Making Distributed Applications Robust*

Chi Ho[1], Danny Dolev[2], Robbert van Renesse[1]

[1]Dept. of Computer Science
Cornell University, Ithaca, NY
{chho, rvr}@cs.cornell.edu

[2]School of Engineering and Computer Science
The Hebrew University of Jerusalem, Israel
dolev@cs.huji.ac.il

Abstract. We present a novel translation of systems that are tolerant of crash failures to systems that are tolerant of Byzantine failures in an asynchronous environment, making weaker assumptions than previous approaches. In particular, we assume little about how the application is coded. The translation exploits an extension of the Srikanth-Toueg protocol, supporting ordering in addition to authentication and persistent delivery. We illustrate the approach by synthesizing a version of the Castro and Liskov Practical Byzantine Replication protocol from the Oki and Liskov Viewstamped Replication protocol.

Keywords: Byzantine Fault Tolerance, Ordered Broadcast

1 Introduction

Developing applications that span multiple administrative domains is difficult if the environment is asynchronous and machines may exhibit arbitrary failures. Yet, this is a problem that many software developers face today. While we know how to build replicated data stores that tolerate Byzantine behavior (*e.g.*, [4]), most applications go well beyond providing a data store. Tools like Byzantine consensus may help developing such applications, but most software developers find dealing with arbitrary failures extremely challenging. They often make simplifying assumptions like a crash failure model, relying on careful monitoring to detect and fix problems that occur when such assumptions are violated.

We are interested in techniques that automatically transform crash-tolerant applications into Byzantine-tolerant applications that do not require careful monitoring and repair.

This paper makes the following contributions. First we present a novel ordered broadcast protocol that we will use as a building block. The protocol is an extension of the Srikanth and Toueg authenticated broadcast protocol often used in Byzantine consensus protocols [11], adding consistent ordering for messages from the same sender even in the face of Byzantine behavior. Second,

* This work is supported by AFOSR grants FA9550-07-1-0304, FA8750-06-2-0060, FA9550-06-1-0019, FA9550-06-1-0244, DHS award 2006-CS-001-000001 (I3P), and the National Science Foundation under grant 0424422, and ISF, ISOC, and CCR. Any opinions expressed in this publications are those of the authors and do not necessarily reflect the views of the funding agencies.

we present a new way of translating a distributed application that is tolerant of crash failures into one that tolerates the same number of Byzantine failures, while imposing fewer restrictions on how the application is constructed than previous approaches. Third, we show how a version of the Castro and Liskov Practical Byzantine Replication protocol [4] can be derived from the Oki and Liskov Viewstamped Replication protocol [10] using our translation technique, something not possible with previous approaches.

We present background in Sect. 2. After describing a system model in Sect. 3, we introduce three mechanisms used for translation: Authenticated Reliable broadcast (Sect. 4), Ordered Authenticast Reliable broadcast (Sect. 5), and the translation mechanism itself (Sect. 6). Correctness proofs for these appear in the appendix. In Sect. 7 we demonstrate the translation mechanism.

2 Background

The idea of automatically translating crash-tolerant systems into Byzantine systems can be traced back to the mid-eighties. Gabriel Bracha used a translation similar to ours to generate a consensus protocol tolerant of t Byzantine failures out of $3t+1$ hosts [3]. Brian Coan also presents a translation [6] that is similar to Bracha's. The most important restriction in these approaches is that input protocols are required to have a specific style of execution, and in particular they have to be round-based with each participant awaiting the receipt of $n-t$ messages before starting a new round. These requirements exclude, for example, protocols that designate roles to senders and receivers such as the primary role used in Viewstamped Replication [10]. Our approach makes no such assumptions, and we will demonstrate our approach for Viewstamped Replication.

Toueg, Neiger and Bazzi worked on an extension of Bracha's and Coan's approaches for translation of synchronous systems [9, 2, 1]. Their approach takes advantage of synchrony to detect faulty hosts and eliminate them from the protocol. The extension can be applied to our scheme as well.

Most recently, Mpoeleng et al. [8] present a translation that is intended for synchronous systems, and transforms Byzantine faults to so-called *signal-on-failure* faults. They replace each host with a pair, and assume only one of the hosts in each pair may fail. They require $4t+2$ hosts, but the system may break with as few as two failures no matter how large t is chosen.

3 System Model

In order to be precise we present a simple model to talk about machines, processes, and networks. The model consists of *agents* and *links*. An agent is an active entity that maintains state, receives messages on incoming links, performs some processing based on this input and its state, possibly updating its state and producing output messages on outgoing links.

Links are abstract unidirectional FIFO channels between two agents. Agents can interact across links only. In particular, an agent can *enqueue* a message on one of its outgoing links, and it can *dequeue* messages from one of its incoming links (assuming a message is available there).

We use agents and links to model various activities and interactions. Processes that run on hosts are agents, but the network is also an agent—one that forwards messages from its incoming links to its outgoing links according to some policy. Agents are named by lower-case Greek letters $\alpha, \beta, ...$ For agents that are processes, we will use subscripts on names to denote which hosts they run on. For example, β_i is an agent that runs on host h_i.

Hosts are containers for agents, and they are also the unit of failure. Hosts are either *honest*, executing programs as specified, or *Byzantine* [7], exhibiting arbitrary behavior. We also use the terms *correct* and *faulty*, but not as alternatives to honest and Byzantine. A correct host is honest and always eventually makes progress. A faulty host is a Byzantine host or an honest host that has crashed or will eventually crash. Honest and Byzantine are mutually exclusive, as are correct and faulty. However, a host can be both honest and faulty.

