How Far Can Synchronous BFT Consensus Go?

Nenad Milošević
Università della Svizzera italiana (USI)

Consensus

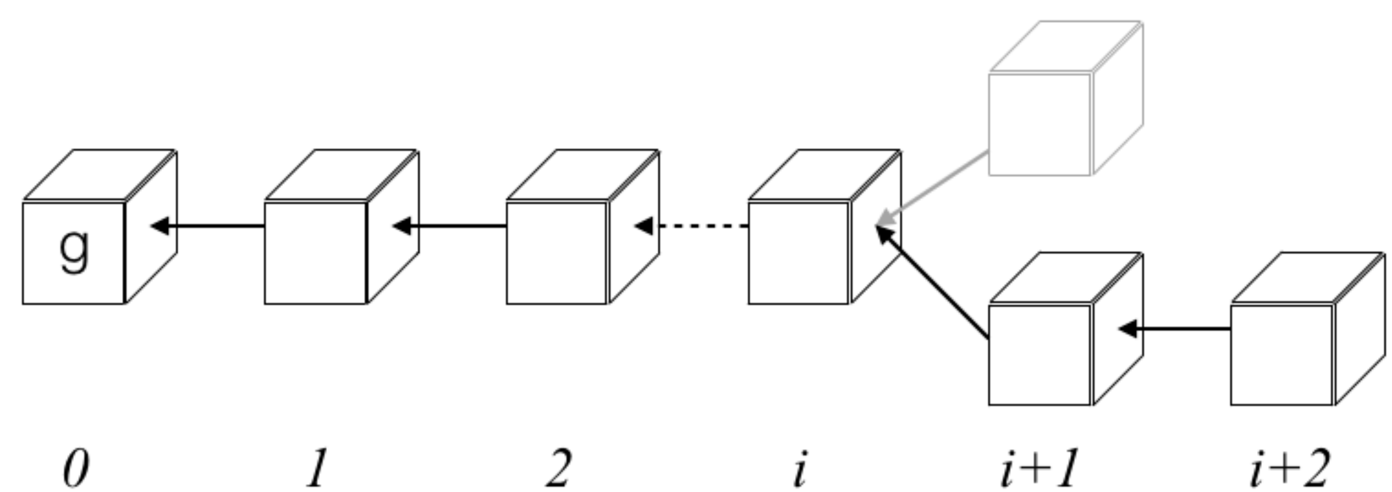
- Distributed computing primitive that allows a set of replicas to agree on a common value, while some of them may fail
  - ✦ Safety: No two honest replicas decide on different values
  - ✦ Liveness: Eventually all honest replicas decide

Consensus and SMR

- At the core of State Machine Replication (SMR)
- Consensus and SMR have been deployed for decades to replicate core components of distributed systems:
  - ✦ Distributed databases
  - ✦ Cloud computing
  - ✦ **Blockchain systems**

Blockchain

- Distributed system where 100s or 1000s of mutually untrusted parties build an immutable, ordered history of transactions/requests
- The history is represented as a data structure called blockchain
  - ✦ Each block has a cryptographic link to the previous block



Consensus meets Blockchain

- Consensus and SMR ensure all replicas maintain a consistent view of the blockchain
- Consensus determines which block should be appended to the blockchain
- Byzantine fault-tolerant (BFT) consensus
  - ✦ Faulty replicas can fail arbitrarily, even be malicious
- New environment:
  - ✦ Large scale (100s or 1000s), global setup (WAN)
  - ✦ Multiple administrative domains
- New environment requires new solutions!

System model

- Set of assumptions about the environment
- Synchrony assumptions
    - ✦ The upper bound on process execution time, Φ
    - ✦ The upper bound on message transmission time, Δ
- They allow proving that if these assumptions are met our protocol will work properly
- FLP: There is no consensus algorithm that can tolerate even 1 crash failure in an asynchronous network!

Partially synchronous BFT consensus

- Partially synchronous system model:
  - The bound on message delay Δ exists, but holds only eventually, after an unknown point in time, called Global Stabilization Time (GST)
- Partially synchronous BFT consensus algorithms:
  - Tendermint, HotStuff
  - Rely on Δ to ensure liveness but not for safety
  - Safe even when messages break Δ, when the network is asynchronous
  - Tolerate less than 1/3 of Byzantine replicas

Synchronous BFT consensus

- Difinity, Sync HotStuff…
- **Tolerate less than 1/2 of Byzantine replicas**
- Mostly of theoretical interest
- Rely on Δ to ensure both safety and liveness
  - ✦ Messages breaking Δ (synchrony violations) can potentially lead to the safety violations
  - ✦ Δ impacts performance, especially latency

Synchronous bound Δ

- Determining Δ requires greater accuracy than for partially synchronous protocols
- Conservative Δ
  - ✦ High percentiles (e.g., 99.99%) or significantly higher values (e.g., 10x observed delays)
  - ✦ Minimizes the risk of synchrony violations, favor correctness
  - ✦ Negatively impacts protocol performance
- **Tradeoff:** Balancing correctness and performance is a key challenge when determining Δ in synchronous systems!

