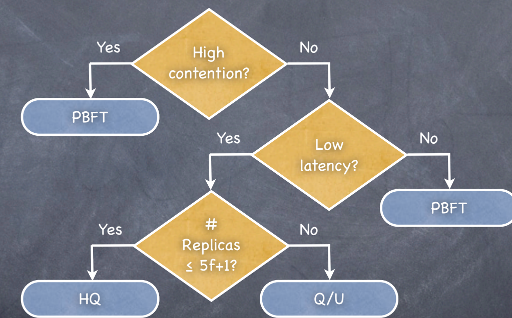
# Zyzzyva

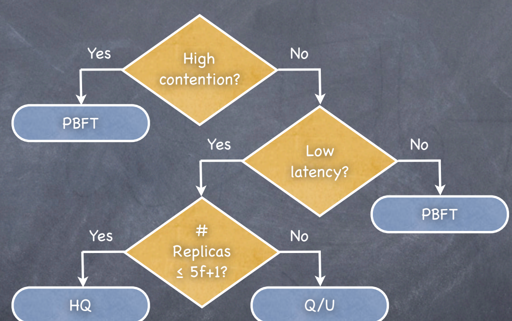
# Why then another BFT protocol?




- Complex decision tree hampers BFT adoption

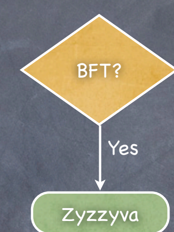
# “Simplify, simplify”

H.D. Thoreau




# “Simplify, simplify”

H.D. Thoreau



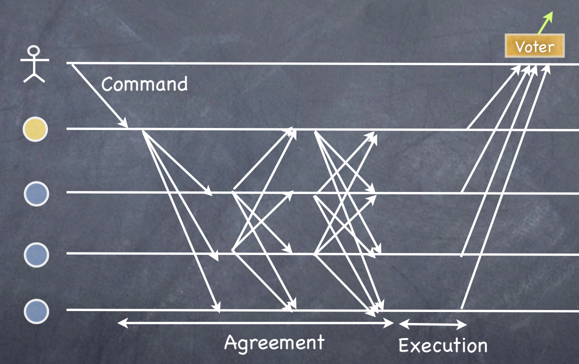

- One protocol that matches or tops its competitors in

✓ latency ✓ throughput ✓ cost of replication

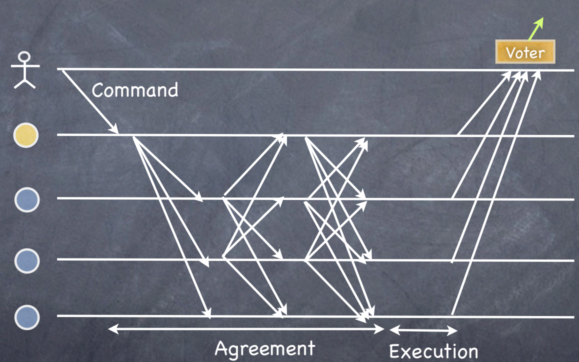
# Replica coordination

- All correct replicas execute the same sequence of commands
- For each received command $c$, correct replicas:
  - Agree on $c$'s position in the sequence
  - Execute $c$ in the agreed upon order
  - Replies to the client

# How it is done now




# How Zyzzyva does it




# Stability

- A command is stable at a replica once its position in the sequence cannot change

**RSM Safety**

Correct clients only process replies to stable commands

**RSM Liveness**

All commands issued by correct clients eventually become stable and elicit a reply

# Enforcing safety

- RSM safety requires:
  - Correct clients only process replies to stable commands
- ...but RSM implementations enforce instead:
  - Correct replicas only execute and reply to commands that are stable
- Service performs an output commit with each reply

# Speculative BFT: "Trust, but Verify"

- Insight: output commit at the client, not at the service!
- Replicas execute and reply to a command without knowing whether it is stable
  - trust order provided by primary
  - no explicit replica agreement!
- Correct client, before processing reply, verifies that it corresponds to stable command
  - if not, client takes action to ensure liveness

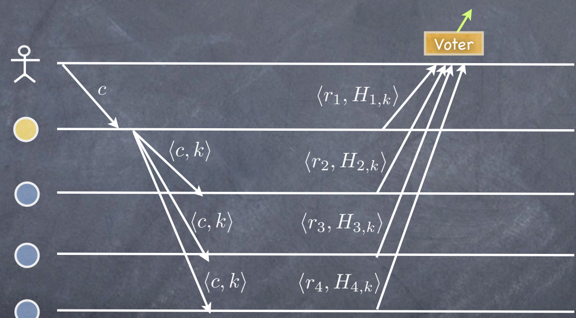
# Verifying stability

- Necessary condition for stability in Zyzzyva:
  A command $c$ can become stable only if a majority of correct replicas agree on its position in the sequence
- Client can process a response for $c$ iff:
  - a majority of correct replicas agrees on $c$'s position
  - the set of replies is incompatible, for all possible future executions, with a majority of correct replicas agreeing on a different command holding $c$'s current position

# Command History

- $H_{i,k}$ = a hash of the sequence of the first $k$ commands executed by replica $i$
- On receipt of a command $c$ from the primary, replica appends $c$ to its command history
- Replica reply for $c$ includes:
  - the application-level response
  - the corresponding command history

# Case 1: Unanimity




- Client processes response if all replies match:
  $r_1 = \ldots = r_4 \wedge H_{1,k} = \ldots = H_{4,k}$

# Safe?

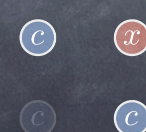
- ✓ A majority of correct replicas agrees on $c$'s position (all do!)
- If primary fails
  - New primary determines k-th command by asking $n-f$ replicas for their $H$

# Safe?

- ✓ A majority of correct replicas agrees on $c$'s position (all do!)
- If primary fails
  - New primary determines k-th command by asking $n-f$ replicas for their $H$


# Safe?

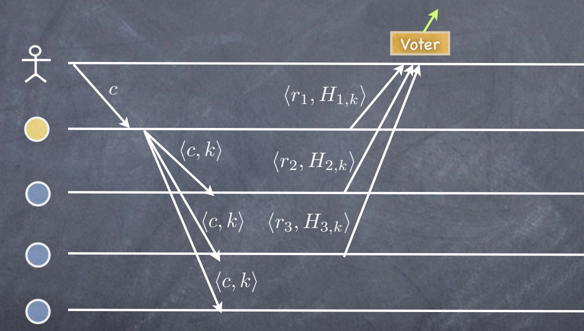
- ✓ A majority of correct replicas agrees on $c$'s position (all do!)
- If primary fails
  - New primary determines k-th command by asking $n-f$ replicas for their $H$



# Safe?

- ✓ A majority of correct replicas agrees on $c$'s position (all do!)
- If primary fails
  - New primary determines $c$'s position by asking $n-f$ replicas for their $H$
- ✓ It is impossible for a majority of correct replicas to agree on a different command for $c$'s position

# Case 2: A majority of correct replicas agree



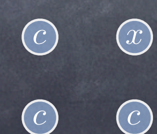

- At least $2f+1$ replies match

# Safe?

- ✓ A majority of correct replicas agrees on $c$'s position
- If primary fails
  - New primary determines k-th command by asking $n-f$ replicas for their $H$

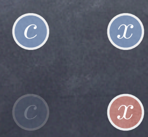
# Safe?

- ✓ A majority of correct replicas agrees on $c$'s position
- If primary fails
  - New primary determines $k$-th command by asking $n-f$ replicas for their $H$



# Safe?

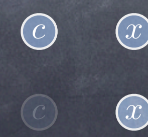
- ✓ A majority of correct replicas agrees on $c$'s position
- If primary fails
  - New primary determines $k$-th command by asking $n-f$ replicas for their $H$





# Safe?

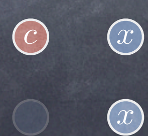
- ✓ A majority of correct replicas agrees on $c$'s position
- If primary fails
  - New primary determines $k$-th command by asking $n-f$ replicas for their $H$



# Safe?

- ✓ A majority of correct replicas agrees on $c$'s position
- If primary fails
  - New primary determines $k$-th command by asking $n-f$ replicas for their $H$

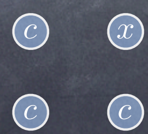


# Safe?

- ✓ A majority of correct replicas agrees on $c$'s position
- If primary fails
  - New primary determines $k$-th command by asking $n-f$ replicas for their $H$


# Safe?

- ✓ A majority of correct replicas agrees on $c$'s position
- If primary fails
  - New primary determines $k$-th command by asking $n-f$ replicas for their $H$



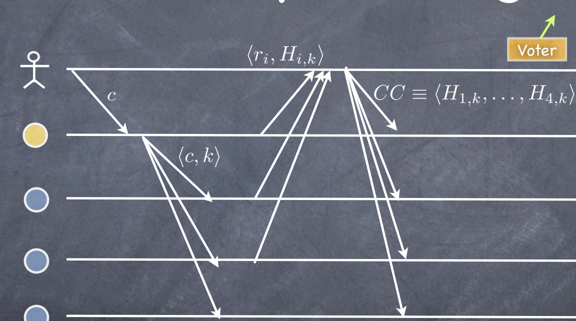

# Safe?

- ✓ A majority of correct replicas agrees on $c$'s position
- If primary fails
  - New primary determines $k$-th command by asking $n-f$ replicas for their $H$


# Safe?

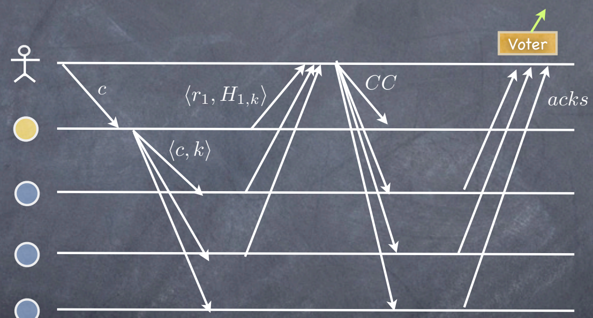
- ✓ A majority of correct replicas agrees on $c$'s position
- If primary fails
  - New primary determines k-th command by asking $n-f$ replicas for their $H$
- Not safe!

# Case 2: A majority of correct replicas agree




- Client sends to all a commit certificate containing $2f+1$ matching histories

# Case 2: A majority of correct replicas agree




- Client processes response if it receives at least $2f+1$ acks

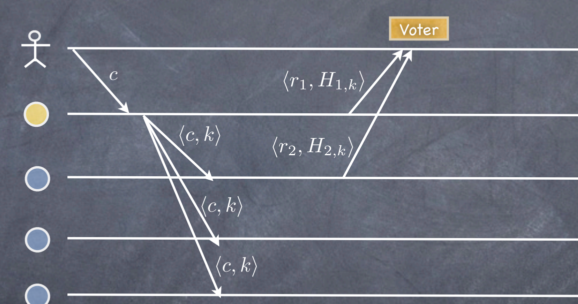
# Safe?

