Proof-of-Execution: Reaching Consensus through Fault-Tolerant Speculation

Suyash Gupta, Jelle Hellings, Sajjad Rahnama, Mohammad Sadoghi

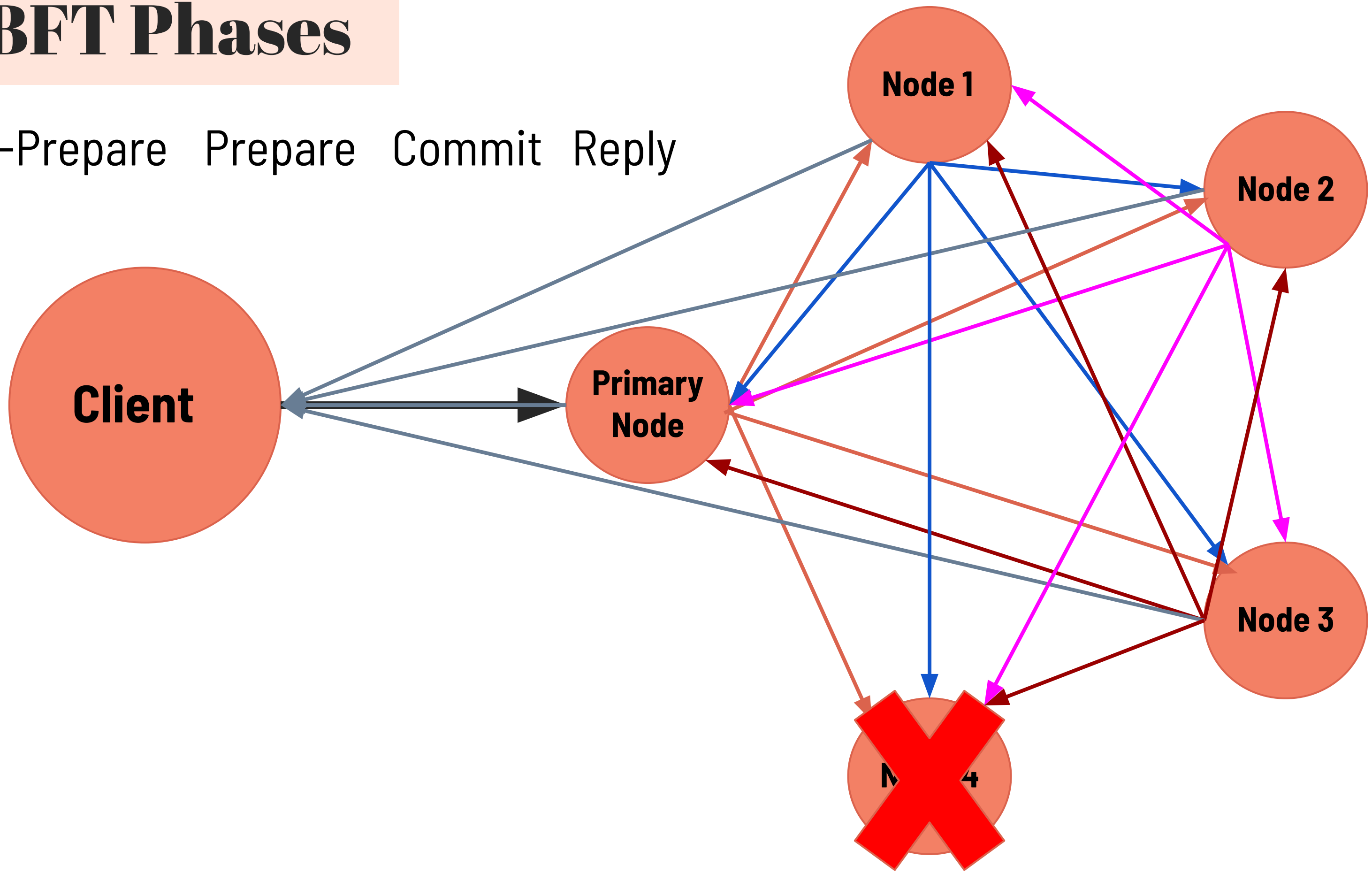
Presented by: Pranjal Mahajan
Utkarsh Kolhe
Yash Deshmukh
Tarang Gujar