Our goal

- Investigate the tradeoff between correctness and performance when determining Δ for synchronous BFT consensus protocol
- Explore how robust synchronous BFT consensus really are
  - Robustness = ability to maintain correctness under synchrony violations

Our approach

1. We took a BFT consensus algorithm (BoundBFT) proven correct in the synchronous system model
2. Analyzed its execution to understand how synchrony violations can compromise its safety and liveness
3. Studied how malicious replicas can exploit synchrony violations
4. Designed Byzantine attacks based on insights from the analysis 2 and 3
5. Implemented and tested the protocol and attacks to evaluate Δ
6. Selected a Δ value that ensures consensus properties hold under attack

Experimental setup

- USI cluster: 60 machines
- Emulated wide area network
- XFT (OSDI 2016): 3 month long experiment, 6 AWS regions, ping (hping)

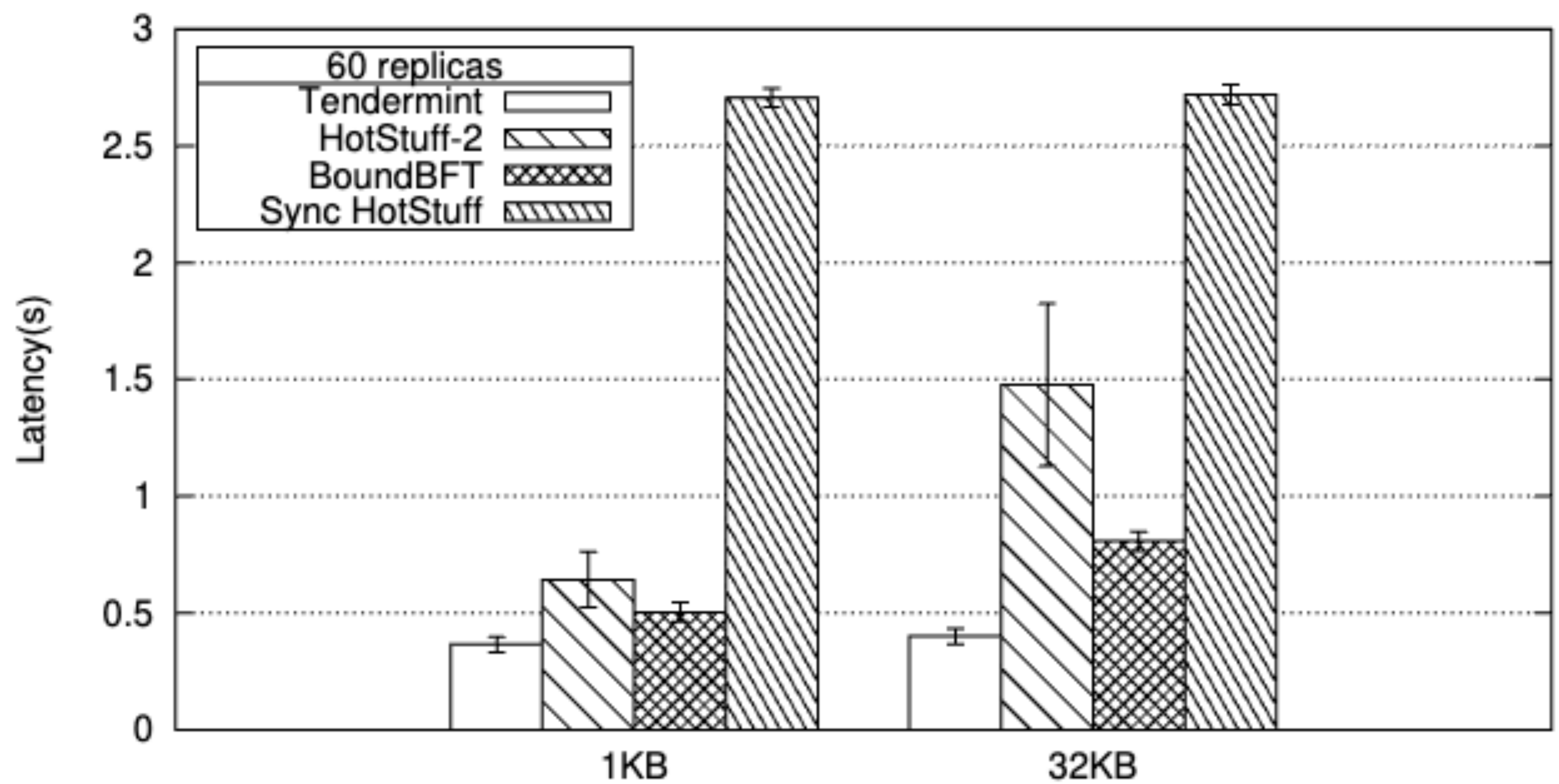
	US East	US West	Europe	Tokyo	Sydney	Sao Paolo
US East	0	44	46	90	134	73
US West	44	0	87	60	93	104
Europe	46	87	0	144	171	117
Tokyo	90	60	144	0	69	197
Sydney	134	93	171	69	0	196
Sao Paolo	73	104	117	197	196	0