- Certificate proves that a majority of correct replicas agreed on $c$'s position
- If primary fails
  - New primary determines k-th command by contacting $n-f$ replicas
  - This set contains at least one correct replica with a copy of the certificate
- ✓ Incompatible with a majority backing a different command for that position

# Stability and command histories

- Stability depends on matching command histories
- Stability is prefix-closed:
  - If a command with sequence number $n$ is stable, then so is every command with sequence number $n' < n$

# Case 3: None of the above

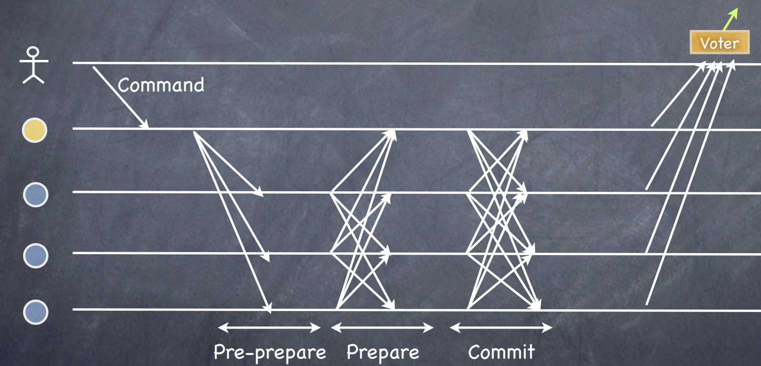



- Fewer than $2f+1$ replies match
- Clients retransmits $c$ to all replicas-hinting primary may be faulty

# Zyzzyva recap

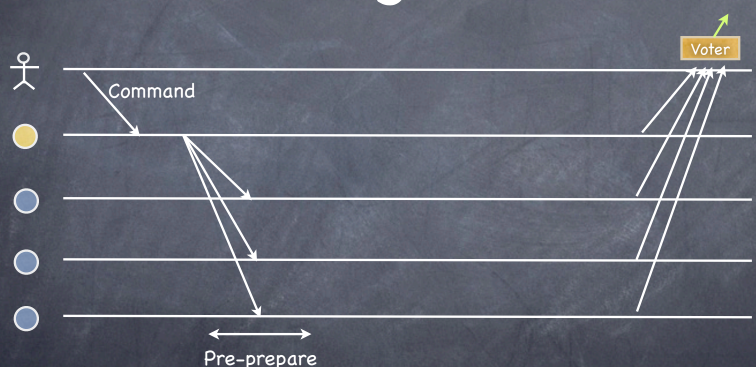
- Output commit at the client, not the service
- Replicas execute requests without explicit agreement
- Client verifies if response corresponds to stable command
- At most 2 phases within a view to make command stable

# The Case of the Missing Phase



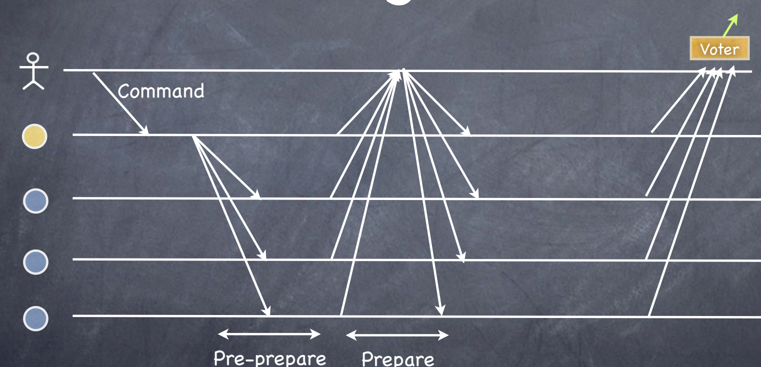

- Client processes response if it receives at least $f+1$ matching replies after commit phase

# The Case of the Missing Phase



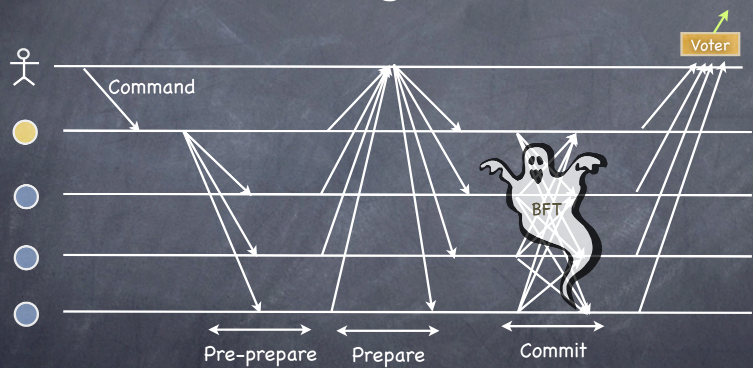

Unanimity

# The Case of the Missing Phase




Majority

# The Case of the Missing Phase




- Where did the third phase go?
- Why was it there to begin with?

# View-Change: replacing the primary

- In PBFT, a replica that suspects primary is faulty goes unilaterally on strike
  - Stops processing messages in the view
  - Third "Commit" phase needed for liveness

# View-Change: replacing the primary

- In PBFT, a replica that suspects primary is faulty goes unilaterally on strike
  - Stops processing messages in the view
  - Third "Commit" phase needed for liveness
- In Zyzzyva, the replica goes on "Technion strike"
  - Broadcasts "I hate the primary" and keeps on working
  - Stops when sees enough hate mail to ensure all correct replica will stop as well
- Extra phase is moved to the uncommon case

# Faulty clients can't affect safety

- Faulty clients cannot create inconsistent commit certificates
  - Clients cannot fabricate command histories, as they are signed by replicas
  - It is impossible to generate a valid commit certificate that conflicts with the order of any stable request
    - Stability is prefix closed!

# "Olly Olly Oxen Free!"

or, faulty clients can't affect liveness

# "Olly Olly Oxen Free!"

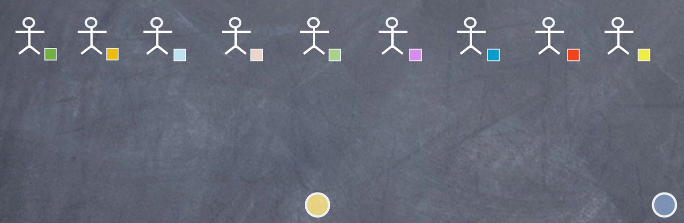
or, faulty clients can't affect liveness

- Faulty client omits to send CC for $c$
- Replicas commit histories are unaffected!
- Later correct client who establishes $c' > c$ is stable "frees" $c$ as well
  - Stability is prefix closed

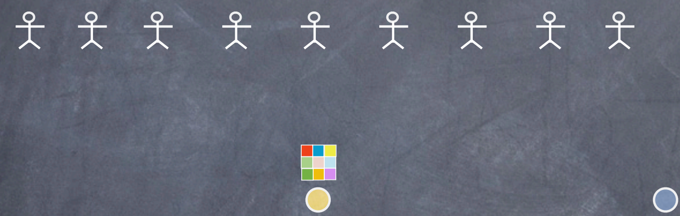
# Optimizations

- Checkpoint protocol to garbage collect histories
- Optimizations include:
  - Replacing digital signatures with MAC
  - Replicating application state at only $2f+1$ replicas
  - Batching
  - Zyzzyva5

# Batching



# Batching



- Only one history digest for all requests in the batch-amortizes crypto operations

# Throughput

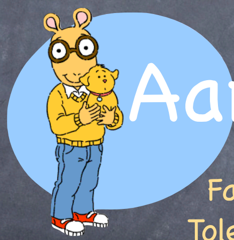
# Throughput

| | Best case |
|---|---|
| PBFT | 62K |
| QU | 24K |
| HQ | 15K |
| Zyzzyva | 80K |

# BFT: From Z To A

Zyzzyva

# BFT: From Z To A



Aardvark

Making Byzantine
Fault Tolerant Systems
Tolerate Byzantine Faults

# Paved with good intentions

- No BFT protocol should rely on synchrony for safety
- FLP: No consensus protocol can be both safe and live in an asynchronous system
    - All one can guarantee is eventual progress

# Paved with good intentions

- No BFT protocol should rely on synchrony for safety
- FLP: No consensus protocol can be both safe and live in an asynchronous system
    - All one can guarantee is eventual progress
- "Handle normal and worst case separately as a rule, because the requirements for the two are quite different:
  the normal case must be fast;
  the worst case must make some progress"
  -- Butler Lampson, "Hints for Computer System Design"

# The road more traveled

- Maximize performance when
    - the network is synchronous
    - all clients and servers behave correctly
- While remaining
    - safe if at most $f$ servers fail
    - eventually live

# The Byzantine Empire (565 AD)

Synchronous, no failures

Asynchronous

Synchronous, with faults!

# The Byzantine Empire (circa 2009 AD)

Synchronous, with or without failures

Asynchronous

# Recasting the problem

- Misguided
- Maximize performance when
  - the network is synchronous
  - all clients and servers behave correctly
- Dangerous
- While remaining
  - safe if at most $f$ servers fail
  - eventually live
- Futile

# Recasting the problem

- Misguided
  - it encourages systems that fail to deliver BFT
- Dangerous
- Futile

# Recasting the problem

- Misguided
  - it encourages systems that fail to deliver BFT
- Dangerous
  - it encourages fragile optimizations
- Futile

# Recasting the problem

- Misguided
  - it encourages systems that fail to deliver BFT
- Dangerous
  - it encourages fragile optimizations
- Futile
  - it yields diminishing return on common case

# BFT: a blueprint

- Build the system around execution path that:
  - provides acceptable performance across the broadest set of executions
  - it is easy to implement
  - it is robust against Byzantine attempts to push the system away from it

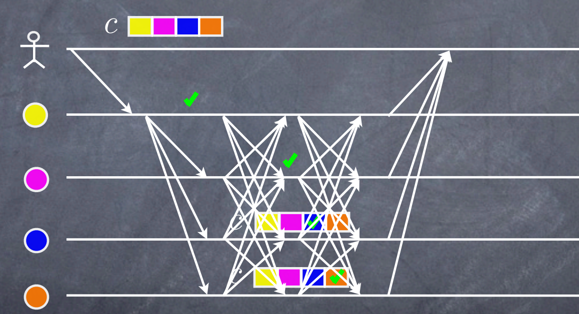
# Revisiting conventional wisdom

- Signatures are expensive - use MACs
- View changes are to be avoided
- Hardware multicast is a boon

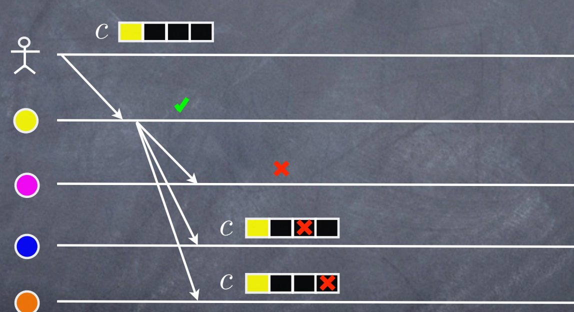
## Revisiting conventional wisdom

- Signatures are expensive – use MACs
  - Faulty clients can use MACs to generate ambiguity
- View changes are to be avoided
- Hardware multicast is a boon

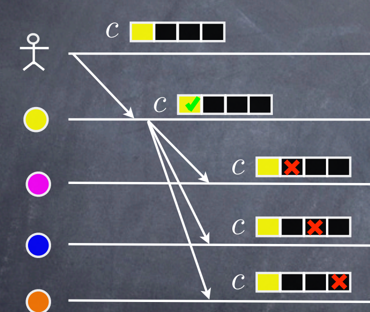
## Big MAC Attack



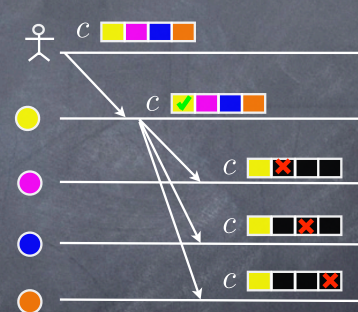

## Big MAC Attack




## Big MAC Attack

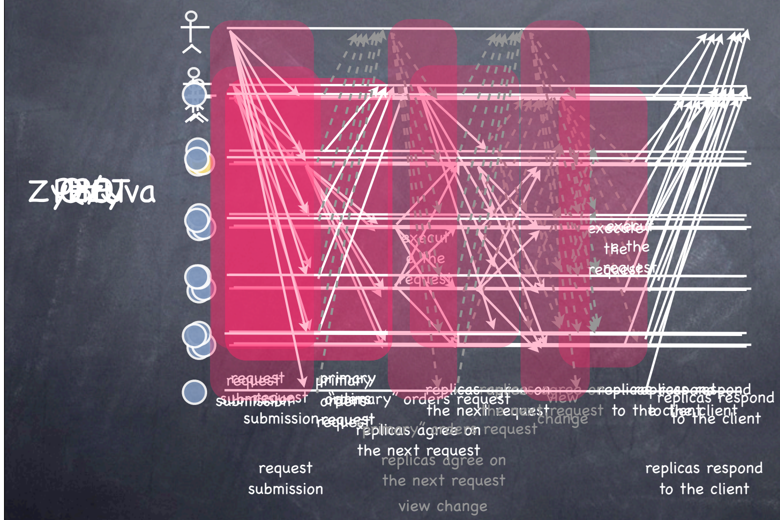



Faulty Client




Faulty Primary

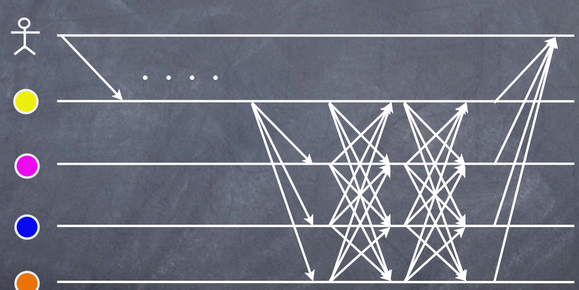
# Big MAC Attack



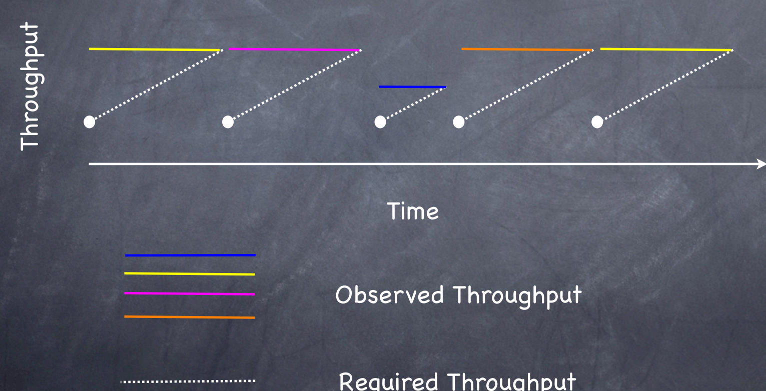

# Revisiting conventional wisdom

- Signatures are expensive - use MACs
  - Faulty clients can use MACs to generate ambiguity
  - Aardvark requires clients to sign requests
- View changes are to be avoided
- Hardware multicast is a boon

