

Distributed Systems 600.437 Intrusion-Tolerant Replication

Department of Computer Science
The Johns Hopkins University

Intrusion Tolerant Replication

Lecture 8

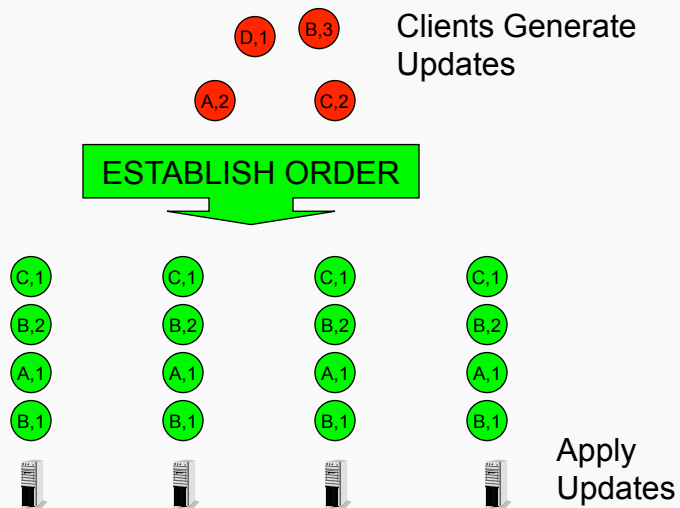
Further readings:

- *Practical Byzantine Fault Tolerance*, Miguel Castro and Barbara Liskov, OSDI 99.
- *Prime: Byzantine Replication Under Attack* IEEE TDSC 2011.
- *Towards a Practical Survivable Intrusion Tolerant Replication System* IEEE SRDS 2014.

State Machine Replication

- Servers **start in the same state**.
- Servers change their state only when they execute an update.
- State changes are deterministic. **Two servers in the same state will move to identical states, if they execute the same update.**
- If servers **execute updates in the same order**, they will progress through exactly the same states. **State Machine Replication!**

State Machine Replication



Outline

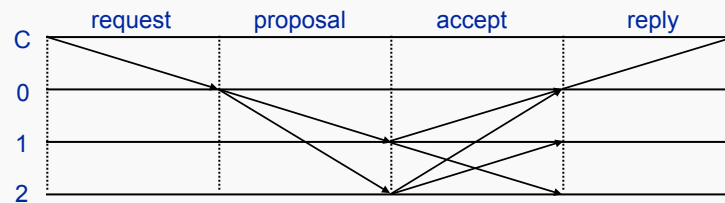
- State Machine Replication
- Byzantine Fault Tolerant Replication (**BFT**)
 - Servers can lie
 - Safety and Liveness properties
 - Byzantine performance failure
- Performance Guarantees while Under Attack (**Prime**)
 - Bounded delay
 - Pre-Ordering and Ordering protocols
 - Suspect-Leader protocol
- Survivable Intrusion Tolerant Replication
 - BFT with performance guarantees under attack
 - Defense across Space and Time
 - Support to large-state application



System Model

- N servers
 - Uniquely identified in $\{1 \dots N\}$
- Asynchronous communication
 - Message **loss**, duplication, and delay
 - Network **partitions**
 - No message corruption
- Benign faults
 - Crash/recovery with stable storage
- Byzantine faults
 - **Byzantine** behavior – up to f servers may lie
 - $N \geq 3f + 1$

Benign Faults: Paxos



- The Part-Time Parliament [Lamport, 98]
- A very resilient protocol. Only a majority of participants are required to make progress.
- Works well on unstable networks.
- Only handles benign failures (not Byzantine).

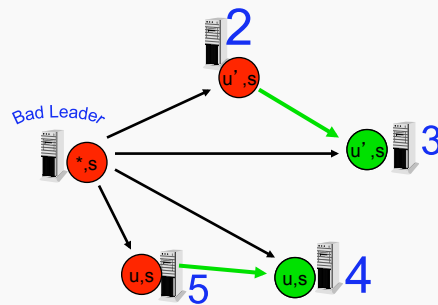
What Happens If Servers Lie?

- Servers must be able to verify who sent each message.
 - **Crypto! Digital Signatures or HMACS**
- **The leader might be bad!**
- What might happen?

What Happens If Servers Lie?

- Servers must be able to verify who sent each message.
 - **Crypto! Digital Signatures or HMACS**
- **The leader might be bad!**
- What might happen?
 - The leader can send $\text{Proposal}(u,s)$ to 2 out of 5 servers and $\text{Proposal}(u',s)$ to 2 out of 5 servers -- can we have a safety violation?
- **Correct servers must make sure the malicious servers do not cause safety errors.**
- The bad servers might send messages or they might not.

Byzantine Leader Example



- **Bad Leader** Sends $\text{Proposal}(u,s)$ to servers 4 and 5.
- **Bad Leader** Sends $\text{Proposal}(u',s)$ to servers 2 and 3.
- Server 4 could **order** (u,s) and server 3 could **order** (u',s) .