Practical Byzantine Fault Tolerance (PBFT)

- A consensus algorithm is a process to attain agreement on a single data value among distributed systems. PBFT is a type of consensus algorithm.
- PBFT is based on the topic of solving consensus, considering byzantine faults.
- PBFT handles less than ⅓ byzantine faults. The system can handle 'f' byzantine faults, where there are minimum (3f+1) nodes.
- The main PBFT algorithm consists of three phases: PRE-PREPARE, PREPARE, COMMIT

PBFT Phases

Request Pre-Prepare Prepare Commit Reply



PoE vs PBFT

- **Why do we need something else?**
  - As already mentioned, PBFT operates in three communication phases, two of which necessitate quadratic communication complexity.
  - PBFT is considered unrealistic in large scale data management systems due to higher complexity.
  - PBFT involves high computational power.
- **Proof of Execution (PoE)**

A novel BFT protocol that achieves resilient agreement in just three linear phases. The paper portrays PoE as a scalable and reliable agreement protocol that shields against malicious attacks. PoE's scalable and resilient design emerges by adding four design elements to PBFT.

Non-Divergent Speculative Execution

PoE achieves faster consensus using speculative execution. In PBFT terminology, PoE replicas execute requests after they get prepared, i.e. they don't broadcast commit messages.

Safe Rollbacks and Robustness under Failures

PoE ensures that if a client receives a full proof of execution, consisting of responses from a majority of the non-faulty replicas, then such a request persists in time. Otherwise, PoE permits replicas to rollback their state if necessary.

Agnostic Signatures and Linear Communication

When few replicas are participating in consensus (up to 16), then a single phase of all-to-all communication is inexpensive and using MACs for such setups can make computations cheap. For larger setups, we employ TSs to achieve linear communication complexity.

Avoid Response Aggregation

In specific, all replicas execute each client request and send their response to the executor. It is the duty of the executor to reply to the client and send a proof that a majority of the replicas outputted the same result. In PoE, we avoid this additional communication between the replicas by allowing each replica to respond directly to the client.

Comparison protocols

PBFT

Practical Byzantine Fault Tolerance

ZYZZYVA

Speculative Byzantine Fault Tolerance

HOTSTUFF

Leader-based Byzantine fault-tolerant

SBFT

Scalable and Decentralized Byzantine Fault-tolerant

Comparison protocols

Protocol	Phases	Messages	Resilience	Requirements
ZYZZYVA	1	$\mathcal{O}(n)$	0	Reliable clients and unsafe
PoE (our paper)	3	$\mathcal{O}(3n)$	f	Sign. agnostic
PBFT	3	$\mathcal{O}(n + 2n^2)$	f	
HOTSTUFF	8	$\mathcal{O}(8n)$	f	Sequential Consensus
SBFT	5	$\mathcal{O}(5n)$	0	Optimistic path

Analysis of Design Principles

The last phase of PBFT enures that non-faulty replicas only execute requests and inform clients when there is a guarantee that such a transaction will be recovered after any failures.

Since we are eliminating this last phase, replicas speculatively execute requests before obtaining recovery guarantees.

This impacts PBFT-style consensus in two ways:

- Clients need a way to determine proof-of-execution after which they have a guarantee that their requests are executed and maintained by the system.
- Since requests are executed before they are guaranteed, replicas need to be able to rollback requests that are dropped during periods of recovery.