# Slow Primary



# Adaptive View Changes



# Revisiting conventional wisdom

- Signatures are expensive – use MACs
  - Faulty clients can use MACs to generate ambiguity
  - Aardvark requires clients to sign requests
- View changes are to be avoided
  - Aardvark uses regular view changes to maintain high throughput despite faulty primaries
- Hardware multicast is a boon

# Revisiting conventional wisdom

- Signatures are expensive – use MACs
  - Faulty clients can use MACs to generate ambiguity
  - Aardvark requires clients to sign requests
- View changes are to be avoided
  - Aardvark uses regular view changes to maintain high throughput despite faulty primaries
- Hardware multicast is a boon
  - Aardvark uses separate work queues for clients and individual replicas

# Throughput

| | Best case | Faulty Client | Client Flood | Faulty Primary | Faulty Replica |
|---|---|---|---|---|---|
| PBFT | 62K | 0 | crash | 1k | 250 |
| QU | 24K | 0 | crash | NA | 19K |
| HQ | 15K | NA | 4.5K | NA | crash |
| Zyzzyva | 80K | 0 | crash | crash | 0 |
| Aardvark | 39K | 39K | 7.8K | 37K | 11K |

# BAR Protocols for MAD Services


Lorenzo Alvisi
University of Texas at Austin


---

How to build a service without an a priori guarantee that any node will follow the protocol?

---

"We were put on this Earth to help others.
Why others were put here is beyond me."
-W. H. Auden


---


"We were put on this Earth to help others.
Why others were put here is beyond me."
-W. H. Auden


# MAD Systems

- Multiple Administrative Domain services
  - Nodes controlled by different entities
- Challenges
  - Nodes may fail
    - How do you build protocols when nodes may fail in arbitrary ways?
  - Black box
    - How do you build protocols when some nodes are black boxes whose internals are unknown?
  - Competing interests
    - How do you build protocols when nodes may have an incentive to cheat?

# Who's to blame


Allen Clement (MPI SWS)


Harry Li (Facebook)


Jean-Philippe Martin (MSR)


Edmund Wong


Lorenzo Alvisi


Mike Dahlin

# Examples

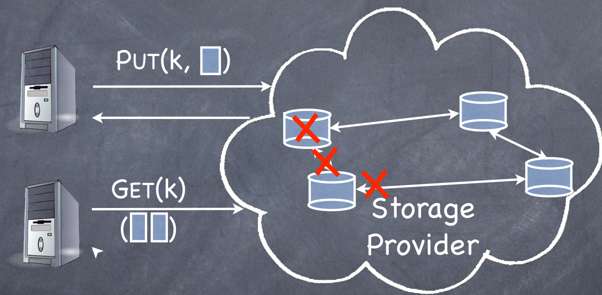
- P2P Services
  - Just TRB [DSN08]
  - BAR Backup [SOSP05]
  - BAR Gossip [OSDI06]
  - Flightpath Live Streaming [OSDI08]
- Cloud Storage
  - SafeStore [USENIX07]
  - Depot [OSDI10]

# This Talk

How to build a service without an a priori guarantee that any nodes will follow the protocol?

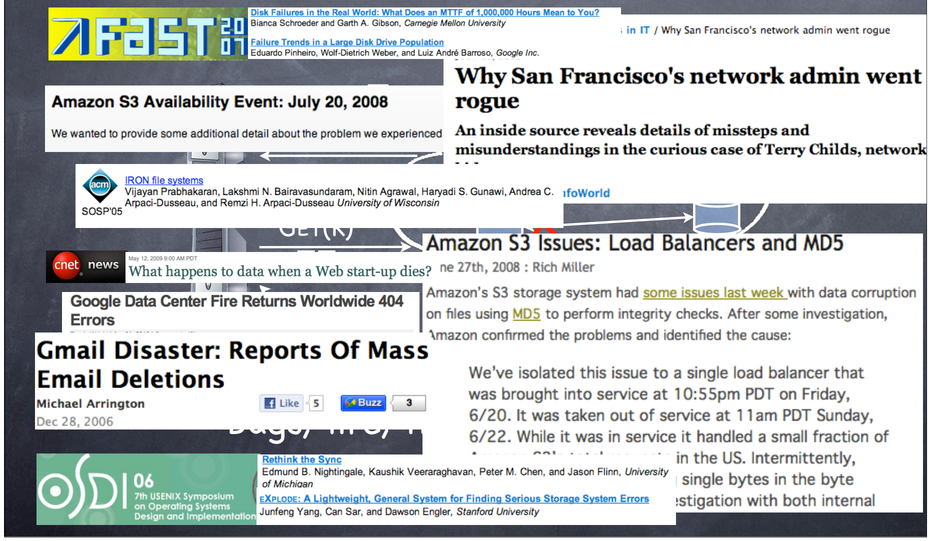
- BAR model
- BAR Services
  - Flightpath (P2P live streaming)
- Open Questions

# Failures 1: Nodes Can Break

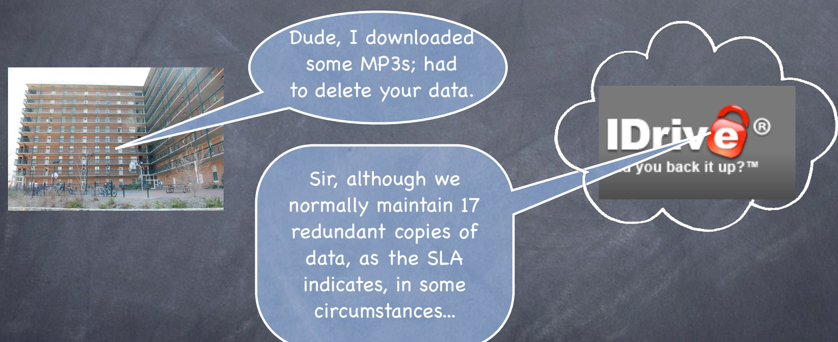



Disk crash, network failure, machine crash, etc.

# Failures 1: Nodes Can Break




# Failures 2: Selfishness




Rational "failure"

- Minimize work, maximize gain

# Byzantine Model

- Tolerates arbitrary deviations from specification
- Limits number $f$ of faulty nodes
  - e.g. Agreement requires $f < n/3$
- Assumes all other nodes are correct

# Byzantine Model

- Tolerates arbitrary deviations from specification
- Limits number $f$ of faulty nodes
  - e.g. Agreement requires $f < n/3$
- Assumes all other nodes are correct

Inappropriate when all nodes
may deviate when in their interest

# Rational Model

- All nodes are rational
- Rational nodes can deviate selfishly from their specification
- Does not tolerate Byzantine behavior
  - Broken nodes may violate assumptions
  - Malicious nodes may cause unbounded damage

Inappropriate when some node
may deviate against its interest

# BAR: A Failure Model for Cooperative Services

- Three classes of nodes
  - Byzantine: Deviate in any way, for any reason
    - Typically bound number of Byzantine nodes
  - Altruistic: Don't deviate (obedient)
  - Rational: Deviate iff in their interest
    - Typically no bound on number of Rational nodes

# BAR Research Agenda

1. New model
   Develop a model in which it is possible to prove properties about MAD services
2. Is the model usable?
   Understand how to simplify the development of MAD services in the new model
3. Is the model practical?
   Demonstrate that MAD services developed under the new model can be efficient, effective

# Outline

How to build a service without an a priori guarantee that any nodes will follow the protocol?

- BAR model
- BAR Services
  - Flightpath (P2P live streaming)
- Open Questions

# P2P Live Streaming

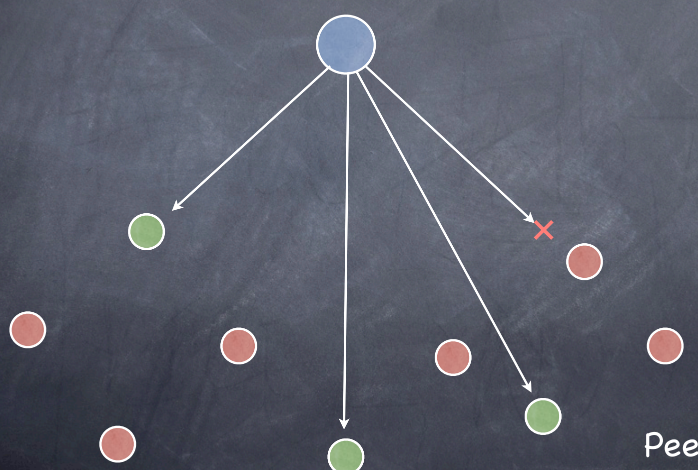
- Examples: Internet radio, NCAA tournament, web concerts, Internet TV

SopCast P2P-Radio livestation Be there now

- Practical challenges:
  - Deliver updates by deadline
  - Minimize jitter
  - Be mindful of bandwidth requirements
  - Tolerate churn
  - Handle Byzantine and rational peers