How Do We Solve this Problem?

- Assume that there are at most **f malicious** servers, which can fail or become malicious. **All of the other servers are correct.**
- Let N denote the number of servers in our system.
- Any correct server can wait for at most $N - f$ messages from servers, because f may fail or be malicious (and not send their messages).
- Can we add **more servers**?

How many servers do we need?

- **Malicious servers can lie.**
- **Good servers tell the truth.**
- We need to guarantee that a malicious server cannot generate two groups of **Accept / Proposal** messages that conflict. (i.e., (u, s) and (u', s)) within the same view.
- We need at least $N = 3f + 1$ servers to do this!!
- We wait for $2f + 1$ messages that say the same thing!
- **The f bad servers can say $\text{Accept}(u, s)$ and $\text{Accept}(u', s)$.**
- **The good servers say only one thing, but a bad leader can lie to them.**
- Let's try to generate the two sets of messages -- Can we do it?
- Liar tells $f + 1$ of the good servers (u, s) , and f of the good servers (u', s) .

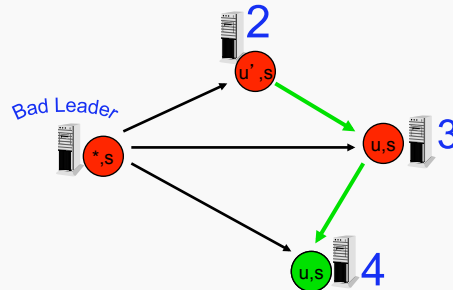
(u, s)
 $f(\text{bad}) + f + 1(\text{good})$

total: $2f + 1$

(u', s)
 $f(\text{bad}) + f(\text{good})$

total: $2f$

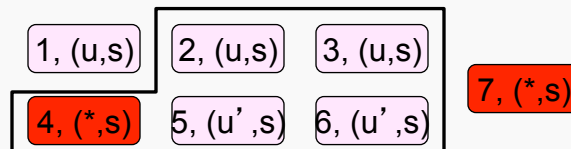
Let's use $N=3f+1$!



- $f = 1, N = 4$
- **Bad Leader** sends $\text{Proposal}(u, s)$ to Server 3 and 4.
- **Bad Leader** sends $\text{Proposal}(u', s)$ to Server 2.
- Can the **Bad Leader** violate safety?

Is the Protocol Live?

- $f = 2, N = 3 \cdot 2 + 1 = 7$
- Bad Leader is Server 7, and Server 4 is bad, too!
- Bad Leader sends $\text{Proposal}(v, u, s)$ to Servers 1, 2, and 3
- Bad Leader sends $\text{Proposal}(v, u', s)$ to Servers 4, 5, and 6
- There is a partition, Servers 2, 3, 4, 5, 6 are together.
- They can't determine which update server 1 ordered.



How Can We Guarantee Liveness?

- **We can add another round to the fault tolerant protocol. The Normal Case Protocol becomes:**
 - The Leader broadcasts a **Pre-Prepare(v,u,s)**
 - If not Leader, Upon receiving a **Pre-Prepare(v,u,s)** that does not conflict with what I know about, broadcast a **Prepare(v,u,s)**
 - Upon receiving **2f Prepare(v,u,s)** and **1 Pre-Prepare(v,u,s)**, broadcast **Commit(v,u,s)**
 - Upon receiving **2f+1** Commit Messages, Order the message
 - Rounds 1 and 2 allow the correct servers to preserve safety within the same view.
 - Round 3 preserves safety across view changes.
- Note that if $N > 3f+1$, then every process must receive at least $n-f$ {Prepare and Pre-prepare messages} as well as $n-f$ {Commit messages}.

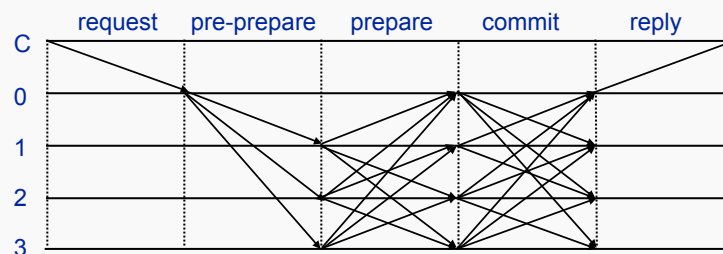
What About Changing Leaders?

- If any server orders (v,u,s) , then **2f+1** servers must have collected a set of **2f Prepare(v,u,s)** messages and **1 Pre-Prepare(v,u,s)**
- We call such a set a **Prepare-Certificate(v,u,s)**.
- If **Prepare-Certificate(v,u,s)** exists, then **Prepare-Certificate(v,u',s)** cannot exist.
- How do we **change Leaders** (View Changes) ?

What About Changing Leaders?