Compatibility with Scalable Design Principles

01 **Out of order execution**

02 **Out of order processing**

03 **Twin Path Consensus**

04 **Primary Rotation**

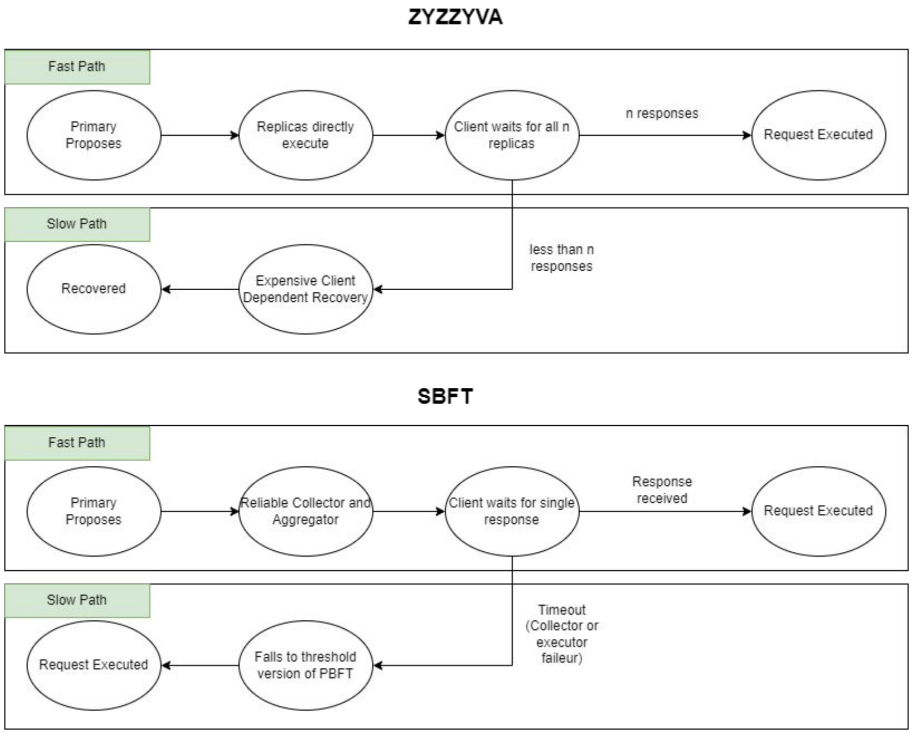
Out of order execution

- In Typical BFT Systems, replicas first agree on unique order of client request and then execute requests in order. This is called order-execute model.
- In few BFT systems, Execution is done prior to ordering. But they need to reverify before committing.
- PoE lies between these two.
- In PoE, replicas speculatively execute only with partial ordering guarantees.
- Minimizes communication costs and latencies.

Out of order processing

- BFT typically execute requests in order but this does not mean they have to process proposals sequentially.
- They support out-of-order processing, which means primary continuously proposes requests without waiting for replicas to process them.
- This eliminates impact of high message delays.
- This is included in PoE.

Twin Path Consensus



- Twin Path Consensus has an optimistic fast path that works when none of the replicas are faulty and require aid to determine optimistic condition.
- Since Twin path requires outside aid, it is in contrast with design of PoE.

Primary Rotation

- Since primary plays a major role consensus, a faulty primary can have a major impact on the process.
- To overcome this, HOTSTUFF replaces primary to using an extra communication phase after every round.
- Primary Replacements require all consensus round are performed in sequential manner. This will eliminate possibility of out-of-order processing

Proof of Execution(PoE)

- Primary replica is responsible for proposing transactions requested by clients to all the backup replicas
- Assuming that the primary replica is behaving correctly, each replica executes the transactions speculatively
- Replicas can recover by rolling back transactions in case a malicious behaviour is detected

