CS 3551: Advanced Topics in Distributed Information Systems - Building Dependable Infrastructure

Day 3: “The Bedrock of Byzantine Fault Tolerance: A Unified Platform for BFT Protocols Analysis, Implementation, and Experimentation”

Dr. Amy Babay, Fall 2024

University of Pittsburgh

Department of Computer Science

School of Computing and Information

The Problem

- There are A LOT of Byzantine Fault Tolerant (BFT) State Machine Replication (SMR) protocols

- This makes it difficult for system designers to determine which protocol is *best* for their needs

Contribution

- **Bedrock** is a tool to enable system designers to understand how different protocols relate to each other and compare them both theoretically and empirically

- Bedrock platform supports:
  - **Analysis:** provides “design space” for BFT protocols that summarizes the ways protocols can differ from each other
  - **Implementation:** provides a domain specific language (DSL) for implementing BFT protocols by specifying the choices they make within the design space, plus roles, phases, states, and message exchanges
  - **Experimentation:** provides a common implementation base and deployment environment for fair comparisons

Approach

- Surveys existing BFT protocols and identifies the important dimensions in which they differ to create the Bedrock *design space*

Table 1: Comparing selected BFT protocols based on different dimensions of Bedrock design space

Protocol	E1. Nodes	E2. Topo.	E3. Auth.	E4. Timers	P1. Strategy	P2. Phases	P3. V-change	P5. Rec.	P6. Client	Q1. Fair.	Q2. Load.	Design Choices
PBFT [73]	$3f+1$	clique	MAC ‖ Sign	τ_1, τ_2, τ_8	pessimistic	3	stable	pro.	Req.	□	□	(11)
Zyzzyva [157]	$3f+1$	star	MAC ‖ Sign	τ_1, τ_2	optimistic (spec): a_1, a_2	1 (3)	stable	-	Rep.	□	□	8, (11)
Zyzzyva5 [157]	$5f+1$	star	MAC ‖ Sign	τ_1, τ_2	optimistic (spec): a_1	1 (3)	stable	-	Rep.	□	□	8, 10, (11)
PoE [135]	$3f+1$	star	MAC ‖ T-Sign	τ_1, τ_2	optimistic (spec): a_2	3	stable	-	Req.	□	□	1, 7, 11
SBFT [131]	$3f+1$	star	T-Sign	τ_1, τ_2, τ_3	optimistic: a_2	3 (5)	stable	-	Req.	□	□	1, 6, 11
HotStuff [252]	$3f+1$	star	T-Sign	τ_1, τ_2	pessimistic	7	rotating	-	Req.	□	□	1, 3, 11
Tendermint [66]	$3f+1$	clique	Sign	$\tau_1, \tau_2, \tau_5, \tau_6$	optimistic: a_6	3	rotating	-	Req.	□	□	4, 11
Themis [149]	$4f+1$	star	T-Sign	τ_1, τ_2, τ_6	pessimistic	$1+7$	rotating	-	Req.	■	□	1, 3, 13, 11
Kauri [202]	$3f+1$	tree	T-Sign	τ_1, τ_2	optimistic: a_3	$7h$	stable*	-	Req.	□	■	(3), 14, 11
CheapBFT [146]	$2f+1$	clique	MAC	τ_1, τ_2	optimistic: a_2	3	stable	-	Req.	□	□	5
FaB [190]	$5f+1$	clique	(Sign)	τ_1, τ_2	pessimistic	2	stable	-	Req.	□	□	2
Prime [24]	$3f+1$	clique	Sign	$\tau_1, \tau_2, \tau_6, \tau_7$	robust	6	stable	-	Req.	◨	□	11, 12
Q/U [5]	$5f+1$	star	MAC	τ_1, τ_2	optimistic: a_4, a_5	1 (3)	stable	-	Rep.	□	□	9, 10
FLB	$5f-1$	clique	Sign	τ_1, τ_2	pessimistic	2	stable	-	Req.	□	□	1, 2, 11
FTB	$5f-1$	tree	T-Sign	τ_1, τ_2	optimistic: a_3	$3h$	stable	-	Req.	□	■	1, 2, 14, 11

Hint: "T-Sign": threshold signatures, "Req": requester client, "Rep": repairer client, "Pro": proactive recovery. The number of phases in the slow path of protocols is shown in parentheses. While Kauri is implemented on top of HotStuff, it does not use rotating leaders. Prime provides partial fairness.

