Efficient Reconciliation and Flow Control for Anti-Entropy Protocols

Robbert van Renesse
Dan Dumitriu
Valient Gough
Chris Thomas

Work done at Amazon.com
(2006)

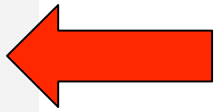
amazon web services SERVICE HEALTH DASHBOARD

Amazon S3 Availability Event: July 20, 2008

We wanted to provide some additional detail about the problem we experienced on Sunday, July 20th.

At 8:40am PDT, error rates in all Amazon S3 datacenters began to quickly climb and our alarms went off. By 8:50am PDT, error rates were significantly elevated and very few requests were completing successfully. By 8:55am PDT, we had multiple engineers engaged and investigating the issue. Our alarms pointed at problems processing customer requests in multiple places within the system and across multiple data centers. While we began investigating several possible causes, we tried to restore system health by taking several actions to reduce system load. We reduced system load in several stages, but it had no impact on restoring system health.

At 9:41am PDT, we determined that servers within Amazon S3 were having problems communicating with each other. As background information, Amazon S3 uses a gossip protocol to quickly spread server state information throughout the system. This allows Amazon S3 to quickly route around failed or unreachable servers, among other things. When one server connects to another as part of processing a customer's request, it starts by gossiping about the system state. Only after gossip is completed will the server send along the information related to the customer request. On Sunday, we saw a large number of servers that were spending almost all of their time gossiping and a disproportionate amount of servers that had failed while gossiping. With a large number of servers gossiping and failing while gossiping, Amazon S3 wasn't able to successfully process many customer requests.



S3 Outage Highlights Fragility of Web Services

Om Malik, Sunday, July 20, 2008 at 7:46 PM PT

Comments (59)

Updated with Statement from Amazon: Amazon's S3 cloud storage service **went offline** this morning for an extended period of time — the second big outage at the service this year. **In February**, Amazon suffered a major outage that knocked many of its customers offline.

It was no different this time around. I first learned about today's outage when avatars and photos (stored on S3) **used by Twinkle**, a Twitter-client for iPhone, vanished.

My big hope was that it would come back soon, but popular S3 clients such as SmugMug were offline for more than eight hours — an awfully long time for Amazon's Web Services division to bring back the

THE WALL STREET JOURNAL.

February 15, 2008, 6:17 pm

Is Amazon's Small Crash a Giant Crash for Cloud Computing?

Posted by Ben Worthen

Today was a bad day for a new computing model that could one day be the norm. Amazon's S3 service – which companies can use to rent data storage on Amazon's tech gear — crashed this morning, knocking many small businesses offline and highlighting one of the model's drawbacks: You're putting your operations in somebody else's hands.

InformationWeek
DEFINING THE BUSINESS VALUE OF TECHNOLOGY

Amazon S3 Crash Raises Doubts Among Cloud Customers

The company's Simple Storage Service suffered a similar outage lasting about two hours in February, an incident that at the time led many to question its dependability.

By Thomas Claburn
InformationWeek

July 21, 2008 03:30 PM

Amazon (NSDQ: **AMZN**) Web Service's Simple Storage Service (S3) suffered a service failure for about eight hours on Sunday, causing outages at online companies that depend on S3 for file storage.

S3 suffered a similar outage lasting about two hours in February, an incident that led many to question the dependability of the increasingly fashionable cloud computing model.

