RBFT: Redundant Byzantine Fault Tolerance

Pierre-Louis Aublin, Sonia Ben Mokhtar, and Vivien Quema

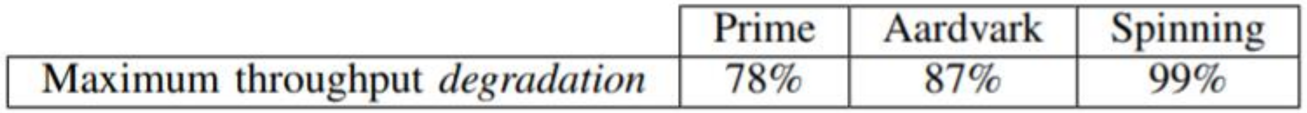
Grenoble University, CNRS – LIRIS, Grenoble INP

presenter = Muhammad Awad

ECS265 - Paper Presnetaion

Goal and Motivation:

- Robust BFT protocol (achieves good performance when faults occur).
- We can detect and recover from a malicious primary, but the primary can be *smartly* malicious.
- What is the throughput of a non-malicious primary?
  - We can't tell.



	Prime	Aardvark	Spinning
Maximum throughput *degradation*	78%	87%	99%

TABLE I: Performance degradation of "robust" BFT protocols under attack.

RBFT: High-level Idea

- Leverage multicore architectures.
- Multiple (f + 1) instances of a BFT protocol are executed in parallel.
- Each instance has its own primary replica.

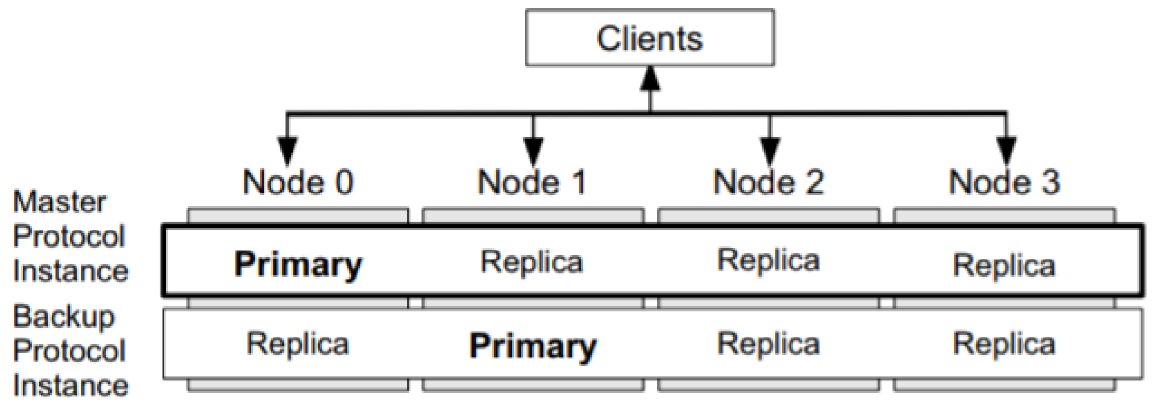


Fig. 4: RBFT overview ($f = 1$).

RBFT: High-level Idea

- All protocols order requests, but the ***master instance*** executes them.
- Other instances ***(backup instances)*** order requests and compares the highest throughput to the master instance throughput.
- If majority (2f + 1) concludes that master is slower, it’s considered malicious and new primaries are elected for each instance.

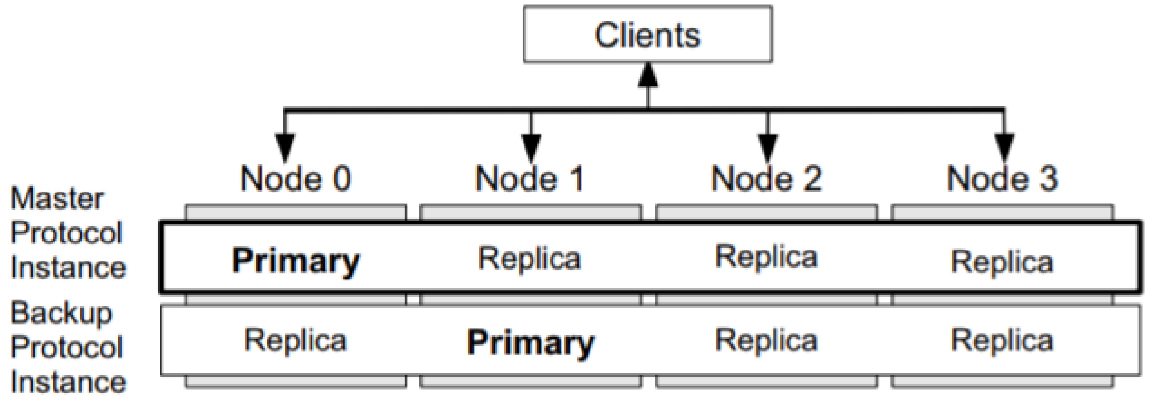


Fig. 4: RBFT overview ($f = 1$).