# P2P Streaming Setup

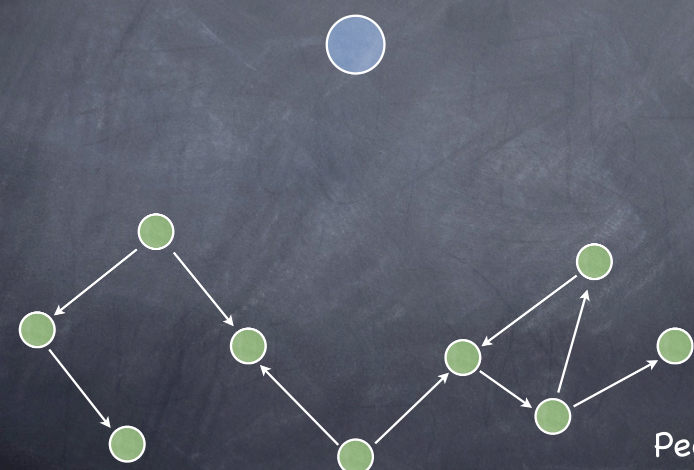
Broadcaster



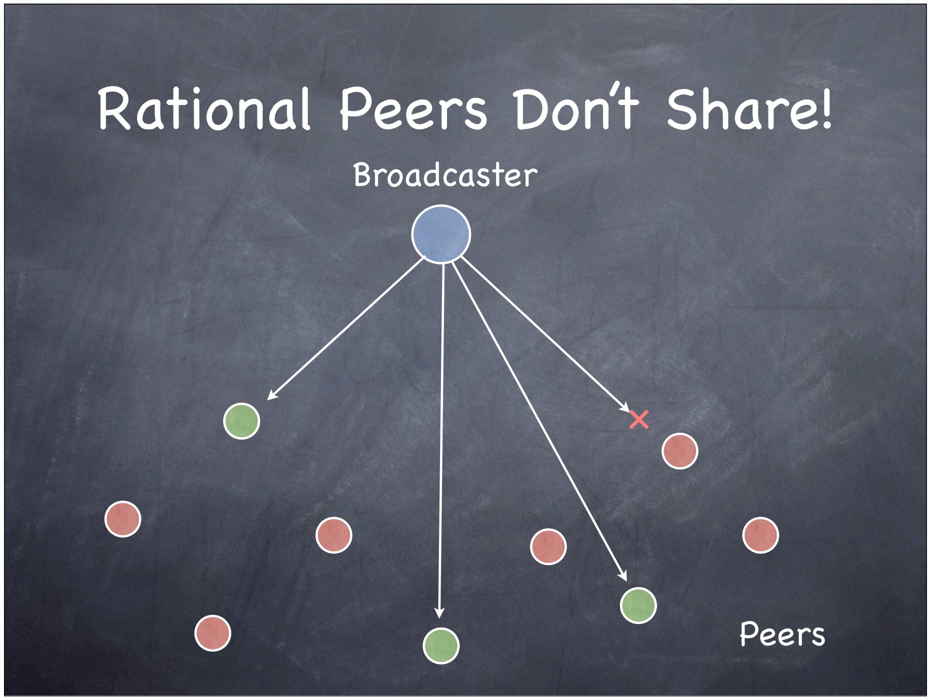
Peers

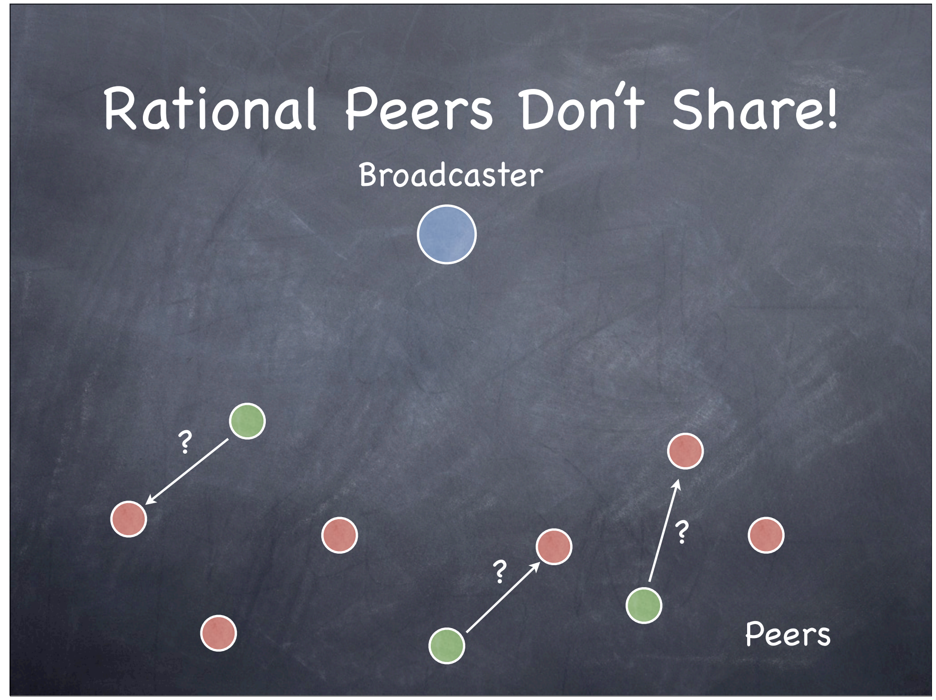
# P2P Streaming Setup

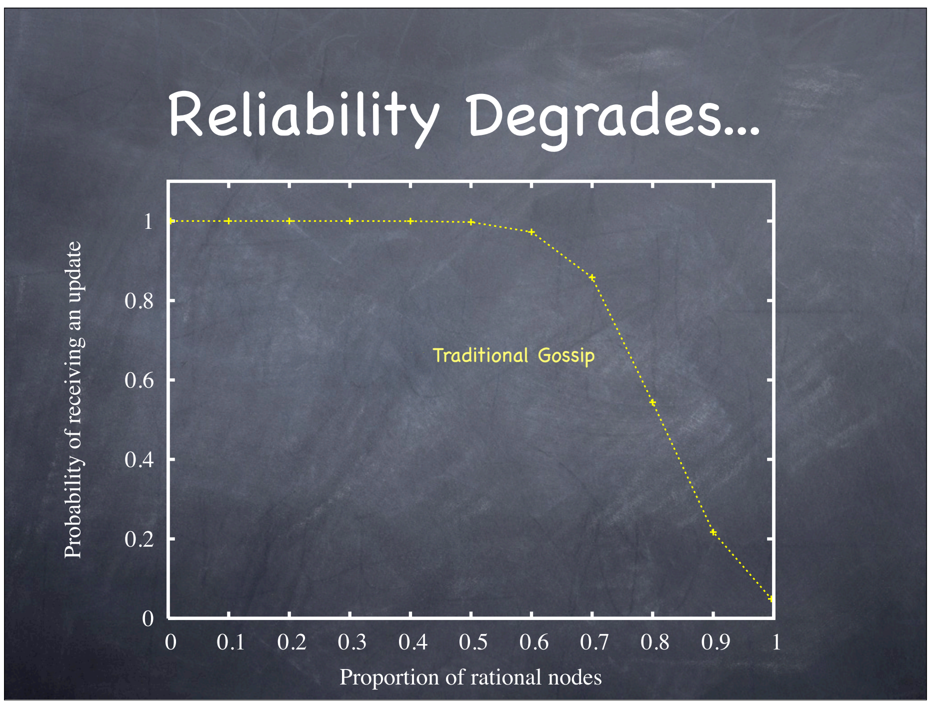
Broadcaster

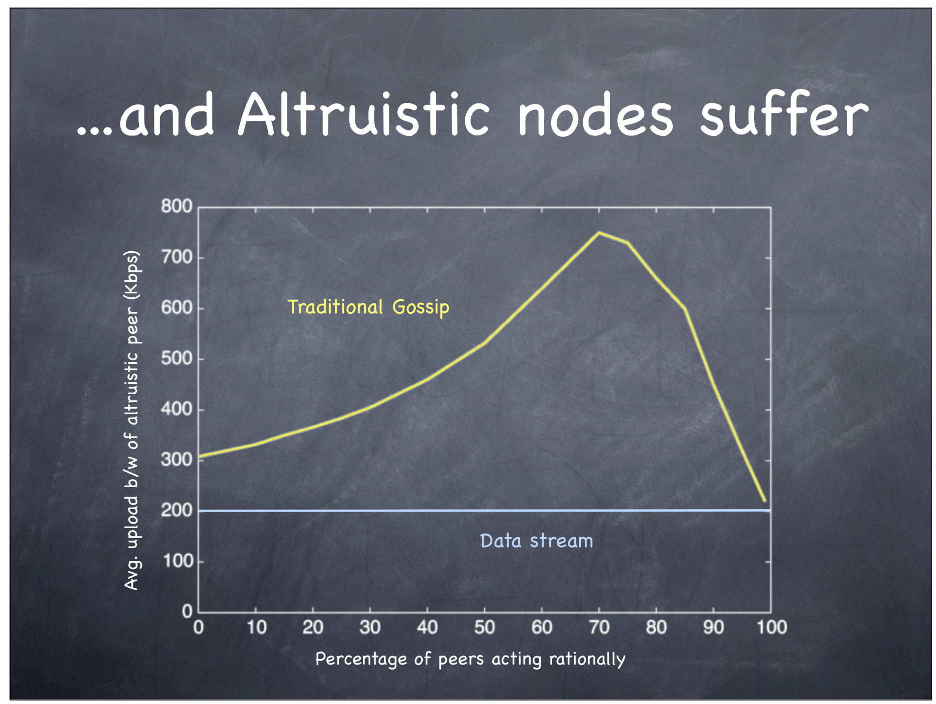


Peers


Rational Peers Don't Share!
Broadcaster
Peers


Rational Peers Don't Share!
Broadcaster
?
?
?
Peers


Reliability Degrades...
Probability of receiving an update
Proportion of rational nodes
Traditional Gossip
0
0.2
0.4
0.6
0.8
1
0.1
0.3
0.5
0.7
0.9


...and Altruistic nodes suffer
Avg. upload b/w of altruistic peer (Kbps)
Percentage of peers acting rationally
Traditional Gossip
Data stream
0
100
200
300
400
500
600
700
800
10
20
30
40
50
60
70
80
90

# BAR Gossip

- First BAR tolerant gossip protocol
- Design game for Nash equilibrium
  - No peer gains from unilateral deviation
  - Benefit: Delivering stream packets
  - Cost: Bandwidth
- Key protocol: Balanced Exchange

# Design Principles

- Restrict choice
  - Eliminate non-determinism
  - Evict provably deviant peers
- Balance costs
  - $Cost_{divergence} = Cost_{obedience}$
- Delay gratification
  - Postpone payoff

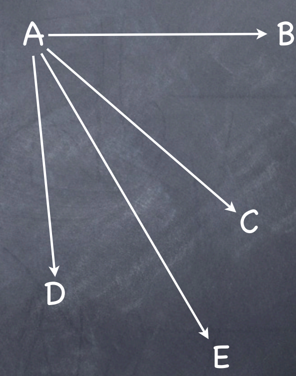
# Balanced Exchange is a Nash Equilibrium

Theorem: A balanced exchange is incentive compatible for strategies that maximize the number of useful updates received in that exchange

- Partner selection
- History exchange
- Update exchange

# (1) Partner Selection

Problem: Gossip relies on randomness
- Rational node may
  - Choose nearby partner
  - Choose well-connected partner
  - Choose multiple partners
  - ...



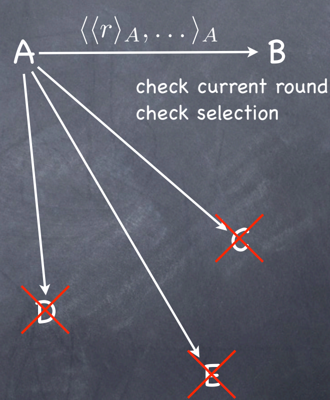

Q: How do we limit a peer to one uniformly selected partner per round?

# (1) Partner Selection

A: Restrict Choice
Verifiable pseudo-random partner selection

A's PRNG seed in round: $r : \langle r \rangle_A$

- Retains strength of randomness:
  - ✓ Uniform selection of partners
  - ✓ Unpredictability
- Supports Nash equilibrium
  - ✓ Unilateral deviation not useful

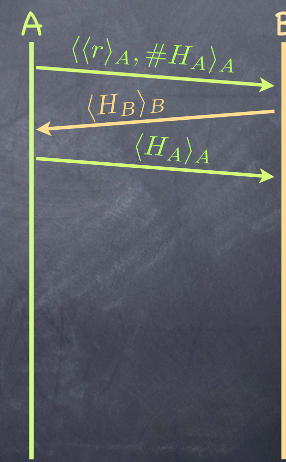



# (2) History Exchange

Negotiate update exchange
- Deterministic function of histories

Problem: Strategic client might
- Under-report
- Over-report
- Etc.

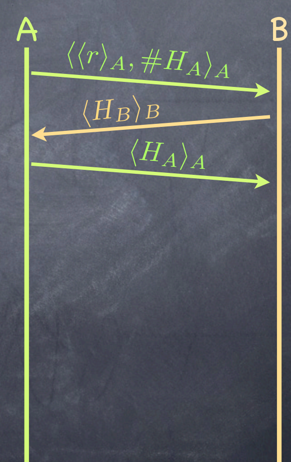
Q: How do we handle a client lying about its history?




# (2) History Exchange

A: Restrict Choice
Client commits to a history before discovering partner's history

- Correct reporting maximizes useful exchange
- Under-reporting decreases number of useful updates exchanged
- Over-reporting risks eviction

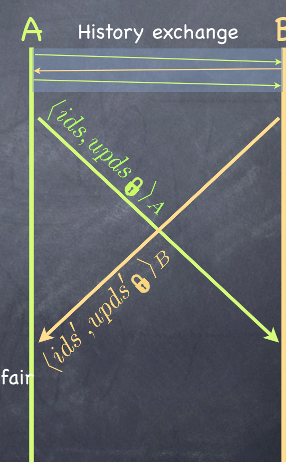



# (3) Briefcase Exchange

Q: How do we encourage a rational client to send a briefcase?

A1: Fair exchange is impossible*

A2: Fair enough exchange




* Without a trusted third party
B. Garbinato and I. Rickebusch. Impossibility results on fair exchange. Tech. Rep. DOP-20051122, Université de Lausanne, Distributed Object Programming Lab.