Gossip at Amazon

- Ubiquitous
  - Monitoring and Configuration (Astrolabe)
  - Eventual Consistency (Dynamo)
  - Failure Detection (S3)
  - …

Gossip Protocols

- Basic idea: each node executes

  periodically

  p := selectRandomPeer();

  peerState := *p*.getState();

  myState := *me*.getState();

  newState := **merge**(*myState*, *peerState*);

  p.putState(*newState*);

  me.putState(*newState*);

Gossip cont'd

- Pioneered by Al Demers et al. 1987 (includes Doug Terry in audience)
- Salient properties:
  - Propagates in time proportional to log(#peers)
  - Tolerates host failures and message loss
  - Behavior easily modeled

Two types of gossip

- Rumor Mongering
  - Gossip for some time
  - Every message is important
  - Useful for reliable broadcast
- Anti-Entropy
  - Gossip until obsolete
  - Only last update is important
  - Useful for eventual consistency

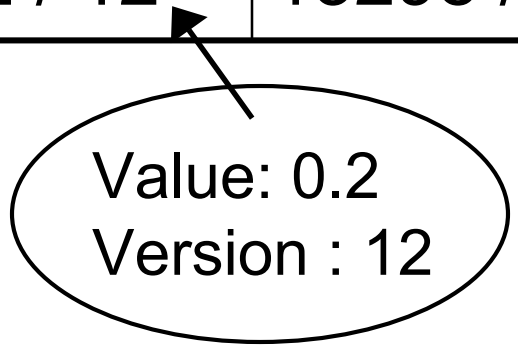
Problems with Anti-Entropy

- *Synchronous communication channel*
  - Capacity limited by available network capacity and CPU for handling updates
  - When overloaded, updates may back up
  - Tuning involves
    - Setting gossip rate
    - Setting maximum message size
  - Tuning affects the capacity of the channel

State of a Gossiper

(may only write own row)

	Request Rate	Number of Items	Number of Clients
Venus	0.5 / 21	2300 / 12	3 / 25
Mars	1.3 / 11	1432 / 24	4 / 12
Jupiter	0.2 / 12	13298 / 3	10 / 13

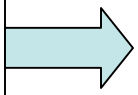


Only last versions are relevant

State Merge

	Request Rate	Number of Items	Number of Clients
Venus	0.5 / 21	2300 / 12	3 / 25
Mars	1.3 / 11	1432 / 24	5 / 14

	Request Rate	Number of Items	Number of Clients
Venus	0.5 / 21	2400 / 13	3 / 25
Jupiter	0.2 / 12	13298 / 3	10 / 13



	Request Rate	Number of Items	Number of Clients
Venus	0.5 / 21	2400 / 13	3 / 25
Mars	1.3 / 11	1432 / 24	5 / 14
Jupiter	0.2 / 12	13298 / 3	10 / 13

Merge protocol exchanges deltas

Bandwidth Limited

- Limited available b/w per gossip exchange
  - Can't send all deltas every time (or even ever)
    - Limited bandwidth, limited CPU
    - Need to prioritize
- Two parts to this talk
  1. Initially assume b/w is fixed and consider merge
  2. Then assume b/w depends on background load, and consider flow control

