Where we are in the semester…

- Ordering

- Consensus

- Virtual Synchrony

- Replication

Failure models

- Fail-stop

- Fail-crash

- Byzantine

Byzantine Techniques

1. The Byzantine Generals Problem

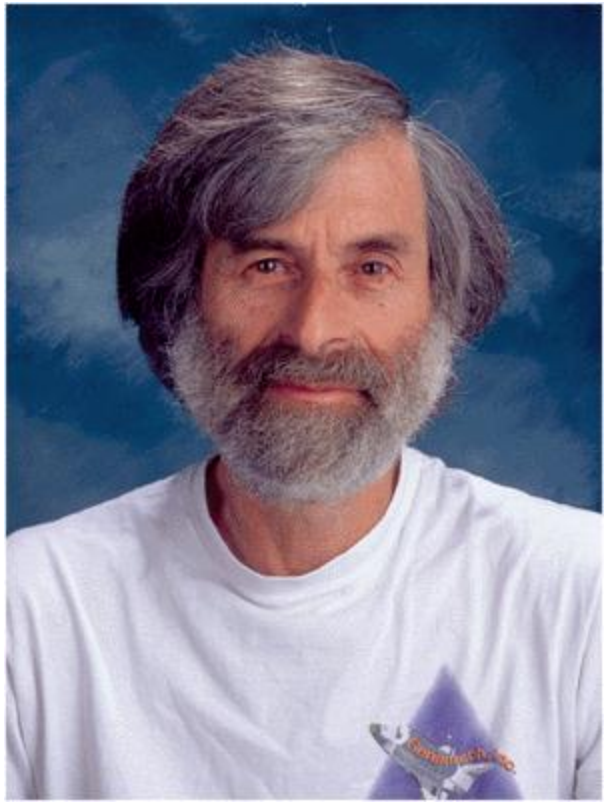
2. Practical Byzantine Fault Tolerance

Zhiyuan Teo
slides adapted from Srivatsan Ravi
and Eleanor Birrell

Discussion overview

1. Definition of problem
2. Relevance to computer systems
3. Naïve solutions
4. Solution with oral messages
5. Solution with signed messages
6. Other considerations

About the authors



No info?

Leslie Lamport, MSR

Robert Shostak, Vocera

Marshall Pease

What is the Byzantine army's problem?

- Some divisions of the Byzantine army are camped outside an enemy city.

- Each division is led by a general.

- Generals can communicate with one another via messages.

- They need a unanimous decision to attack or retreat.

- But some generals may be traitors!

Relevance to computer systems

- We would like reliable systems: use redundancy.

- Components can fail.

- Arbitrary behavior.

- We want consensus.

- Solution: employ some kind of majority voting system, but how?

A naïve solution

"Let every general tell every other general what they want to do. Each of them can independently and correctly come to consensus by looking at the majority."

- Traitors can give arbitrary answers to different generals.

- No distributed consensus.

Formal statement of naïve solution

- Each general G(i) sends his value v(i) to every other G.

- Each G collects every v(i) to form a set of values v(1), v(2), … v(n).

- Each G performs Majority(v(1), v(2), … v(n)) on his V.

- Problem: each general may have a different V.

Towards a tractable solution

- Reduce this to the problem of 1 general sending his orders to (n-1) generals.

Interactive Consistency

IC1: all loyal generals obey the same order.

IC2: if the commander is loyal, then every loyal general obeys the issued order.

- If the commander is loyal, IC2 implies IC1.

Impossibility with 3 generals

- Impossible to reach consensus with 1 traitor.

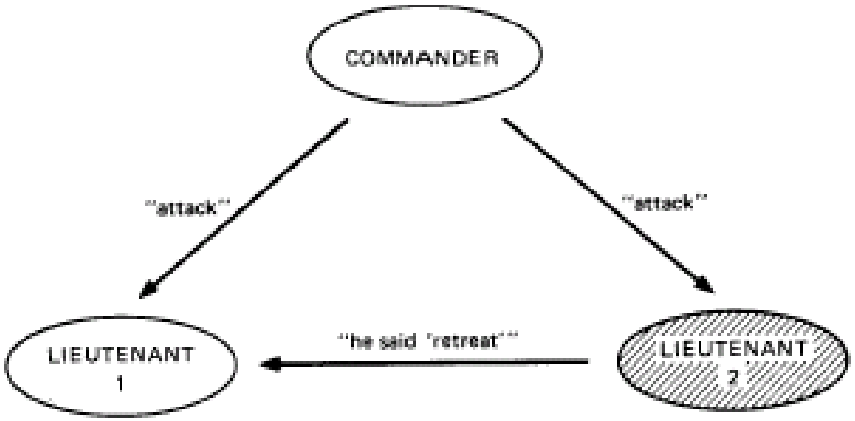


Fig. 1. Lieutenant 2 a traitor.

