TinyBFT: Byzantine Fault-Tolerant Replication for Highly Resource-Constrained Embedded Systems

30th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2024)

05/15/2024

Harald Böhm Tobias Distler Peter Wägemann

Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)

Lehrstuhl für Informatik 4
Systemsoftware

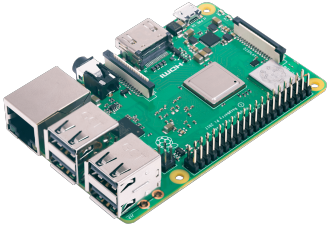
FAU

BMWK Grant Number 20E2122B
(LuFo VI-2, BALu)
DFG Project Number 502947440
(Watwa)

BFT Replication Systems Are Getting Smaller

BFT Replication Systems Are Getting Smaller

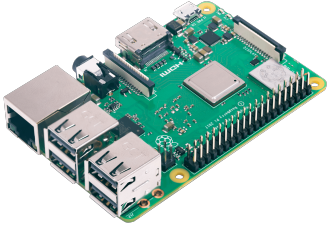
$\gg$32 GB RAM



$\sim$4 GB RAM

BFT Replication Systems Are Getting Smaller

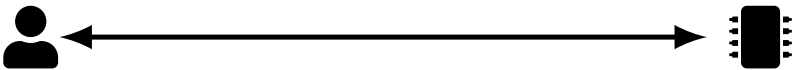
$\gg$32 GB RAM



$\sim$4 GB RAM

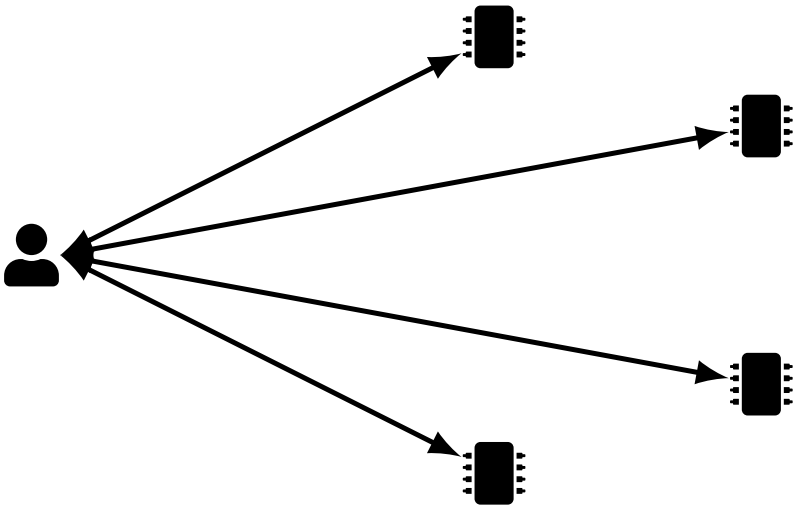
< 1 MB RAM

Replication

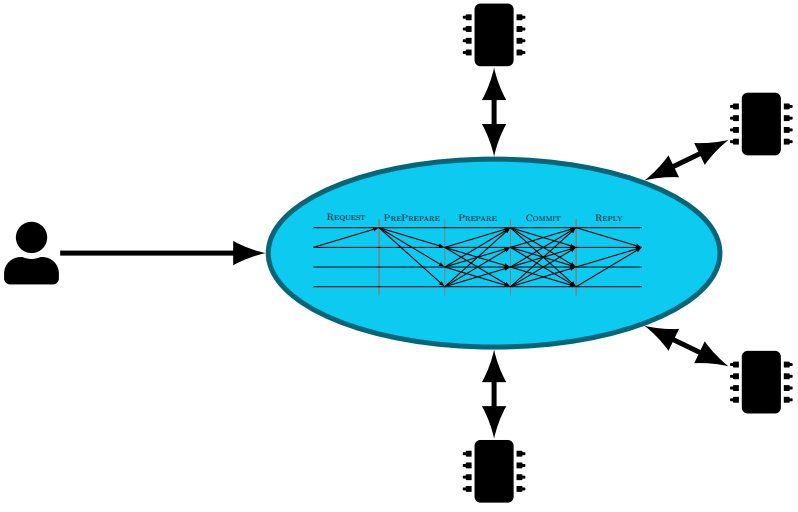


Replication

Replication



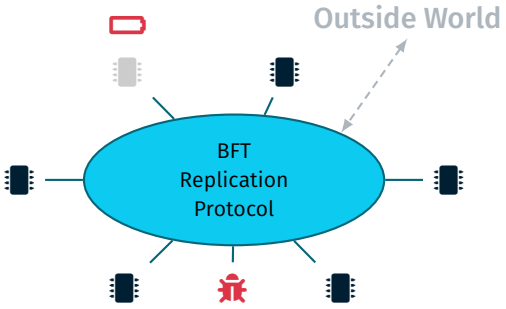
Replication



System Model and Fault Model

System Model

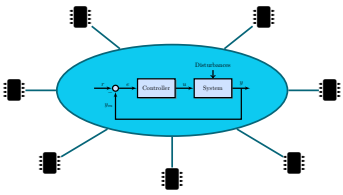
- Limited RAM **(< 1MB)**
- Limited energy
- Wireless communication
- Response times up to a few sec



Fault Model