System Model and Notations

- Set **R** of replicas that process client requests
- Each **r ∈ R** is assigned a unique identifier id(r) where $0 \le id(r) < |R|$
- **F ⊆ R** is the set of faulty replicas
- **n = |R|** is the number of replicas
- **f = |F|** is the number of faulty replicas
- **nf = |R \ F|** is the number of non-faulty replicas
- Assumption: **n > 3f => nf > 2f**
- Authenticated communication is the minimal assumption - Byzantine replicas are able to impersonate each other, non-faulty replicas can't be impersonated
- Based on the message type and number of replicas either message authentication code (MAC) or threshold signatures (TS) is used

Definition

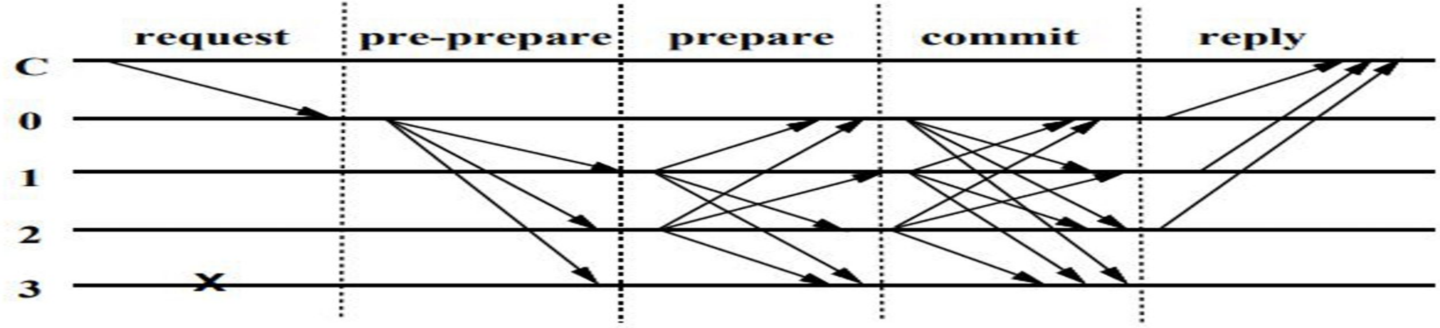
A single run of any consensus protocol should satisfy the following requirements -

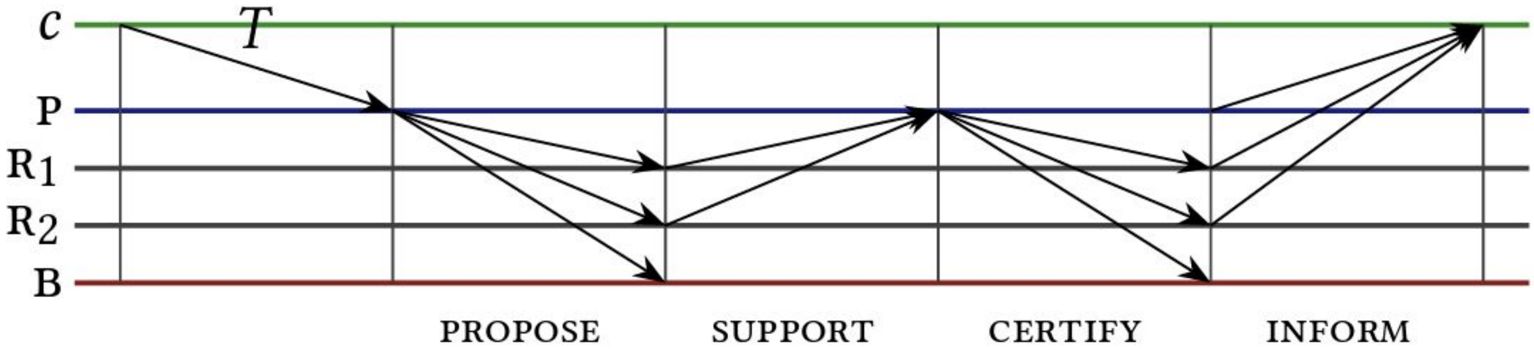
- Termination(liveness): Each non-faulty replica executes a transaction
- Non-divergence(safety): All non-faulty replicas execute the same transaction

In PoE, execution is speculative - replicas can execute and rollback

- Speculative non-divergence: If $nf - f \geq f + 1$ non-faulty replicas accept and execute the same transaction T, then eventually all the non-faulty replicas will accept and execute T.

