# Byzantine Techniques

Michael George

November 29, 2005

# Reliability and Failure

*"There can be no unity without agreement, and there can be no agreement without conciliation"*
*— René Maowad*

# Reliability and Failure

*"There can be no unity without agreement, and there can be no agreement without conciliation"*
*— René Maowad*

- We want reliable systems
- Until now, we've assumed that failures are fail-stop
- What happens if failures are arbitrary?

# Reliability and Failure

*"There can be no unity without agreement, and there can be no agreement without conciliation"*
*— René Maowad*

- We want reliable systems
- Until now, we've assumed that failures are fail-stop
- What happens if failures are arbitrary?
- . . . or even malicious?

# Today's Presentation

We will discuss two papers that address this worst-case scenario:

1. The Byzantine General's Problem [Lamport et. al. 1982]
    - Phrases the problem in terms of Byzantine Generals
    - Shows a tight upper bound on fault tolerance
    - Explores bounds under modified assumptions

# Today's Presentation

We will discuss two papers that address this worst-case scenario:

1. The Byzantine General's Problem [Lamport et. al. 1982]
    - Phrases the problem in terms of Byzantine Generals
    - Shows a tight upper bound on fault tolerance
    - Explores bounds under modified assumptions
2. Practical Byzantine Fault Tolerance [Castro and Liskov 1999]
    - Implements fault-tolerant state-machine replication
    - Aggressively optimizes the implementation
    - Layers replicated NFS over state-machine
    - Shows performance penalty is reasonable

# The Basic Problem

A group of Byzantine Generals are surrounding an enemy city.

- They need to jointly decide whether to attack or retreat.
- But some of them might be traitors.
- Want them to agree on a decision.

# The Basic Problem

A group of Byzantine Generals are surrounding an enemy city.

- They need to jointly decide whether to attack or retreat.
- But some of them might be traitors.
- Want them to agree on a decision.
- Decision must be good.

# Reducing Decision Making to Information Propogation

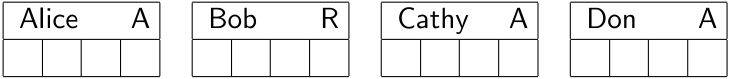
If a single commander can send information to some lieutenants such that:

- IC1 All loyal lieutenants recieve the same value
- IC2 If commander is loyal, then all lieutenants recieve value she sent

# Reducing Decision Making to Information Propogation

If a single commander can send information to some lieutenants such that:

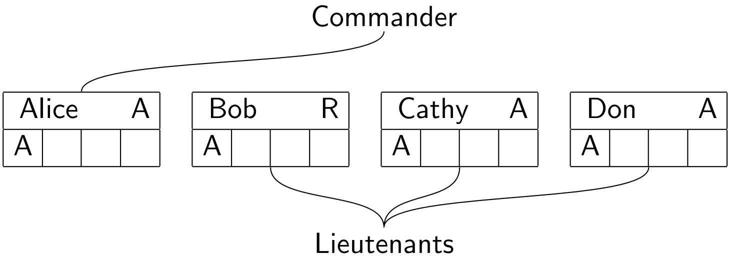
- **IC1** All loyal lieutenants recieve the same value
- **IC2** If commander is loyal, then all lieutenants recieve value she sent



# Reducing Decision Making to Information Propogation

If a single commander can send information to some lieutenants such that:

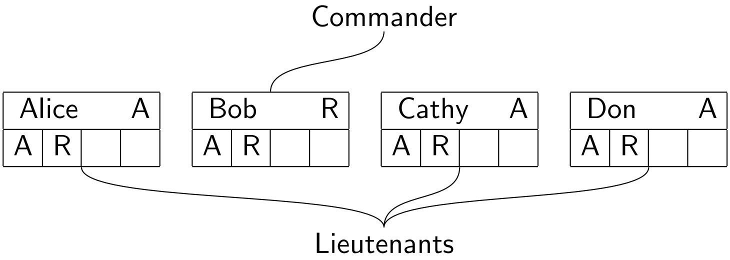
- **IC1** All loyal lieutenants recieve the same value
- **IC2** If commander is loyal, then all lieutenants recieve value she sent