- Crashes
- Byzantine faults
- $n \geq 3f + 1$

Why do we bother?



1. Resilient control applications [3]
2. Blockchain-based data recording
3. Eco-friendly room booking

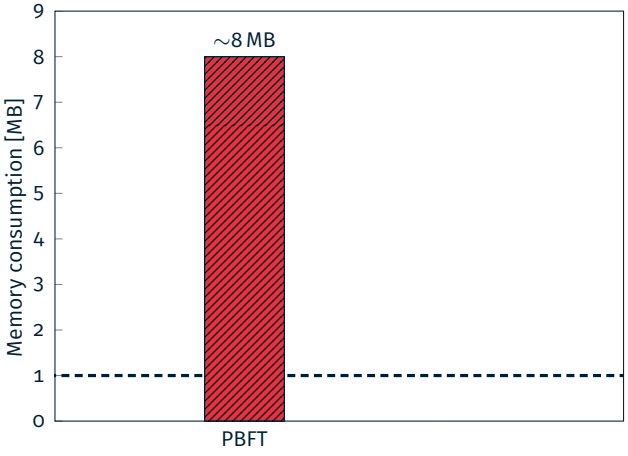
Existing Replication Software Is Not Ready For These Systems

- Implementations target *server-grade* hardware
- Use of large runtime environments (e.g., JRE)
- **Memory plays (almost) no role in design**

< 1 MB RAM

Existing Replication Software Is Not Ready For These Systems

< 1 MB RAM



Existing Replication Software Is Not Ready For These Systems

< 1 MB RAM

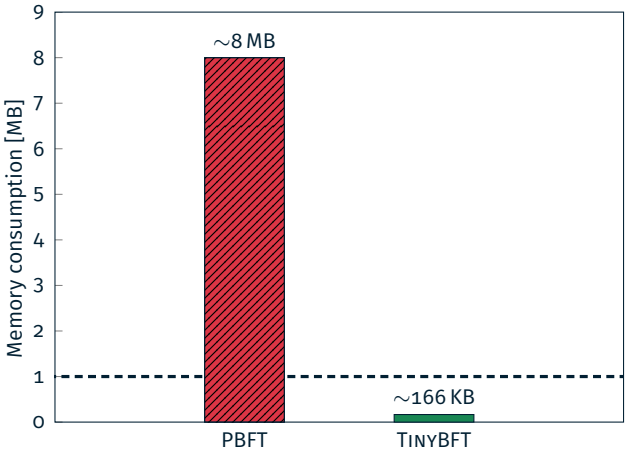


Table of Contents

Small Devices, Unique Challenges

Three key challenges running BFT replication in target environment:

Challenge 1

Selecting the right protocol design

Challenge 2

Reducing protocol's memory footprint

Challenge 3

Exploiting additional (non-volatile) memory modules

Small Devices, Unique Challenges

Three key challenges running BFT replication in target environment:

Challenge 1 ✓

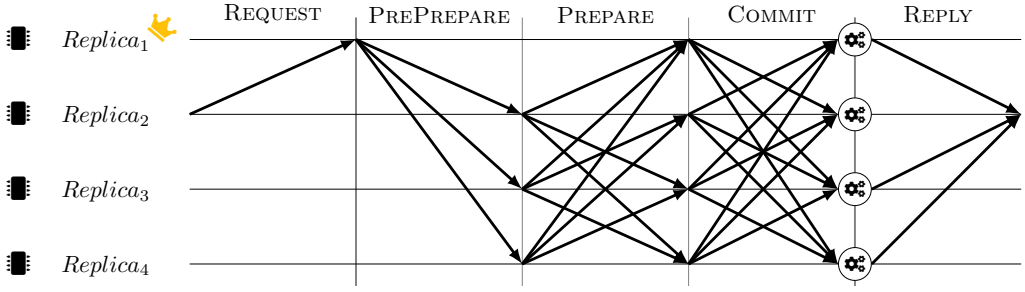
Carefully selected PBFT as best fit for our use-case