RBFT: Detailed Protocol Steps

1. Client sends request to all nodes: <<REQUEST, *o, rid, c*>$_{\sigma c}$, *c*>$_{\mu c}$

- Nodes authenticate the MAC (make sure it was sent by the client).
- If valid, validate the signature of the request (if not valid blacklist client).
  - If already executed send back reply.

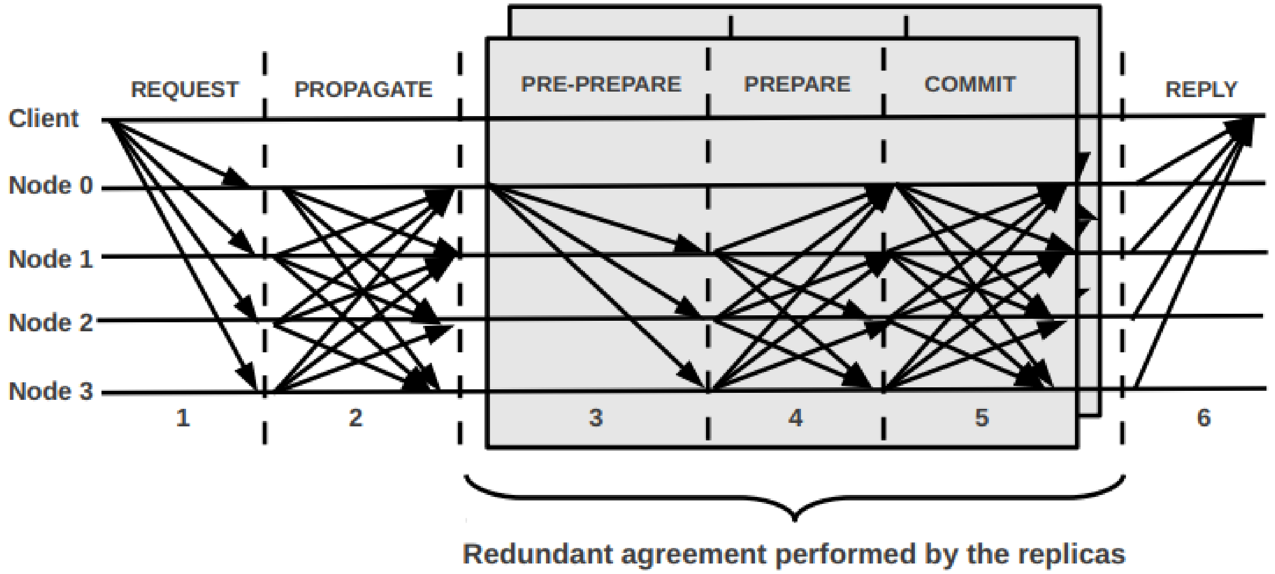


Fig. 5: RBFT protocol steps ($f = 1$).

RBFT: Detailed Protocol Steps

2. Propagate request: <PROPAGATE <REQUEST, *o, s, c*>$_{\sigma c}$, *i*>$_{\mu c}$

- After reciving f + 1 propagate requests, a node give requests to local replicas.

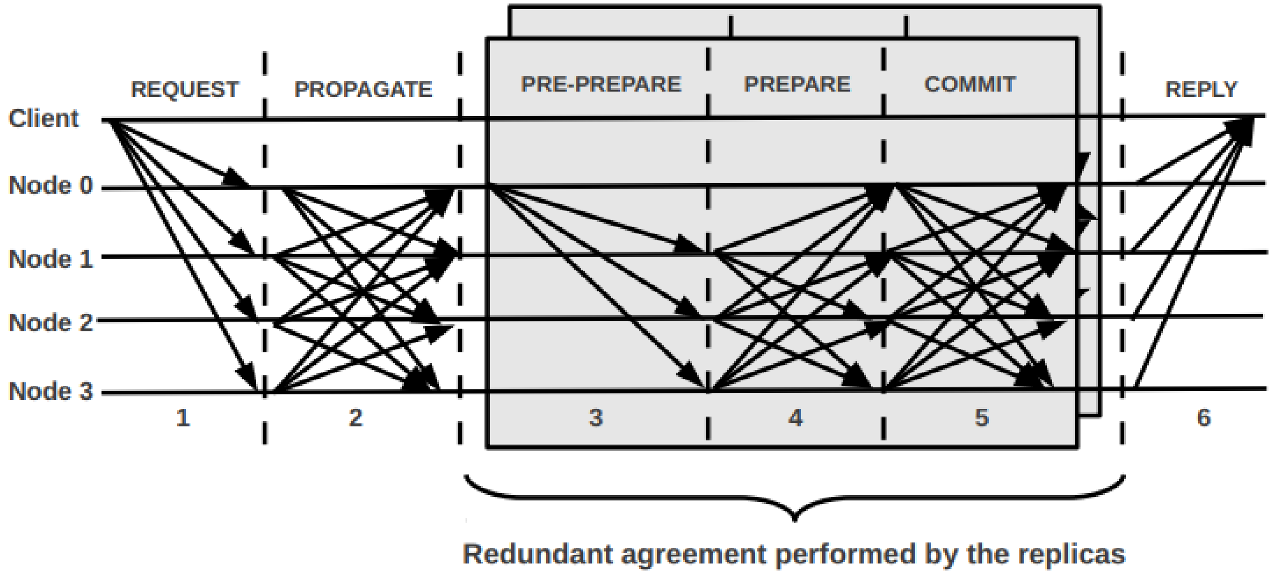


Fig. 5: RBFT protocol steps ($f = 1$).