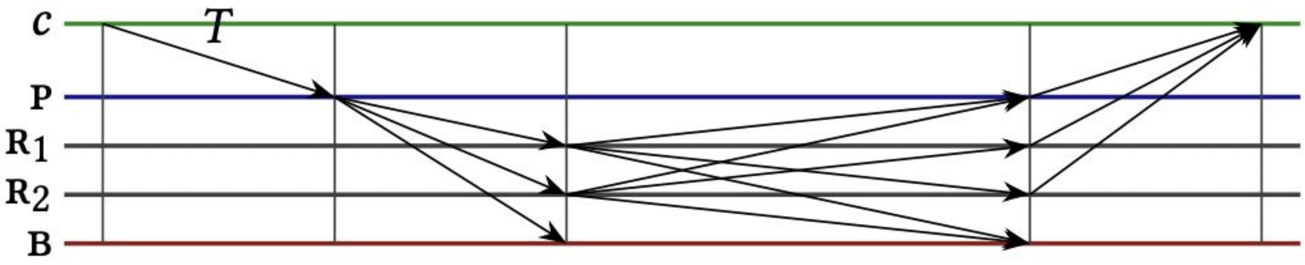
Normal-Case Algorithm



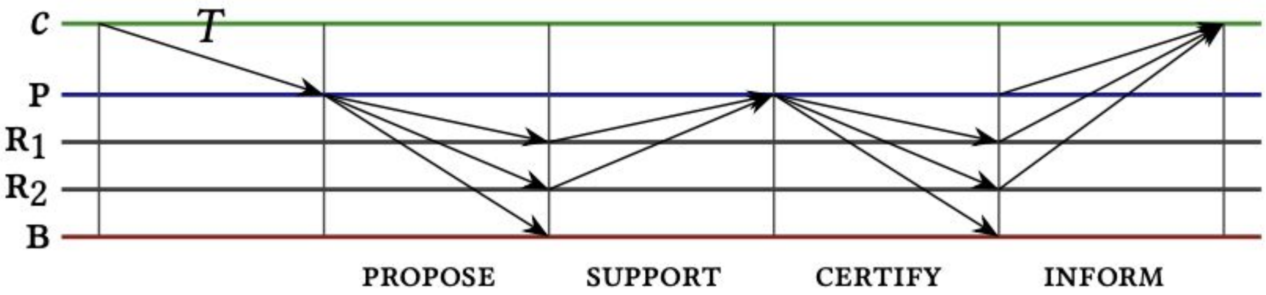


PoE using TSs.