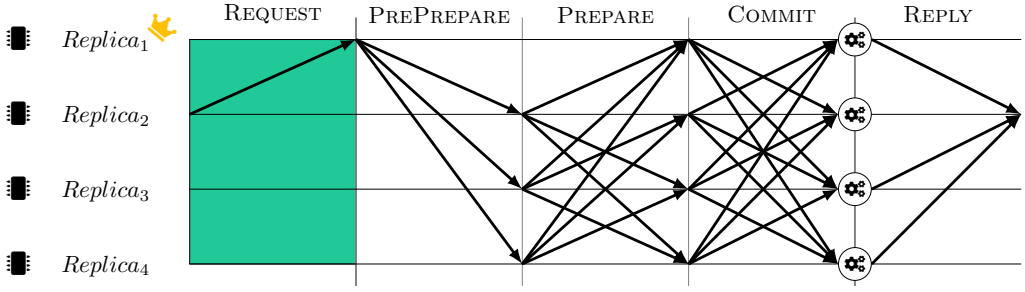
Challenge 2

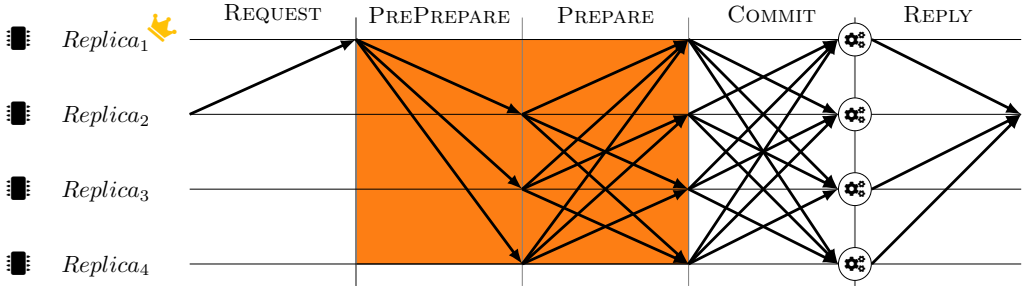
Reducing protocol's memory footprint

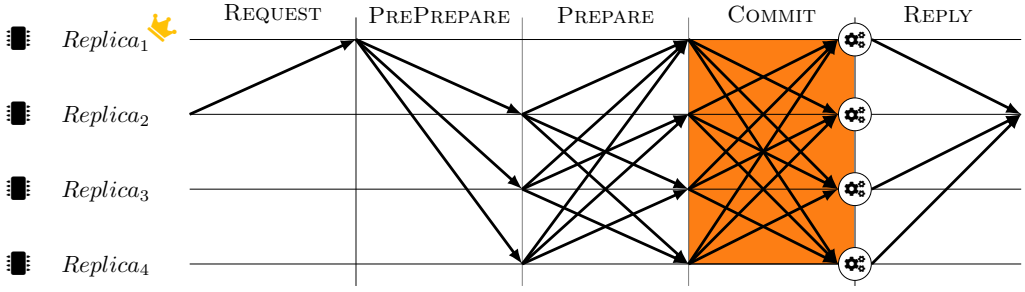
Challenge 3

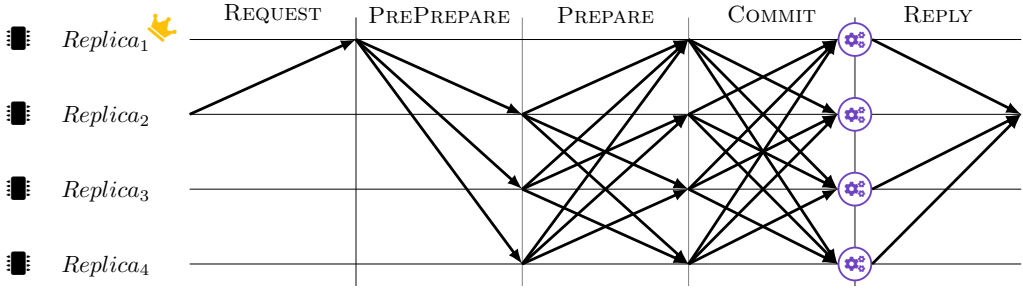
Exploiting additional (non-volatile) memory modules

