the validity and agreement properties of the pairwise atomic multicast imply that r must atomically deliver m^*. This is the required contradiction. $\square$

8.6 Pairwise atomic multicast with no inconsistency

Our protocol that prevents inconsistency with respect to pairwise atomic multicasts is derived from the first protocol that prevents contamination.

To atomically multicast a message $m = (p, c, g, data)$ to group g at time c, process p *multicasts* $(basic, m)$ to g.

If process q *delivers* $(basic, m)$, by time $c + \Delta_g$, then q sends the message $(q, m, AllM_q^c)$ to all in g, where $AllM_q^c$ is the set of all messages that q has basic-delivered by time c. Informally, as in the simple atomic broadcast protocol that prevented inconsistency (see Chapter 5), the set $AllM_q^c$ contains all the messages that q intends to atomically deliver before it atomically delivers m.

At time $c + \Delta_g + \lambda$, q checks to see if, for *every* group g' to which it belongs, a majority of processes in the intersection of g and g' agree with it on the sequence of messages that should be atomically delivered in g' before m is atomically delivered. If so, then q atomically delivers $(p, c, g, data)$; otherwise, q is faulty, and to avoid becoming inconsistent, q halts.

The protocol is formally presented in Figure 8.9. The proof of correctness is similar to the proofs of correctness of our first pairwise atomic protocol that prevented contamination (Figure 8.5) and our non-optimal atomic broadcast protocol that ensured consistency (Figure 7.3), and hence is omitted.

Theorem 8.36 *Suppose for all groups g and g', if $g \bigcap g' \neq \phi$, then $n_{g,g'} > 2f_{g,g'}$. The protocol in Figure 8.9 is a pairwise atomic multicast protocol that prevents inconsistency, tolerates omission failures and has a latency of $\Delta_g + \lambda$ in each group g.*

/* p atomically multicasts a message $m = (p, c, g, data)$ at time c to group g */

To multicast $(pairwise\ atomic, m = (p, c, g, data))$

p **multicasts** $(basic, m)$

/* p delivers basic multicast messages at c */

$M_p^c \equiv \{m \mid ts(m) = c - \Delta_{group(b)},\ p$ **delivered** $(basic, m)\}$

$AllM_p^c = \bigcup_{c' \leq c} M_p^c$

/* p sends messages at c */

For all $m \in M_p^c$ p **sends** $(p, m, AllM_p^c)$ to all in $group(m)$

/* p receives all messages at c */

For all $m \in M_p^{c-\lambda}$ **for all** groups $g \in belongs(p)$

$ack_{p,m,g} \equiv \{q \mid p$ **received** $(q, m, AllM_q^{c-\lambda})$,

and $\forall m^*, group(m^*) = g, m^* \in AllM_q^{c-\lambda} \iff m^* \in AllM_p^{c-\lambda}\}$

/* p atomically delivers messages at c */

For all $m \in M_p^{c-\lambda}$, $g^* \equiv group(m)$

if for any group $g \in belongs(p)$: $|ack_{p,m,g}| < n_{g,g^*} - f_{g,g^*}$ **then halt**

$D \equiv$ **sort** $M_p^{c-\lambda}$ in timestamp order

p **delivers** $(pairwise\ atomic, D)$

Figure 8.9: Pairwise atomic multicast protocol that prevents inconsistency
Executed by process p at time c

Appendix A

Other Fault-tolerant Broadcasts

In this appendix, we present sufficient conditions to prevent inconsistency and/or contamination with respect to *reliable broadcast*, *causal broadcast* and *causal atomic broadcast*. We only consider the prevention of D-inconsistency, VBD-inconsistency and BD-contamination. Note that as with atomic broadcast, BD-consistency cannot be ensured with these broadcasts.