RBFT: Detailed Protocol Steps

3. Primary replica of each instance sends PRE-PREPARE messages.

- Non-primary waits for the PRE-PREPARE message from instance primary.

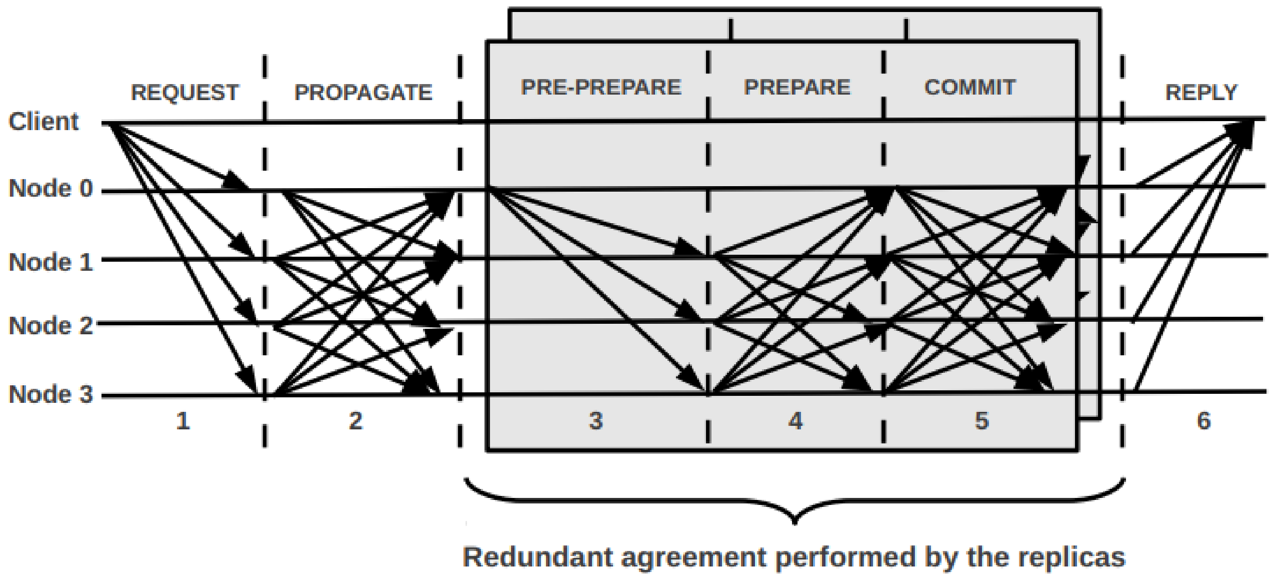


Fig. 5: RBFT protocol steps ($f = 1$).

RBFT: Detailed Protocol Steps

4. Replicas of each instance receives PRE-PREPARE messages.

- Verifies validity of MAC
- Wait for the f + 1 instances before sending PREPARE messages (i.e. avoid fake throughput boost).

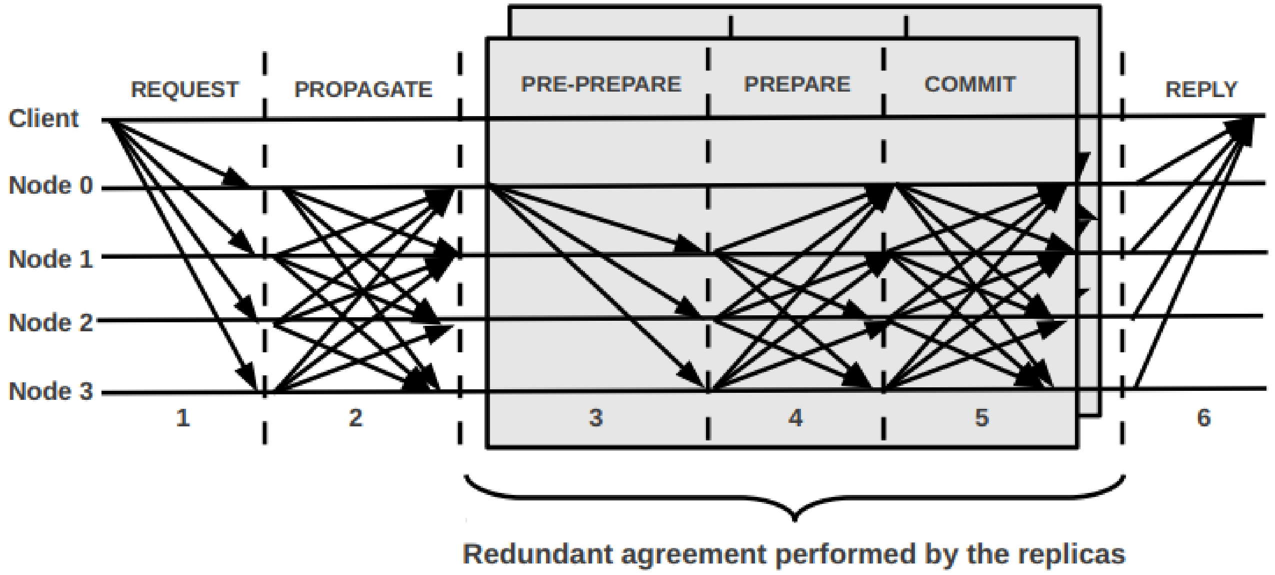


Fig. 5: RBFT protocol steps ($f = 1$).