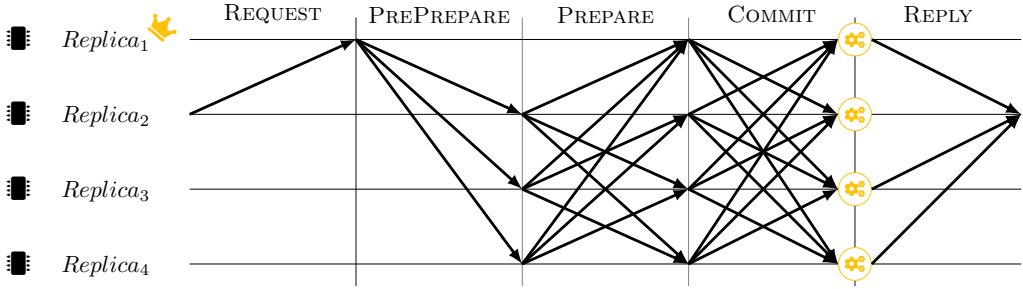

Request
PrePrepare
Prepare
Commit
Reply
$Replica_1$
$Replica_2$
$Replica_3$
$Replica_4$

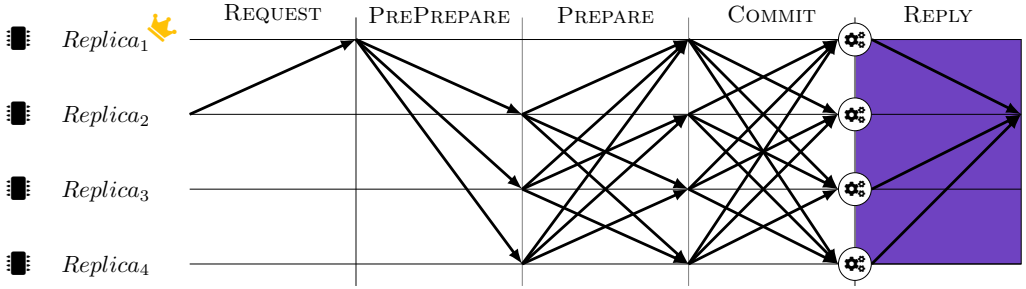

Request
PrePrepare
Prepare
Commit
Reply
$Replica_1$
$Replica_2$
$Replica_3$
$Replica_4$


Request
PrePrepare
Prepare
Commit
Reply
$Replica_1$
$Replica_2$
$Replica_3$
$Replica_4$

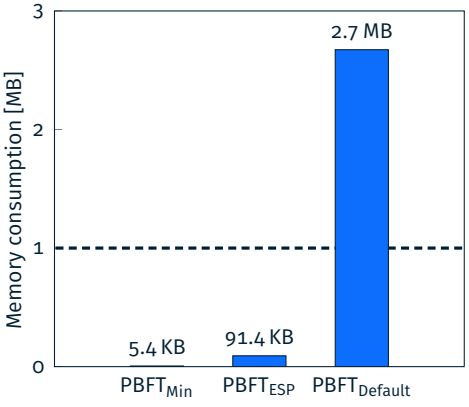

Request
PrePrepare
Prepare
Commit
Reply
$Replica_1$
$Replica_2$
$Replica_3$
$Replica_4$


Request
PrePrepare
Prepare
Commit
Reply
$Replica_1$
$Replica_2$
$Replica_3$
$Replica_4$


Request
PrePrepare
Prepare
Commit
Reply
$Replica_1$
$Replica_2$
$Replica_3$
$Replica_4$


Request
PrePrepare
Prepare
Commit
Reply
$Replica_1$
$Replica_2$
$Replica_3$
$Replica_4$

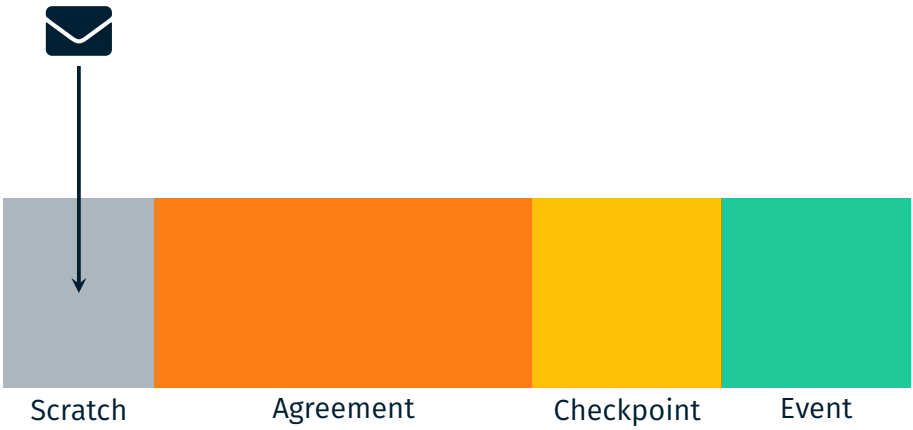
- Theoretical memory demand
- Assess demand of configurations
- Baseline for implementations