# Reducing Decision Making to Information Propogation

If a single commander can send information to some lieutenants such that:

- **IC1** All loyal lieutenants recieve the same value
- **IC2** If commander is loyal, then all lieutenants recieve value she sent

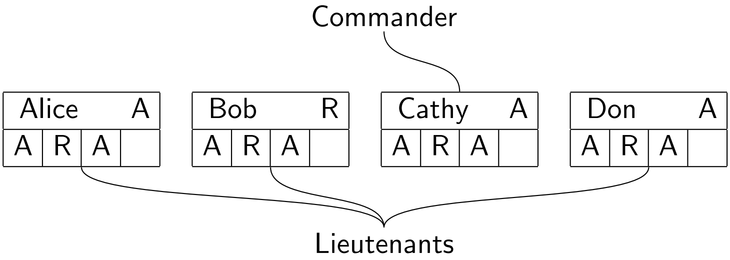


# Reducing Decision Making to Information Propogation

If a single commander can send information to some lieutenants such that:

IC1 All loyal lieutenants recieve the same value

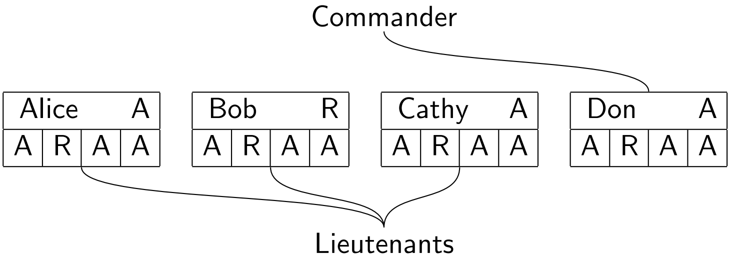
IC2 If commander is loyal, then all lieutenants recieve value she sent



# Reducing Decision Making to Information Propogation

If a single commander can send information to some lieutenants such that:

- **IC1** All loyal lieutenants recieve the same value
- **IC2** If commander is loyal, then all lieutenants recieve value she sent



# Reducing Decision Making to Information Propogation

If a single commander can send information to some lieutenants such that:

- IC1 All loyal lieutenants recieve the same value
- IC2 If commander is loyal, then all lieutenants recieve value she sent

| Alice | | | A |
|---|---|---|---|
| A | R | A | A |

| Bob | | | R |
|---|---|---|---|
| A | R | A | A |

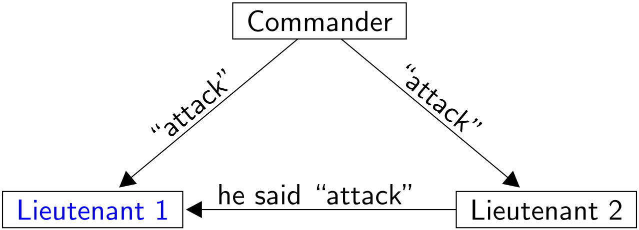
| Cathy | | | A |
|---|---|---|---|
| A | R | A | A |

| Don | | | A |
|---|---|---|---|
| A | R | A | A |

The *Byzantine General's Problem* is to send information in a way that satisfies IC1 and IC2.

