Chapter 9
State Machine Replication with Byzantine Faults

Christian Cachin

Abstract This chapter gives an introduction to protocols for state-machine replication in groups that are connected by asynchronous networks and whose members are subject to arbitrary or “Byzantine” faults. It explains the principles of such protocols and covers the following topics: broadcast primitives, distributed cryptosystems, randomized Byzantine consensus protocols, and atomic broadcast protocols.

9.1 Introduction

Coordinating a group of replicas to deliver a service, while some of them are actively trying to prevent the coordination effort, is a fascinating topic. It stands at the heart of Pease, Shostak, and Lamport’s classic work [24] on reaching agreement in the presence of faults, which ignited an impressive flow of papers elaborating on this problem over the last 30 years.

In this chapter, we survey protocols to replicate a state machine in an asynchronous network over a group of $n$ parties or replicas, of which up to $t$ are subject to so-called Byzantine faults. No assumptions about the behavior of the faulty parties are made; they may deviate arbitrarily from the protocol, as if corrupted by a malicious adversary. The key mechanism for replicating a deterministic service among the group is a protocol for the task of atomic broadcast [16,31,32]. It guarantees that every correct party in the group receives the same sequence of requests from the clients. This approach allows to build highly resilient and intrusion-tolerant services on the Internet, as discussed in Chapter 8.

The model considered here is motivated by practice. The parties are connected pairwise by reliable authenticated channels. Protocols may use cryptographic methods, such as public-key cryptosystems and digital signatures. A trusted entity takes care of initially generating and distributing private keys, public keys, and certificates, such that every party can verify signatures by all other parties, for example. The system is asynchronous: there are no bounds on the delivery time of messages and no synchronized clocks. This is an important aspect because systems whose cor-
rectness relies on timing assumptions are vulnerable to attackers that simply slow down the correct parties or delay the messages sent between them.

The chapter is organized as follows. We first introduce some building blocks for atomic broadcast; they consist of two broadcast primitives, distributed cryptosystems, and randomized Byzantine consensus protocols. Then we present the structure of some recent asynchronous atomic broadcast protocols. Finally, we illustrate some issues with service replication that arise specifically in the presence of Byzantine faults. We focus on the asynchronous model and leave out many other protocols that have been formulated for synchronous networks.

9.2 Building Blocks

9.2.1 Broadcast Primitives

We present two broadcast primitives, which are found in one way or other in all consensus and atomic broadcast protocols tolerating Byzantine faults. As such protocols usually invoke multiple instances of a broadcast primitive, every message is tagged by an identifier of the instance in practice (and where applicable, the identifier is also included in every cryptographic operation).

Every broadcast instance has a designated sender, which broadcasts a request \( m \) to the group at the start of the protocol. All parties should later deliver \( m \), though termination is not guaranteed with a faulty sender. To simplify matters, we assume that the sender is a member of the group (i.e., that requests from clients to the service are relayed through one replica) and that all requests are unique.

**Consistent Broadcast**

Consider a group of \( n \) parties \( P_1, \ldots, P_n \). In consistent broadcast, a designated sender \( P_s \) first executes \( c\text{-broadcast} \) with request \( m \) and thereby starts the protocol. All parties terminate the protocol by executing \( c\text{-deliver} \) with request \( m \). Consistent broadcast ensures only that the delivered request is the same for all receivers. In particular, it does not guarantee that every party delivers a request with a faulty sender.

The following definition is implicit in the work of Bracha and Toueg \[37, 2\] but has been formulated more recently \[3\] to be in line with the corresponding notions for systems with crash failures \[12\]. Recall that it models only one instance of consistent broadcast.

**Definition 9.1 (Consistent Broadcast).** A protocol for consistent broadcast satisfies:

Validity: If a correct sender \( P_s \) \( c\text{-broadcasts} \) \( m \), then all correct parties eventually \( c\text{-deliver} \) \( m \).

Consistency: If a correct party \( c\text{-delivers} \) \( m \) and another correct party \( c\text{-delivers} \) \( m' \), then \( m = m' \).

Integrity: Every correct party \( c\text{-delivers} \) at most one request. Moreover, if the sender \( P_s \) is correct, then the request was previously \( c\text{-broadcast} \) by \( P_s \).