Interactive Consistency

IC1: all loyal generals obey the same order.

IC2: if the commander is loyal, then every loyal general obeys the issued order.

- Case 1: lieutenant is a traitor. IC 2 not satisfied.

Impossibility with 3 generals

- Impossible to reach consensus with 1 traitor.

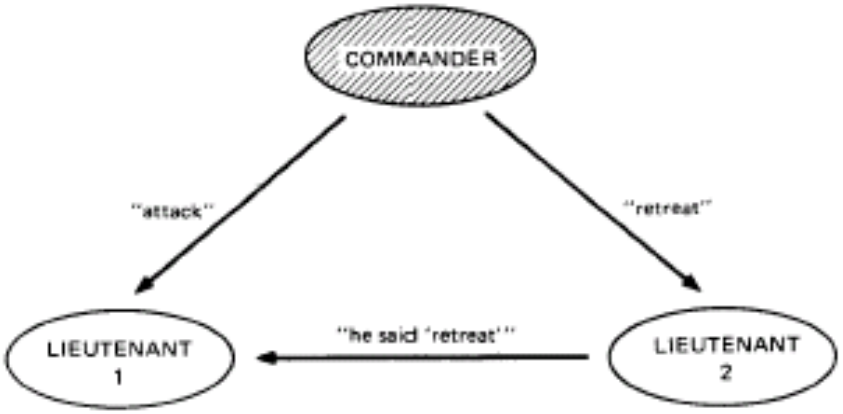


Fig. 2. The commander a traitor.

Interactive Consistency

IC1: all loyal generals obey the same order.

IC2: if the commander is loyal, then every loyal general obeys the issued order.

- Case 2: commander is a traitor. IC 1 not satisfied.

Impossibility results

- No solution for 3n+1 generals, if more than n are traitors.

- The proof is by contradiction and reduction.

- Assume a solution exists for a group of 3n or fewer generals.

- Set n=1, and this reduces to the original BGP problem.

- But this is impossible, so no solution exists for n traitors with fewer than 3n+1 generals.

Solution with oral messages

Assumptions:

A1. Each message sent is delivered correctly.

A2. Receiver knows who sent the message.

A3. Absence of a message can be detected.

- Also assume all-to-all connectivity.

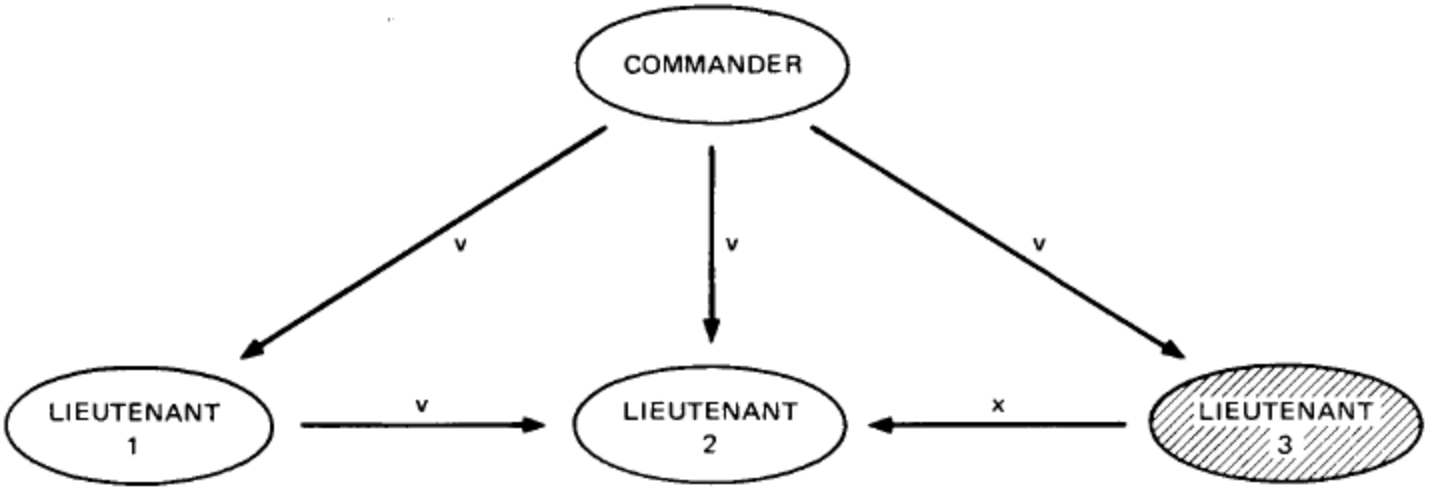
Algorithm for oral messages (OM)