# (3) Briefcase Exchange

Q: How do we encourage a rational client to send a briefcase?

A: Defer gratification
Client gives key only after swapping briefcases


# Valid Briefcase Exchange

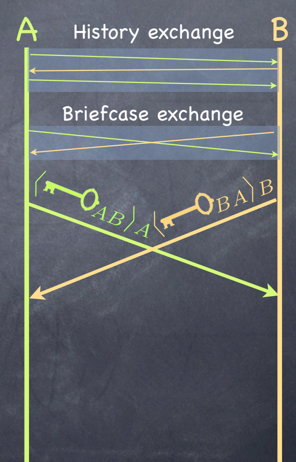
Q: How do we encourage a rational client to send only appropriate briefcases?

A: Restrict choice
Hold client accountable for contents

- Briefcase contains ids of updates and encrypted updates
- Inconsistencies risk eviction
- Decryption key is reproducible by broadcaster


# Key Exchange

Q: How do we encourage a rational client to send the appropriate key?




# Key Exchange

Q: How do we encourage a rational client to send the appropriate key?

A: Balance costs
Repeated Key Requests

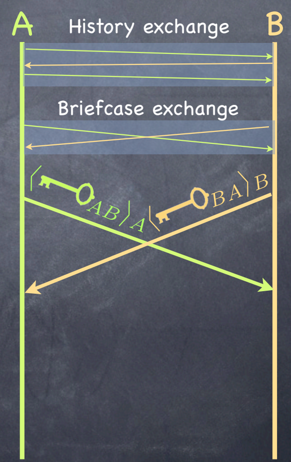
- Rational client minimizes cost by sending key

# Key Exchange

Q: How do we encourage a rational client to send the appropriate key?

A: Balance costs
Repeated Key Requests

- Rational client minimizes cost by sending key
- Rational client proactively sends key



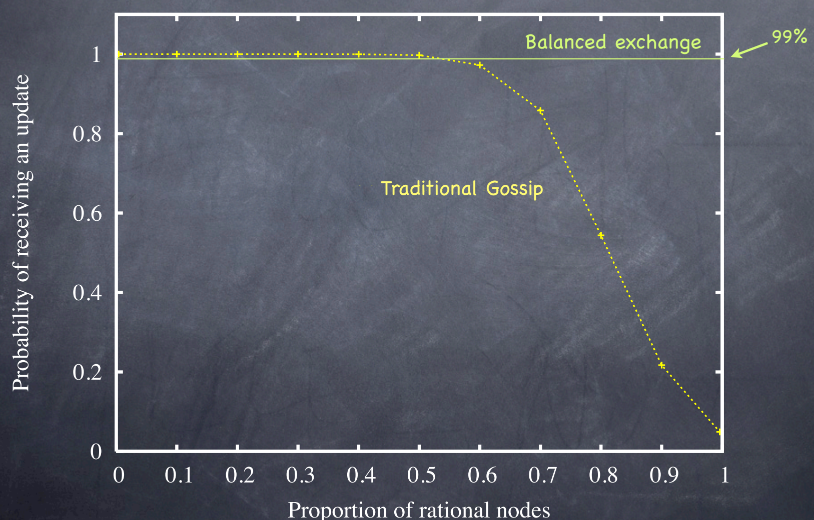

# Balanced Exchange

In each round

- Select a partner
- Exchange histories
- Trade equal number of updates
  - Exchange briefcases
  - Exchange keys
    - pester to nudge unresponsive Rational nodes

fair enough exchange

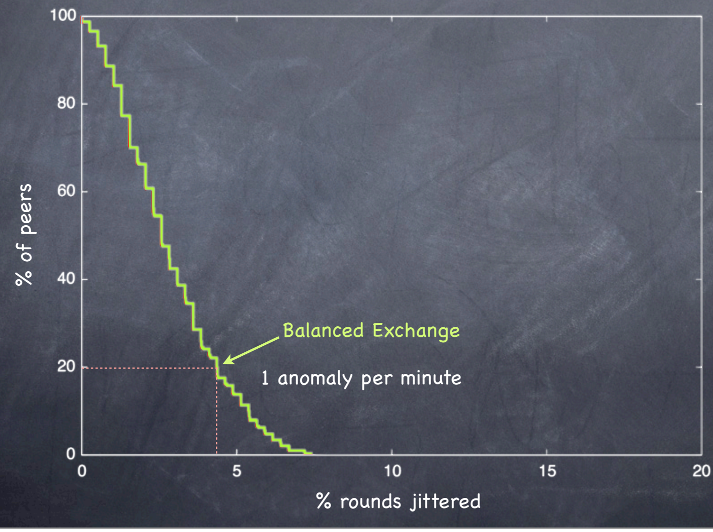
# Reliability is way up!




# "Obedience sums up our entire duty" Hosea Ballou

| | | |
|---|---|---|
| Restrict choice | Deterministic partner<br>Deterministic items | Static membership<br>Inflexible communication patterns |
| Balance costs | Fixed message sizes<br>Even exchange | Extra overheads<br>Garbage messages |
| Defer gratification | Chain steps of protocol | |
| Prove Equilibrium | "Balanced exchange is Nash" | Micro-manage protocol<br>Ignore cross-round strategies |

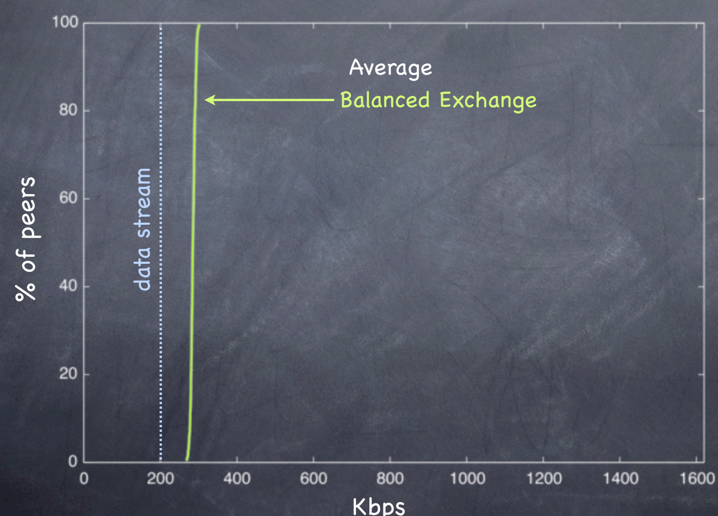
# Jitterbug



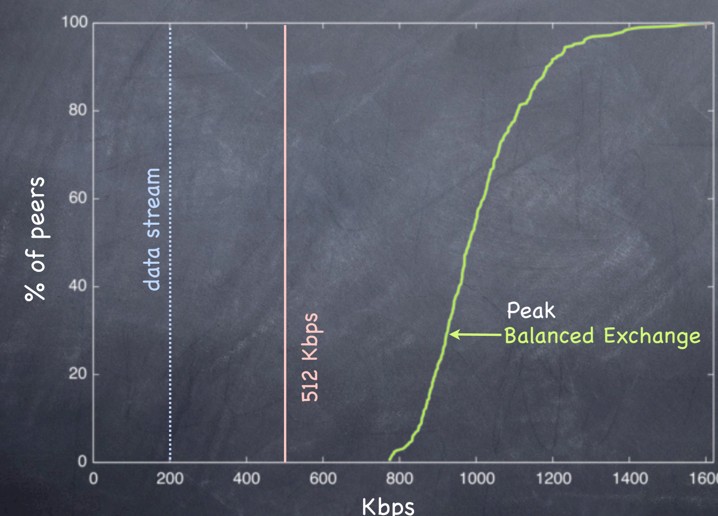

# "Obedience sums up our entire duty" Hosea Ballou

| | | |
|---|---|---|
| Restrict choice | Deterministic partner<br>Deterministic items | Static membership<br>Inflexible communication patterns |
| Balance costs | Fixed message sizes<br>Even exchange | Extra overheads<br>Garbage messages |
| Defer gratification | Chain steps of protocol | |
| Prove Equilibrium | "Balanced exchange is Nash" | Micro-manage protocol<br>Ignore cross-round strategies |

# Average Bandwidth




# Peak Bandwidth



# "Obedience sums up our entire duty" Hosea Ballou

**Restrict choice**
- Deterministic partner
- Deterministic items
- Static membership
- Inflexible communication patterns

**Balance costs**
- Fixed message sizes
- Even exchange
- Extra overheads
- Garbage messages

**Defer gratification**
- Chain steps of protocol

**Prove Equilibrium**
- "Balanced exchange is Nash"
- Micro-manage protocol
- Ignore cross-round strategies

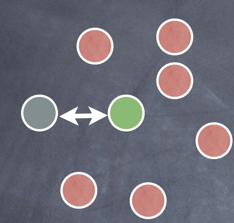
# BAR Research Agenda

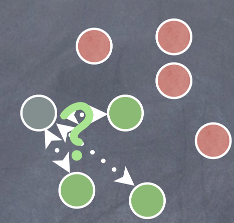
1. New model
   Develop a model in which it is possible to prove properties about MAD services
2. Is the model usable?
   Understand how to simplify the development of MAD services in the new model
3. Is the model practical?
   Demonstrate that MAD services developed under the new model can be efficient, effective

# Flightpath

- Obedience vs Choice
  - Nash --> $\varepsilon-$Nash
    - Deviate only if doing so increases utility by more than $\varepsilon$
  - Similar to BAR Gossip
    - Partner selection --> choose "good" partner
    - History exchange --> prioritize important updates
    - Briefcase exchange --> allow limited imbalance