We do not assume timing bounds on execution of agents. Latency in the network is modeled as execution delay in a network agent. Note that this prevents hosts from accurately detecting crashes of other hosts.

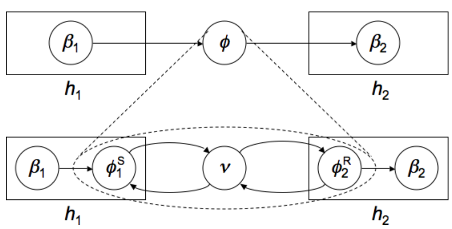


Fig. 1. An agent model and a refinement.

Figure 1 depicts an example of an agent model and a *refinement.* Agents are represented by circles, links by arrows, and hosts by rectangles. The top half models two application agents β_1 and β_2 running on two hosts h_1 and h_2 communicating using a FIFO network agent ϕ. The bottom half refines the FIFO network using an unreliable network agent ν and two protocol agents $\phi_1^{\mathbf{S}}$ and $\phi_2^{\mathbf{R}}$ that implement ordering and retransmission using sequence numbers, timers, and acknowledgment messages. This kind of refinement will be a common theme throughout this paper.

4 The ARcast Mechanism

The first mechanism we present is Authenticated Reliable broadcast (ARcast). This broadcast mechanism was suggested by Srikanth and Toueg, and they present an implementation that does not require digital signatures in [11]. Their implementation requires $n > 3t$. As shown below, it is also possible to develop an implementation that uses digital signatures, in which case n only has to be larger than $2t$.

4.1 ARcast Definition

Assume $\beta_i, ...$ are agents communicating using ARcast on hosts $h_i, ...$ Then ARcast provides the following properties:

1. *bc-Persistence.* If two hosts h_i and h_j are correct, and β_i sends a message m, then β_j delivers m from β_i;
2. *bc-Relay.* If h_i is honest and h_j is correct, and β_i delivers m from β_k, then β_j delivers m from β_k (host h_k is not necessarily correct);
3. *bc-Authenticity.* If two hosts h_i and h_j are honest and β_i does not send m, then β_j does not deliver m from β_i.

Informally, ARcast ensures that a message is reliably delivered to all correct receivers in case the sender is correct (bc-Persistence) or in case another honest receiver has delivered the message already (bc-Relay). Moreover, a Byzantine host cannot forge messages from an honest host (bc-Authenticity).

4.2 ARcast Implementation

We assume there is a single sender β_i on h_i. We model ARcast as a network agent ξ_i, which we refine by replacing it with the following agents (see Fig. 2):

- $\xi_i^{\mathbf{S}}$ *sender* agent that is in charge of the sending side of the ARcast mechanism;
- $\xi_*^{\mathbf{R}}$ *receiver* agents that are in charge of the receive side;
- ϕ *FIFO network* agent that provides point-to-point authenticated FIFO communication between agents.

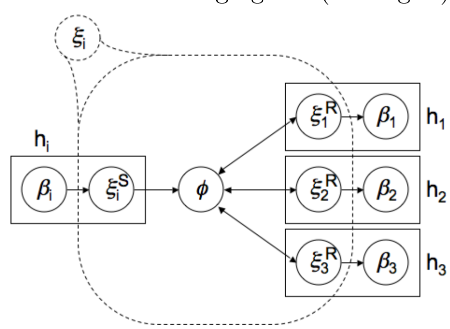


Fig. 2. Architecture of the ARcast implementation if the sender is on host h_i.

The mechanism has to be instantiated for each sender. The sending host h_i runs the ARcast sender agent $\xi_i^{\mathbf{S}}$. Each receiving host h_j runs a *receiver* agent $\xi_j^{\mathbf{R}}$. There have to be at least $2t+1$ receiving hosts, one of which may be h_i. When $\xi_i^{\mathbf{S}}$ wants to ARcast a message m, it sends $\langle \mathsf{echo}\ m, i\rangle_i$, signed by h_i using its public key signature, to all receivers. A receiver that receives such an `echo` message for the first time forwards it to all receivers. On receipt of $t+1$ of these correctly signed echoes for the same m from different receivers (it can count an echo from itself), a receiver delivers m from i.

Due to space considerations, we omit the (simple) correctness proof.

5 The OARcast Mechanism

ARcast does not provide any ordering. Even messages from a correct sender may be delivered in different orders at different receivers. Next we introduce a broadcast mechanism that is like ARcast, but adds delivery order for messages sent by either honest or Byzantine hosts.

5.1 OARcast Definition

OARcast provides, in addition to the ARcast properties, the following:

4. *bc-FIFO.* If two hosts h_i and h_j are honest and β_i sends m_1 before m_2, and β_j delivers m_1 and m_2 from β_i, then β_j delivers m_1 before m_2;
5. *bc-Ordering.* If two hosts h_i and h_j are honest and β_i and β_j both deliver m_1 from β_k and m_2 from β_k, then they do so in the same order (even if h_k is Byzantine).

As a result of bc-Ordering, even a Byzantine sender cannot cause two honest receivers to deliver OARcast messages from the same source out of order. bc-FIFO ensures that messages from honest hosts are delivered in the order sent. OARcast does not guarantee any order among messages from different sources, and is thus weaker than consensus.