- If any server orders (v, u, s) , then $2f+1$ servers must have collected a set of $2f$ $\text{Prepare}(v, u, s)$ messages and 1 $\text{Pre-Prepare}(v, u, s)$
- We call such a set a **Prepare-Certificate** (v, u, s) .
- If **Prepare-Certificate** (v, u, s) exists, then **Prepare-Certificate** (v, u', s) cannot exist.
- How do we **change Leaders** (View Changes) ?
 - The new leader collects information from $2f+1$ servers. The servers supply Prepare-Certificates. If something was ordered, the new leader will find out.
 - The new leader needs to send this information to all of the correct servers, otherwise the correct servers will not participate in the protocol.
- A Prepare-Certificate can be viewed as a trusted message (agreed upon by all of the servers). We use it like we use a Proposal message in Paxos.

We have: BFT



- Byzantine Fault Tolerance [Castro and Liskov, 99]
- **Excellent LAN performance.** Over 1000 updates/sec. (without stable storage costs)
- $2/3$ **total servers** + 1 are required to make progress
- Three rounds of message exchanges

The Downside of Asynchrony

- Existing correctness criteria: **safety** and **liveness**
 - **Safety**: servers remain consistent.
 - **Liveness**: each update is eventually executed.
- Protocols are designed to be **safe in all executions**.
 - Do not rely on synchrony for safety!
 - Guarantee liveness only when the network is **sufficiently stable**.
- Real systems are not completely asynchronous.
 - **Systems can satisfy much stronger performance guarantees than liveness during stable periods.**
- Consequence: Performance attacks!
 - **An attacker can exploit the gap between what is promised during stable periods (liveness) and what is possible.**

Byzantine Performance Failures

Previously Considered Byzantine Failures

Failure Type	Failure Behavior	Mitigated by
Value Domain	Sending incorrect, conflicting, or invalid messages	Cryptography, agreement protocols
Time Domain	Messages arrive after timeouts or not at all	Timeouts, view change

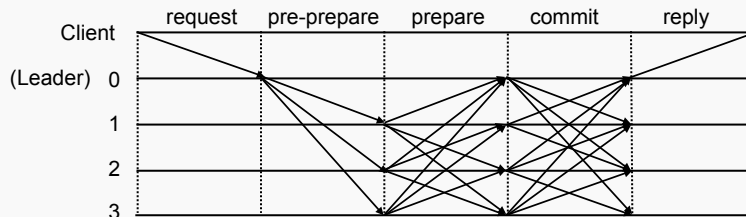
- If the adversary cannot violate safety and liveness, the next best thing is to **slow down the system beyond usefulness**.
- **Performance failures: send correct messages slowly but without triggering timeouts.**

A Problem: Performance Under Attack

- **BFT systems are vulnerable to performance attacks.**
 - A small number of faulty servers can cause the system to make progress at an extremely slow rate -- indefinitely!
- **Leader-based protocols** are vulnerable to performance attacks by a malicious leader.
 - Problem is **magnified in wide-area networks**, where it is difficult to predict the performance that should be expected of the leader.
- **Main challenges:**
 - Developing **meaningful performance metrics** for evaluating Byzantine replication protocols.
 - Designing protocols that **perform well according to these metrics**, even when the system is under attack.

Case Study: BFT Under Attack

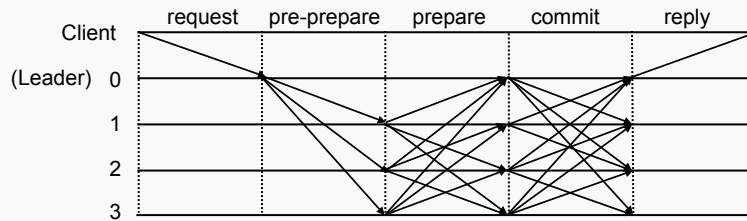
[Castro and Liskov 99]



- **Attack 1: Pre-Prepare Delay**
 - **Malicious leader can add delay into the ordering path by withholding its Pre-Prepare.**
 - Non-leaders maintain a FIFO queue of pending updates.
 - Use timeouts to monitor the leader.
 - Timeout placed on execution of first update in queue.
 - Malicious leader can stay in power by ordering one update per queue per timeout period!

Case Study: BFT Under Attack

[Castro and Liskov 99]



- Attack 2: Timeout Manipulation
 - Timeout doubles every time the leader is replaced.
 - **Use a denial of service attack to increase the timeout, then stop on a malicious leader.**
- Each update is eventually executed, but performance is much worse than if there were only correct servers.