- Latencies: 40ms - 200ms

BoundBFT’s Δ - Equivocation attack

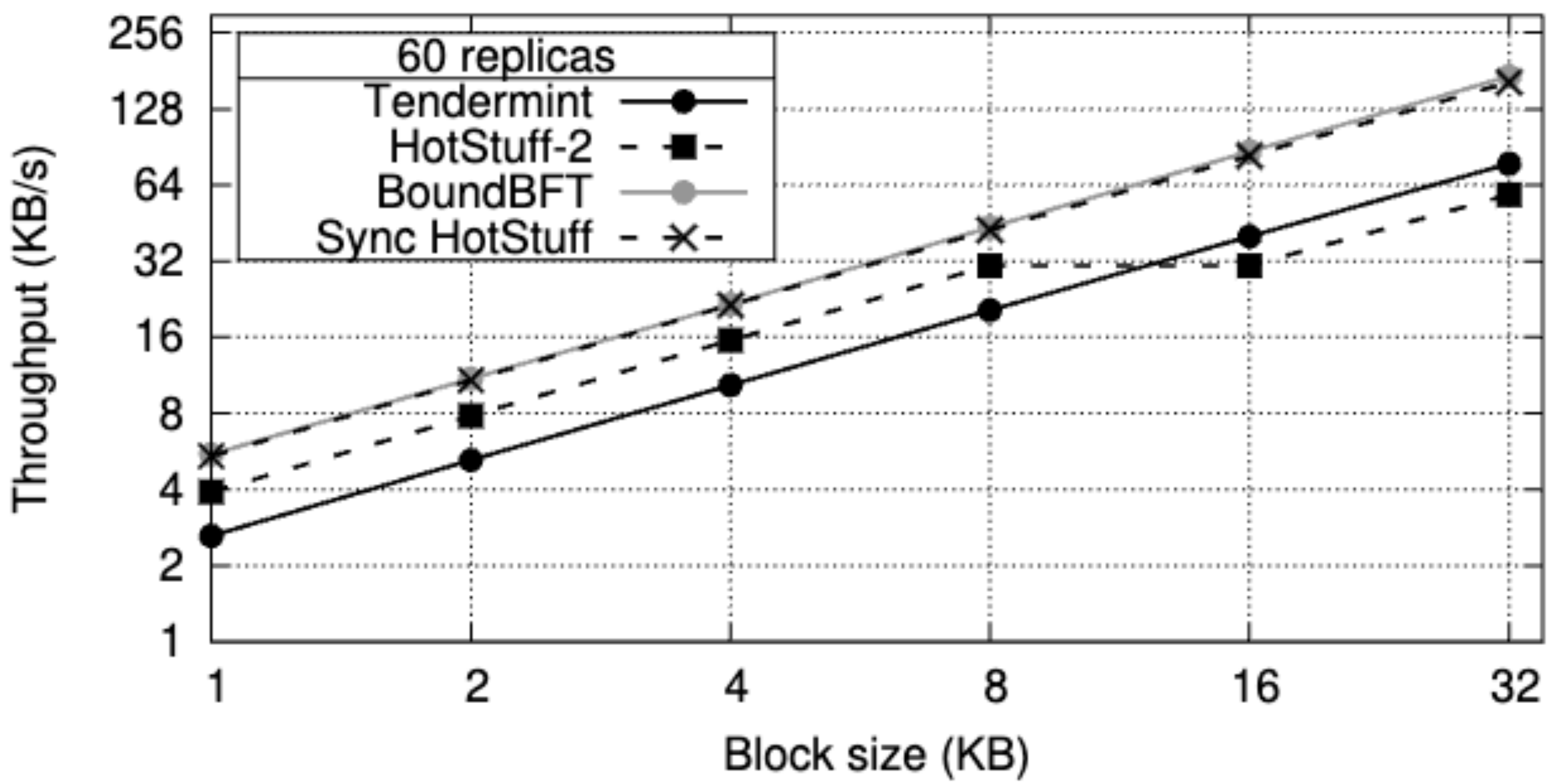
	Δ (ms)	Equivocation attack					
		f=1		f=19		f=29	
		Safety	Liveness	Safety	Liveness	Safety	Liveness
99.99% (XFT) =>	1250	0%	0%	0%	0%	0%	0%
	600	0%	0%	0%	0%	0%	0%
	300	0%	0%	0%	0%	0%	0%
Synchrony violations	150	0%	0%	0%	0%	0%	3%
	100	0%	0%	0%	0%	6%	6%
	50	0%	65%	8%	54%	31%	60%