- Recursive algorithm with parameter m (number of traitors).

- OM(0): no traitors
  - commander sends his value to every other general.
  - each general uses the received value, or some default value if nothing was received.

- O(m): m > 0 traitors
  - commander sends his value to every general.
  - each general acts as a commander for OM(m-1), and sends the original commander's value to every other general.
  - at the end of this round, every general receives a vector of values corresponding to what all generals have claimed to be the commander's orders.
  - run majority() on these values to get the actual commander's orders.
  - but the algorithm doesn't end because we still need the votes from other generals!

Example for n=4, m=1

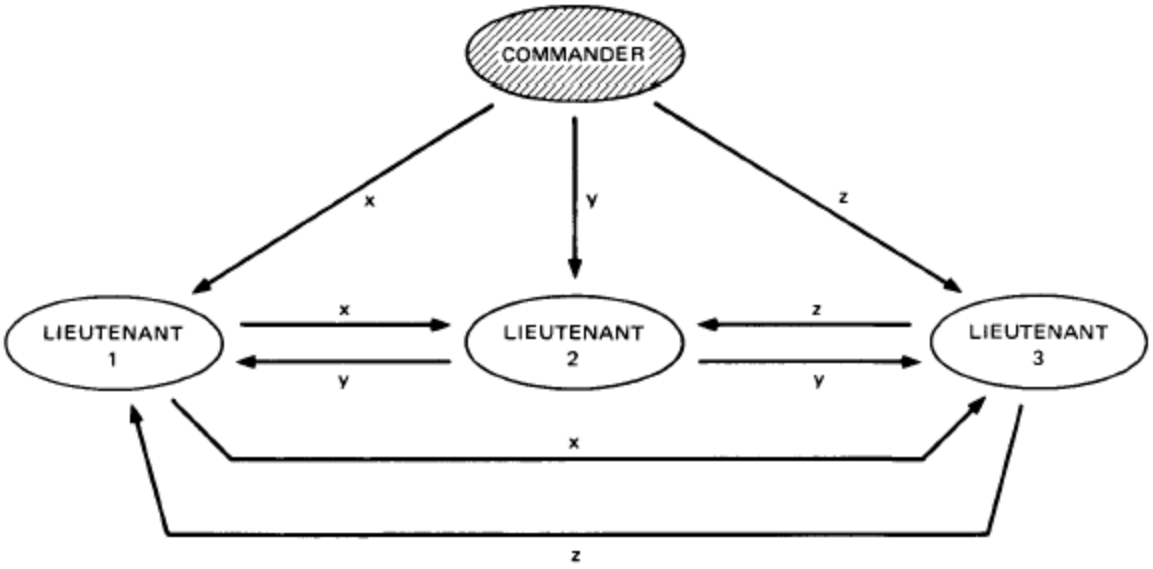


- this diagram is an incomplete summary of the protocol!

- Generals 1 and 2 receive (v, v, x).

- All loyal generals will obey the same orders. (IC1 & 2)

Example for n=4, m=1



- All generals receive (x, y, z), so majority() returns correct value for each general.

- IC1 is met, all loyal generals execute the same action.

Algorithm complexity

- What's the cost?

- OM(m) invokes (n-1) OM(m-1).
- OM(m-1) invokes (n-2) OM(n-2).
- …
- OM(m-k) will be called (n-1)(n-2)…(n-k) times.