Outline

- State Machine Replication
- Byzantine Fault Tolerant Replication (**BFT**)
 - Servers can lie
 - Safety and Liveness properties
 - Byzantine performance failure
- Performance Guarantees while Under Attack (**Prime**)
 - Bounded delay
 - Pre-Ordering and Ordering protocols
 - Suspect-Leader protocol
- Survivable Intrusion Tolerant Replication
 - BFT with performance guarantees under attack
 - Defense across Space and Time
 - Support to large-state application



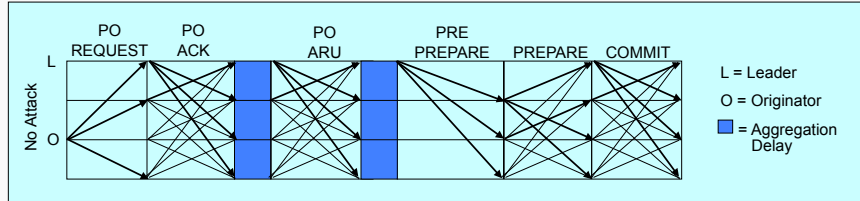
The Prime Replication System

- **Performance-Oriented Replication in Malicious Environments**
 - Leader-based protocol providing **Bounded-Delay**, a stronger guarantee than liveness, when the network is stable.
- **System components:**
 - **Prime Ordering Protocol** (Preordering phase, Global ordering phase)
 - **Suspect-Leader Protocol** for detecting malicious leaders.
- **Main Ideas:**
 - Resources needed by the leader to do its job are bounded and independent of system throughput.
 - Leader has “no excuse” for not sending timely messages.
 - Non-leader servers compute a threshold level of acceptable performance that the leader should meet.
 - **Upper-bounded by a function of the latency between correct servers after the network stabilizes.**

Bounded Delay

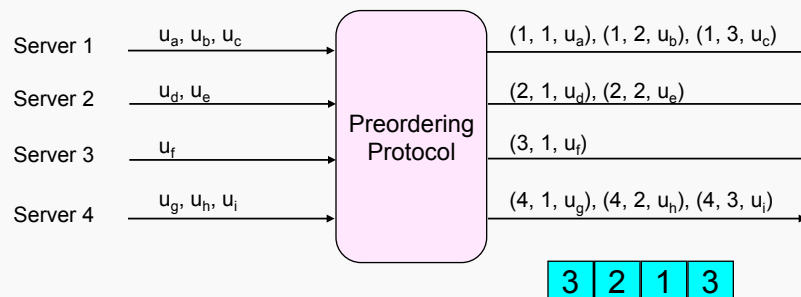
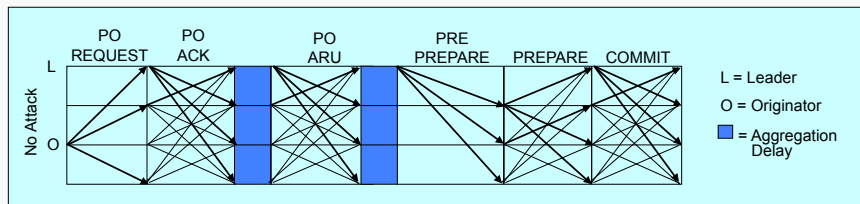
- **Prime-Stability:** There is a time after which the following condition holds for a set of at least $2f+1$ correct servers (the stable servers):
 - For each pair of stable servers r and s , there exists a value $\text{Min_Lat}(r,s)$, unknown to the servers, such that if r sends a message to s , it will arrive with delay $\Delta_{r,s}$, where
$$\text{Min_Lat}(r,s) \leq \Delta_{r,s} \leq \text{Min_Lat}(r,s) * K_{Lat}$$
- **Bounded-Delay:** There exists a time after which the update latency for any update initiated by a stable server is upper-bounded.

Prime: Ordering Protocol

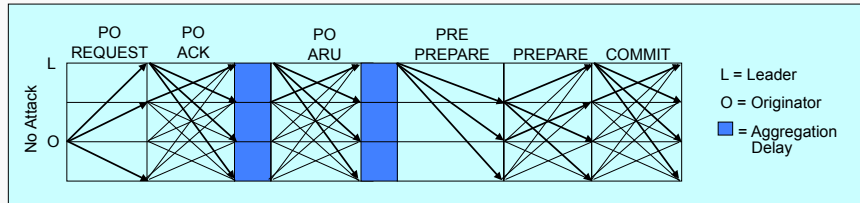


- **Preordering (PO) Phase:**
 - Each originating server o , disseminates its updates to the other servers (PO-Request).
 - Agreement protocol binds update u to **preorder identifier** (o, i) , where u is the i^{th} update originated by server o (PO-ACK).
 - Each server cumulatively acknowledges the updates it preorders (PO-ARU).