# Impossibility With Three Generals

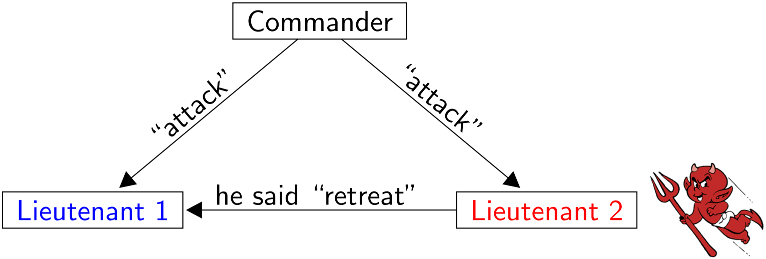
Consider the following:




| Commander says | Lieutenant 2 says | Lieutenant 1 concludes |
|---|---|---|
| "attack" | "attack" | "attack" |

# Impossibility With Three Generals

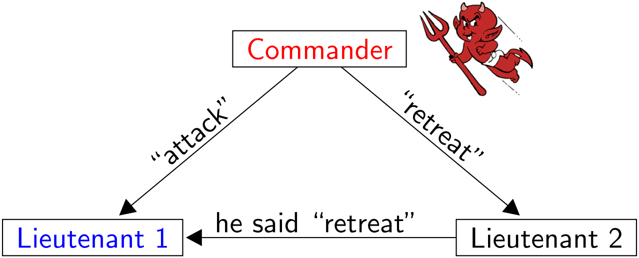
Consider the following:




| Commander says | Lieutenant 2 says | Lieutenant 1 concludes |
|---|---|---|
| “attack” | “attack” | “attack” |
| “attack” | “retreat” | “attack” |

# Impossibility With Three Generals

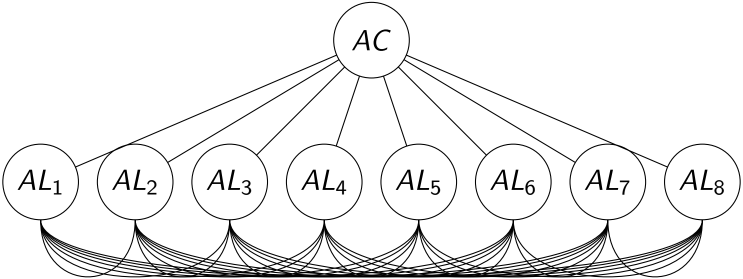
Consider the following:




| Commander says | Lieutenant 2 says | Lieutenant 1 concludes |
|---|---|---|
| "attack" | "attack" | "attack" |
| "attack" | "retreat" | "attack" |
| "attack" | "retreat" | "retreat" |

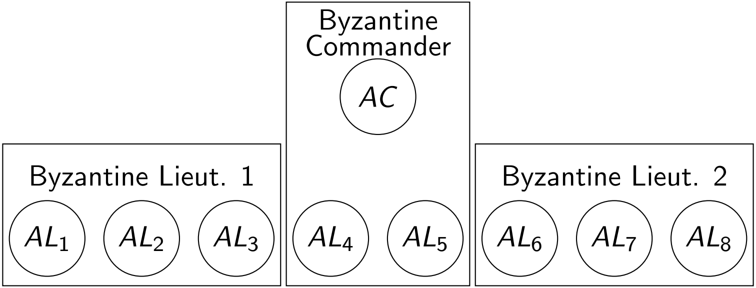
# Impossibility With $3m$ Generals

What if we can solve for $3m$ Albanian generals with $m$ failures?



# Impossibility With $3m$ Generals

What if we can solve for $3m$ Albanian generals with $m$ failures?



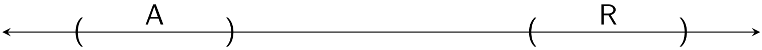

Then we can implement three Byzantine generals with one failure!

# Impossibility With Approximate Agreement

Can we do approximate (within a given $\delta$) agreement?

# Impossibility With Approximate Agreement

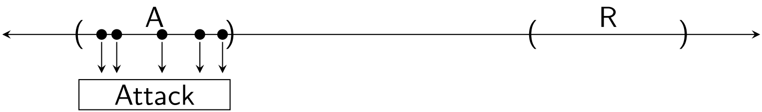
Can we do approximate (within a given $\delta$) agreement?




No - just have general choose points further then $2\delta$ apart.

# Impossibility With Approximate Agreement

Can we do approximate (within a given $\delta$) agreement?