A.1 Reliable broadcast

Informally, reliable broadcast requires that all correct processes deliver the same set of messages, and this set must include all messages broadcast by correct processes. Formally, *reliable broadcast* is defined to be the conjunction of the validity, agreement and integrity properties (defined in Chapter 2 and repeated in Chapter 5).

We use the following notation (defined in Chapter 5 and repeated below). Suppose $B = (\beta, \delta)$ is a broadcast history. $\text{BCAST}_p^c(B)$ denotes the set of messages that process p broadcasts by time c in B; formally, $\text{BCAST}_p^c(B) = \bigcup_{c' \leq c} \beta(p, c')$. $\overrightarrow{\text{DLVD}}_p^c(B)$ denotes the *sequence* of messages that p delivers by time c (defined in Chapter 2). $\text{DLVD}_p^c(B)$ denotes the *set* of messages that p delivers by time c; $\text{DLVD}_p^c(B) = \{m \mid m \in \overrightarrow{\text{DLVD}}_p^c(B)\}$. The superscript c is omitted when referring to all the messages broadcast or delivered during a complete history.

Lemma A.1 *Let $B = (\beta, \delta) \in \mathcal{B}$ and F be a failure set, such that B satisfies reliable broadcast with respect to F.*

- *All correct processes deliver the same set of messages:*
  $\forall p \notin F, \forall q \notin F : \text{DLVD}_p(B) = \text{DLVD}_q(B)$.
- *If a correct process broadcasts a message m, then all correct processes eventually deliver m:*

$\forall p \notin F, \forall q \notin F:$ $\text{BCAST}_p(B) \subseteq \text{DLVD}_q(B)$.

Proof: Directly from the definition of reliable broadcast. □

Suppose B is a broadcast history and F is a failure set such that B satisfies reliable broadcast with respect to F. $\overrightarrow{\text{DLVD}}_{\overline{F}}(B)$ and $\text{DLVD}_{\overline{F}}(B)$ respectively denote the sequence of messages and the set of messages that the correct processes deliver during B.

Lemma A.2 *Let $A \in \mathcal{A}$, B be the broadcast history of A, and F be such that B satisfies reliable broadcast with respect to F. If for all processes p, for all times c, $\text{DLVD}_p^c(B) \subseteq \text{DLVD}_{\overline{F}}(B)$, then A is a* D*-consistent history with respect to reliable broadcast and F.*

Note that although an application history A may be a D-consistent history with respect to reliable broadcast and a failure set F, there may be a faulty process p, and a time c, such that $\text{DLVD}_p^c(B) \not\subseteq \text{DLVD}_{\overline{F}}(B)$. Informally, this occurs when p delivers a message at time c that no correct process delivers, and then halts *before* changing state at time c.

Lemma A.3 *Let $A \in \mathcal{A}$, B be the broadcast history of A, and F be such that B satisfies reliable broadcast with respect to F. If B satisfies the fifo order property with respect to F, and for all processes p, for all times c, $\text{DLVD}_p^c(B) \subseteq \text{DLVD}_{\overline{F}}(B)$, then A is a* VBD*-consistent history with respect to reliable broadcast and F.*

Lemma A.4 *Let $B = (\beta, \delta) \in \mathcal{B}$ and F be such that B satisfies reliable broadcast with respect to F. Suppose B satisfies fifo order with respect to F. A correct process p is* BD*-contamination-free until c with respect to reliable broadcast and F if and only if:*

- *if p delivers any message m at or before time c, then $\text{DLVD}_{bc(m)}^{ts(m)}(B) \subseteq \text{DLVD}_p(B)$.*

A.2 Causal broadcast

Reliable broadcast does not impose any requirement on the order of message delivery; thus, if the broadcast of a message m *causally precedes* the broadcast of a message m' [Lam78], a correct process may deliver m before m' even though another correct process delivers m' before m. We now define causal broadcast, which orders message delivery in a way that is consistent with the causal precedence relation. That is, if the broadcast of a message m causally precedes the broadcast of a message m', and a correct process delivers both m and m', then it delivers m before m'.