BoundBFT’s latency



- BoundBFT’s latency for 1KB and 32KB blocks, respectively:
  - 5.4× and 3.4× lower than Sync HotStuff
  - 1.3x and 1.8x lower than HotStuff-2
  - 1.4x and 2x higher than Tendermint

BoundBFT's throughput



- Similar to Sync HotStuff
- Higher than partially synchronous protocols
  - From 1.4x to 3x

Key takeaways

- BoundBFT can tolerate some synchrony violations
- As a result, BoundBFT can operate with a significantly lower Δ than typical conservative estimates
- With this refined Δ, BoundBFT achieves performance comparable to partially synchronous protocols while tolerating more Byzantine failures

Large values

Δ (ms)	Equivocation attack (128KB)					
	f=1		f=19		f=29	
	Safety	Liveness	Safety	Liveness	Safety	Liveness
99.99% (XFT) => 1250	0%	0%	0%	0%	1%	0%
600	0%	0%	0%	0%	6%	0%
300	0%	0%	0%	0%	7%	1%
Synchrony violations 150	0%	1%	0%	1%	14%	29%
Synchrony violations 100	0%	23%	0%	48%	33%	64%
Synchrony violations 50	0%	89%	0%	77%	61%	56%

Study on message delays

- We implemented our own ping program: processes exchange messages and calculate message round trip times (RTT)
- Various message sizes (from 1KB to 1MB)
- Different setups:
  - Single-region
  - Large-machines
  - Cross-region
  - Different-provider
  - Cross-vendor
- The study spawned the period of three months

Key observation

	99.99%			MAX		
	2KB	**128KB**	**Diff**	**2KB**	**128KB**	**Diff**
Single-Region	5.13	120.48	23.49×	10.87	180.10	16.57×
Large-Machines	1.01	3.99	3.94×	6.64	107.34	16.15×
Cross-Region	197.50	1399.00	7.08×	2008.50	7295.50	3.63×
Different-Provider	383.00	4953.50	12.93×	591.50	5879.00	9.94×
Cross-Vendor	1114.00	5976.00	5.36×	4625.50	6558.00	1.42×

- Small messages exhibits **low and stable delays**
- Large messages experience higher and more variable delays — in some cases up to 23× higher than small messages
- This pattern was consistently observed across all experimental setups

Synchronous protocols and message size

- Synchronous protocols must set their bound Δ to accommodate for the delays of large message => this will hurt performance a lot!
- 99.99% for 2KB messages is 250ms while 99.99% for 128KB and 1MB are 2825ms and 6099ms, respectively

Our idea

- Messages should be treated differently depending on their size
- We defined two types of messages:
  - ✦Type S - stands for small messages
  - ✦Type L - stands for large messages