TinyBFT: Optimizations


Scratch
Agreement
Checkpoint
Event


Scratch
Agreement
Checkpoint
Event


Scratch
Agreement
Checkpoint
Event

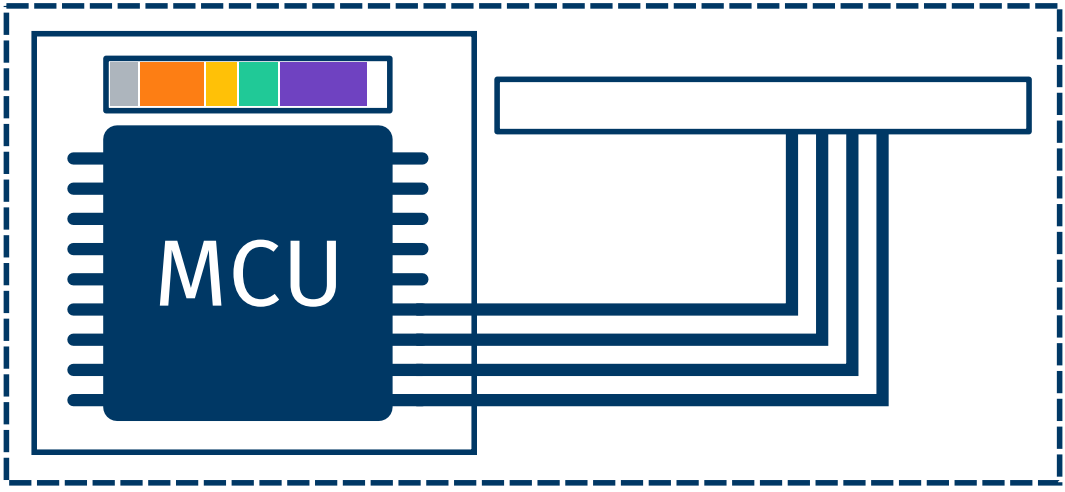

Scratch
Agreement
Checkpoint
Event

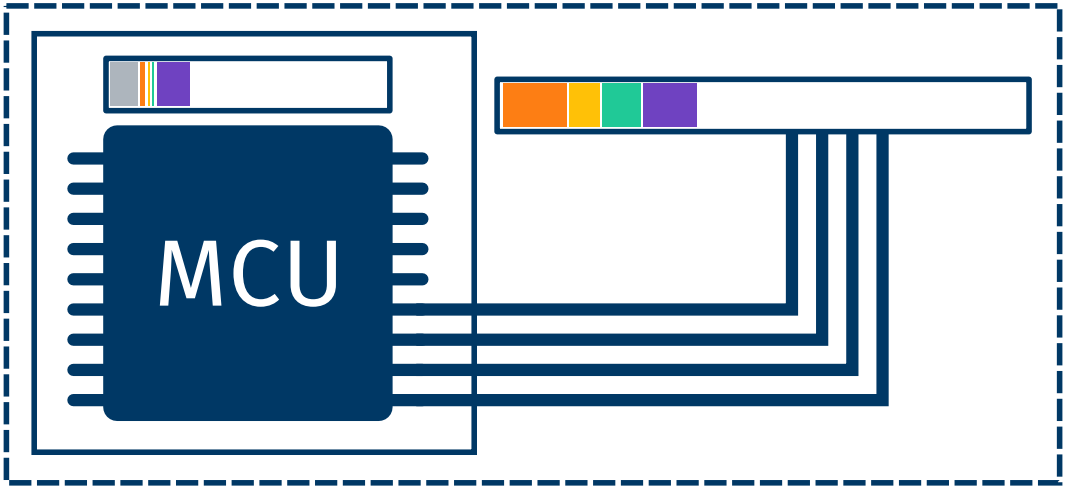

Scratch
Agreement
Checkpoint
Event

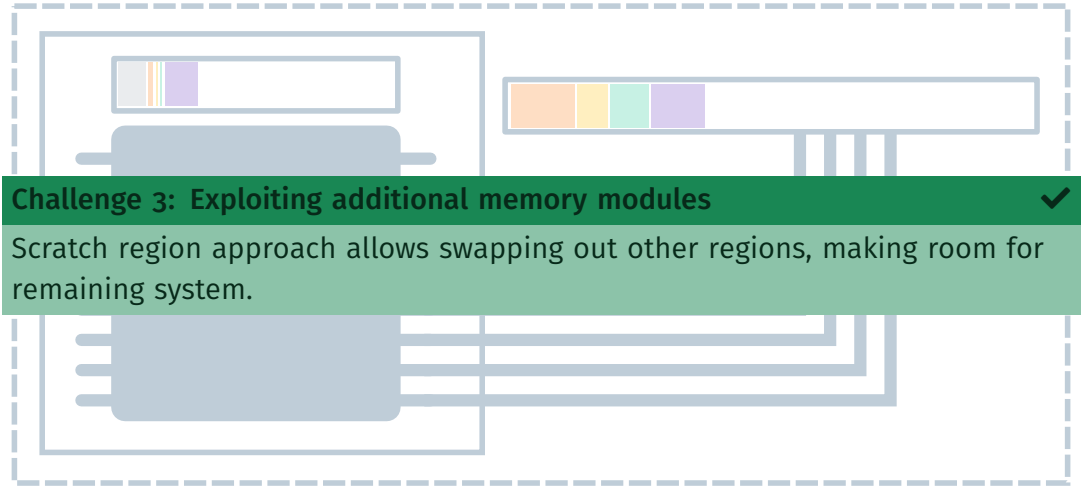

Scratch
Agreement
Checkpoint
Event


Scratch
Agreement
Checkpoint
Event


MCU


MCU

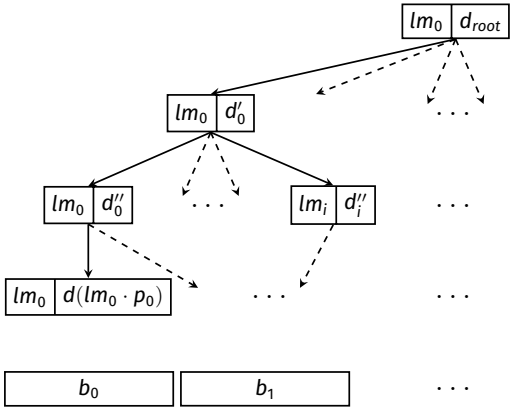

MCU



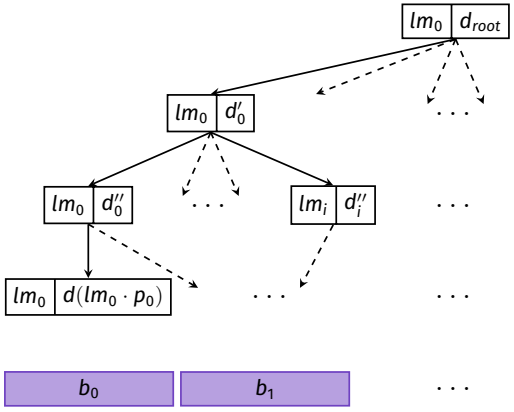
Challenge 3: Exploiting additional memory modules ✓