RBFT: Detailed Protocol Steps

5. Sending COMMIT messages.

- After receiving 2f matching PREPARE from replicas (same protocol instance), consistent with PRE-PREPARE message, a replica sends COMMIT messages

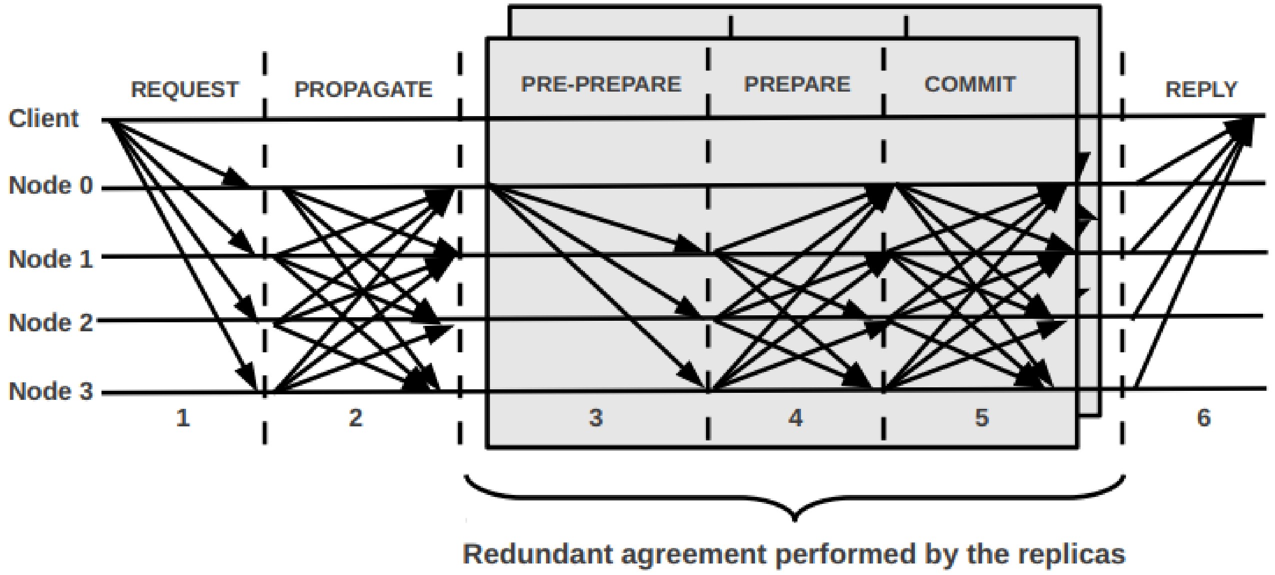


Fig. 5: RBFT protocol steps ($f = 1$).

RBFT: Detailed Protocol Steps

5. Receiving COMMIT messages.

- After receiving 2f+1 matching COMMIT messages (same protocol instance), a replica gives the ordered request to its node.