- Long story short: algorithm complexity is $O(n^m)$.
  (note: m = number of failures)

Can we improve on this?

- Problem with oral messages: “He said she said…”

- Can’t tell if a relayed message was modified.

- What if we can prevent a relayed message from being changed?

- Use signatures.

Solution with signed messages

- One additional assumption required:

Assumptions:

A1. Each message sent is delivered correctly.

A2. Receiver knows who sent the message.

A3. Absence of a message can be detected.

A4. A loyal general's signature cannot be forged.

Solution with signed messages

- Commander sends a signed order to each general.

- Each general that receives the order verifies the signature and then:
  - puts the order into V if it has not seen that value before.
  - signs the message and then sends it on to other generals who have not received that message.

- Run choice (V).

Solution with signed messages

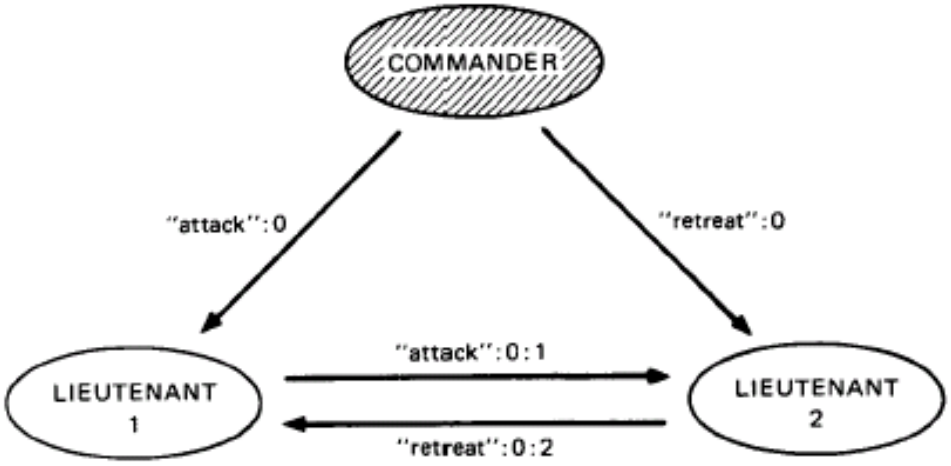


Fig. 5. Algorithm SM(1); the commander a traitor.

- Both generals 1 and 2 receive V = { attack, retreat }

- Since both have the same vector of values, choice (V) will be the same for both generals.

What's the benefit of signed messages?

- Improved resistance to traitors.

- You can have any number of traitors!

- In technical terms: SM(m) is resistant to m traitors.

Missing communication paths

- What if all-to-all communications isn’t possible?

- For oral messages with m traitors: 3m regular graph.

- For signed messages, connected graph.

x-regular graphs

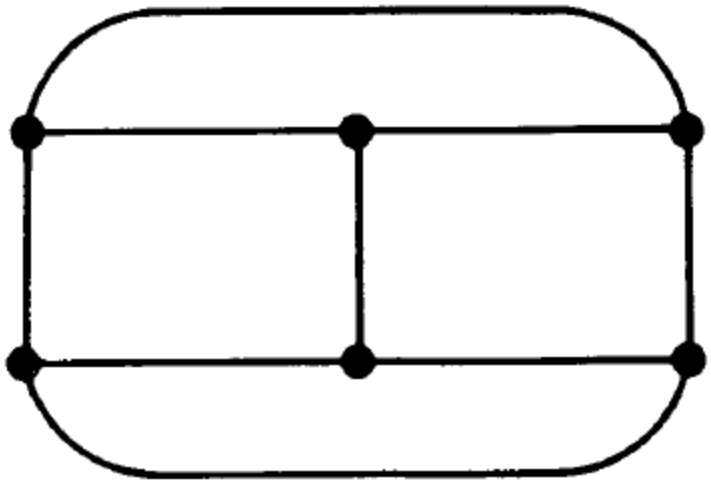


Fig. 6. A 3-regular graph.