Causal broadcasts are sufficient for many applications [Sch88], and they are central to many experimental systems such as *Isis* [BCJ$^+$90], the *Lazy Replication* system [LLS90], and *Psync* [PBS89].

Formally, a causal broadcast is defined to be the conjunction of the validity, agreement and integrity properties, as well as the *causal order* property defined below for a broadcast history B and a failure set F:

- *Causal order:* For all correct processes p, if p delivers messages m and m', and $(bc(m), ts(m)) \rightarrow_B (bc(m'), ts(m'))$, then p delivers m before m'.

Lemma A.5 *Let $A \in \mathcal{A}$, B be the broadcast history of A, and F be such that B satisfies causal broadcast with respect to F. If for all processes p, for all times c:*

- $\text{DLVD}_p^c(B) \subseteq \text{DLVD}_{\overline{F}}(B)$, *and*
- *if message $m' \in \overrightarrow{\text{DLVD}}_p^c(B)$, and there is a message m such that $(bc(m), ts(m)) \rightarrow_B (bc(m'), ts(m'))$, then p delivers m before m'.*

then A is a VBD*-consistent history with respect to causal broadcast and F.*

Lemma A.6 *Let $B = (\beta, \delta) \in \mathcal{B}$ and F be such that B satisfies causal broadcast with respect to F. A correct process p is* BD*-contamination-free until c with respect to causal broadcast and F if and only if:*

- *if p delivers any message $m = (q, c', -)$ at or before time c, then:*
  - $\text{DLVD}_q^{c'}(B) \subseteq \text{DLVD}_p(B)$, *and*
  - *if $m' \in \overrightarrow{\text{DLVD}}_q^{c'}(B)$, and there is a message $m'' \in \text{DLVD}_p(B)$ such that $(bc(m''), ts(m'')) \rightarrow_B (q, c')$, then q delivers m'' before m'.*

A.3 Causal atomic broadcast

Neither atomic broadcast nor causal broadcast is a more stringent specification than the other. In particular, ensuring that atomic broadcast's total order property is satisfied is not equivalent to ensuring that causal broadcast's causal order property is satisfied. *Causal atomic broadcast* is a fault-tolerant broadcast that is stronger than both atomic broadcast and causal broadcast; formally, it is defined to be the conjunction of the validity, agreement, integrity, total order and causal order properties.

Lemma A.7 *Let $A \in \mathcal{A}$, B be the broadcast history of A, and F be such that B satisfies causal atomic broadcast with respect to F. If for all processes p, for all times c:*

- $\overrightarrow{\text{DLVD}}_p^c(B) \preceq \overrightarrow{\text{DLVD}}_{\overline{F}}(B)$ *then A is a* VBD*-consistent history (and hence also* D*-consistent history) with respect to causal atomic broadcast and F.*

Lemma A.8 *Let $B = (\beta, \delta) \in \mathcal{B}$ and F be such that B satisfies causal atomic broadcast with respect to F. A correct process p is* BD-*contamination-free until c with respect to causal broadcast and F if and only if:*

- *if p delivers any message m at or before time c, then $\overrightarrow{\mathrm{DLVD}}^{ts(m)}_{bc(m)}(B) \preceq \overrightarrow{\mathrm{DLVD}}_p(B)$.*

Bibliography

[BCJ+90] Kenneth P. Birman, Robert Cooper, Thomas A. Joseph, Kenneth P. Kane, and Frank Bernhard Schmuck. *ISIS - A Distributed Programming Environment*, June 1990.

[BGT90] Navin Budhiraja, Ajei Gopal, and Sam Toueg. Early-stopping distributed bidding and applications. In *Proceedings of the Fourth International Workshop on Distributed Algorithms*. Springer-Verlag, September 1990. In press.

[BJ87] Kenneth P. Birman and Thomas A. Joseph. Reliable communication in the presence of failures. *ACM Transactions on Computer Systems*, 5(1):47–76, February 1987.