Scratch region approach allows swapping out other regions, making room for remaining system.

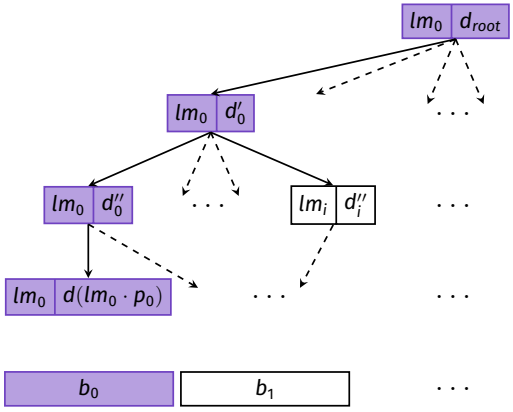
- State divided into blocks
- Tree structure to track changes
- Enables efficient state transfer
- PBFT's tree is fixed size
- TinyBFT's tree adjusted to state size



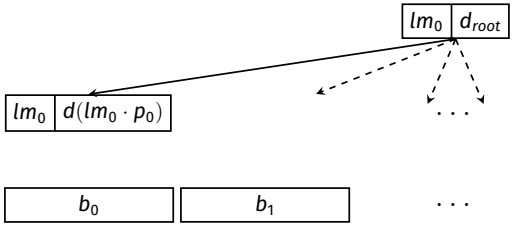
- State divided into blocks
- Tree structure to track changes
- Enables efficient state transfer
- PBFT's tree is fixed size
- TinyBFT's tree adjusted to state size