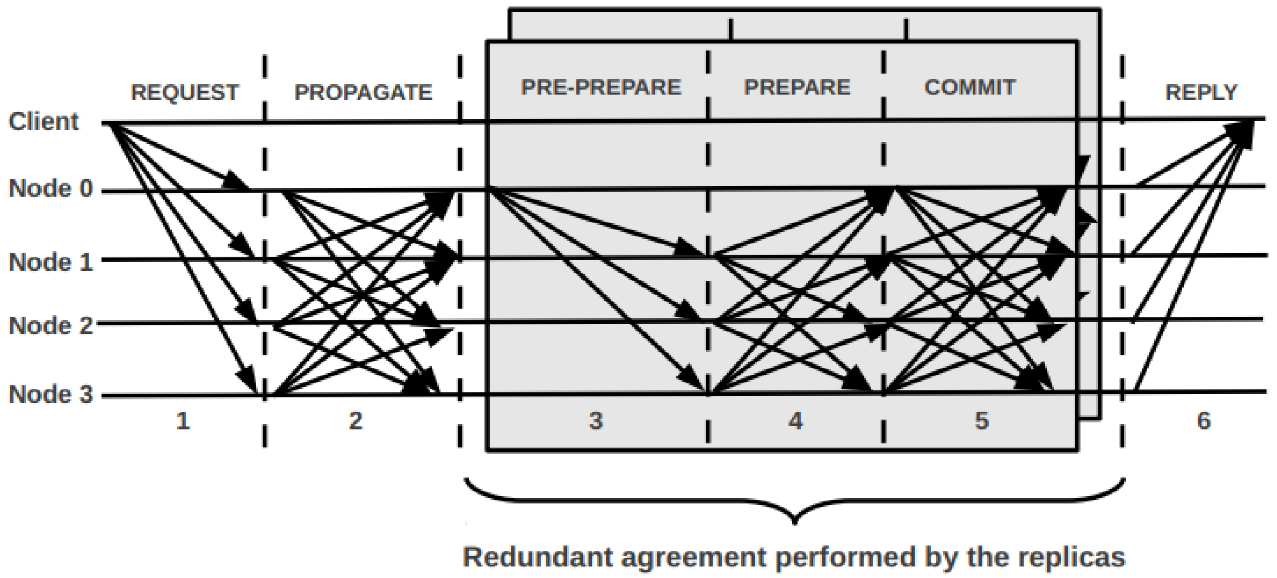


Fig. 5: RBFT protocol steps ($f = 1$).

RBFT: Detailed Protocol Steps

6. Execution and reply.

- Each time a node receives request from its replicas (master instance) it executes the operation, then it sends the reply.
- Once the client receives f+1 valid and matching replies it accepts the result of the request execution.

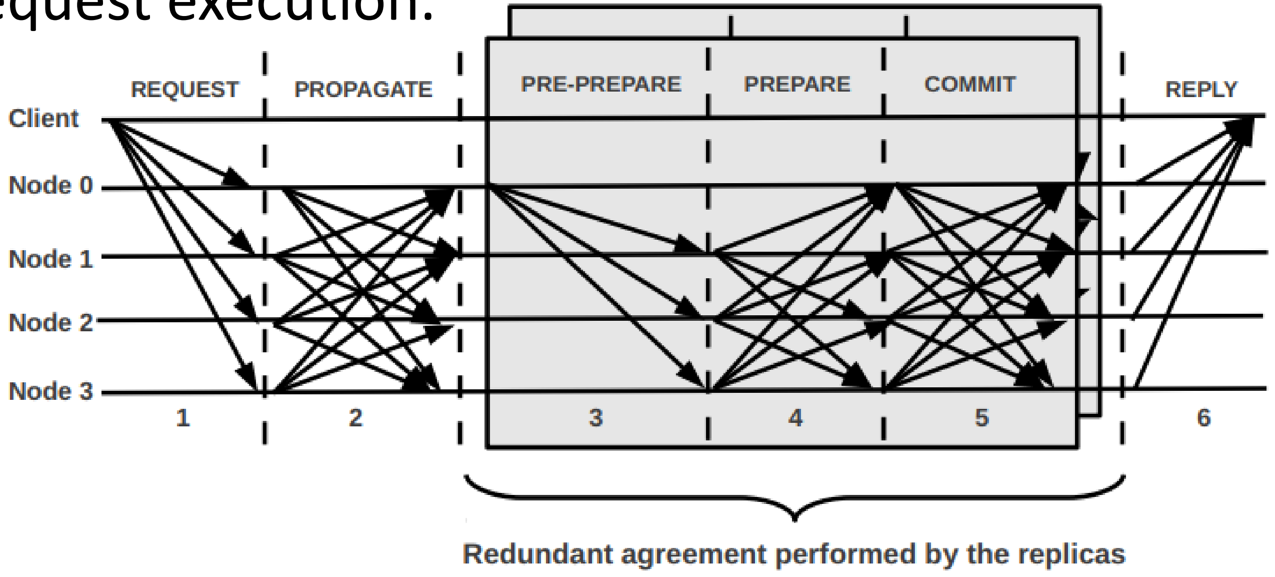


Fig. 5: RBFT protocol steps ($f = 1$).

RBFT: Monitoring Mechanism

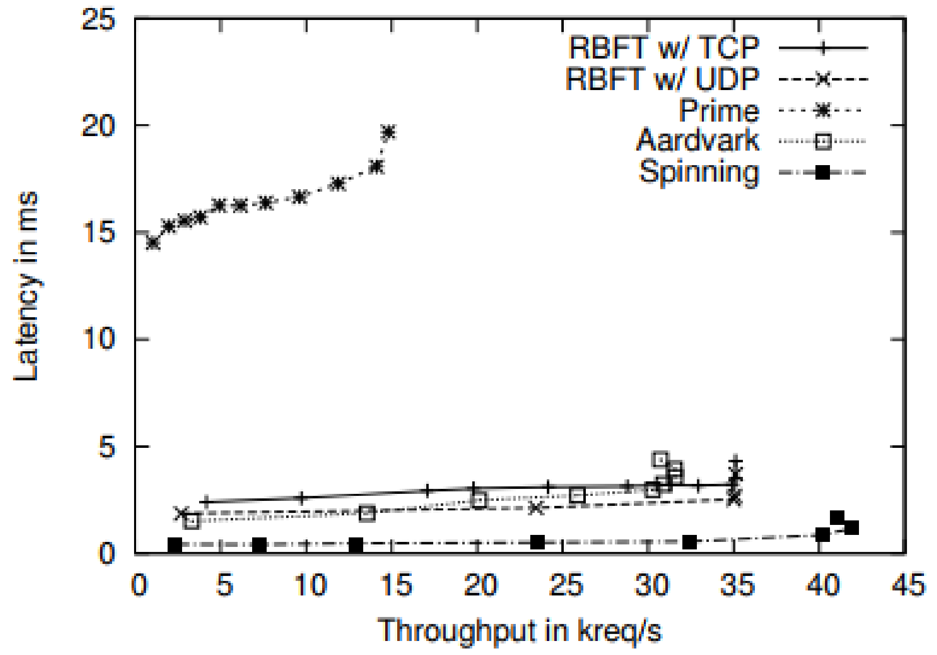
- **Detect faulty master protocol instance.**
  - **Each nodes keeps a counter for each protocol instance to compute throughput of master instance.**
  - $nbreqs_i$ = number of requests ordered by replicas that received 2f+1 commit messages.
  - If the ratio between throughput of master and average of backups is less than a threshold, node request instance change.
- **Ensure fairness between clients**
  - Nodes measures latency of *each request* and *average latency for each client* (between node's replicas) and a client's *average requests latency*.
  - If request ordered by master instance and request latency > request_threshold
  - Or, average client requests latency compared between instances > threshold
    - -> Request instance change