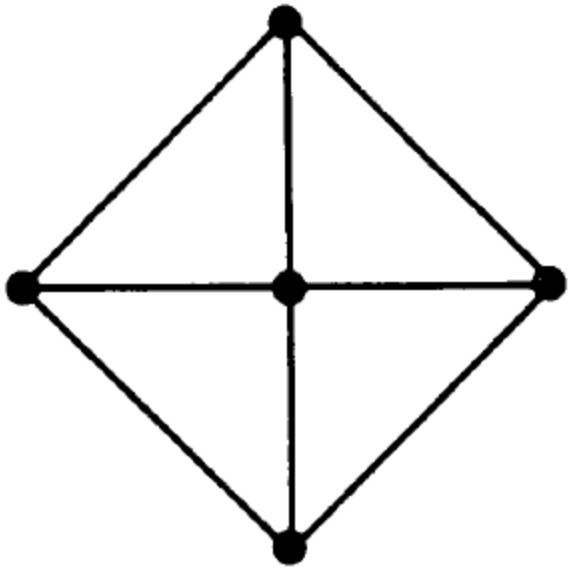


Fig. 7. A graph that is not 3-regular.

Concluding thoughts on BGP

- Large communication overheads.

- Need many replicas for fault tolerance using oral messages.

- What if you don’t know how many nodes will be prone to byzantine failures?

- What other problematic assumptions does this paper make?

Practical Byzantine fault tolerance

- We've seen the theory, but can we design something that actually works in practice?

About the authors

Miguel Castro, MSR

Barbara Liskov, MIT

Practical Byzantine fault tolerance

- Basic problem remains the same: provide a reliable answer despite Byzantine faults and arbitrary failure.

- Client should be able to:
  - send a request
  - wait for some small number of replies
  - be able to conclude that the answer is the correct one (as if the full consensus algorithm is ran).

What is wrong with other approaches?

- Theoretically feasible but inefficient.

- Many systems assume synchrony for correctness, which requires bounds on message delays and process speeds.

Assumptions made in this paper

- Unreliable network.

- Faulty nodes can behave arbitrarily.

- Strong cryptographic techniques for signed messages.

- A strong adversary is allowed.

Overview

- Byzantine fault tolerance through state machine replication.

- Replicas maintain service state.

- Use of cryptography to detect message corruption.

Views

- Replicas move through successive configurations called views.

- In each view, some node will be the primary; others are backups.

- The primary node is given by p = v mod n
  where v = view number and n = number of nodes.

Nodes

- Maintain state:
  - log
  - view number
  - state

- Every node can perform a set of operations
  - need not be simple reads/writes
  - but must be deterministic
  - and must start in the same state

How the algorithm works

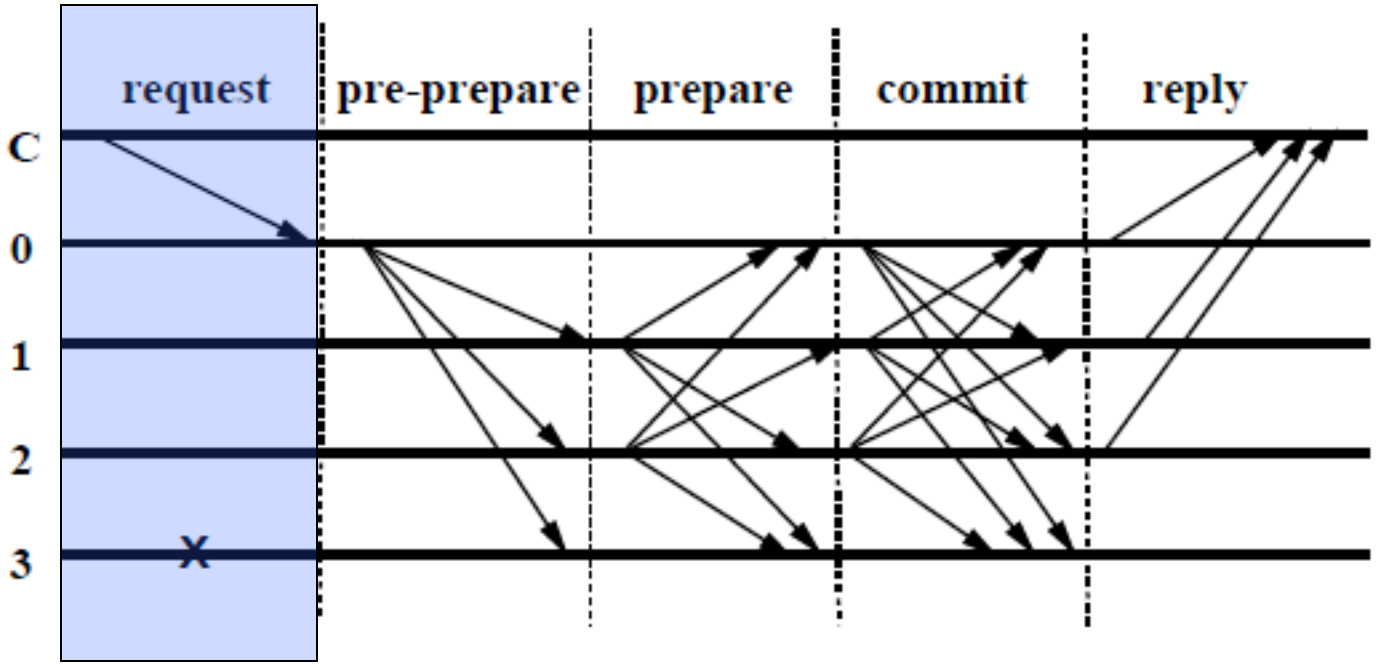
- Client issues the primary a request.