Prime: Ordering Protocol

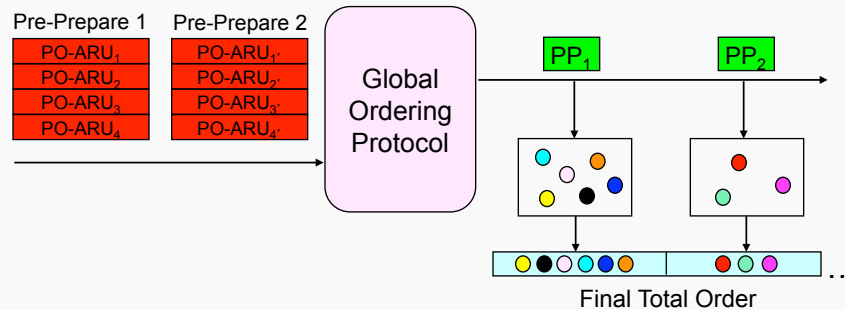
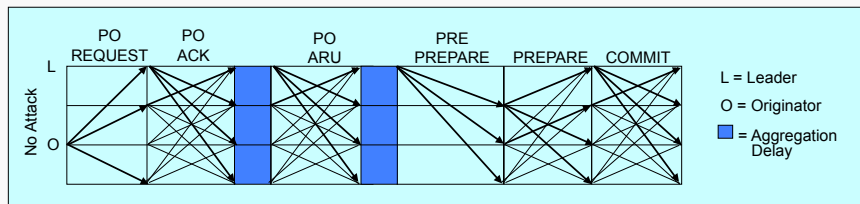


Prime: Ordering Protocol



- **Global Ordering Phase:**
 - Similar to BFT (Pre-Prepare, Prepare, Commit)
 - Leader periodically sends a Pre-Prepare containing a **proof matrix** (vector of PO-ARU messages).
 - Each globally ordered Pre-Prepare maps to a batch of preordered updates based on contents of proof matrix.
 - Final total order is obtained by deterministically ordering the updates in each batch based on preorder identifier.

Prime: Ordering Protocol



Suspect-Leader Protocol

- **Protocol Strategy:**
 - Dynamically determine an acceptable turn-around time based on roundtrip measurements (**TAT_acceptable**).
 - Use turn-around times measured in the current view to compute a measure of the current leader's performance (**TAT_leader**).
 - Suspect the leader if **TAT_leader > TAT_acceptable**.
- **Design Challenges:**
 - Malicious servers can lie to try to lower expectation of acceptable performance.
 - Leader could remain in power while going slowly.
 - Malicious servers can lie to make a correct leader look bad.
 - Would lead to continuous view changes.

Suspect-Leader: Key Properties

- **Any server that retains a role as leader must provide a TAT to at least one correct server that is no more than**

$$B = 2K_{Lat}L^* + \Delta_{agg}$$

- **Maximum update latency:**

$$\beta = 6L^* + B + 2\Delta_{agg}$$

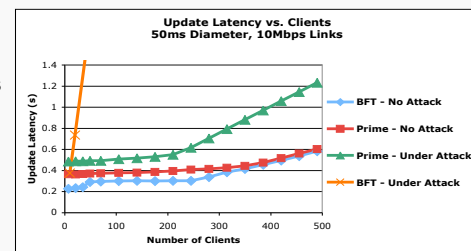
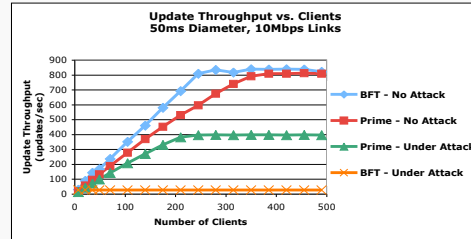
Bounded-Delay!

L^* = Maximum delay
between correct servers
 Δ_{agg} = Aggregation delay

- **There exists a set of at least f+1 correct servers that will not be suspected by any correct server if elected leader.**
 - Aggressive but not overly aggressive.

Experimental Results

- 7 servers ($f = 2$)
- Symmetric network
 - 50ms diameter, 10 Mbps links
- Leader performs just well enough to stay in power.
- BFT: aggressive timeout (300ms)
- BFT: Pre-Prepare delay
- Prime:
 - Leader adds as much delay as possible.
 - Non-leader servers force as much reconciliation as possible.



Yair Amir

Fall 16 / Lecture 8

33

Prime - Recap

- BFT replication protocols are vulnerable to performance attacks.
 - Liveness is not a meaningful performance metric for evaluating Byzantine replication protocols.
- **Bounded-Delay**: a new performance metric.
 - Can we provide stronger guarantees?
 - Can we guarantee a minimum throughput?
- **Prime**: a Byzantine replication protocol with performance guarantees while under attack.
 - Achieves Bounded-Delay when the network is sufficiently stable.

Yair Amir

Fall 16 / Lecture 8

34

Outline

- State Machine Replication
- Byzantine Fault Tolerant Replication (**BFT**)
 - Servers can lie
 - Safety and Liveness properties
 - Byzantine performance failure
- Performance Guarantees while Under Attack (**Prime**)
 - Bounded delay
 - Pre-Ordering and Ordering protocols
 - Suspect-Leader protocol
- Survivable Intrusion Tolerant Replication
 - BFT with performance guarantees under attack
 - Defense across Space and Time
 - Support to large-state application