RBFT: Protocol Instance Change Mechanism

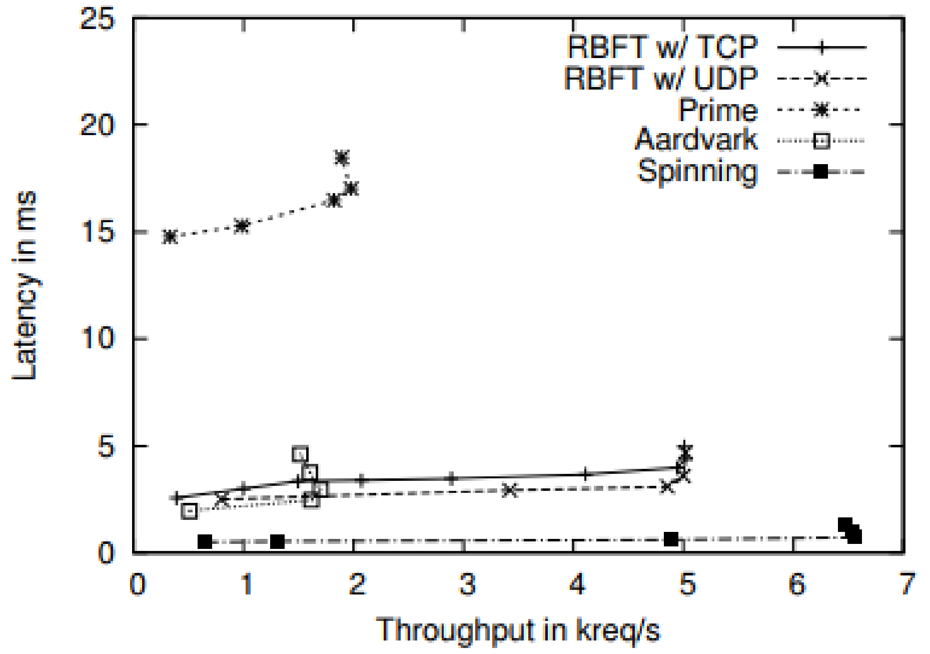
- **Nodes keeps counter of instance change messages *(cpi_i).***
- **Once a node detects that it needs instance change it sends an INSTANCE_CHANGE message to all other nodes.**
- **When a node j receives INSTANCE_CHANGE message*,***
  - If message $cpi_i < cpi_j$ it ignores the messages (older instance change message)
  - Else, it checks if it needs to send the INSTANCE_CHANGE message. If it does, it sends the message to all replicas.
- **Once a node receives 2f+1 valid and matching INSTANCE_CHANGE message it initiates view change for every protocol and increment its instance change messages counter.**

Results

1. Fault free.



(a) With requests of 8B.



(b) With requests of 4kB.

Fig. 7: Latency vs. throughput for various robust BFT protocols ($f = 1$).