- Primary multicasts the request to all backup replicas.

- Replicas execute the request and returns a reply to the client.

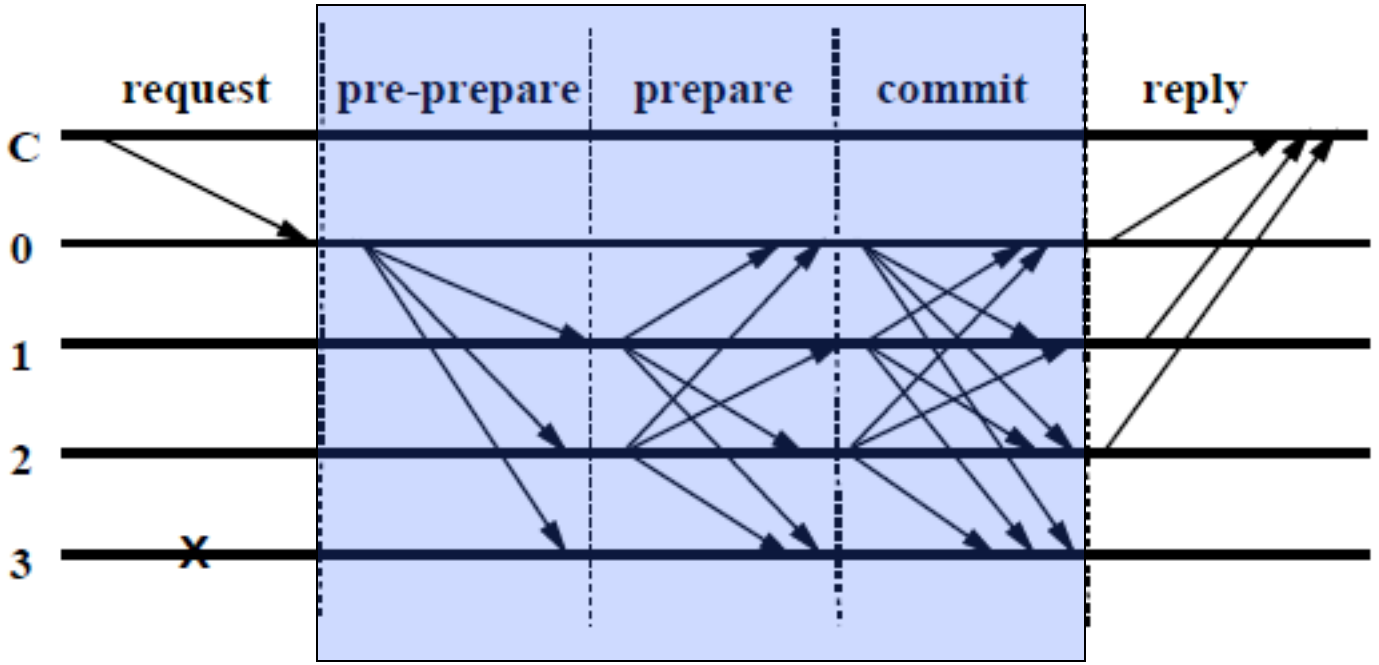
- Client waits for f+1 replies with the same result.