Designing PoE using MACs



PoE using MACs



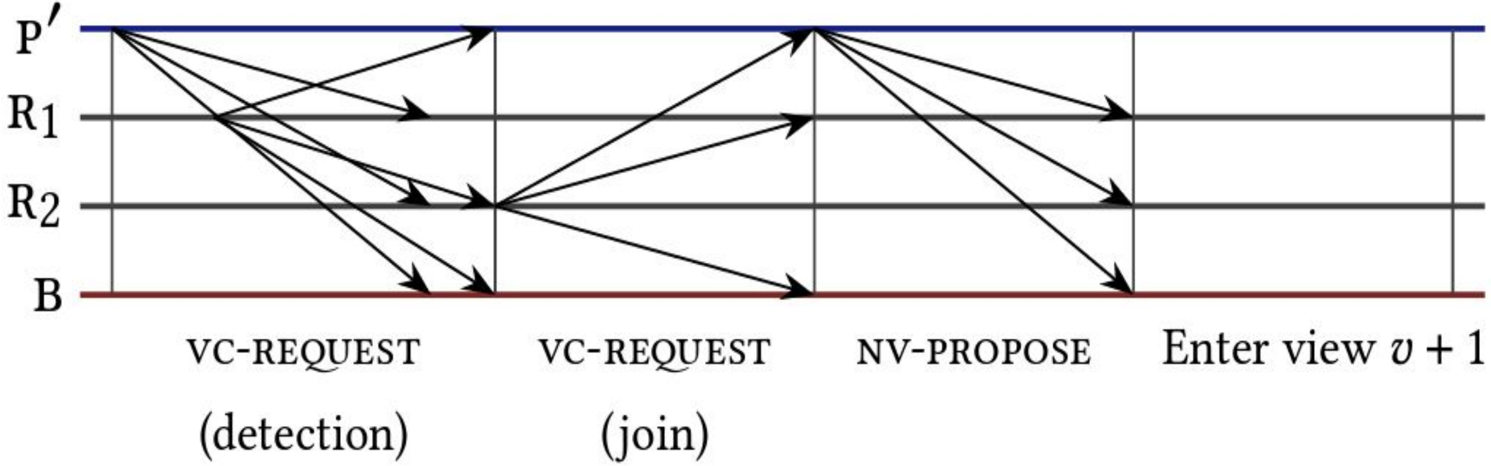
PoE using TSs.

View-Change Algorithm

There are three stages in the view-change algorithm

- **Failure detection and View-change requests**: broadcasts a message VC-Request(v,E) where E is the summary of all the transactions executed. There are two ways of identifying a failure -
  - ‘r’ timeouts e.g ‘r’ forwards a client request to primary and the primary fails to propose this request on time
  - ‘r’ receives VC-Request from at least (f+1) distinct replicas
- **Proposing the New View**: new primary p’ needs to propose a new view by determining valid lists of requests that need to be preserved. p’ waits until it receives valid VC-Request from nf distinct replicas.
- **Move to the New View**: After a replica ‘r’ receives NV-Propose from the new primary p’, it validates the content of the message. After validation, it matches the state of the new view and enters the new view ‘v+1’

View-Change Algorithm - Example



Correctness of PoE

Consider a system in view v, in which the first $k-1$ transactions have been executed by all non-faulty replicas, in which the primary is non-faulty, and communication is reliable. If the primary received $\langle T \rangle c$, then the primary can use the algorithm to ensure that

(1) there is non-divergent execution of T;

(2) c considers T executed as the k-th transaction; and

(3) c learns the result of executing T (if any),

this independent of any malicious behavior by faulty replicas.

Optimizations

- To reach nf signature shares,the primary can generate one itself. Hence, it only needs nf – 1 shares of other replicas.

- The propose, support, inform, and nv-propose messages are not forwarded and only need MACs to provide message authentication. The certify messages need not be signed, as tampering them would invalidate the threshold signature. The vc-request messages need to be signed, as they need to be forwarded without tampering.

ResilientDB -Batching

- ResilientDB helps us implement and test different consensus protocols.
- Aggregating several client requests in a single batch.
- The primary replica assigns the client requests a sequence number and enqueues these requests in batch queue.
- Each batching-thread also hashes the requests in a batch to create a unique digest.

ResilientDB-Ledger Management

- Consider a Block $B_i := \{k, d, v, H(B_{i-1})\}$
- ResilientDB requires the first primary replica to create a genesis block.
- To create a block, the execute-thread hashes the previous block in the blockchain and creates a new block.
- In PoE, such a proof includes the threshold signature sent by the primary as part of the certify message.