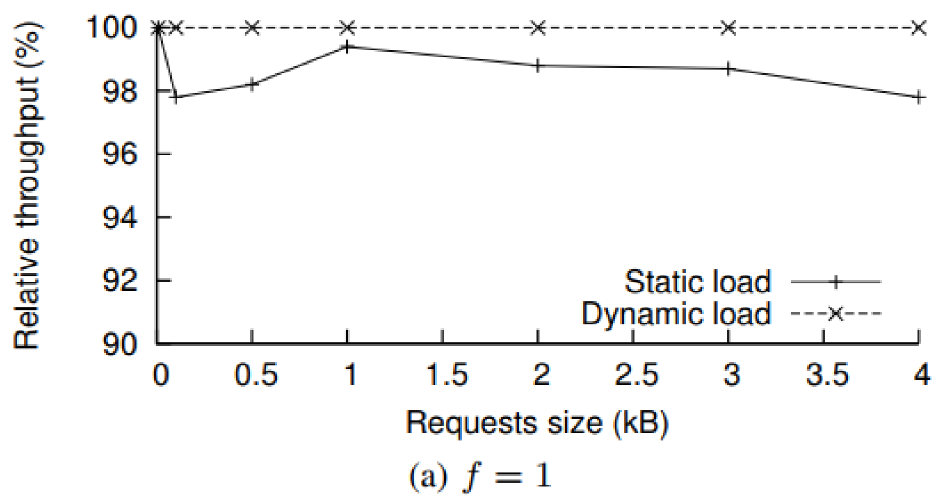
Results

2. Worst-attack I (faulty f nodes – primary instance correct)

- **Decrease throughput w/o requiring view change**
- **Client sends verifiable requests to all replicas except master's primary instance node *p*.**
- **Faulty nodes sends PROPAGATE messages to *p***
- **Faulty replicas of master instance sends invalid messages or don't take part in the protocol.**

Results

2. Worst-attack I (faulty f nodes – primary instance correct)



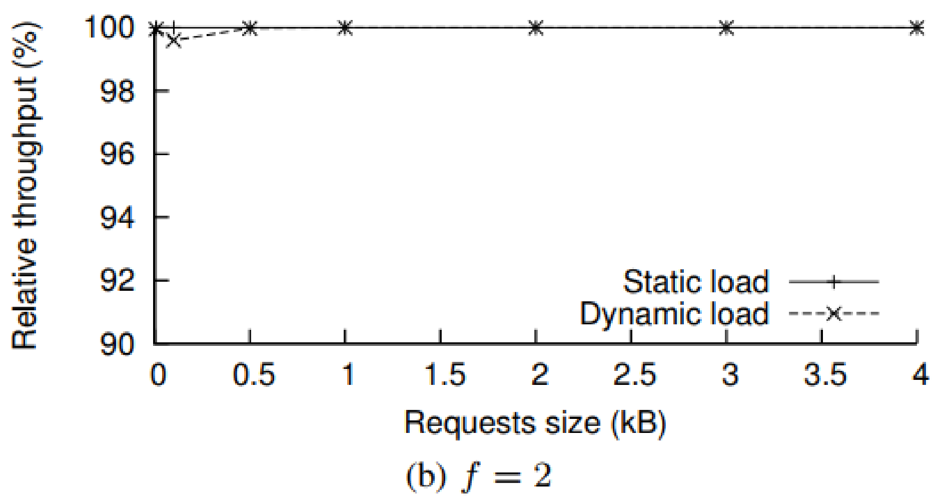


Fig. 8: RBFT throughput under *worst-attack-1* relative to the throughput in the fault-free case, for both a static and a dynamic load.