5.2 OARcast Implementation

We describe how OARcast may be implemented using ARcast. Again, we show the implementation for a single sender β_i on host h_i. With multiple senders, the implementation has to be instantiated for each sender separately. We refine the OARcast network agent ω_i by replacing it with the following agents (see Fig. 3):

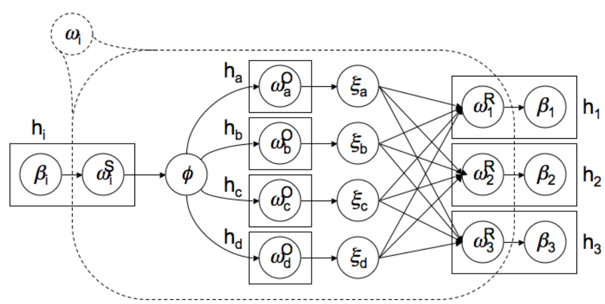


Fig. 3. Architecture of the OARcast implementation if the sender is on host h_i.

- $\omega_i^{\mathbf{S}}$ *sender* agent that is in charge of the sending side of the OARcast mechanism;
- $\omega_*^{\mathbf{O}}$ *orderer* agents that are in charge of ordering;
- $\omega_*^{\mathbf{R}}$ *receiver* agents that are in charge of the receive side;
- ϕ *FIFO network* agent that provides point-to-point authenticated FIFO communication from the sender agent to each orderer agent;
- ξ_* *ARcast network* agents each provides ARcast from a particular orderer agent to all receiver agents.

We need to run $3t+1$ orderers on separate hosts, of which no more than t may fail. A host may end up running a sender, a receiver, as well as an orderer. A receiver $\omega_j^{\mathbf{R}}$ maintains a sequence number c_j, initially 0. An orderer $\omega_k^{\mathbf{O}}$ also maintains a sequence number, t_k, initially 0.

To OARcast a message m, $\omega_i^{\mathbf{S}}$ sends m to each orderer via ϕ. When an orderer $\omega_k^{\mathbf{O}}$ receives m from $\omega_i^{\mathbf{S}}$, it ARcasts $\langle \mathsf{order}\ m, t_k, i\rangle$ to each of the receivers, and increments t_k. A receiver $\omega_j^{\mathbf{R}}$ awaits $2t+1$ messages $\langle \mathsf{order}\ m, c_j, i\rangle$ from different orderers before delivering m from $\omega_i^{\mathbf{S}}$. After doing so, the receiver increments c_j.

We prove the correctness of this implementation in Appendix A.

6 The Translation Mechanism

In this section, we describe how an *arbitrary* protocol tolerant only of crash failures can be translated into a protocol that tolerates Byzantine failures.

6.1 Definition

Below we use the terms *original* and *translated* to distinguish the system before and after translation, respectively. The original system tolerates only crash failures, while the translated system tolerates Byzantine failures as well. The original system consists of n hosts, each of which runs an *actor agent*, $\alpha_1, \ldots, \alpha_n$. Each actor α_i is a state machine that maintains a running state s^i, initially s^i_0, and, upon receiving an input message m, executes a deterministic *state transition function* F^i: $(\overline{m_o}, s^i_{c+1}) := F^i(m, s^i_c)$ where

- c indicates the number of messages that α_i has processed so far;
- s^i_c is the state of α_i before processing m;
- s^i_{c+1} is the next state of s^i_c as a result of processing m (called $F^i(m, s^i_c)$`.next`);
- $\overline{m_o}$ is a finite, possibly empty set of output messages (called $F^i(m, s^i_c)$`.output`).

The state transition functions process one input message at a time and may have no computational time bound.

Actors in the original system communicate via a FIFO network agent ϕ. Each actor maintains a pair of input-output links with the FIFO network agent. When an actor α_i wants to send a message m to another actor α_j (may be itself), α_i formats m (detailed below) and enqueues it on α_i's output link. We call this action α_i *sends* m *to* α_j. ϕ dequeues m from the link and places it into the message buffer that ϕ maintains. Eventually ϕ removes m from its buffer and enqueues m on the input link of α_j. When α_j dequeues m we say that α_j *delivers* m *from* α_i. The original system assumes the following of the network:

1. α-*Persistence.* If two hosts h_i and h_j are correct and α_i sends m to α_j, then α_j delivers m from α_i.
2. α-*Authenticity.* If two hosts h_i and h_j are honest and α_i does not send m to α_j, then α_j does not deliver m from α_i.
3. α-*FIFO.* If two hosts h_i and h_j are honest and α_i sends m_1 before m_2, and α_j delivers m_1 and m_2 from α_i, then α_j delivers m_1 before m_2;

Note that in the original system all hosts are honest. However, for the translation we need to be able to generalize these properties to include Byzantine hosts.

Messages in the original system are categorized as internal or external. *Internal messages* are sent between actors and are formatted as $\langle d, i, j\rangle$, where d is the data (or payload), i indicates the source actor, and j indicates the destination actor. *External messages* are from clients to actors and are formatted as $\langle d, \perp, j\rangle$, similar to the format of internal messages except the source actor is empty ($\perp$). Internal and external messages are in general called α-*messages*, or simply *messages* when the context is clear.

In the original system all actors produce output messages by making transitions based on input as specified by the protocol. We call such output messages *valid*. We formalize validity below.