- State divided into blocks
- Tree structure to track changes
- Enables efficient state transfer
- PBFT's tree is fixed size
- TinyBFT's tree adjusted to state size

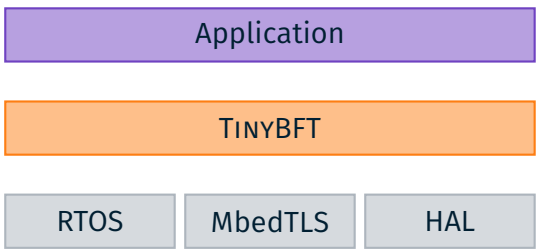


- State divided into blocks
- Tree structure to track changes
- Enables efficient state transfer
- PBFT's tree is fixed size
- TinyBFT's tree adjusted to state size



- Flexible configuration mechanism
- Tailor protocol to specific needs
- Sanity checks during compile time

- Based on PBFT prototype [1, 2]
- MBedTLS for Cryptography
- HAL for hardware timers



Evaluation

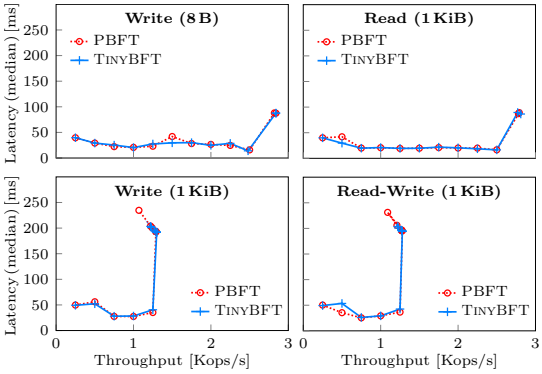
Protocol Part	PBFT (Intel)	TinyBFT (ESP32-C3)	Regions
Client handling	5,952 B	11,460 B	
Agreement	1,320 B	11,908 B	
Execution	7,395,588 B	115,780 B	
View	1,992 B	7,920 B	
Other	554,984 B	18,447 B	
Total	**7,959,872 B**	**165,515 B**	

Protocol Part	PBFT (Intel)	TinyBFT (ESP32-C3)	Regions
Client handling	5,952 B	11,460 B	
View	1,992 B	7,920 B	
Other	554,984 B	18,447 B	
Total	**7,959,872 B**	**165,515 B**	

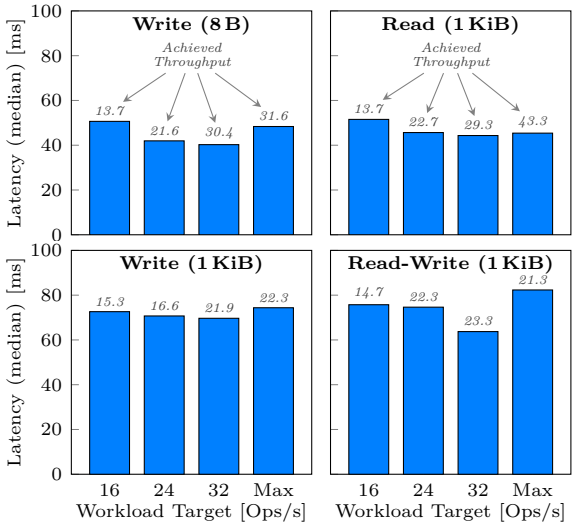
Challenge 2: Reducing PBFT's memory footprint ✓

TinyBFT is able to run on target hardware.

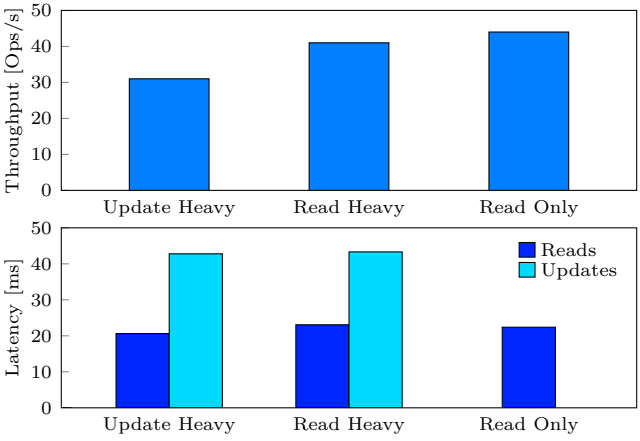
- **Platform:** Intel Xeon (Skylake) (3.6 GHz, 16 GB RAM)
- **Application:** Key-value Store
- $f = 1$