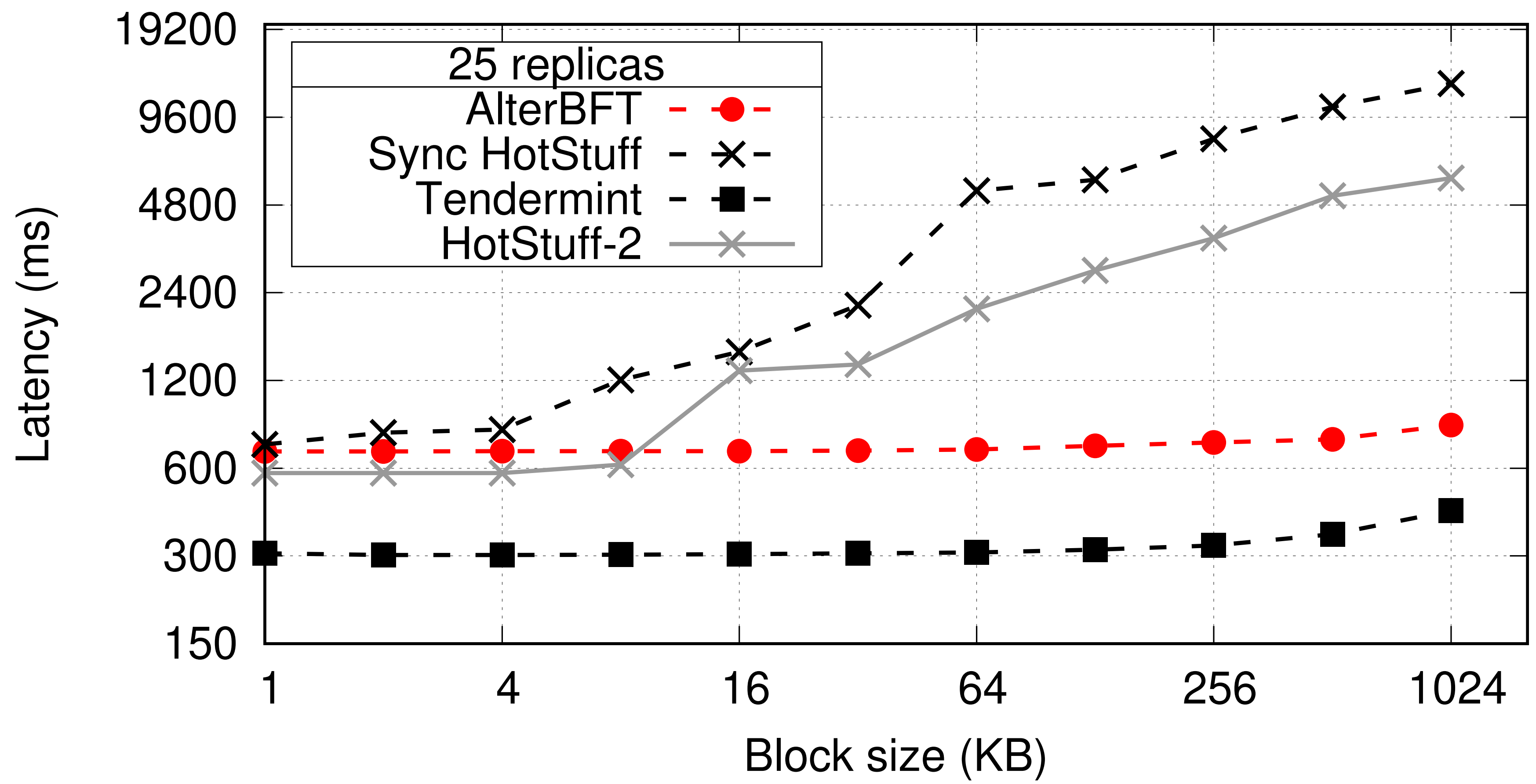
New system model

- ***Hybrid synchronous system model***:
  - *Type S messages* will always respect the specified bound Δ_S (synchronous system model)
  - *Type L messages* will respect the time bound Δ_L only eventually, after GST (partially synchronous system model)

AlterBFT

- First BFT consensus protocol in the new model
- Key idea:
  - ✦ Safety relies on the timely delivery of small messages within Δ_S
  - ✦ Liveness relies on eventually timely delivery of big messages within Δ_L
- *Tolerates the same number of Byzantine failures as synchronous protocols, up to 1/2*
- *Achieves better performance, especially latency, because its performance only depends on Δ_S*

AlterBFT’s latency



- Up to 15x lower latency compare to Sync HotStuff
- Comparable to partially synchronous protocols

Key takeaways

- The message size has a huge effect on message delays: Delays tend to increase and vary more as the message size increase
- Hybrid model captures assumes small messages will be timely and large message will be eventually timely
- AlterBFT is a new BFT consensus protocol in the hybrid model whose safety relies on small messages and that requires timely large messages only for progress
- AlterBFT achieves comparable performance to partially synchronous protocols while tolerating more Byzantine failures

Final remark

- A **step** toward understanding the practicality of synchronous protocols
- Opens the door for **new research** to explore and finally answer:
  - Can synchronous protocols be practical?
  - Or should they remain purely theoretical?

Thank you!