How the algorithm works



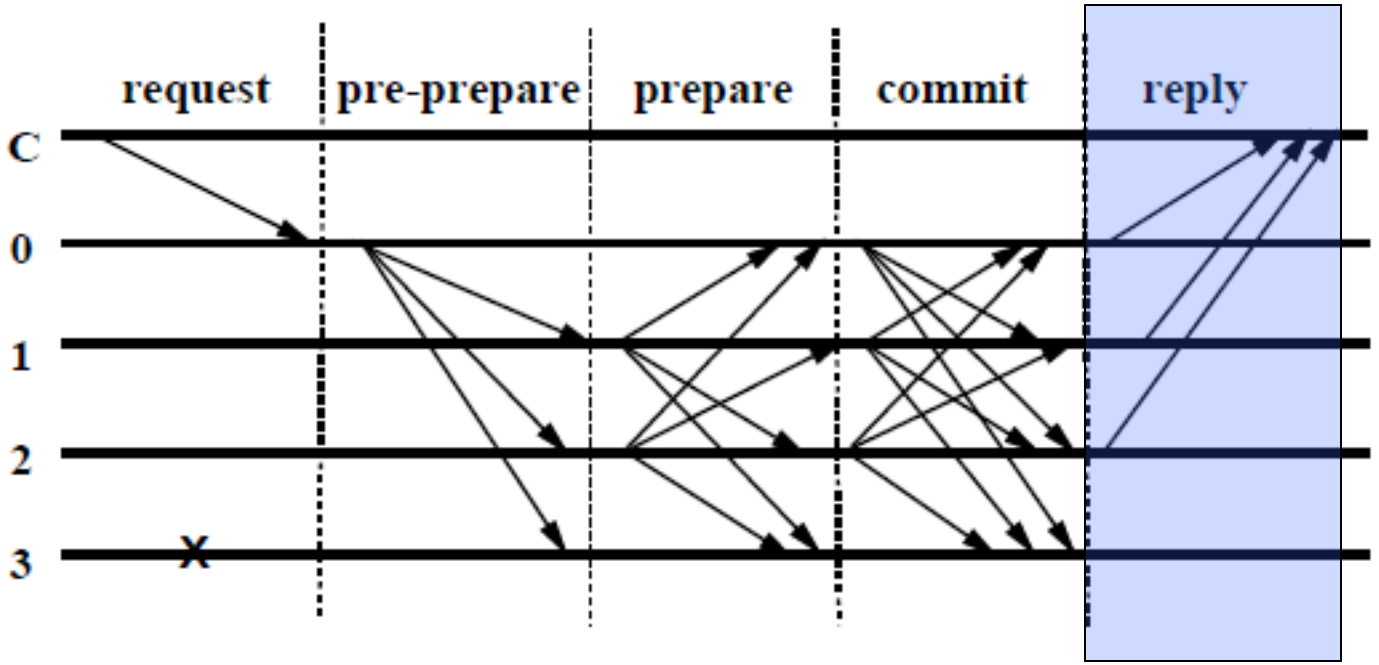
- Client issues the primary a request.

How the algorithm works



- 3 phase commit: primary multicasts the request to the backups.

How the algorithm works



- Replicas execute the request and reply the client.

Why the algorithm works

- Replicas start in the same state.

- Primary picks the ordering of operations.

- Operations are deterministic.

- f+1 similar responses ensures that the operation is correct.

- Doesn’t matter if the primary behaves incorrectly.

Byzantine Fault Tolerant File System

- BFS is implemented using a replication library.

- User-level relay processes communicate with NFS client and primary/replicas.

- When relay receives NFS requests, it invokes procedure in replication library and returns the result back.

- Only 3% performance penalty!