No - just have general choose points further then $2\delta$ apart.
Now we've solved the exact problem.

# Oral Messages

Some assumptions:

- A1 Every message that is sent is delivered correctly
- A2 The reciever of a message knows who sent it
- A3 The absence of a message can be detected

# Oral Messages

Some assumptions:

- **A1** Every message that is sent is delivered correctly
- **A2** The reciever of a message knows who sent it
- **A3** The absence of a message can be detected
- **implicit** Only the sender and reciever can read a message

# Oral Messages

Some assumptions:

- A1 Every message that is sent is delivered correctly
- A2 The reciever of a message knows who sent it
- A3 The absence of a message can be detected
- implicit Only the sender and reciever can read a message

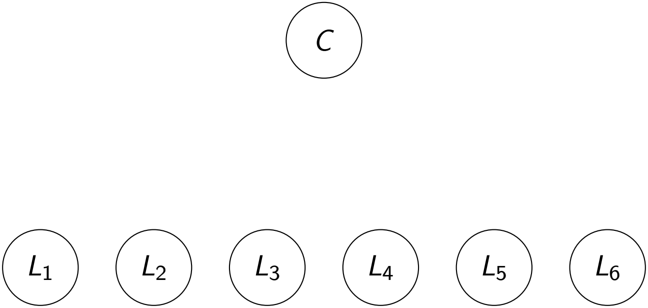
Also need a *majority* function:

- If a majority of $v_i$'s are $v$ then $majority(\vec{v}) = v$
- Can use the "majority or default" function or the median function

# The Oral Messages Algorithm

IC1 All loyal lieutenants recieve the same value

IC2 If commander is loyal, then all lieutenants recieve value she sent

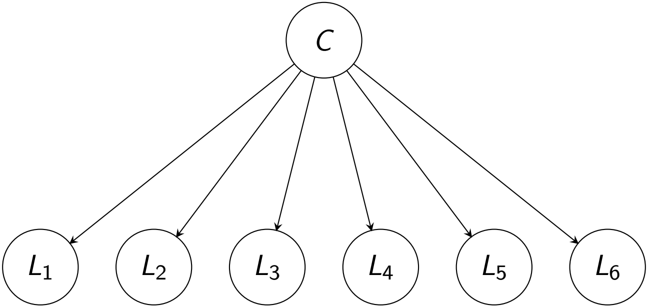


# The Oral Messages Algorithm

IC1 All loyal lieutenants recieve the same value

IC2 If commander is loyal, then all lieutenants recieve value she sent

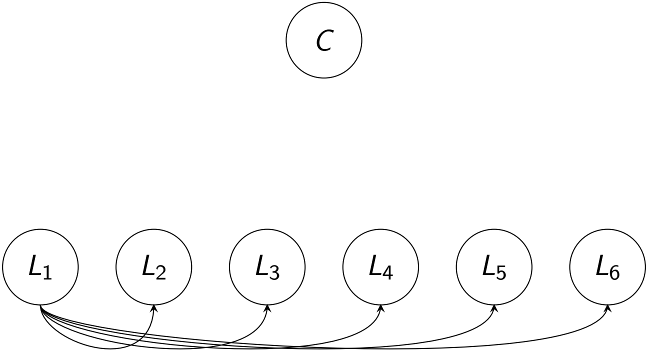
Step 1



# The Oral Messages Algorithm

- **IC1** All loyal lieutenants recieve the same value
- **IC2** If commander is loyal, then all lieutenants recieve value she sent