Evaluation

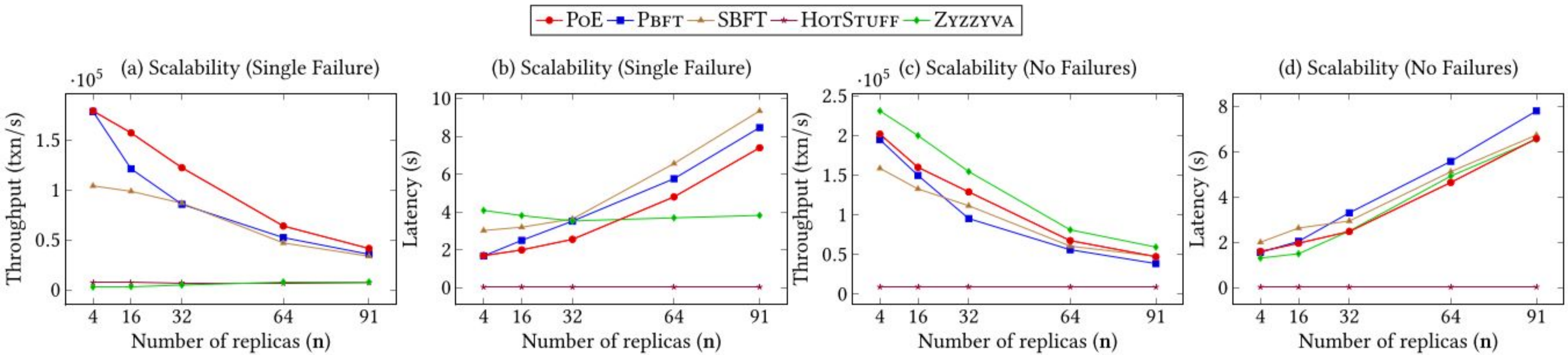
Comparison Protocols:

- ZYZZYVA
- PBFT
- SBFT
- HOTSTUFF

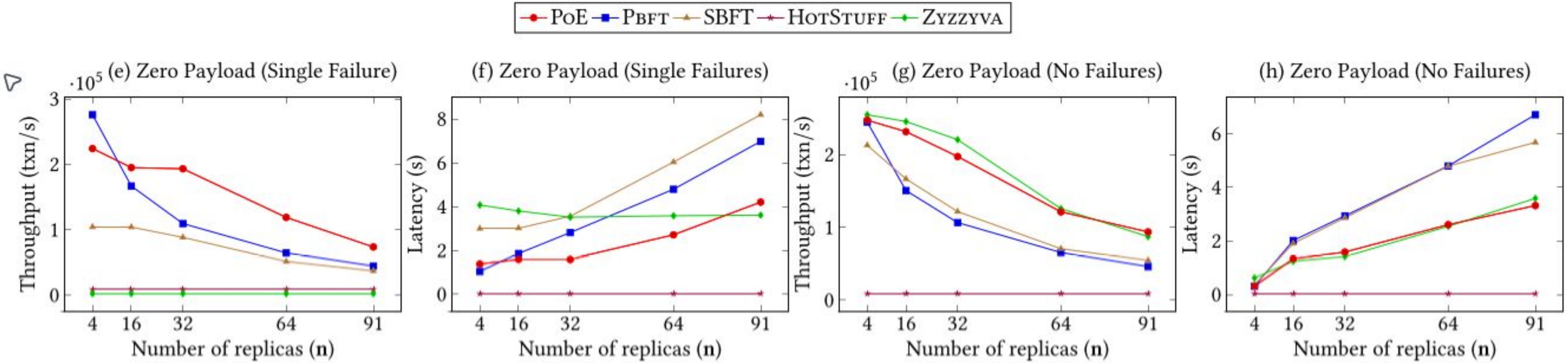
Evaluation questions:

- How PoE compares with other protocols under failures?
- Does PoE benefits from batching client requests?
- How PoE performs under zero payload?
- POE is how much scalable on increasing replicas?