Table omits P4 (Checkpointing) and Performance Optimizations (Appendix)

Approach

- Surveys existing BFT protocols and identifies the important dimensions in which they differ to create the Bedrock *design space*

Protocol structure

P1. Commitment strategy
P2. Number of commitment phases
P3. View-change
P4. Checkpointing
P5. Recovery
P6. Types of clients

Environmental Settings

E1. Number of replicas
E2. Communication topology
E3. Authentication
E4. Responsiveness, synchronization, and timers

Quality of Service

Q1. Order-fairness
Q2. Load balancing

Performance Optimization

O1. Out-of-order processing
O2. Request pipelining
O3. Parallel ordering
O4. Parallel execution
O5. Read-only requests processing
O6. Separating ordering and execution
O7. Trusted hardware
O8. Request/reply dissemination

From the authors' slides: https://www.usenix.org/system/files/nsdi24_slides-amiri.pdf

Approach

- Identifies *design choices* that can transform one valid protocol to another valid protocol at a different point in the design space

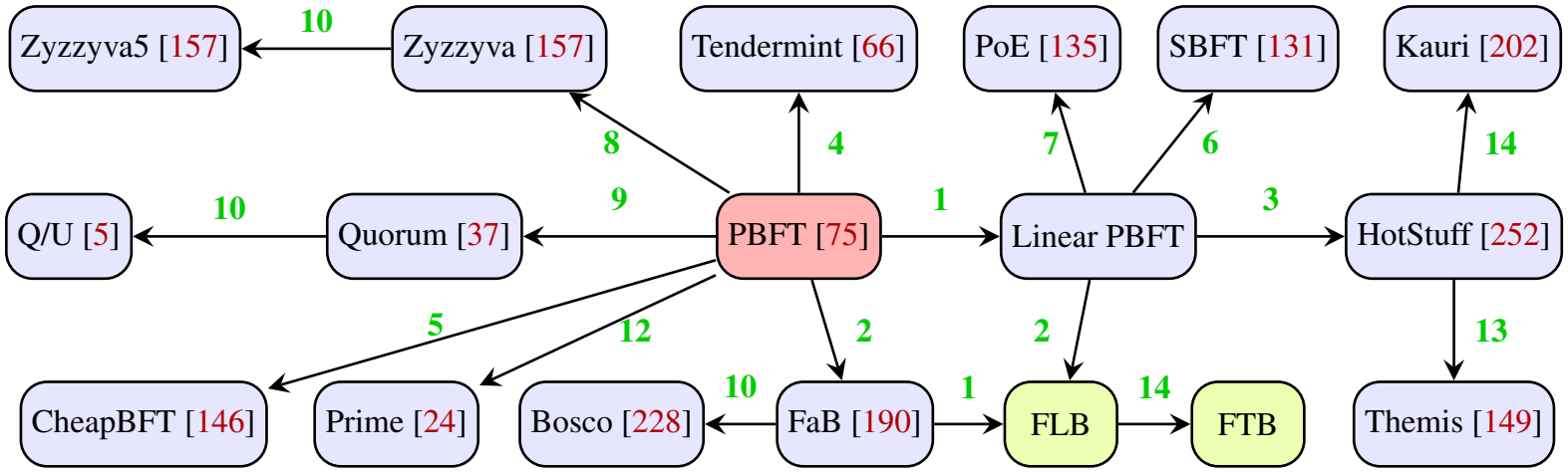

Figure 4: Derivation of protocols from PBFT using design choices

1. Linearization
2. Phase reduction through redundancy
3. Leader rotation
4. Non-responsive leader rotation
5. Optimistic replica reduction
6. Optimistic phase reduction
7. Speculative phase reduction
8. Speculative execution
9. Optimistic conflict-free
10. Resilience
11. Authentication
12. Robust
13. Fair
14. Tree-based Load Balancer