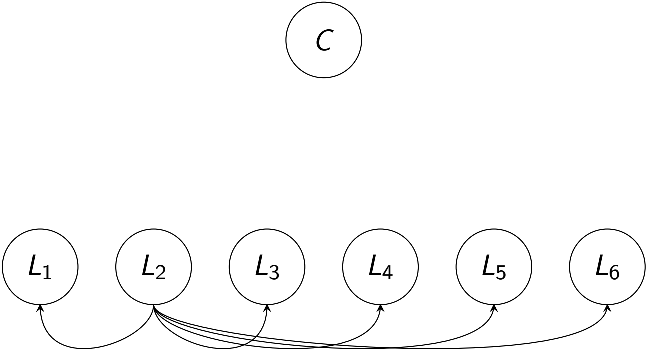
Step 2



# The Oral Messages Algorithm

- IC1 All loyal lieutenants recieve the same value
- IC2 If commander is loyal, then all lieutenants recieve value she sent

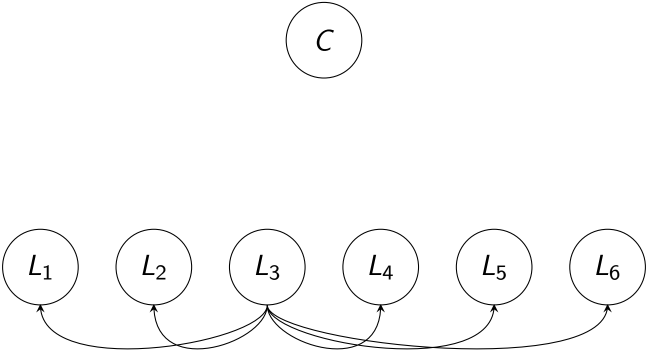
Step 2



# The Oral Messages Algorithm

- IC1 All loyal lieutenants recieve the same value
- IC2 If commander is loyal, then all lieutenants recieve value she sent

Step 2



# With Signed Messages (or broadcast)

Impossibility proof assumes that lieutenants can lie

- Can be prevented with digitial signatures
- Also with broadcast

# With Signed Messages (or broadcast)

Impossibility proof assumes that lieutenants can lie

- Can be prevented with digitial signatures
- Also with broadcast
- Authors provide $m + 2$ general algorithm that thwarts $m$ traitors

# With Restricted Communications

What if generals can only talk to certain (nearby) generals?
Under certain connectivity hypotheses:

- Almost the same basic algorithm works (add forwarding)
- Same bounds on number of traitors/generals
- Signed version also goes through as long as loyal generals connected

# "Certain Connectivity Hypotheses"

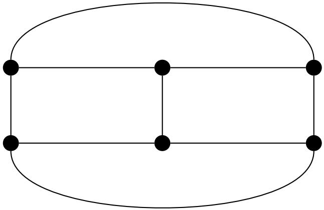
Definition:

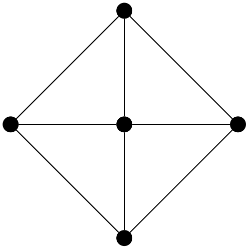
- A set $N$ of neighbors of $v$ is regular if for all $n \in N$ and all $v' \neq v$ there is a path $\gamma_{nv'}$ from $n$ to $v'$ not passing through $v$ or $\gamma_{n'v''}$
- A graph is $p$-regular if every node has a regular set of $p$ neighbors

# "Certain Connectivity Hypotheses"

Definition:

- A set $N$ of neighbors of $v$ is regular if for all $n \in N$ and all $v' \neq v$ there is a path $\gamma_{nv'}$ from $n$ to $v'$ not passing through $v$ or $\gamma_{n'v''}$
- A graph is $p$-regular if every node has a regular set of $p$ neighbors





# Problems With Lamport et. al.

The first paper was *theoretical*:

- Algorithms provided only as proof of existence
- Very impractical; synchronous execution
- Assume network is reliable

# Problems With Lamport et. al.

The first paper was *theoretical*:

- Algorithms provided only as proof of existence
- Very impractical; synchronous execution
- Assume network is reliable

The second paper aims for *practicality*.