Survivable Intrusion-Tolerant Replication Defense across Space and Time

- BFT with performance guarantees while under attack is a **short-term solution**
 - The adversary can exploit a single vulnerability to compromise all replicas
- We need to **diversify** the execution environment
 - **Static diversity** [Rodrigues2001, Castro2002, Sousa2008-2010, Roeder2010, ...]
 - Complexity for the adversary: from $O(1)$ to $O(n)$
 - Not survivable over long system lifetime
- **Proactive recovery** to clean the system from potential intrusions
- Survivability requires **defense across space and time: dynamic diversity + proactive recovery**
 - A rejuvenated replica is different from all previous replica instances
 - Complexity for the adversary: from $O(n)$ over the system lifetime to $O(n)$ within a bounded time (i.e. rejuvenation cycle)

Novelty Claims (SRDS 2014)

- The **first** practical survivable intrusion-tolerant replication system
 - A replication engine with **performance guarantees while under attack**
 - **Defense across space** (i.e. dynamic diversity)
 - **Defense across time** (i.e. proactive recovery)
- Support for **large state** applications (e.g. 1 terabyte)
 - Two novel state transfer strategies
 - Reducing latency
 - Reducing bandwidth consumption
- **Theoretical model** that computes the resiliency of the system over its lifetime
 - Based on the rejuvenation rate, the size of the state, the number of replicas, and the strength of a single replica
 - Model assumes diversity provides complete independence

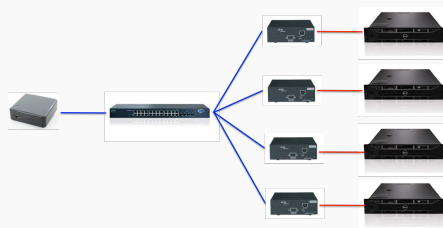
Dynamic Diversity [Homescu2013]

- **MultiCompiler** from UC Irvine (<https://github.com/secaresystemslab>)
 - NOP insertion
 - stack padding
 - shuffling the stack frames
 - substituting instructions for equivalent instructions
 - randomizing the register allocation
 - randomizing instruction scheduling
- Generate different versions of the program starting from its bitcode (no source code required)

Proactive Recovery

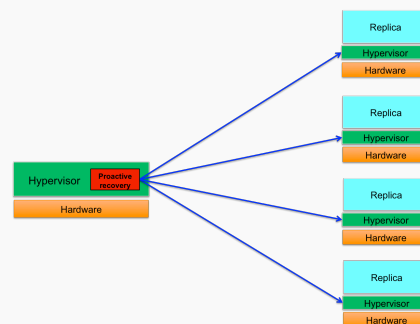
- A component trusted to periodically initiate proactive recovery in a round robin manner by **rejuvenating** a replica from a **clean state**
- Each correct replica completes recovery before the beginning of the rejuvenation of the next replica
- The system may not be available if the f replica fails and a correct replica is rejuvenating.
- We solve this problem by adding more replicas in the system
 - $3f+2k+1$ replicas as in [Sousa2010], with k replicas that rejuvenate at the same time

Physical and Virtualized Approaches



Proactive recovery logic runs in an isolated Next Unit of Computing (NUC). Periodically, the NUC activates a remote power switch, which cycles the power to restart the server that hosts a Prime replica, rebooting a fresh copy from a read-only device

Proactive recovery runs in a hypervisor installed in an isolated server. Periodically a replica is refreshed by instantiating a new virtual machine.



Proactive Recovery Operation Sequence

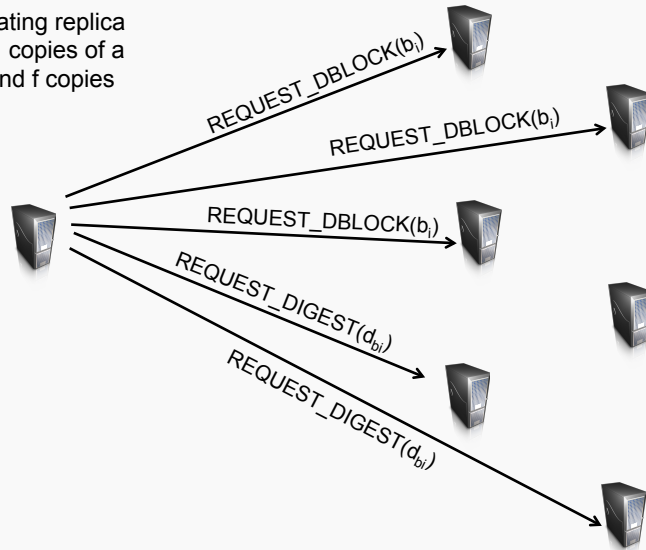
- Replica rejuvenation
 - The replica is restarted periodically from a fresh copy of OS and application code from read-only memory
 - Getting a random number from the Trusted Platform Module (TPM) and use of fine-grained diversity
- Session key replacement
 - If the replica was malicious, its private key can be used to forge messages
 - Session key is based on the TPM
- State validation
- State transfer if needed
- Client updates transfer