Approach

- Implements the Bedrock platform for specifying BFT protocols, and implements a wide range protocols within that framework

  - The core unit
    - Defines entities, e.g., clients and nodes, and maintains the application logic and data
    - Defines workloads and benchmarks

  The state manager
  - Enables the core unit to track the states and transitions of each entity according to the protocol
  - Defines a domain-specific language (DSL) to rapidly prototype BFT protocols

  The plugin manager
  - Implements protocol-specific behaviors that cannot be handled by the protocol config
  - Enables users to define their own dimensions/values or to update existing dimensions without requiring changes to the platform code or rebuilding the platform binaries

  The run-time unit
  - Manages the run-time execution
  - E.g., manages benchmarks, setups all entities, enables plugins to run, reports results

From the authors' slides: https://www.usenix.org/system/files/nsdi24_slides-amiri.pdf

Results

- Claim: Bedrock provides new insights into the space of possible BFT SMR protocol designs
- Evidence: two new protocols
  - Fast Linear BFT (FLB):
    - **PBFT base**: 3f+1 replicas, 3 ordering phases (linear, quadratic, quadratic)
    - **Phase reduction through redundancy**: use 5f-1 replicas to reduce message exchange to 2 phases (linear, quadratic)
    - **Linearization**: split quadratic message exchange into 2 linear exchanges via collect-broadcast pattern
  - Fast Tree-based BFT (FTB):
    - **FLB + Tree-based Load Balancer / Kauri with modifications**