- Algorithm is implemented as general-purpose library
- Assumptions model reality better
- Implementation is optimized and benchmarked

# Theoretical Limitations

Some hard limitations:

- Previous paper: need $3m + 1$ generals

# Theoretical Limitations

Some hard limitations:

- Previous paper: need $3m + 1$ generals
- FLP result: need synchrony

# Theoretical Limitations

Some hard limitations:

- Previous paper: need $3m + 1$ generals
- FLP result: need synchrony
- Can't avoid failures that are correct according to protocol

# Theoretical Limitations

Some hard limitations:

- Previous paper: need $3m + 1$ generals
- FLP result: need synchrony
- Can't avoid failures that are correct according to protocol

Given these limitations, the authors design a state machine replication protocol

# State Machine Replication

Replicated state machines are an abstract framework for distributed systems

- There is a shared global "state" of the system
- Events modify the state in a *deterministic* way
  - Client requests
  - Membership changes / failure

# State Machine Replication

Replicated state machines are an abstract framework for distributed systems

- There is a shared global "state" of the system
- Events modify the state in a *deterministic* way
  - Client requests
  - Membership changes / failure
- Replicated servers maintain local copy of state
  - Can act on state transitions

# State Machine Replication

Replicated state machines are an abstract framework for distributed systems

- There is a shared global "state" of the system
- Events modify the state in a *deterministic* way
  - Client requests
  - Membership changes / failure
- Replicated servers maintain local copy of state
  - Can act on state transitions
- If all replicas start in same state and all events propogated, then all replicas remain in the same state

# Normal Operation

The algorithm is a 3-phase commit protocol:

0. Client sends request to primary
   - If primary is down, broadcast request

# Normal Operation

The algorithm is a 3-phase commit protocol:

0. Client sends request to primary
   - If primary is down, broadcast request
1. Primary broadcasts PRE-PREPARE message to replicas
   - Just contains a sequence number, a view, and a signature
   - Message is piggybacked

# Normal Operation

The algorithm is a 3-phase commit protocol:

0. Client sends request to primary
   - If primary is down, broadcast request
1. Primary broadcasts PRE-PREPARE message to replicas
   - Just contains a sequence number, a view, and a signature
   - Message is piggybacked
2. When a replica recieves a PRE-PREPARE it broadcasts a PREPARE message

# Normal Operation

The algorithm is a 3-phase commit protocol:

0. Client sends request to primary
   - If primary is down, broadcast request
1. Primary broadcasts PRE-PREPARE message to replicas
   - Just contains a sequence number, a view, and a signature
   - Message is piggybacked
2. When a replica recieves a PRE-PREPARE it broadcasts a PREPARE message
3. When a replica recieves $2f$ PREPARE messages, it sends a COMMIT message

# Normal Operation

The algorithm is a 3-phase commit protocol:

0. Client sends request to primary
   - If primary is down, broadcast request
1. Primary broadcasts PRE-PREPARE message to replicas
   - Just contains a sequence number, a view, and a signature
   - Message is piggybacked
2. When a replica recieves a PRE-PREPARE it broadcasts a PREPARE message
3. When a replica recieves $2f$ PREPARE messages, it sends a COMMIT message
4. When a replica recieves $2f + 1$ commit messages, it changes its' local state

# View Changes

Like Paxos, we maintain a view of primary

1. When a replica thinks current primary has failed, broadcasts a VIEW-CHANGE message
   - Contains its best estimate of primary's state upon failure

# View Changes

Like Paxos, we maintain a view of primary

1. When a replica thinks current primary has failed, broadcasts a VIEW-CHANGE message
   - Contains its best estimate of primary's state upon failure
2. When the new primary recieves $2f$ VIEW-CHANGE messages it broadcasts NEW-VIEW to all other replicas
   - Contains proof that it really recieved VIEW-CHANGE messages

# Optimizations

Some optimizations to reduce communication delay:

- Client designates single server for reply; others send digest

# Optimizations

Some optimizations to reduce communication delay:

- Client designates single server for reply; others send digest
- Client can accept $2f + 1$ tentative replies instead of waiting for $f + 1$ actual replies

# Optimizations

Some optimizations to reduce communication delay:

- Client designates single server for reply; others send digest
- Client can accept $2f + 1$ tentative replies instead of waiting for $f + 1$ actual replies
- Reduced interaction in read-only case

Also use message authentication codes instead of public-key crypto for common case.