[BSS90] Kenneth P. Birman, Andre Schiper, and Pat Stephenson. Fast causal multicast. Technical Report 90-1105, Department of Computer Science, Cornell University, April 1990.

[CASD85] Flaviu Cristian, Houtan Aghili, H. Raymond Strong, and Danny Dolev. Atomic broadcast: From simple message diffusion to Byzantine agreement. In *Proceedings of the Fifteenth International Symposium on Fault-Tolerant Computing*, pages 200–206, June 1985. A revised version appears as IBM Research Laboratory Technical Report RJ5244 (April 1989).

[Cri87] Flaviu Cristian. Issues in the design of highly available computing services. In *Annual Symposium of the Canadian Information Processing Society*, pages 9–16, July 1987. Also IBM Research Report RJ5856, July 1987.

[DRS86] Danny Dolev, Rüdiger Reischuk, and H. Raymond Strong. Early stopping in Byzantine agreement. Technical Report RJ5406, IBM Research Laboratory, December 1986.

[FLP85] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, April 1985.

[GSTC90] Ajei Gopal, Ray Strong, Sam Toueg, and Flaviu Cristian. Early-delivery atomic broadcast. In *Proceedings of the Ninth ACM Symposium on Principles of Distributed Computing*, pages 297–310, August 1990.

[GT89] Ajei Gopal and Sam Toueg. Reliable broadcast in synchronous and asynchronous environments (preliminary version). In J.-C. Bermond and M. Raynal, editors, *Proceedings of the Third International Workshop on Distributed Algorithms*, volume 392 of *Lecture Notes on Computer Science*, pages 110–123. Springer-Verlag, September 1989.

[GT91] Ajei Gopal and Sam Toueg. Inconsistency and contamination (preliminary version). In *Proceedings of the Tenth ACM Symposium on Principles of Distributed Computing*, pages 257–272, August 1991.

[Had84] Vassos Hadzilacos. *Issues of Fault Tolerance in Concurrent Computations.* Ph.D. dissertation, Harvard University, June 1984. Department of Computer Science Technical Report 11-84.

[Lam78] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

[Lam84] Leslie Lamport. Using time instead of timeout for fault-tolerant distributed systems. *ACM Transactions on Programming Languages and Systems*, 6(2):254–280, April 1984.

[LLS90] Rivka Ladin, Barbara Liskov, and Liuba Shrira. Lazy replication: Exploiting the semantics of distributed services. In *Proceedings of the Ninth ACM Symposium on Principles of Distributed Computing*, pages 43–58, August 1990.

[LSP82] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, July 1982.

[NT90] Gil Neiger and Sam Toueg. Automatically increasing the fault-tolerance of distributed algorithms. *Journal of Algorithms*, 11(3):374–419, September 1990.

[PBS89] L. L. Peterson, N. C. Bucholz, and Richard D. Schlichting. Preserving and using context information in interprocess communication. In *ACM Transactions on computer systems 7,3*, pages 217–246, August 1989.

[Per85] Kenneth J. Perry. *Early Stopping Protocols for Fault-Tolerant Distributed Agreement.* Ph.D. dissertation, Cornell University, February 1985. Department of Computer Science Technical Report 85-662.

[PT84] Kenneth J. Perry and Sam Toueg. An authenticated Byzantine generals algorithm with early stopping. Technical Report 84-620, Department of Computer Science, Cornell University, June 1984.

[Sch86] Fred B. Schneider. The state machine approach: a tutorial. Technical Report 86-800, Department of Computer Science, Cornell University, December 1986. Revised June 1987.

[Sch88] Frank Bernhard Schmuck. *The Use of Efficient Broadcast Protocols in Asynchronous Distributed Systems.* Ph.D. dissertation, Cornell University, August 1988. Department of Computer Science Technical Report 88-928.