State transfer

- The state transfer protocol has to be efficient
 - A compromised replica completes recovery quickly
 - Replicas can be rejuvenated more often
 - The adversary does not have enough time to compromise more than f replicas
- Two strategies
 - Reducing latency
 - Reducing bandwidth usage in the best case
- The state is logically partitioned into data blocks of fixed size
- Assumption: the adversary totally compromises the state (i.e. all data blocks)

State transfer – Reducing latency

The rejuvenating replica requests $f+1$ copies of a data block and f copies of its digest



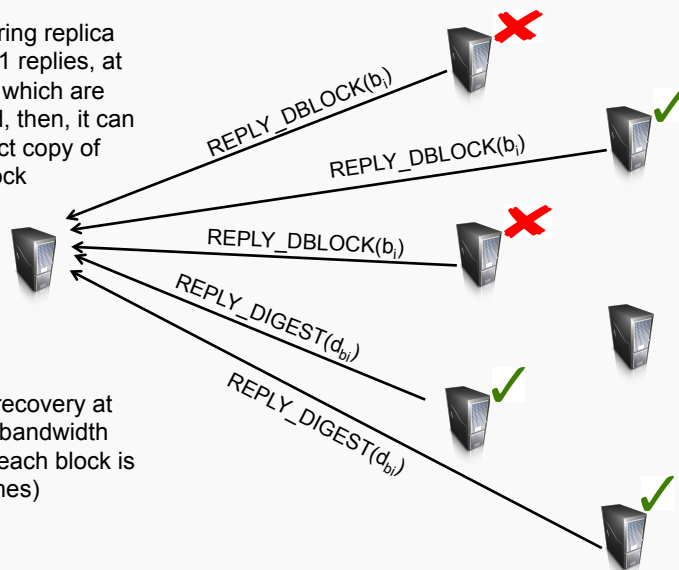
Yair Amir

Fall 16 / Lecture 8

43

State transfer – Reducing latency

The recovering replica collects $2f+1$ replies, at least $f+1$ of which are correct, and, then, it can find a correct copy of the data block



Fast state recovery at the cost of bandwidth overhead (each block is sent $f+1$ times)

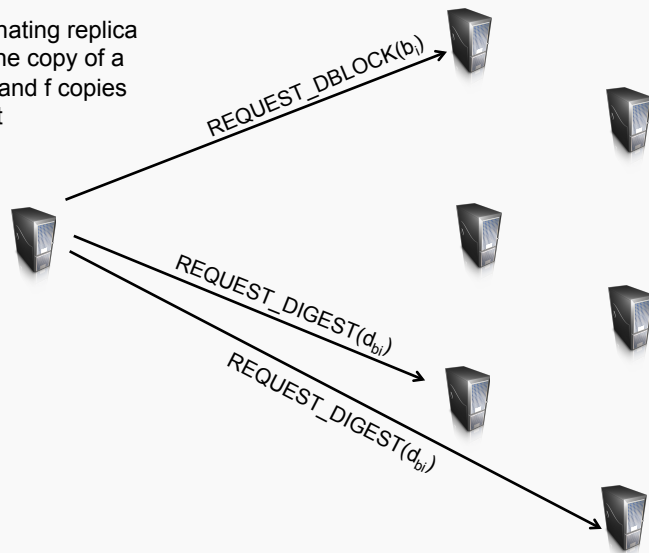
Yair Amir

Fall 16 / Lecture 8

44

State transfer – Reducing bandwidth usage

The rejuvenating replica requests one copy of a data block and f copies of its digest



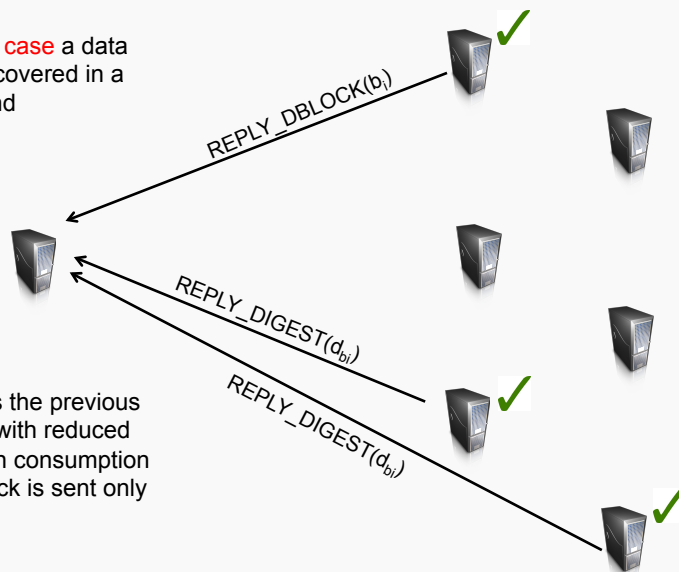
Yair Amir

Fall 16 / Lecture 8

45

State transfer – Reducing bandwidth usage

In the **best case** a data block is recovered in a single round



As fast as the previous strategy, with reduced bandwidth consumption (each block is sent only once)