External messages are assumed to be valid. For example, we may require that clients sign messages. An internal message m sent by actor α_i is valid if and only if there exists a sequence of valid messages $m_1^i, \ldots, m_c^i$ delivered by α_i such that $m \in F^i(m_c^i, F^i(m_{c-1}^i, F^i(\ldots, F^i(m_1^i, s_0^i)\texttt{.next}\ldots)\texttt{.next})\texttt{.next})\texttt{.output}$. The expression means that actor α_i sends m after it has processed the first c input messages, be they internal or external. Note that external input forms the base case for this recursive definition, as actors produce no internal messages until at least one delivers an external message.[1]

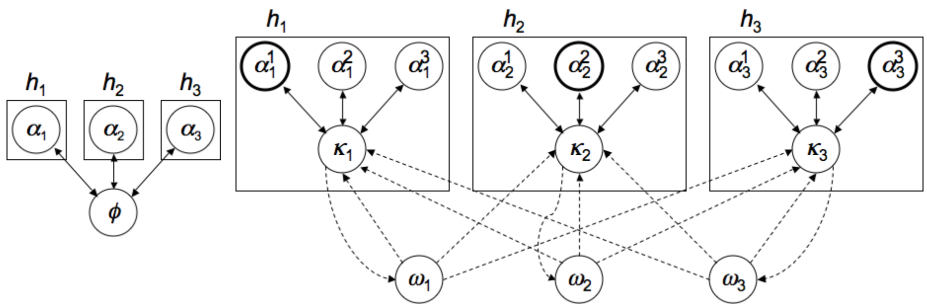


Fig. 4. Translation: the original system (left) is simulated at each host in the translated system (right). Dark circles are master actors. Dashed lines represent OARcast communication.

In order for the original system to work correctly, each actor needs to make transitions based on valid input. More formally,

4. *α-Validity.* If h_i is honest and α_i delivers m from α_j, then m is valid.

The property is granted to the original system by default, because it is in an environment where faulty hosts follow the protocol faithfully until they crash.

Besides the four α–properties, the original system requires no other assumptions about communication among actors. However, the original system may require non-communication assumptions such as "up to t hosts can fail."

The Translation mechanism transforms a crash-tolerant system in which all hosts require the four α-properties into a Byzantine-tolerant system that preserves the α-properties.

6.2 Implementation

In the original system, each actor α_i runs on a separate host h_i. In the translated system each host simulates the entire original system (see Fig. 4). That is, a host

[1] We model periodic processing not based on input by external *timer* messages.

runs a replica of each of the n actors and passes messages between the actors internally using a simulated network agent, called *coordinator*, that runs on the host. We denote the coordinator running on host h_i as κ_i.

To ensure that the different hosts stay synchronized, the coordinators agree on the order in which messages are delivered to replicas of the same actor. The replica of α_i on host h_j is called α^i_j. We designate α^i_i as the *master* replica and α^i_j $(i \neq j)$ as *slave* replicas. On honest hosts, the replicas of each actor start in the same initial state.

Each coordinator replaces ϕ of the original system by OARcast, i.e., OARcast is used to send messages. OARcast guarantees that coordinators agree on the delivery of messages to replicas of a particular actor. Coordinators wrap each α-message in a κ-*message*. κ-messages have the form $\langle tag\ m, i\rangle$, where tag is either $\mathsf{internal}$ or $\mathsf{external}$, m is an α-message, and i indicates the destination actor.

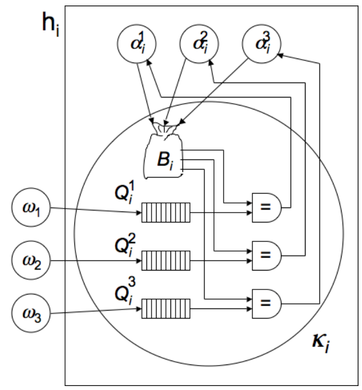


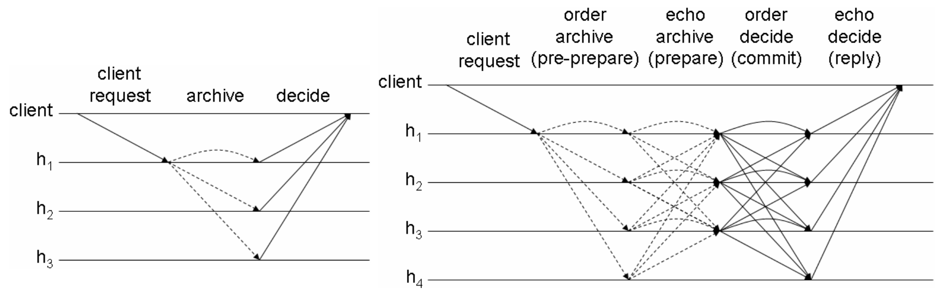
Fig. 5. Anatomy of host h_i in the translated system.

Each coordinator maintains an unordered *message bag* and n per-actor-replica *message queues*. By B_i we denote the message bag at host i and by Q^j_i we denote the message queue for actor α^j_i at host i (see Fig. 5). The pseudo-code for a coordinator κ_i appears in Fig. 6. κ_i intercepts messages from local actors, and it receives messages from remote coordinators. κ_i places α-messages sent by local actor replicas in B_i, and places α-messages received within κ-messages from κ_j in Q^j_i. When there is a match between a message m in the bag and the head of a queue, the coordinator enqueues m for the corresponding actor.