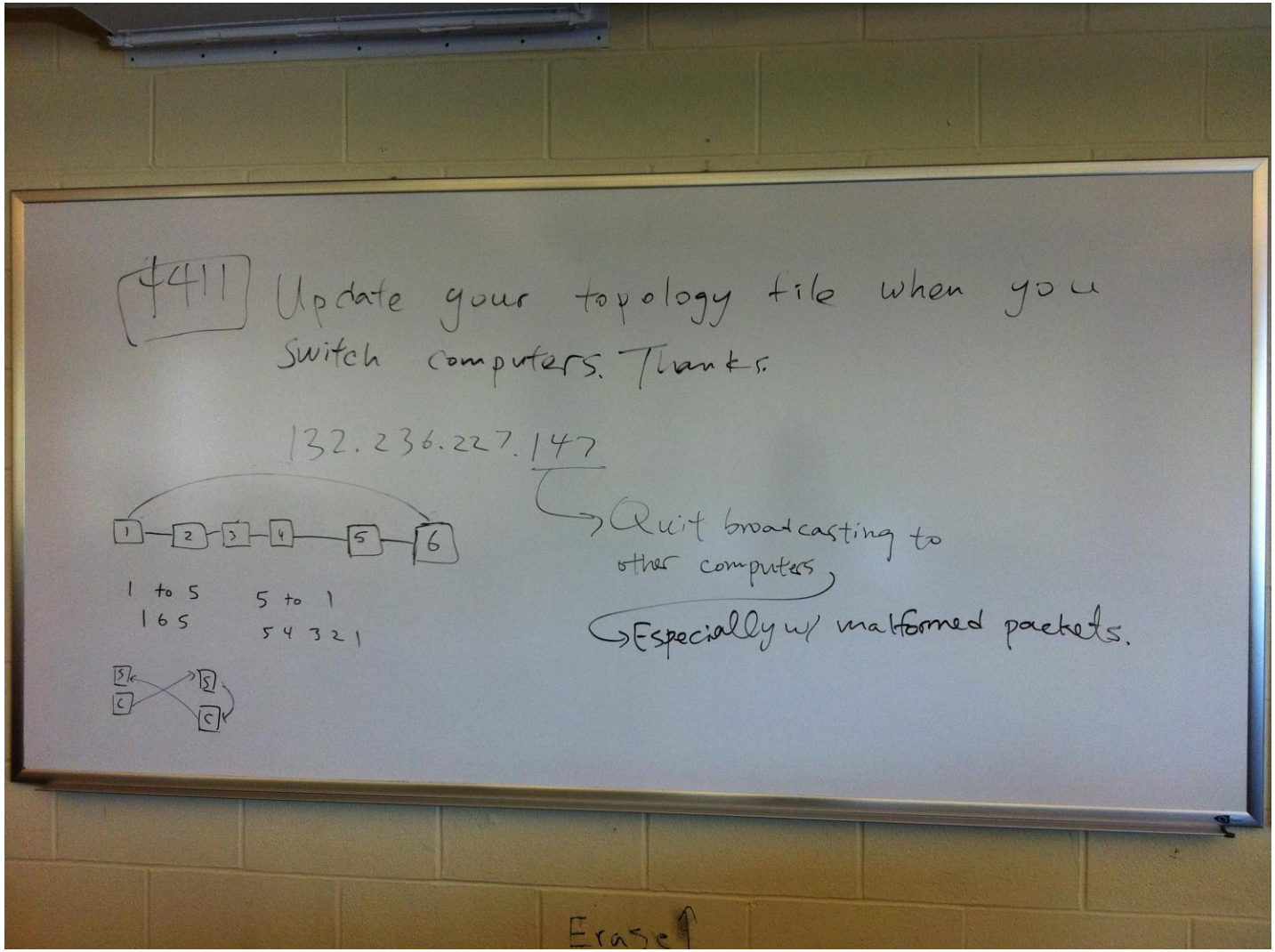
Concluding Thoughts

- Are the assumptions valid? N-versioning?

- Replication library not fully implemented. No N-versioning!

- Progress and performance aren’t the only important metrics: privacy.

Just by coincidence…



A quick comparison

m = traitors, n = total	Synchronous	Asynchronous
Oral messages: fails if	n <= 3m	m >=1
works if	n >= 3m+1	no guarantee
Signed messages: fails if	won't fail unless no correct processes	m >= 1
works if	n >= 1	no guarantee

3 simple takeaways

- 3f+1 required if messages are unsigned.

- If messages are signed, can tolerate any failure.

- Need synchronous network operation.