# The Power of Choice

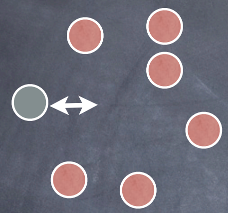


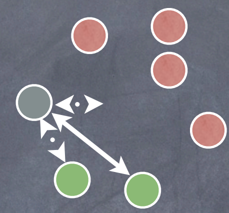


**BAR Gossip**
- One pre-determined partner for each round

**Flightpath**
- One of O(logN) buckets per round
- Choose a partner from bucket
  - Flightpath specifies heuristics

# The Power of Choice





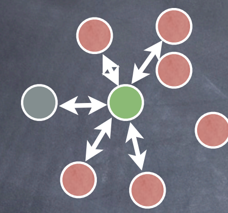
BAR Gossip

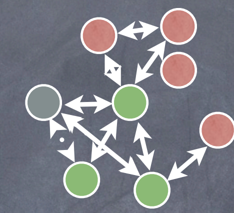
- Node failure -->
  Miss round

Flightpath

- Node failure -->
  Pick another node

# The Power of Choice





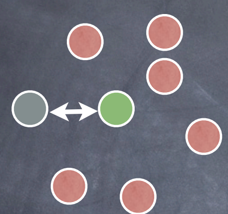
BAR Gossip

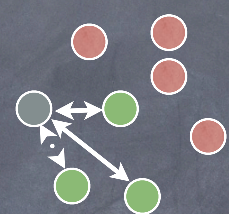
- Node overload -->
  Exceed max BW

Flightpath

- Node overload -->
  Pick another node

# The Power of Choice





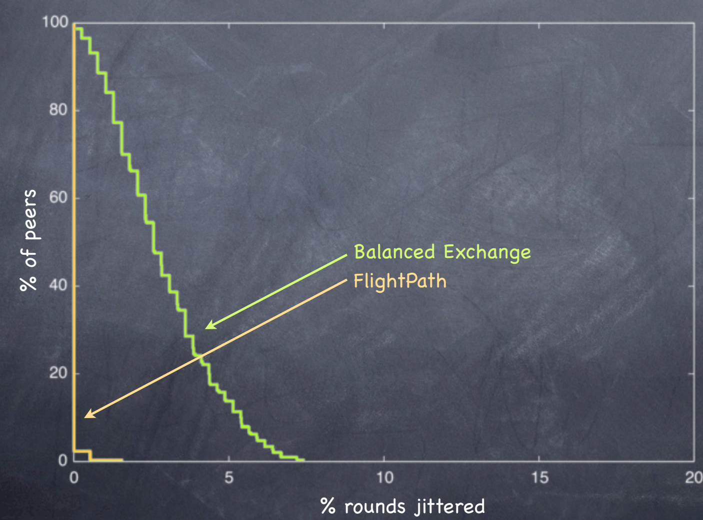
BAR Gossip

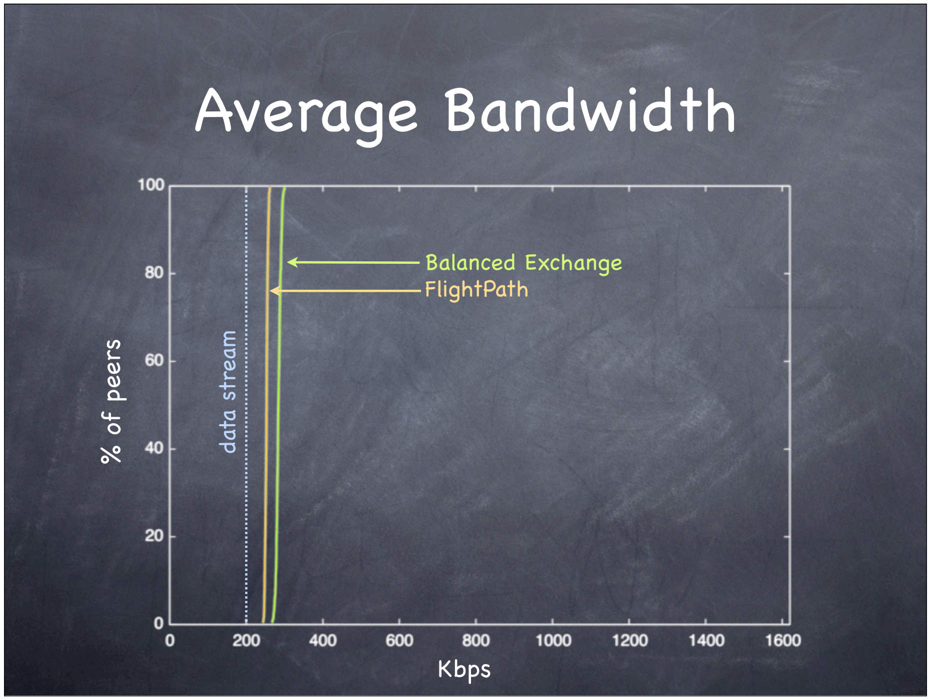
- Fall behind -->
  Too bad

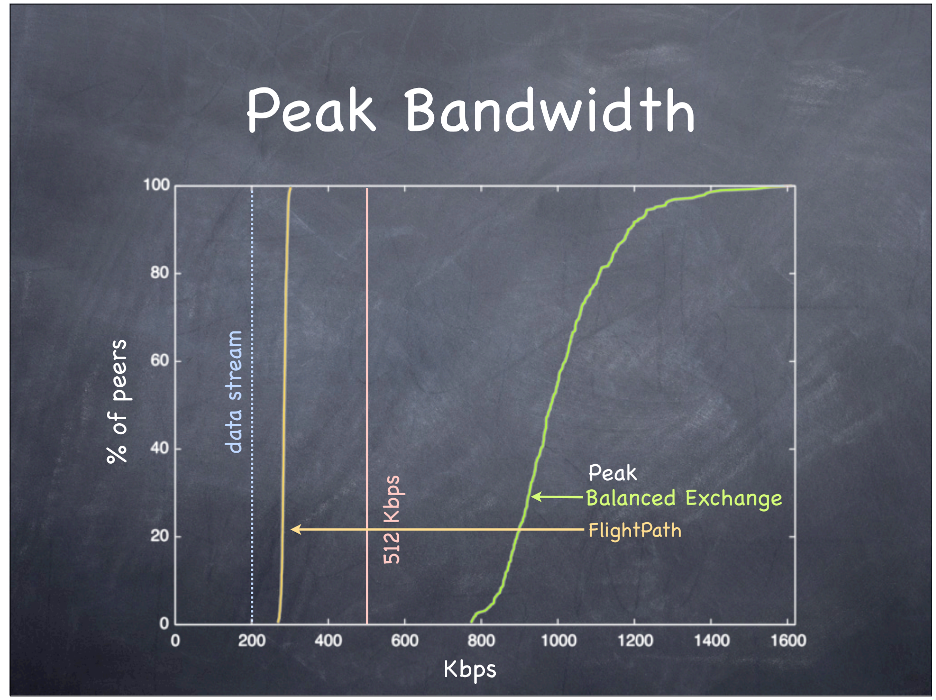
Flightpath

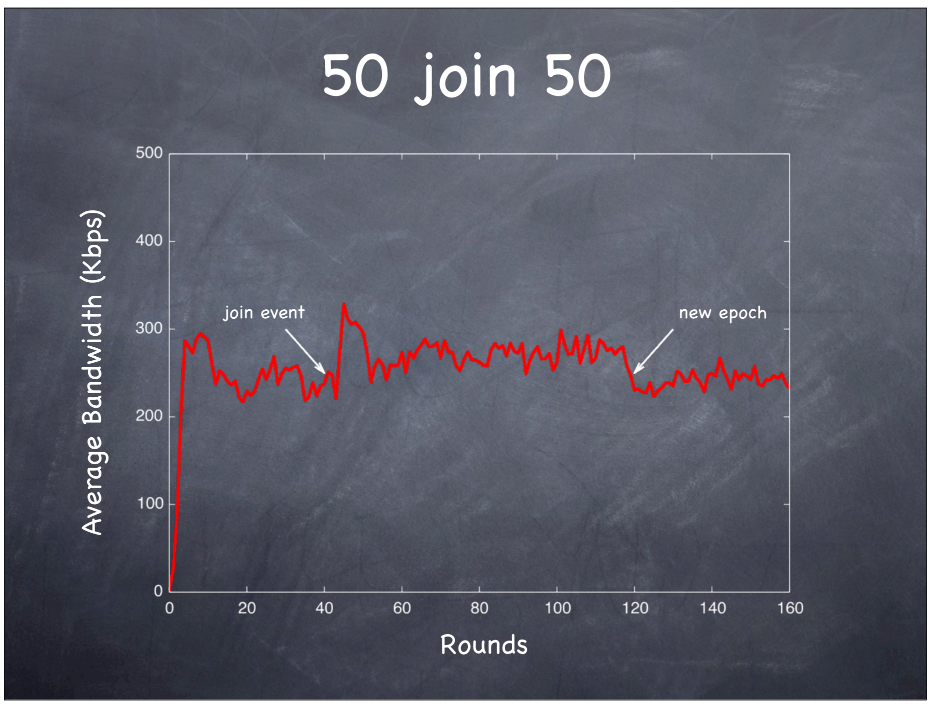
- Fall behind -->
  Initiate extra exchanges

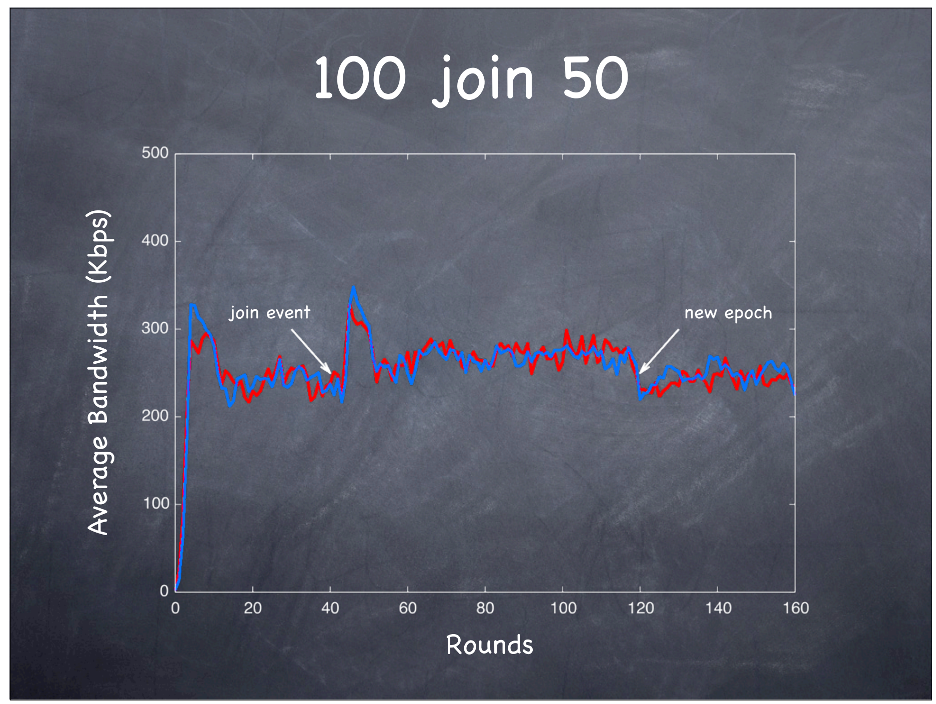
# Jitterdämmerung



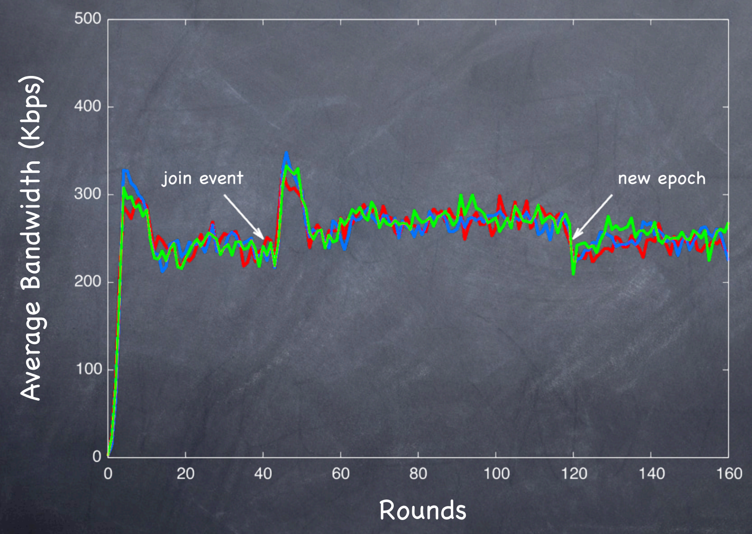

Average Bandwidth
% of peers
data stream
Balanced Exchange
FlightPath
Kbps
0
20
40
60
80
100
0
200
400
600
800
1000
1200
1400
1600


Peak Bandwidth
% of peers
data stream
512 Kbps
Peak
Balanced Exchange
FlightPath
Kbps
0
20
40
60
80
100
0
200
400
600
800
1000
1200
1400
1600

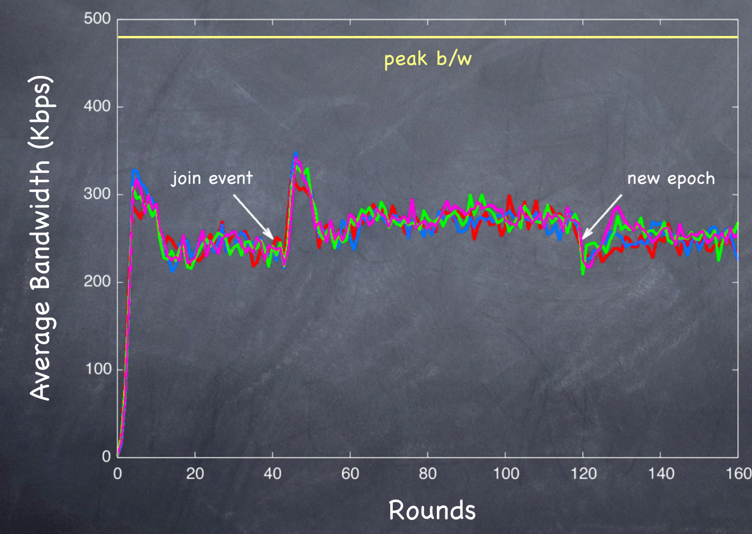

50 join 50
Average Bandwidth (Kbps)
join event
new epoch
Rounds
0
100
200
300
400
500
0
20
40
60
80
100
120
140
160


100 join 50
Average Bandwidth (Kbps)
join event
new epoch
Rounds
0
100
200
300
400
500
0
20
40
60
80
100
120
140
160