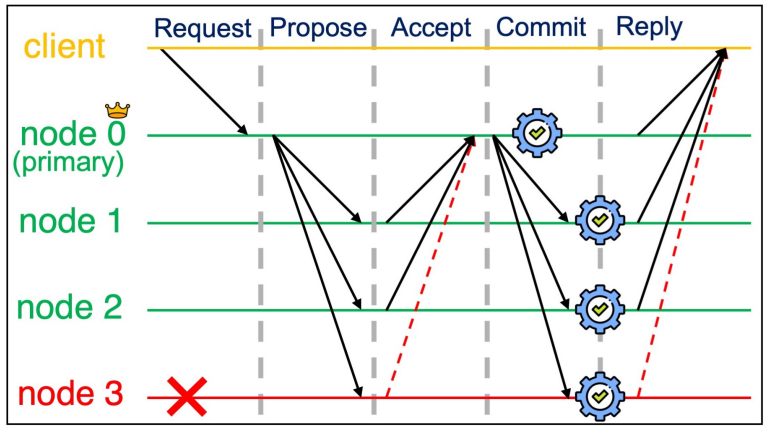

Figure 24: FLB

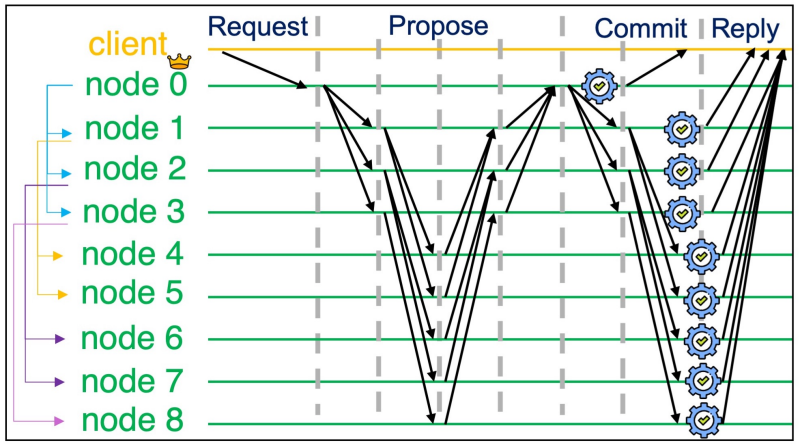

Figure 25: FTB

Results

- Claim: Bedrock makes it easier to implement BFT protocols

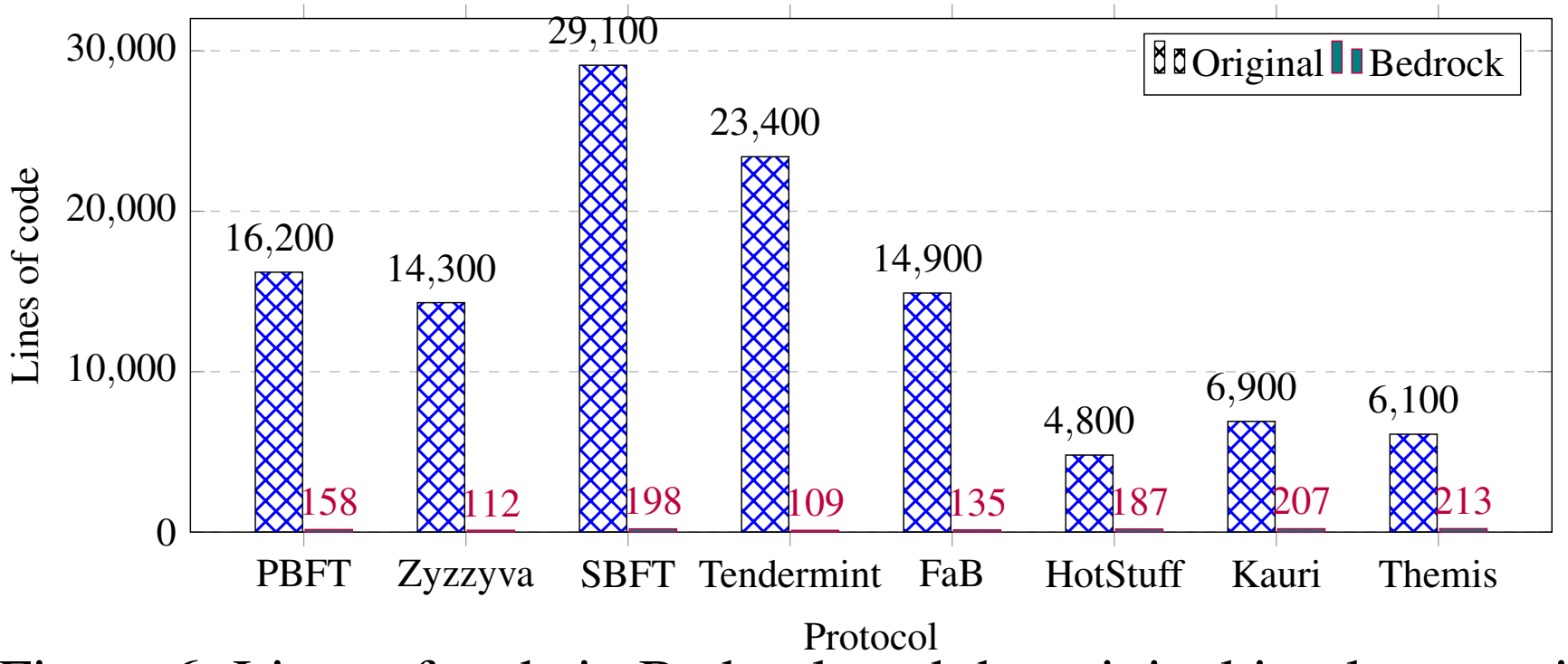

Figure 6: Lines of code in Bedrock and the original implementation

Results

- Claim: Bedrock makes it easier to empirically compare the performance of different BFT SMR protocols