The translated system guarantees α-Persistence, α-Authenticity, α-FIFO, and α-Validity to all master actors on honest hosts. Appendix B contains a proof of correctness.

```
// Message from external client
On receipt of msg m = ⟨x, ⊥, i⟩:
    κ_i.send(⟨external m, i⟩);

// Message from actor j to actor k
On α_i^j.send(⟨d, j, k⟩):
    B_i.add(⟨d, j, k⟩);
    if k = i then
        κ_i.send(⟨internal ⟨d, j, i⟩, i⟩);

// κ-message from j
On κ_i.deliver(⟨tag m, j⟩):
    Q_i^j.enqueue(m);

// Head of queue matches msg in bag
When ∃j : Q_i^j.head() ∈ B_i:
    m = Q_i^j.dequeue();
    B_i.remove(m);
    α_i^j.deliver(m);

// Head of message queue is external
When ∃j, d : Q_i^j.head() = ⟨d, ⊥, j⟩:
    m = Q_i^j.dequeue();
    α_i^j.deliver(m);
```

Fig. 6. Pseudo-code of the Translation Mechanism for coordinator κ_i.



(a) before translation

(b) after translation

Fig. 7. A normal case run of (a) the original system and (b) the translated system. Dashed arrows indicate the `archive` message from the primary. Between brackets we indicate the corresponding BFT message types.

7 Illustration: BFT

In 1999 Castro and Liskov published "Practical Byzantine Fault Tolerance," a paper about a replication protocol (BFT) for a Byzantine-tolerant NFS file system [4]. The paper shows that BFT is indeed practical, adding relatively little overhead to NFS. In this section we show, informally, that a protocol much like BFT can be synthesized from the Viewstamped Replication protocol by Oki and Liskov [10] and the transformations of the current paper. The main difference is that our protocol is structured, while BFT is largely monolithic. In our opinion, the structure simplifies understanding and enhances the ability to scrutinize the protocol. The BFT paper addresses several practical issues and possible optimizations that can be applied to our scheme as well, but omitted for brevity.

Viewstamped Replication is a consensus protocol. A normal case execution is shown in Fig. 7(a).[2] A client sends a request to a server that is elected primary. The primary server sends an `archive` message to each server in the system. If a quorum responds to the client, the request is completed successfully. In the case of failures, a possibly infinite number of rounds of this consensus protocol may be necessary to reach a decision.

If we were to apply translation literally as described, we would end up with a protocol that sends significantly more messages than BFT. The reason for this is two-fold. First, our translation does nothing to group related information from a particular sender to a particular receiver in single messages. Instead, all pieces of information go out, concurrently, in separate small messages. While explicit optimizations could eliminate these, FIFO protocols like TCP automatically aggregate concurrent traffic between a pair of hosts into single messages for

[2] Slightly optimized for our purpose by sending `decide` messages back to the client instead of the primary.

efficiency, obviating the need for any explicit optimizations. Note that while these techniques reduce the number of messages, the messages become larger and the number of rounds remains the same.

Second, the translation would produce a protocol that solves *uniform Byzantine consensus* [5], guaranteeing that if two honest servers decide on an update, they decide on the same update. In a Byzantine environment, one may argue that this property is stronger than needed. We only need that if two *correct* servers decide on an update, they decide the same update. The reason for this is that clients of the system have to deal with the results from Byzantine servers, and because Byzantine and crashing hosts are both counted towards t it is not usually a problem that an honest server makes a "mistake" before crashing. Such servers would be outvoted by correct servers.

BFT does not provide uniform consensus, but Viewstamped Replication does. Our translation maintains uniformity. This arises in the bc-Relay property, which requires that if an honest host delivers a message, then all correct hosts have to do the same. For our purposes, it would be sufficient to require that if a correct host delivers a message, all correct hosts have to follow suit.

If we revisit the ARcast implementation, we see that the protocol maintains the original uniform bc-Relay property by having a receiver await $t + 1$ copies of a message before delivery. Doing so makes sure that one of the copies was sent by a correct receiver that forwards a copy to all other correct receivers as well. For non-uniform bc-Relay this is unnecessary because the receiver itself, if correct, is guaranteed to forward the message to all other correct receivers, and thus a receiver can deliver the message as soon as the first copy is received. The `echo` traffic can be piggybacked on future traffic.

Using this modification, Fig. 7(b) demonstrates a normal run of the translated system for $t = 1$. The figure only shows the traffic that is causally prior to the reply received by the client and thus essential to the latency that the client experiences. In this particular translation we used t additional hosts for OARcast only, but a more faithful translation would have started with $3t + 1$ servers. Nevertheless, the run closely resembles that of a normal run of BFT (see Figure 1 of [4]).

8 Conclusion

We presented a mechanism to translate a distributed application that tolerates only crash failures into one that tolerates Byzantine failures. Few restrictions are placed on the application, and the approach is applicable not only to consensus but to a large class of distributed applications. The approach makes use of a novel broadcast protocol. We have illustrated how the approach may be used to derive a version of the Castro and Liskov Practical Byzantine Replication protocol, showing that our translation mechanism is pragmatic and more powerful than previous translation approaches.