# 200 join 50




# 400 join 50




# Approximate Equilibrium

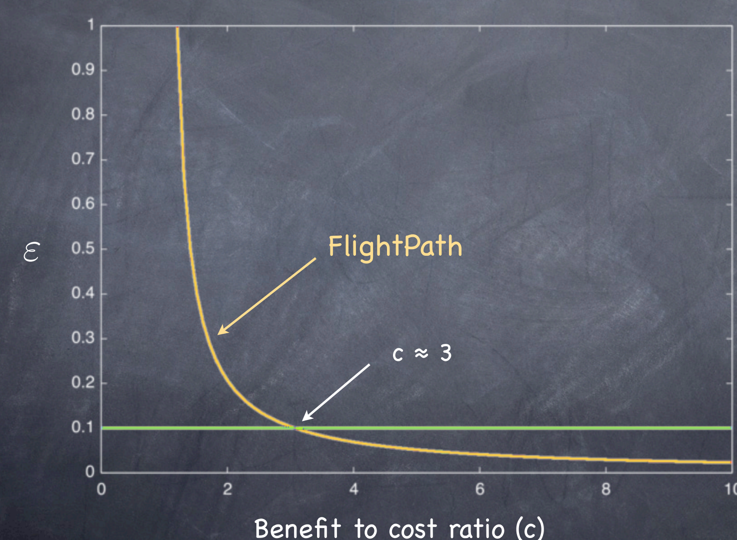
## Nash

- Peer cannot gain by deviating
  - Extra trades, ignore newly joined, nodes, etc...
  - Bet: a Flightpath peer can gain by deviating

## $\varepsilon -$ Nash

- Peer cannot gain much by deviating
  - Increase benefit (i.e decrease jitter)
    - Flightpath already has minimal jitter
  - Reduce cost (i.e. reduce bandwidth)
    - Rational peer must pay at least $\left[\frac{1}{1+\alpha}\right] x$ for $x$ updates

# FlightPath's $\varepsilon$



# Limitations, Open Questions

## Theory

- Refine solution concepts
  - E.g., Weaken assumption that rational nodes are "Risk averse"
- Altruism in BAR systems [DISC '10]
  - Altruism encourages free riders
  - Free riders encourage more free riders
- Simplify proofs, protocol design
  - Rules of thumb -- balance cost, restrict choice, ...
  - $\varepsilon$–Nash helps a lot!

## Practice

- Additional applications
- Cloud vs p2p

# Conclusions

- Modern distributed systems are MAD
  - Fault-tolerance and game theory must come together to bring about some sanity
- BAR is a compelling model for reasoning about robust MAD systems
  - Solid theoretical foundation
  - Supports practical implementations of diverse systems
- Many open challenges
  - If you build it, they will come...

# Keeping the A in BAR

# The amazingly shrinking...

BAR

Current BAR protocols
neither depend on nor leverage
altruistic nodes

# Good will considered harmful

- Unselfishness encourages free riders
- Free riders encourage more free riders
  - "Be a hero" does not scale...

# File sharing on Gnutella (2000)

| The top ____ hosts | Share ____ of all files |
|---|---|
| 1% | 37% |
| 5% | 70% |
| 10% | 87% |
| 15% | 94% |
| 20% | 98% |
| 25% | 99% |
| 34% | 100% |

66% of users are selfish free-riders

but

34% of users are giving away their content freely!

From E. Adar and B. Huberman, "Free Riding on Gnutella", First Monday, 2000.

# File sharing on Gnutella (2005)

| The top ____ hosts | Share ____ of all files |
|---|---|
| | |
| | |
| | |
| | |
| | |
| | |
| 15% | 100% |

66% of users are selfish free-riders

but

34% of users are giving away their content freely!

In 2005, free-riders had grown to 85%!

From E. Adar and B. Huberman, "Free Riding on Gnutella", First Monday, 2000.

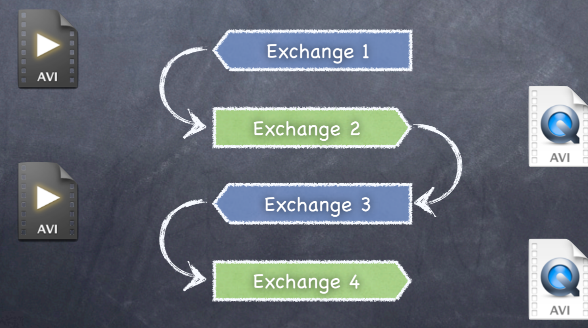
# What is the role of altruism?

Altruism is both necessary and sufficient to trigger rational cooperation

# Cooperation in P2P services

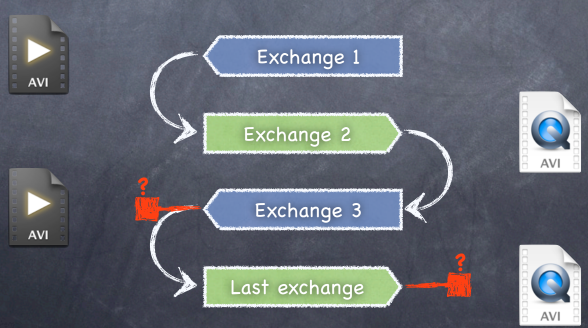

# Why contribute at all?

Because doing so provides future benefits (or lower costs).



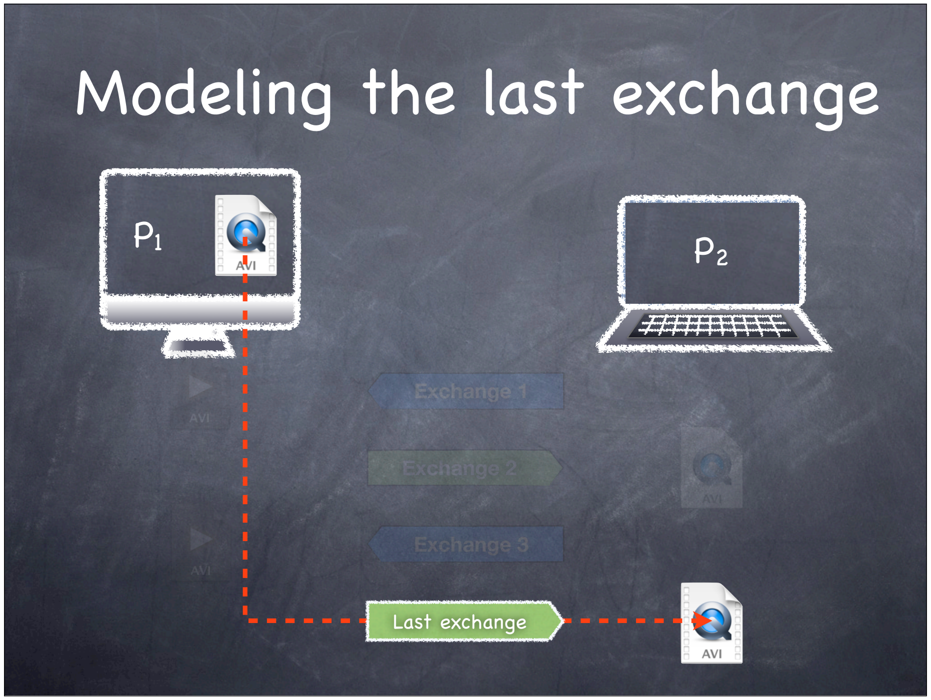

# ...but how about the last exchange?

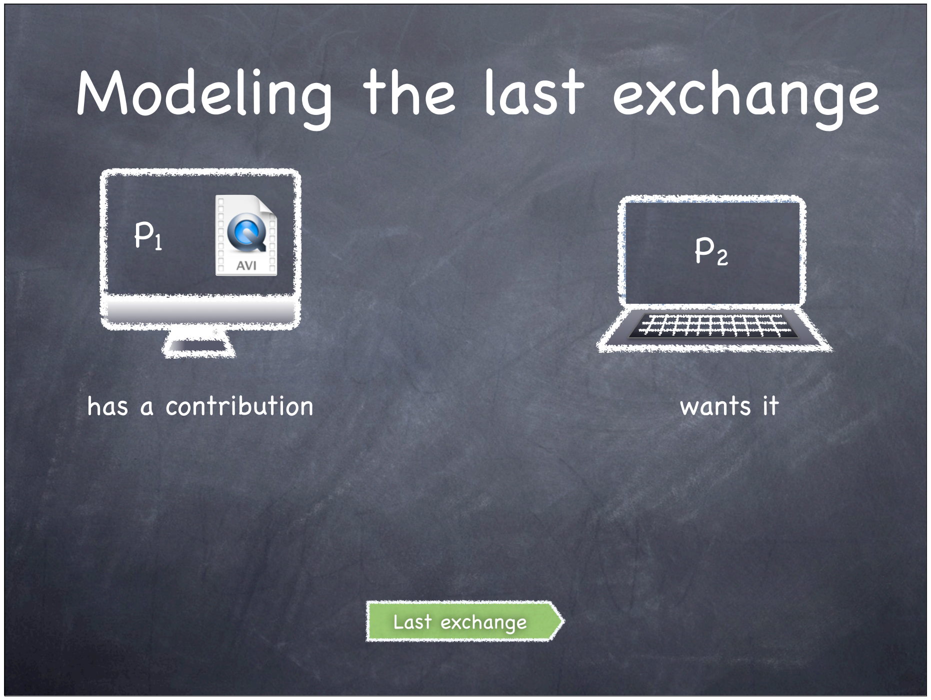
What incentive is there for a participant to contribute?

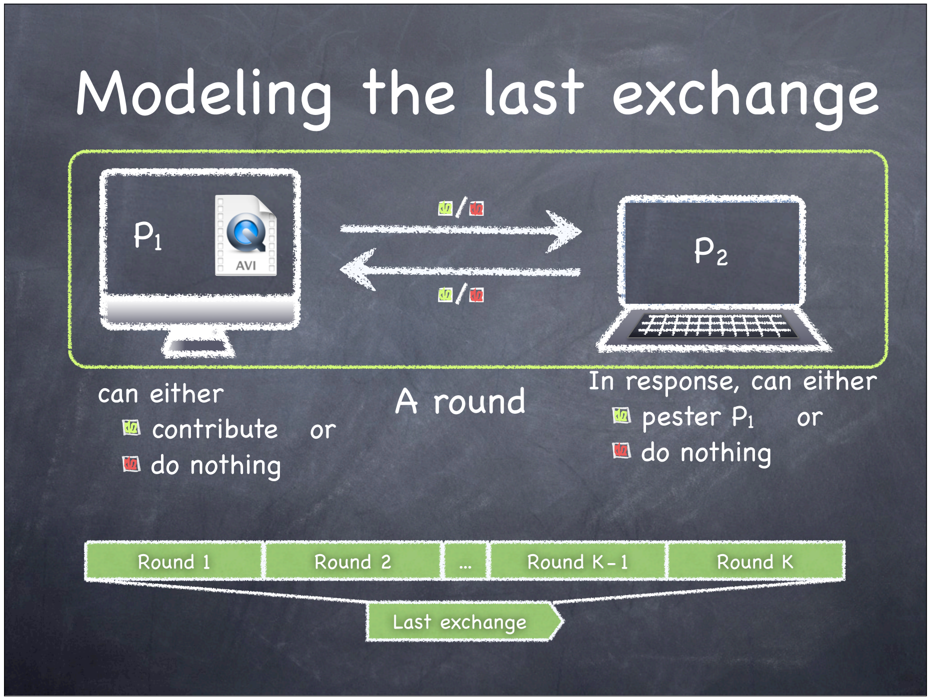


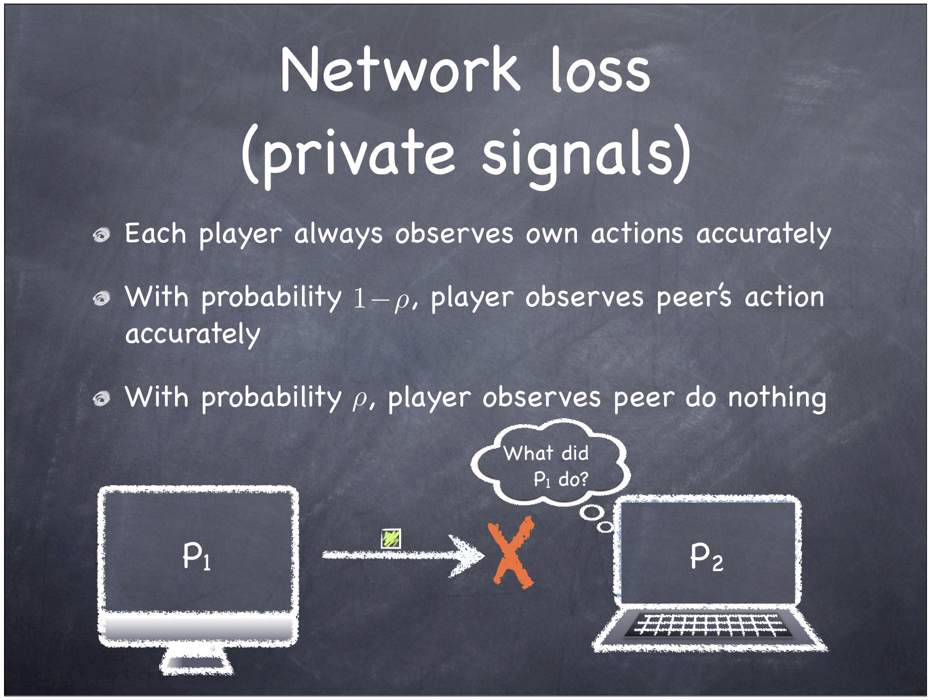

# How to induce rational cooperation

- Ignorance
  - infinite horizon
- Apathy
  - no deviation unless significant gain
- Threat
  - pester