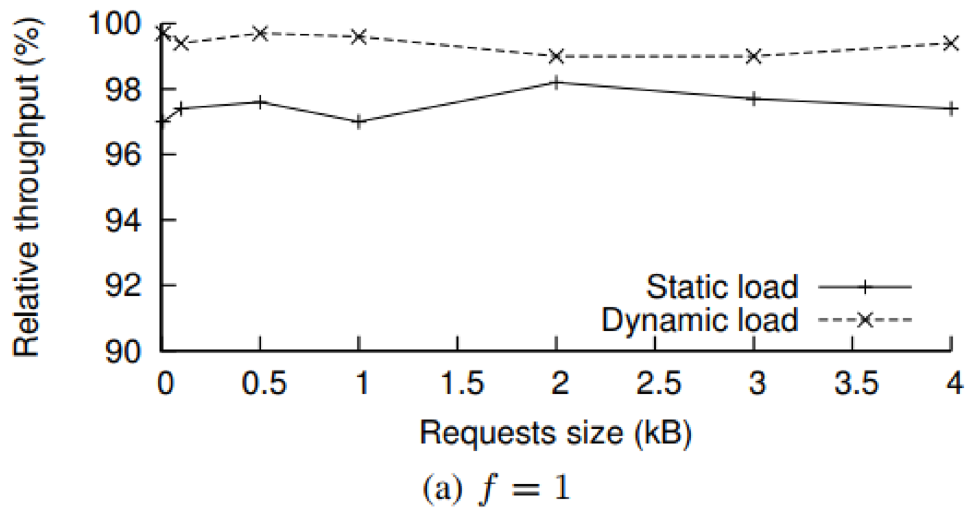
Results

3. Worst-attack II (faulty f nodes – primary instance not correct)

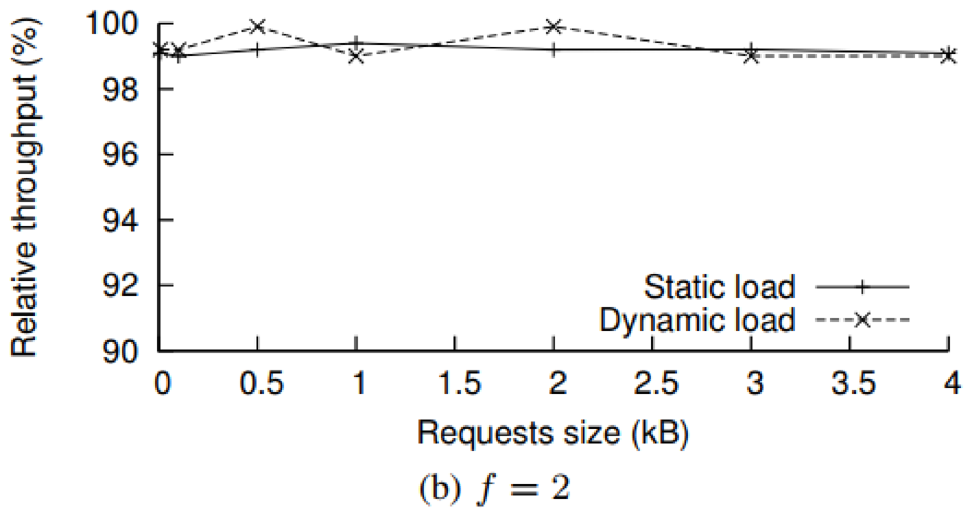
- **Primary of master instance runs on faulty node**
- **Faulty client is trying to reduce backup throughput. So,**
  - **Faulty clients send invalid requests to correct nodes**
  - **Faulty nodes send invalid messages to non-faulty ones and don't participate in propagate phase.**
  - **Replicas of the backup instance faulty nodes send invalid messages and don't take part in the protocol.**

Results

3. Worst-attack II (faulty f nodes – primary instance not correct)



(a) $f = 1$



(b) $f = 2$

Fig. 10: RBFT throughput under *worst-attack-2* relative to the throughput in the fault-free case, for both a static and a dynamic load.

Results

4. Fairness

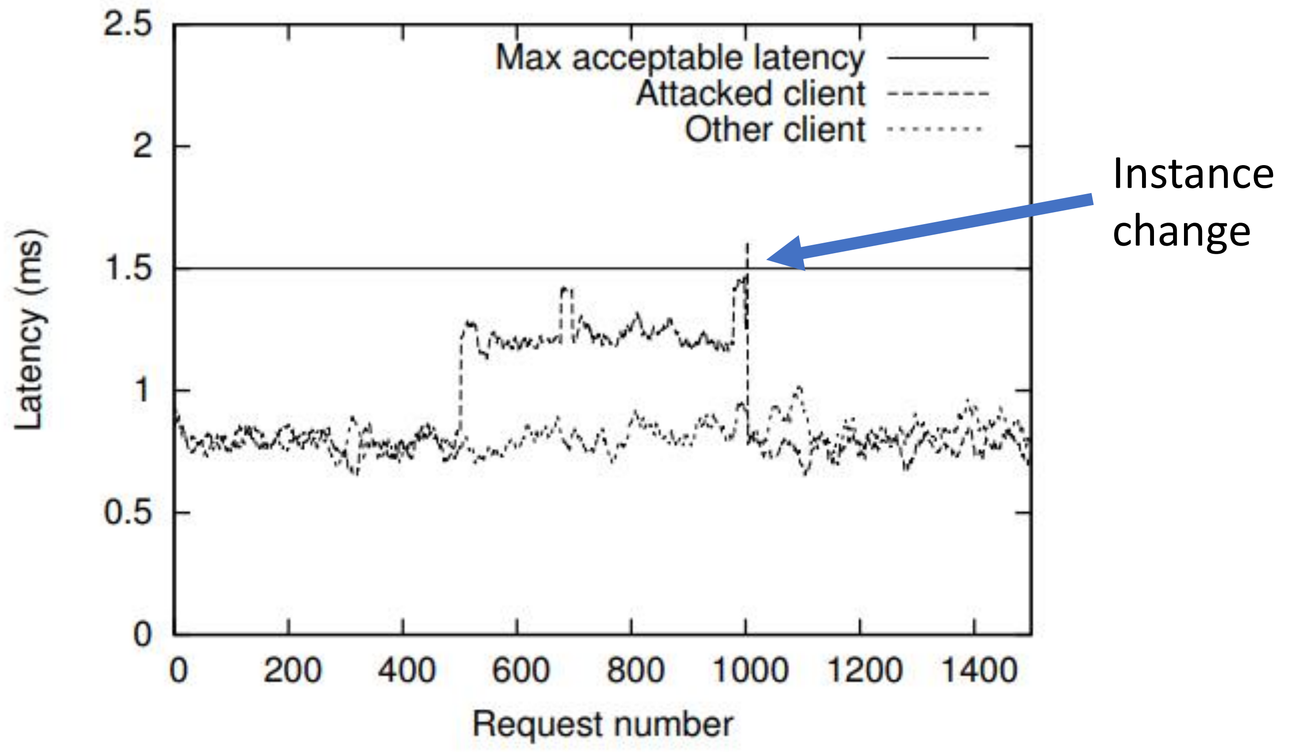


Fig. 12: Ordering latencies for the requests of two clients on the master protocol instance with an unfair primary, which starts to delay the request of one of the clients after 500 requests.

Conclusion

1. **State-of-the-art robust BFT protocols are not robust.**
2. **RBFT leverages multicore architectures and runs multiple instances of BFT protocols; while monitoring throughput and fairness.**
3. **During an attack, the throughput degradation due to malicious primary is 3%.**
4. **Degradation gets lower as f increase.**