References

1. R.A. Bazzi. *Automatically increasing fault tolerance in distributed systems.* PhD thesis, Georgia Institute of Technology, Atlanta, GA, USA, 1995.
2. R.A. Bazzi and G. Neiger. Simplifying fault-tolerance: providing the abstraction of crash failures. *J. ACM*, 48(3):499–554, 2001.
3. G. Bracha. Asynchronous byzantine agreement protocols. *Inf. Comput.*, 75(2):130–143, 1987.
4. M. Castro and B. Liskov. Practical Byzantine Fault Tolerance. In *Proc. of the 3rd Symposium on Operating Systems Design and Implementation (OSDI)*, New Orleans, LA, February 1999.
5. B. Charron-Bost and A. Schiper. Uniform consensus is harder than consensus. *Journal of Algorithms*, 51(1):15–37, April 2004.
6. B.A. Coan. A compiler that increases the fault tolerance of asynchronous protocols. *IEEE Transactions on Computers*, 37(12):1541–1553, 1988.
7. L. Lamport, R. Shostak, and M. Pease. The Byzantine generals problem. *Trans. on Programming Languages and Systems*, 4(3):382–401, July 1982.
8. D. Mpoeleng, P. Ezhilchelvan, and N. Speirs. From crash tolerance to authenticated byzantine tolerance: A structured approach, the cost and benefits. *Int. Conf. on Dependable Systems and Networks*, 2003.
9. G. Neiger and S. Toueg. Automatically increasing the fault-tolerance of distributed systems. In *PODC '88: Proceedings of the Seventh Annual ACM Symposium on Principles of Distributed Computing*, pages 248–262, New York, NY, USA, 1988. ACM Press.
10. B.M. Oki and B.H. Liskov. Viewstamped replication: A general primary-copy method to support highly-available distributed systems. In *Proc. of the 7th ACM Symp. on Principles of Distributed Computing*, pages 8–17, Toronto, Ontario, August 1988. ACM SIGOPS-SIGACT.
11. T.K. Srikanth and S. Toueg. Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. *Distributed Computing*, 2(2):80–94, June 1987.

A Correctness of OARcast

Lemma 1. *Say h_i and h_j are honest and m is the c^{th} message that ω_j^{R} delivers from ω_i^{S}, then m is the c^{th} message that ω_i^{S} sent.*

Proof. Say m is not the c^{th} message sent by $\omega_i^{\mathbf{S}}$, but it is the c^{th} message delivered by $\omega_j^{\mathbf{R}}$. $\omega_j^{\mathbf{R}}$ must have received $2t+1$ messages of the form $\langle \mathsf{order}\ m, c-1, i\rangle$ from different orderers. Because only t hosts may fail, and because of bc-Authenticity of ARcast, at least one of the order messages comes from a correct orderer. Because communication between $\omega_i^{\mathbf{S}}$ and this orderer is FIFO, and because the sender does not send m as its c^{th} message, it is not possible that the orderer sent $\langle \mathsf{order}\ m, c-1, i\rangle$. $\square$

Lemma 2. *Say m is the c^{th} message that a correct sender ω_i^{S} sends. Then all correct receivers receive at least $2t+1$ messages of the form $\langle \mathsf{order}\ m, c-1, i\rangle$ from different orderers.*

Proof. Because the sender is correct, each of the correct orderers will deliver m. As all links are FIFO and m is the c^{th} message, it is clear that for each orderer $\omega_k^{\mathbf{O}}$, $t_k = c-1$. Each correct orderer $\omega_k^{\mathbf{O}}$ therefore sends $\langle \mathsf{order}\ m, c-1, i\rangle$ to all receivers. Because at least $2t+1$ of the orderers are correct, and because of ARcast's bc-Persistence, each correct receiver receives $2t+1$ such order messages. □

Theorem 1. *OARcast satisfies bc-Persistence.*

Proof. Assume the sending host, h_i, is correct, and consider a correct receiving host h_j. The proof proceeds by induction on c, the number of messages sent by $\omega_i^{\mathbf{S}}$. Consider the first message m sent by $\omega_i^{\mathbf{S}}$. By Lemma 2, $\omega_j^{\mathbf{R}}$ receives $2t+1$ messages of the form $\langle \mathsf{order}\ m, 0, i\rangle$. By Lemma 1 it is not possible that the first message that $\omega_j^{\mathbf{R}}$ delivers is a message other than m. Therefore, $c_j = 0$ when $\omega_j^{\mathbf{R}}$ receives the order messages for m and will deliver m.

Now assume that bc-Persistence holds for the first c messages from $\omega_i^{\mathbf{S}}$. We show that bc-Persistence holds for the $(c+1)^{\text{st}}$ message sent by $\omega_i^{\mathbf{S}}$. By Lemma 2, $\omega_j^{\mathbf{R}}$ receives $2t+1$ messages of the form $\langle \mathsf{order}\ m, c, i\rangle$. By the induction hypothesis, $\omega_j^{\mathbf{R}}$ will increment c_j at least up to c. By Lemma 1 it is not possible that the c^{th} message that $\omega_j^{\mathbf{R}}$ delivers is a message other than m. Therefore, $c_j = c$ when $\omega_j^{\mathbf{R}}$ receives the order messages for m and will deliver m. □

Theorem 2. *OARcast satisfies bc-Authenticity.*

Proof. This is a straightforward corollary of Lemma 1. □

Theorem 3. *OARcast satisfies bc-Relay.*

Proof. By induction on the sequence number. Say some correct receiver $\omega_j^{\mathbf{R}}$ delivers the first κ-message m from $\omega_i^{\mathbf{S}}$. Therefore, $\omega_j^{\mathbf{R}}$ must have received $2t+1$ messages of the form $\langle \mathsf{order}\ m, 0, i\rangle$ from different orderers when $c_j = 0$. Because of the bc-Relay property of ARcast, all correct receivers receive the same order messages from the orderers. By Lemma 1 it is not possible that a correct receiver $\omega_{j'}^{\mathbf{R}}$ delivered a κ-message other than m, and therefore $c_{j'} = 0$ when $\omega_{j'}^{\mathbf{R}}$ receives the order messages. Thus $\omega_{j'}^{\mathbf{R}}$ will also deliver m.