Yair Amir

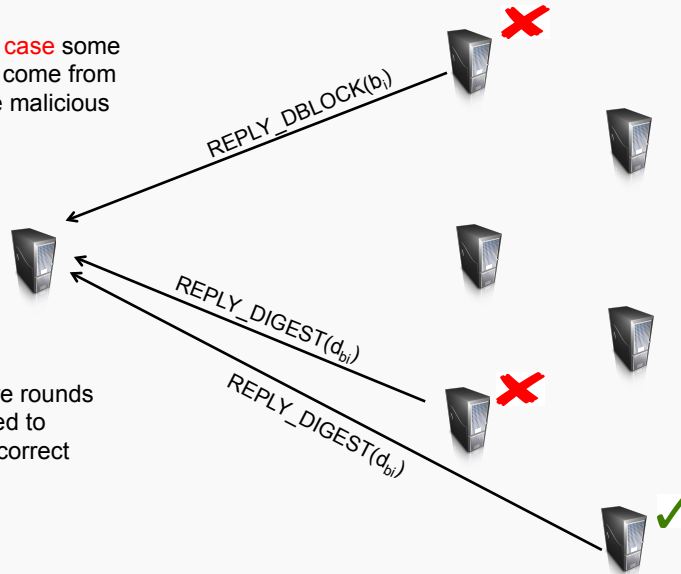
Fall 16 / Lecture 8

46

State transfer – Reducing bandwidth usage

In the **worst case** some replies may come from one or more malicious replicas

Some more rounds are required to retrieve a correct data block



State transfer – Reducing bandwidth usage

- We define two variants to retrieve a correct data block in the presence of incorrect replies
- **Variant 1**
 - The recovering replica keeps requesting a copy of the data block at a time until a correct copy is found (at most $f-1$ additional requests)
- **Variant 2**
 - The recovering replica requests f additional copies of the data block in a single round
 - The recovering replica can find a correct copy among $2f+1$ replies ($f+1$ copies of the same block and f digests)
- We blacklist the senders of invalid replies
 - **The impact of malicious replicas is negligible**

State transfer - Experimental results

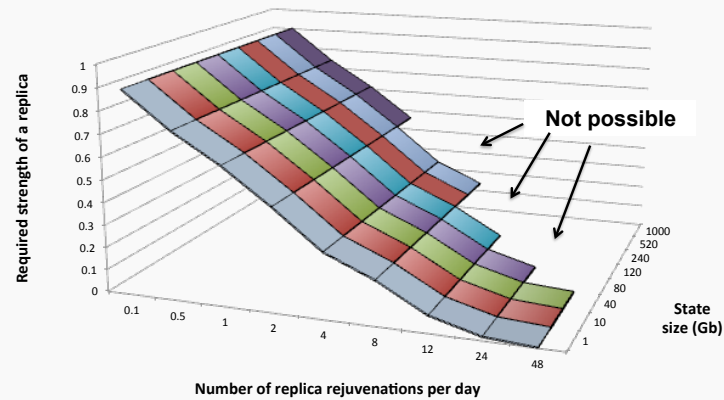
- Time taken to validate and transfer the state (if compromised) after rejuvenation
- The state is fragmented in blocks of fixed size (1 Mbyte)
- Data blocks are transferred in parallel (5 at a time)

state size	state reading	state transfer		
		4 replicas	7 replicas	10 replicas
1 Gb	9 sec	36 sec	25 sec	23 sec
10 Gb	1 m, 27 sec	6 m	4 m	4 m
40 Gb	5 m, 47 sec	24 m	15 m	15 m
80 Gb	11 m, 30 sec	48 m	31 m	31 m
120 Gb	17 m, 15 sec	1 h, 12 m	48 m	48 m
240 Gb	34 m, 30 sec	2 h, 24 m	1 h, 38 m	1h, 36 m
520 Gb	1 h, 14 m	5 h, 9 m	3 h, 28 m	3 h, 24 m
1 Tb	2 h, 24 m	9 h, 50 m	6 h, 17 m	6 h, 17 m

Theoretical model: Resiliency through Proactive Recovery

- A model to compute how strong a replica should be to meet the desired confidence in the system over its lifetime (e.g. 30 years)
- Assumptions
 - No more than f simultaneous failures
 - Replicas get compromised independently (due to diversity)
- Input parameters
 - Number of replicas: $3f+1$
 - Desired confidence in the system over its lifetime
 - Rejuvenation rate (i.e. number of rejuvenations per day)
 - System lifetime
- Output
 - Required strength of a replica (i.e. probability that a replica remains correct over a year)

Replica strength to get 95% of confidence over 30 years (7 replicas system – can sustain 2 compromised replicas)



The strength of a replica – the probability that a replica will not be compromised over a year

Can be estimated with CERT alerts, bug reports, or other historical information, as in [Garcia2011]

Required Replica strength to get 95% of confidence over 30 years (40 Gigabytes state size)