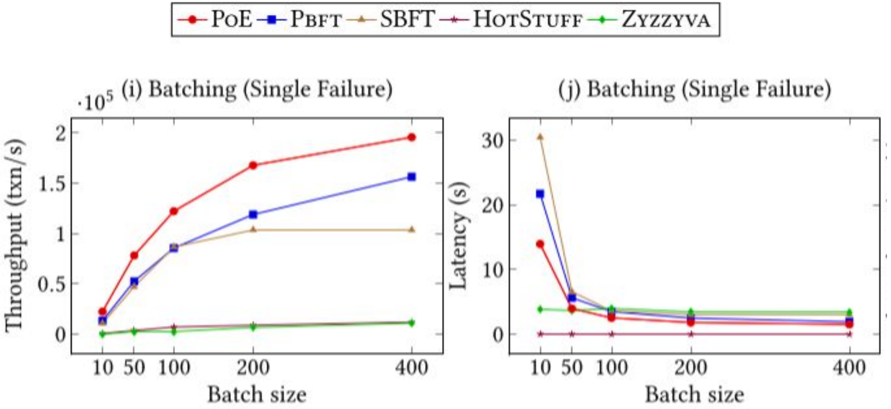
Scalability



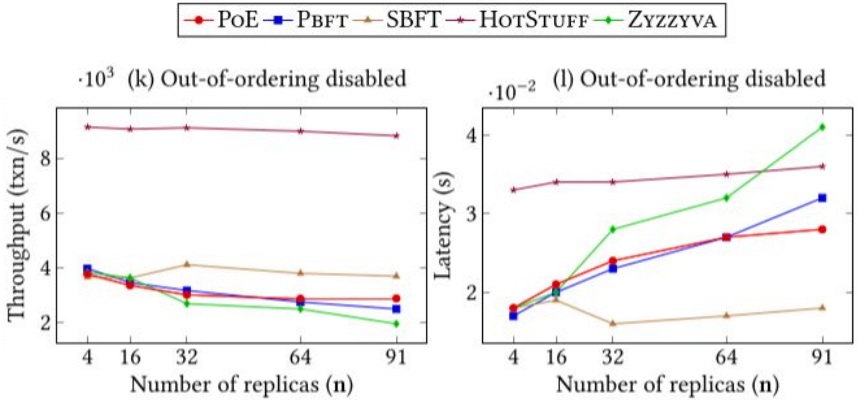
Impact of Zero Payload



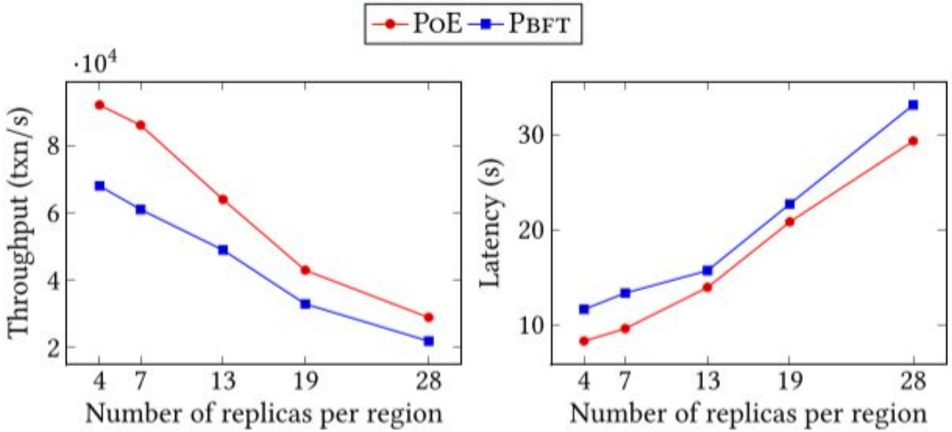
Impact of Batching



Disabling Out of Order



WAN Scalability



CONCLUSION

- PoE guarantees safety and liveness in three linear phases
- PoE performs out of order execution saving impacts of high communication delays through speculative execution.
- PoE performs better than existing protocols on many fronts.