Now assume the theorem holds for the first c κ-messages sent by $\omega_i^{\mathbf{S}}$. Say some correct receiver $\omega_j^{\mathbf{R}}$ delivers the $(c+1)^{\text{st}}$ κ-message m from $\omega_i^{\mathbf{S}}$. Therefore, $\omega_j^{\mathbf{R}}$ must have received $2t+1$ messages of the form $\langle \mathsf{order}\ m, c, i\rangle$ from different orderers when $c_j = c$. Because of the bc-Relay property of ARcast, all correct receivers receive the same order messages from the orderers. Because of the induction hypothesis, the correct receivers deliver the first c κ-messages. By Lemma 1 it is not possible that a correct receiver $\omega_{j'}^{\mathbf{R}}$ delivered a κ-message other than m, and therefore $c_j = c$ when $\omega_{j'}^{\mathbf{R}}$ receives the order messages. Thus $\omega_{j'}^{\mathbf{R}}$ will also deliver m. □

Lemma 3. *Say m is the c^{th} message that an honest receiver $\omega_j^{\mathbf{R}}$ delivers from $\omega_i^{\mathbf{S}}$, and m' is the c^{th} message that another honest receiver $\omega_{j'}^{\mathbf{R}}$ delivers from $\omega_i^{\mathbf{S}}$. Then $m = m'$ (even if h_i is Byzantine).*

Proof. Say not. $\omega_j^{\mathbf{R}}$ must have received $2t+1$ messages of the form $\langle \mathsf{order}\ m, c-1, i\rangle$ from different orderers, while $\omega_{j'}^{\mathbf{R}}$ must have received $2t+1$ messages of the form $\langle \mathsf{order}\ m', c-1, i\rangle$ from different orderers. As there are only $3t+1$ orderers, at least one correct orderer must have sent one of each, which is impossible as correct orderers increment their sequence numbers for each new message. □

Theorem 4. *OARcast satisfies bc-Ordering.*

Proof. Corollary of Lemma 3. □

Theorem 5. *OARcast satisfies bc-FIFO.*

Proof. Evident from the FIFOness of messages from senders to orderers and the sequence numbers utilized by orderers and receivers. □

B Correctness of Translation

We prove correctness of the Translation mechanism assuming the bc-properties. In particular, we show that the collection of coordinators and slave replicas that use the Translation mechanism preserves the α-properties: *α-Persistence*, *α-Authenticity*, *α-FIFO*, and *α-Validity*, for the master replicas $\{\alpha_i^i\}$.

For convenience, we combine bc-Relay and bc-Ordering to state that coordinators on correct hosts deliver the same sequence of κ-messages from any κ_k, even if h_k is Byzantine. This is put more formally in the following lemma:

Lemma 4. *For any i, j, and k, if h_i and h_j are correct, then κ_i and κ_j deliver the same sequence of messages from κ_k.*

Proof. bc-Relay guarantees that κ_i and κ_j deliver the same set of messages from κ_k. bc-Ordering further guarantees that the delivery order between any two messages is the same at both κ_i and κ_j. □

In the proof we need to be able to compare states of hosts. We represent the state of host h_i by a vector of counters, $\Phi_i = (c_i^1, \ldots, c_i^n)$, where each c_i^k is the number of messages that (the local) actor α_i^k has delivered. As shown below, within an execution of the protocol, replicas of the same actor deliver the same sequence of messages. Thus from c_i^k and c_j^k we can compare progress of replicas of α_k on hosts h_i and h_j.

Lemma 5. *Given are that hosts h_i and h_j are correct, α_i^k delivers $m_1, \ldots, m_c$, and α_j^k delivers $c' \leq c$ messages. Then the messages that α_j^k delivers are $m_1, \ldots, m_{c'}$.*

Proof. By the Translation mechanism, the first c' messages that α_i^k and α_j^k deliver are the contents of the first c' κ-messages that κ_i and κ_j delivered from κ_k, resp. By Lemma 4, the two κ-message sequences are identical. This and the fact that links from coordinators to actors are FIFO imply that the first c' messages that α_i^k and α_j^k deliver are identical. □

In the remaining proof we use the following definitions and notations:

- h_i *reaches* $\Phi = (c_1, \ldots, c_n)$, denoted $h_i \rightsquigarrow \Phi$, if $\forall_j \; c_i^j \geq c_j$;
- $\Phi = (c_1, \ldots, c_n)$ *precedes* $\Phi' = (c'_1, \ldots, c'_n)$, denoted $\Phi < \Phi'$, if $(\forall_i \; c_i \leq c'_i) \wedge (\exists_j \; c_j < c'_j)$;
- $\Phi = (c_1, \ldots, c_n)$ *produces* m if $m \in \bigcup_{i=1}^{n} \bigcup_{c=1}^{c_i} (F^i(m_c^i, s_{c-1}^i).\texttt{output})$, where m_c^i is the c^{th} message to α_i and s_{c-1}^i is the state of α_i after it processes the first $c-1$ input messages.

Corollary 1. *If Φ produces m on a correct host, Φ produces m on all correct hosts that reach Φ.*

Proof. By Lemma 5 and because replicas of the same actor start in the same state and are deterministic, if Φ produces m on a correct host, Φ produces m on all correct hosts that reach Φ. $\square$

We now show that if a correct host <u>*is*</u> in a particular state then all other correct hosts will <u>*reach*</u> this state.

Lemma 6. *If there is a correct host h_i in state Φ, then, eventually, all correct hosts reach Φ.*

Proof. By induction on Φ. All correct hosts start in state $\Phi^0 = (0, \ldots, 0)$, and $\forall \Phi \neq \Phi^0 : \Phi^0 < \Phi$.
<u>Base case</u>: All correct hosts reach Φ^0 by definition.
<u>Inductive case</u>: Say that correct host h_i is in state $\Phi = (c_1, \ldots, c_n)$, and the lemma holds for all $\Phi' < \Phi$ (Induction Hypothesis). We need to show that any correct host h_j reaches Φ.