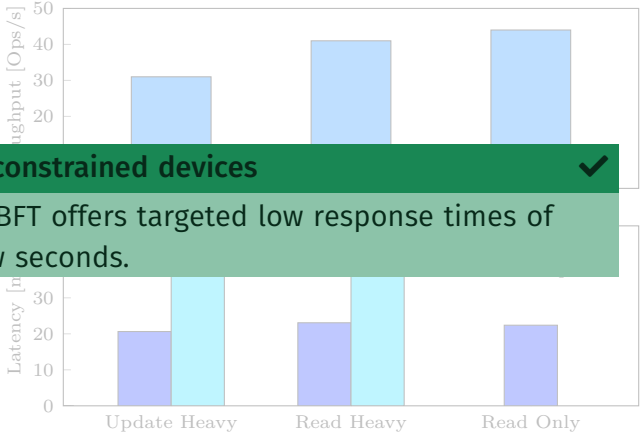
- **Platform:** ESP32-C3 (160 MHz, 400 KB RAM)
- **Application:** Key-value Store
- $f = 1$



- **Platform:** ESP32-C3 (160 MHz, 400 KB RAM)
- **Application:** Key-value Store
- $f = 1$
- YCSB Benchmark
- 256 B Records



- **Platform:** ESP32-C3
- YCSB Benchmark
- 256 B Records



Performance on highly resource-constrained devices ✓

With latencies below 100ms, TinyBFT offers targeted low response times of hundreds of milliseconds to a few seconds.

Conclusion

Conclusion

TinyBFT is the first BFT library for highly resource-constrained systems

Conclusion

TinyBFT is the first BFT library for highly resource-constrained systems

Challenge 1: Selecting the right protocol design ✓

Carefully selected PBFT as best fit for our use-case.

Conclusion

TinyBFT is the first BFT library for highly resource-constrained systems

Challenge 1: Selecting the right protocol design ✓

Carefully selected PBFT as best fit for our use-case.

Challenge 2: Reducing PBFT's memory footprint ✓

Optimizations reduce memory demand by $\approx 97\%$ to ≈ 166 KB.

Conclusion

TinyBFT is the first BFT library for highly resource-constrained systems

Challenge 1: Selecting the right protocol design ✓

Carefully selected PBFT as best fit for our use-case.

Challenge 2: Reducing PBFT's memory footprint ✓

Optimizations reduce memory demand by $\approx 97\%$ to ≈ 166 KB.

Challenge 3: Exploiting additional memory modules ✓

Scratch region approach allows swapping out other regions.

Conclusion

TinyBFT is the first BFT library for highly resource-constrained systems

Challenge 1: Selecting the right protocol design ✓

Carefully selected PBFT as best fit for our use-case.

Challenge 2: Reducing PBFT's memory footprint ✓

Optimizations reduce memory demand by $\approx 97\%$ to ≈ 166 KB.

Challenge 3: Exploiting additional memory modules ✓

Scratch region approach allows swapping out other regions.

Conclusion

TinyBFT is the first BFT library for highly resource-constrained systems

Challenge 1: Selecting the right protocol design

Carefully selected PBFT as best fit for our use-case.

Challenge 2: Reducing PBFT's memory footprint ✓

Optimizations reduce memory demand by $\approx 97\%$ to ≈ 166 KB.

Challenge 3: Exploiting additional memory modules ✓

Scratch region approach allows swapping out other regions.

Thank you for your attention! Questions?

Source available at: `https://gitos.rrze.fau.de/tinybft`

Appendix

Protocol Part	PBFT (Intel)	TinyBFT (Intel)	TinyBFT (ESP32-C3)
Client handling	5,952 B	12,428 B	11,460 B
Agreement	1,320 B	11,684 B	11,908 B
Execution	7,395,588 B	110,448 B	115,780 B
View	1,992 B	7,688 B	7,920 B
Other	554,984 B	40,320 B	18,447 B
Total	**7,959,872 B**	**181,604 B**	**165,515 B**

References (1)


M. Castro and B. Liskov.
Practical Byzantine fault tolerance.
In *Proceedings of the 3rd Symposium on Operating Systems Design and Implementation (OSDI '99)*, pages 173–186, 1999.


M. Castro and B. Liskov.
Practical byzantine fault tolerance and proactive recovery.
ACM Transactions on Computer Systems (TOCS), 20(4):398–461, 2002.


A. Gujarati, N. Yang, and B. B. Brandenburg.
In-ConcReTeS: Interactive consistency meets distributed real-time systems, again!
In *Proceedings of the 43rd Real-Time Systems Symposium (RTSS '22)*, pages 211–224, 2022.

References (2)