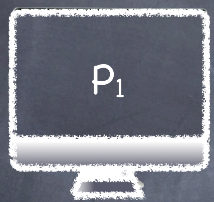

Modeling the last exchange
P1
AVI
P2
Exchange 1
Exchange 2
Exchange 3
Last exchange
AVI


Modeling the last exchange
P1
AVI
P2
has a contribution
wants it
Last exchange


Modeling the last exchange
P1
AVI
P2
can either
contribute or
do nothing
A round
In response, can either
pester P1 or
do nothing
Round 1
Round 2
...
Round K-1
Round K
Last exchange

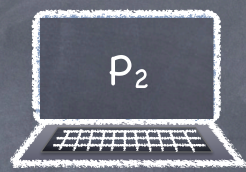

Network loss
(private signals)
Each player always observes own actions accurately
With probability $1-\rho$, player observes peer's action accurately
With probability $\rho$, player observes peer do nothing
What did P1 do?
P1
P2

# Utilities (for rational $P_1$ and $P_2$)




Contributing has positive cost.
Being pestered has positive cost.

- Minimize contributing.
- Minimize receiving pester.




Pestering has positive cost.
Receiving contribution has positive cost and one-time positive benefit.

- Minimize pestering.
- Minimize receiving contribution (but wants it at least once).

Benefit of contribution >> cost of pestering + cost of recv. contribution

# Building the Equilibrium

- $P_i$ starts with an initial belief that $P_{-i}$ is of a given type (B, A, R)
- $P_i$ updates its beliefs using Bayes rule depending on what it observes

| B | A | R |
|---|---|---|
| Do nothing with probability independent of peer's actions | Follow the protocol | Follow if no benefit from deviating |

- Rational players don't expect to be able to affect the strategy of a Byzantine player

# Altruism is necessary for rational cooperation

Theorem 1. In a bounded game, there exists no equilibrium in which a rational $P_1$ contributes or a rational $P_2$ pesters

# Altruism is necessary for rational cooperation

Theorem 1. In a bounded game, there exists no equilibrium in which a rational $P_1$ contributes or a rational $P_2$ pesters

Proof (sketch)

- Some last round $t_c$ and $t_p$
- $P_2$ never pesters after $P_1$ stops contributing: $t_c \geq t_p$
- $P_1$ never contributes after $P_2$ stops pestering: $t_p > t_c$ (at most, $t_p = t_c + 1$)
- $P_1$ never contributes and $P_2$ never pesters

# Altruism is necessary for rational cooperation

**Theorem 1.** In a bounded game, there exists no equilibrium in which a rational $P_1$ contributes or a rational $P_2$ pesters

**Theorem 2.** In an unbounded game in which a rational player believes that there exists some fraction of Byzantine peers that

- never contributes when playing as $P_1$ or
- always pesters when playing as $P_2$

there exists no pure equilibrium in which a rational $P_1$ contributes or a rational $P_2$ pesters

# Altruism to the rescue

Altruistic $P_1$ contributes:
- ❑ Always in the first round.
- ❑ With fixed probability in subsequent rounds.

↓

Rational $P_2$ pesters if:
- ❑ Prob. of altruistic $P_1$ contributing is sufficiently high
- ❑ $P_2$'s belief that $P_1$ is altruistic is sufficiently high.

We derive conditions under which these provisions hold in every round — except the last.

Pestering becomes a credible threat!

# Altruism to the rescue

Altruistic $P_1$ contribute:
- ❑ Always in the first round.
- ❑ With fixed probability in subsequent rounds.

↓ (under certain conditions)

Rational $P_2$ pester if:
- ❑ No contribution has been received; and
- ❑ The last round has not been reached.

Altruistic $P_2$ pester if:
- ❑ No contribution has been received; and
- ❑ The last round has not been reached.

# The cooperative equilibrium

| | |
|---|---|
| Byzantine $P_1$ contributes arbitrarily | Byzantine $P_2$ pesters arbitrarily. |
| Altruistic $P_1$ contributes:<br>❑ Always in the first round.<br>❑ With fixed probability in subsequent rounds. | Altruistic $P_2$ pesters if:<br>❑ Received no contribution<br>❑ Not reached last round |
| Rational $P_1$ contributes if:<br>❑ First round or pestered in previous round<br>❑ Not at end of game<br>❑ Sufficient belief $P_2$ not Byzantine. | Rational $P_2$ pesters if:<br>❑ No contribution has been received;<br>❑ Last round has not been reached. |

# What does this all mean?

# Cooperation is achieved under realistic conditions

- For example, rational peers cooperate in a system where:
  - the network drops 5% of the packets
- and they believe that
  - 50% of the peers are Byzantine
  - fewer than 10% are altruistic

# Does $P_1$ contribute?

| Expected probability of Byzantine $P_2$ pestering | # of times contribution is sent if $P_1$ observes pester every round | |
|---|---|---|
| | Net loss = 0.05 | Net loss = 0.25 |
| 0.1 | 14 | 17 |
| 0.5 | 5 | 9 |
| 1.0 | 2 | 3 |

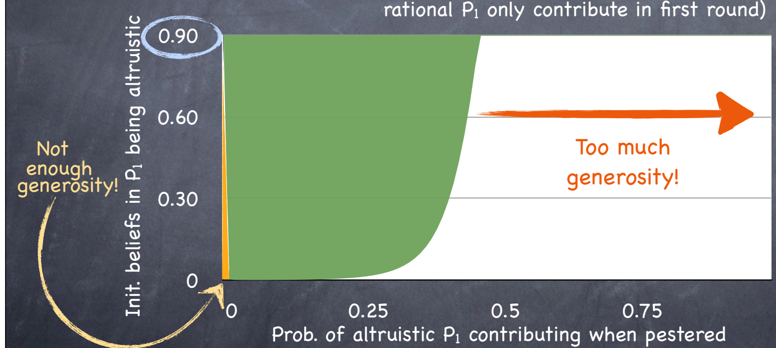
The higher the expected probability, the more likely $P_1$ is going to believe $P_2$ is Byzantine if pestered

The higher the network loss (for reasonable values), the more forgiving $P_1$ is.

$P_1$'s cost of sending contribution (relative to cost of recv. pester) = 2
21 rounds
$P_1$'s initial belief that $P_2$ is Byzantine = 0.1

# When can we guarantee that $P_2$ will pester?

(assuming Byzantine $P_1$ never contribute & rational $P_1$ only contribute in first round)




$P_2$'s cost of sending pester = 1
$P_2$'s initial belief that $P_1$ is Byzantine = 0.1
Benefit of receiving contribution = $10^4$

21 rounds
Network loss = 0.05

# Conclusions

- Modern distributed systems are MAD
  - Fault-tolerance and game theory must come together to bring about some sanity
- BAR is a compelling model for reasoning about robust MAD systems
  - Solid theoretical foundation
  - Supports practical implementations of diverse systems
- Many open challenges
  - Collusion
  - More general treatment of Byzantine nodes
  - Altruism
- If you build it, they will come...

# The amazingly shrinking...

BAR

Current BAR protocols
neither depend on nor leverage
altruistic nodes

# Good will considered harmful

- Unselfishness encourages free riders
- Free riders encourage more free riders
  - "Be a hero" does not scale...

# The Last Exchange

- Two peers: $\mathcal{P}_1$ and $\mathcal{P}_2$
- $\mathcal{P}_1$ has information of value to $\mathcal{P}_2$
- $\mathcal{P}_1$ expects no future benefit from contributing
  - neither expects to interact with the other beyond this exchange
- How can we induce rational cooperation?

## How to induce rational cooperation

- Ignorance
  - infinite horizon
- Apathy
  - no deviation unless significant gain
- Threat
  - pester

## How to induce rational cooperation

- Ignorance
  - infinite horizon
- Apathy
  - no deviation unless significant gain
- Threat
  - pester
  - credible?

## Modeling the Last Exchange

- $\mathcal{P}_1$ and $\mathcal{P}_2$ in a repeated sequential game
- In each round
  - $\mathcal{P}_1$ either contributes $(c)$ or does nothing $(n)$
  - $\mathcal{P}_2$ either pesters $(p)$ or does nothing $(n)$
- No free lunch: sending and receiving costs
- Network loss modeled through private signals
  - what $\mathcal{P}_i$ observes may not be what $\mathcal{P}_{-i}$ played!

## Building the Equilibrium

- $\mathcal{P}_i$ assign a belief to $\mathcal{P}_{-i}$ being of type B, A, R
- $\mathcal{P}_i$'s beliefs depend on what $\mathcal{P}_i$ has observed
- Beliefs evolve using Bayes rule
- Rational players don't expect to be able to affect the strategy of a Byzantine player

# Altruism is necessary for rational cooperation

Theorem 1. In a bounded game, there exists no equilibrium in which a rational $\mathcal{P}_1$ contributes or a rational $\mathcal{P}_2$ pesters

# Altruism is necessary for rational cooperation

Theorem 1. In a bounded game, there exists no equilibrium in which a rational $\mathcal{P}_1$ contributes or a rational $\mathcal{P}_2$ pesters

Theorem 2. In an unbounded game in which a rational player believes that there exists some fraction of Byzantine peers that

- never contributes when playing as $\mathcal{P}_1$ or
- always pesters when playing as $\mathcal{P}_2$

there exists no pure equilibrium in which a rational $\mathcal{P}_1$ contributes or a rational $\mathcal{P}_2$ pesters

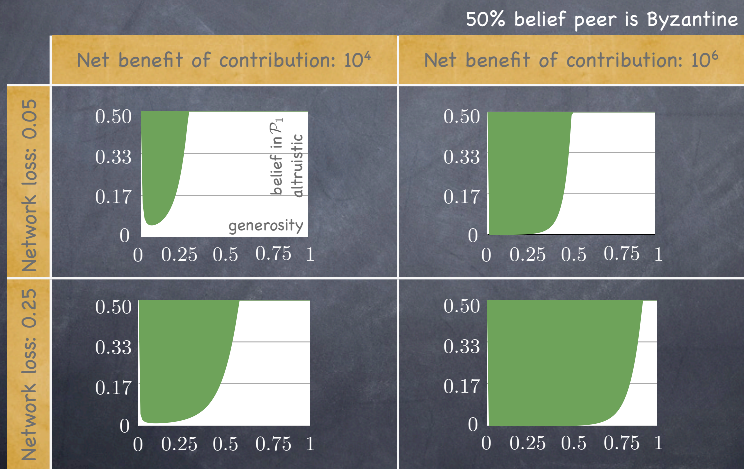
# Altruism to the rescue

- Pestering becomes a credible threat
  - We derive a sufficient condition under which $\mathcal{P}_2$ prefers to pester
- The threat of pestering can lead $\mathcal{P}_1$ to contribute
  - For every round far enough from the end of the game, there exists a belief threshold beyond which contributing yields $\mathcal{P}_1$ a higher expected utility

# Cooperation is achieved under realistic conditions

- For example, rational peers cooperate in a system where:
  - the network drops 5% of the packets
- and they believe that
  - 50% of the peers are Byzantine
  - fewer than 10% are altruistic

# The Dangers of Excessive Generosity




# Conclusions

- Modern distributed systems are MAD
  - fault-tolerance and game theory must come together to bring about some sanity
- BAR is a compelling model for reasoning about robust MAD systems
  - it has a solid theoretical foundation
  - it supports practical implementations of very diverse systems
- Many, many open challenges
  - if you build it, they will come...