# Micro-Benchmarks

| arg./res. (KB) | replicated | | | | without replication | |
|---|---|---|---|---|---|---|
| | read-write | | read-only | | | |
| 0/0 | 3.35 | (309%) | 1.62 | (98%) | 0.82 | (0%) |
| 4/0 | 14.19 | (207%) | 6.98 | (51%) | 4.62 | (0%) |
| 0/4 | 8.01 | (72%) | 5.94 | (27%) | 4.66 | (0%) |

For the "worst-case scenario":

- Tests measure null operations
- Without replication is just "best-effort" (UDP)

The worst is about four times as slow

# Cost of Replication

| phase | BFS strict | | BFS r/o lookup | | BFS-nr | |
|---|---|---|---|---|---|---|
| 1 | 0.55 | (57%) | 0.47 | (34%) | 0.35 | (0%) |
| 2 | 9.24 | (82%) | 7.91 | (56%) | 5.08 | (0%) |
| 3 | 7.24 | (18%) | 6.45 | (6%) | 6.11 | (0%) |
| 4 | 8.77 | (18%) | 7.87 | (6%) | 7.41 | (0%) |
| 5 | 38.68 | (20%) | 38.38 | (19%) | 32.12 | (0%) |
| total | 64.48 | (26%) | 61.07 | (20%) | 51.07 | (0%) |

This benchmark measures the cost of replication:

- BFS-nr is the same as BFS but performs no replication
- It is unsafe because reports that result is stable before it is

# Cost of Fault Tolerance

| phase | BFS strict | | BFS r/o lookup | | NFS-std | |
|---|---|---|---|---|---|---|
| 1 | 0.55 | (-69%) | 0.47 | (-73%) | 1.75 | (0%) |
| 2 | 9.24 | (-2%) | 7.91 | (-16%) | 9.46 | (0%) |
| 3 | 7.24 | (35%) | 6.45 | (20%) | 5.36 | (0%) |
| 4 | 8.77 | (32%) | 7.87 | (19%) | 6.60 | (0%) |
| 5 | 38.68 | (-2%) | 38.38 | (-2%) | 39.35 | (0%) |
| total | 64.48 | (3%) | 61.07 | (-2%) | 62.52 | (0%) |

This test measures the cost of fault tolerance:

- NFS-std is the standard implementation of NFS
- Some numbers are negative (!)
- Best numbers (r/o lookup) not quite fair

# Take-home Messages

First paper:

- Possible to tolerate $m$ traitors with $3m + 1$ generals
- Not possible with fewer

# Take-home Messages

First paper:

- Possible to tolerate $m$ traitors with $3m + 1$ generals
- Not possible with fewer
- Signatures make it much easier
- Connectivity doesn't make it much harder

# Take-home Messages

First paper:

- Possible to tolerate $m$ traitors with $3m + 1$ generals
- Not possible with fewer
- Signatures make it much easier
- Connectivity doesn't make it much harder

Second paper:

- Byzantine techniques are reasonable to use in practice
- Can even improve performance by replacing slow disk with fast distributed processors

# Thoughts for Discussion

- Are byzantine assumptions worthwhile?
  - Who does *n*-version programming anyway?
  - Does it really help?

# Thoughts for Discussion

- Are byzantine assumptions worthwhile?
  - Who does *n*-version programming anyway?
  - Does it really help?
- Can this be done better at a lower level (e.g. broadcast)?
  - Lamport et. al. say no
  - Need to be careful to avoid circularity

# Thoughts for Discussion

- Are byzantine assumptions worthwhile?
  - Who does *n*-version programming anyway?
  - Does it really help?
- Can this be done better at a lower level (e.g. broadcast)?
  - Lamport et. al. say no
  - Need to be careful to avoid circularity
- What about graceful failure?