Consider the last message m that some actor replica α_i^p delivered. Thus, m is the c_p^{th} message that α_i^p delivered. The state of h_i prior to delivering this message is $\Phi' = (c_1, \ldots, c_p - 1, \ldots, c_n)$. It is clear that $\Phi' < \Phi$. By the induction hypothesis $h_j \rightsquigarrow \Phi'$.

By the Translation mechanism we know that $\langle tag\ m, p \rangle$ (for some tag) is the c_p^{th} κ-message that κ_i delivers from κ_p. Lemma 4 implies that $\langle tag\ m, p \rangle$ must also be the c_p^{th} κ-message that κ_j delivers from κ_p. Since $h_j \rightsquigarrow \Phi'$, α_j^p delivers the first $c_p - 1$ α-messages, and thus κ_j must have removed those messages from Q_j^p. Consequently, m gets to the head of Q_j^p. (1)

Now there are two cases to consider. If m is external, then κ_j will directly remove m from Q_j^p and enqueue m on the link to α_j^p. Because α_i^p delivered m after delivering the first $c_p - 1$ messages (Lemma 5), and α_i^p and α_j^p run the same function F^p, α_j^p will eventually deliver m as well, and therefore $h_j \rightsquigarrow \Phi$.

Consider the case where m is internal. By definition, $\Phi' = (c_1, \ldots, c_p - 1, \ldots, c_n)$ *produces* m at host h_i. By Corollary 1, Φ' produces m at host h_j. Thus, eventually κ_j places the message in the message bag B_j. (2)

(1) and (2) provide the matching condition for κ_j to enqueue m on its link to α_j^p. Using the same reasoning for the external message case, $h_j \rightsquigarrow \Phi$. $\square$

We can now show the first two communication properties. (The proof for α-FIFO has been omitted for lack of space.)

Theorem 6. *(α-Persistence.) If two hosts h_i and h_j are correct and α_i^i sends m to α_j, then α_j^j delivers m from α_i.*

Proof. Suppose h_i is in state Φ_i when α_i^i sends m to α_j. By Lemma 6, $h_j \rightsquigarrow \Phi_i$. Thus, α_j^i sends m to α_j as well. By the Translation mechanism, κ_j places m in B_j and OARcasts $\langle \mathsf{internal}\ m, j\rangle$. By bc-Persistence, κ_j delivers $\langle \mathsf{internal}\ m, j\rangle$ (from itself) and places m on its queue Q_j^j. (1)

By the Translation Mechanism, each external message at the head of Q_j^j is dequeued and delivered by α_j^j. (2)

Let us consider an internal message m' at the head of Q_j^j. Since h_j is correct, the Translation mechanism ensures that κ_j has delivered $\langle \mathsf{internal}\ m', j\rangle$ (the κ-message containing m' and from κ_j). bc-Authenticity ensures that κ_j has indeed sent the κ-message. By the Translation mechanism, κ_j always puts a copy of m' in B_j before sending $\langle \mathsf{internal}\ m', j\rangle$. Thus, m' in Q_j^j is matched with a copy in B_j, and α_j^j delivers m'. This together with (2) show that α_j^j delivers all internal messages in Q_j^j. (3)

(1) shows that m sent by α_i^i arrives in Q_j^j, and (3) shows that α_j^j delivers all internal messages in Q_j^j. Together they show that α_j^j delivers m from α_i. □

Theorem 7. *(α-Authenticity.) If two hosts h_i and h_j are honest and α_i^i does not send m to α_j, then α_j^j does not deliver m from α_i.*

Proof. Assume α_j^j delivers m from α_i, but α_i^i did not send m to α_j. By the Translation mechanism, a necessary condition for α_j^j to deliver m from α_i is that κ_j delivers $\langle \mathsf{internal}\ m, i\rangle$. By bc-Authenticity of OARcast, κ_i must have OARcast $\langle \mathsf{internal}\ m, i\rangle$. Then by the Translation mechanism, α_i^i must have sent m, contradicting the assumption. □

We introduce a lemma that helps us show α-*Validity*:

Lemma 7. *Actor replicas on honest hosts only send valid messages.*

Proof. Suppose not. Let m sent by α_j^i be the first invalid message sent by an actor replica on an honest host. Since h_j is honest, there must be a sequence of messages $m_1^i, \dots, m_c^i$ that α_j^i delivered, such that

$$m \in F^i(m_c^i, F^i(m_{c-1}^i, F^i(\dots, F^i(m_1^i, s_0^i).\mathtt{next} \dots).\mathtt{next}).\mathtt{next}).\mathtt{output}$$

Since m is the first invalid message sent by an actor replica, all internal messages in the sequence $m_1^i, \dots, m_c^i$ must be valid. Moreover, external messages are valid by definition. Thus, all messages $m_1^i, \dots, m_c^i$ are valid. But then, m is valid by definition, contradicting the assumption. □

Theorem 8. *(α-Validity.) If h_i is honest and α_i^i delivers m from α_j, then m is valid (even if $j \neq \perp$ and h_j is faulty.)*

Proof. If m is an external message, then it is valid and unforgeable by definition.

If m is an internal message, the fact that α_i^i delivers m from α_j implies that α_i^j has sent m to α_i. By Lemma 7, m is valid. □