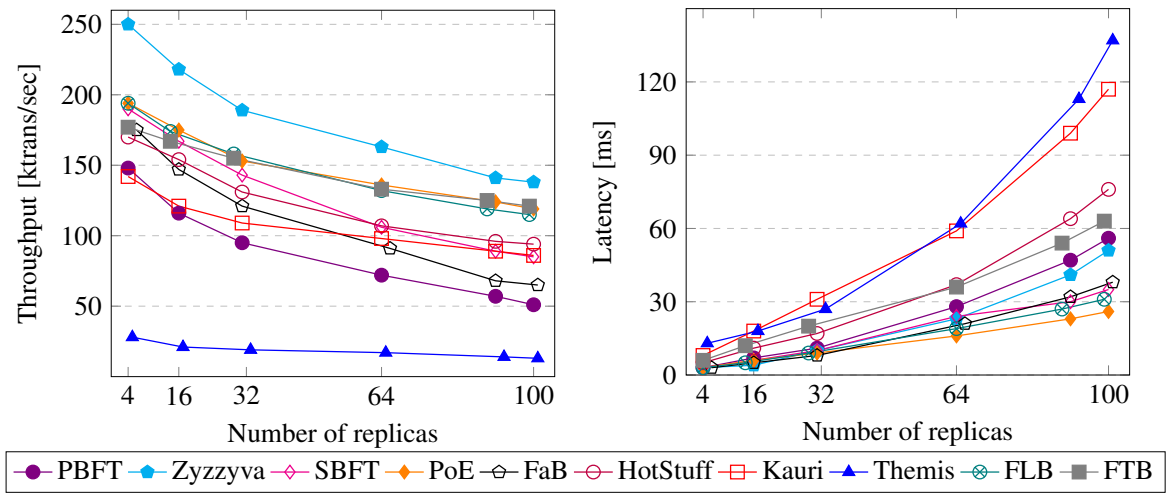

Figure 7: Performance with different number of replicas

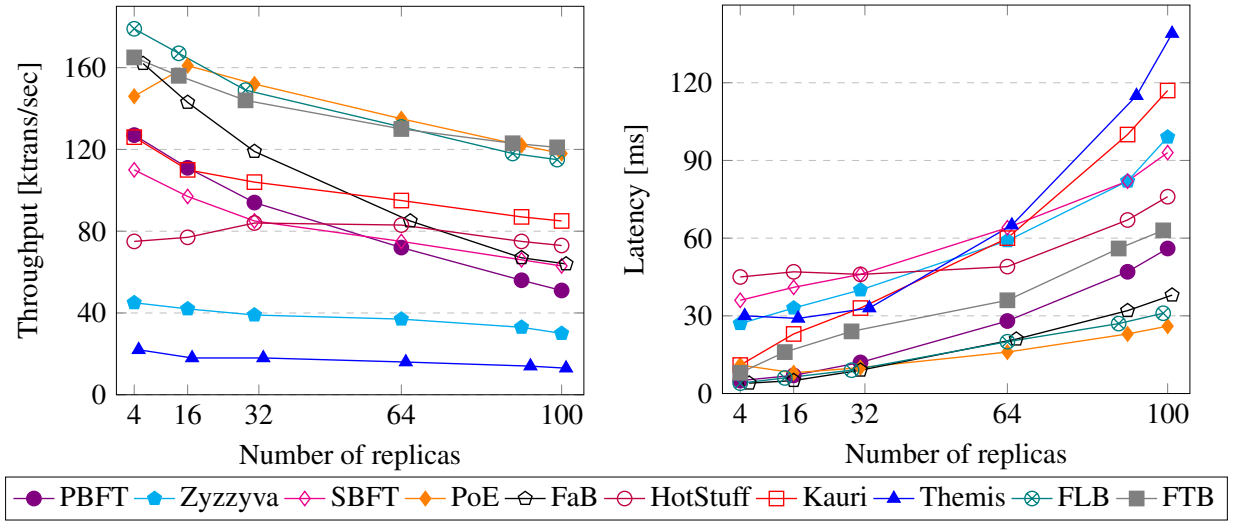

Figure 9: Performance with faulty backups