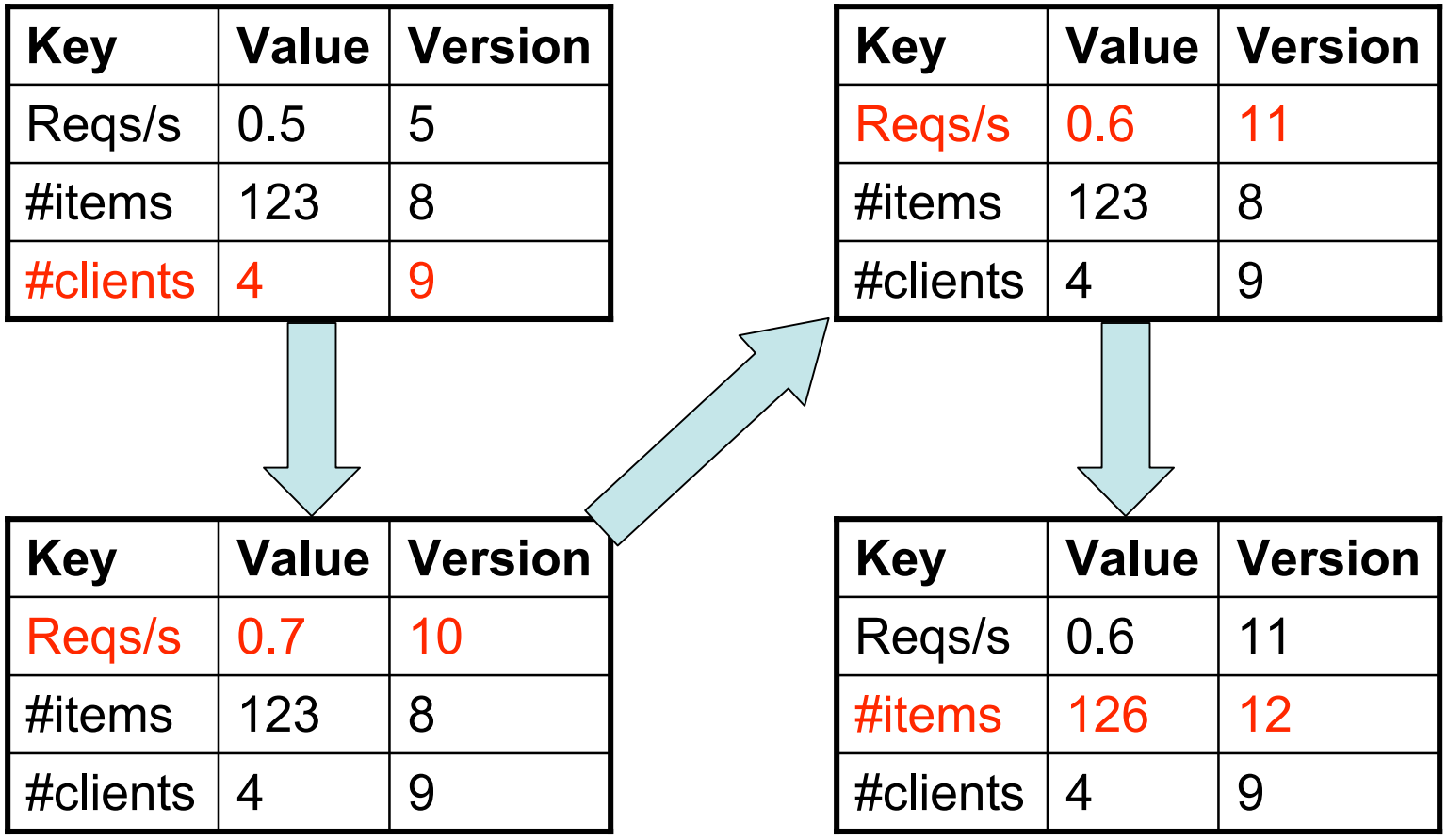
Baseline: Precise Reconciliation

- Focus of much research in the area
  - *Byers, Considine, Mitzenmacher 2002*
  - *Minsky, Trachtenberg, Zippel 2003*
- If bandwidth is limited, can only send subset. Two obvious choices:
  1. Send most out-of-date updates first
     - Seems fair
  2. Send most recent updates first
     - Kills obsolete updates faster, but may lead to starvation
- *Both have high CPU overhead*
  - Hash functions, Bloom filters, Merkle trees, …

Scuttlebutt Reconciliation

- Simple: one version number per participant

Assigning Version Numbers



Note: *never two attributes with the same version number*

Gossiping: Two Phases

I

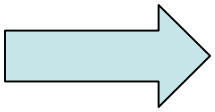
Venus:

	Max(Version)
Venus	6
Mars	12
Jupiter	17

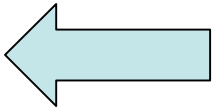
Jupiter:

	Max(Version)
Venus	4
Mars	14
Jupiter	18

II



Venus' attributes with versions 5 and 6



Mars' attributes with versions 13 and 14
Jupiter's attribute with version 18

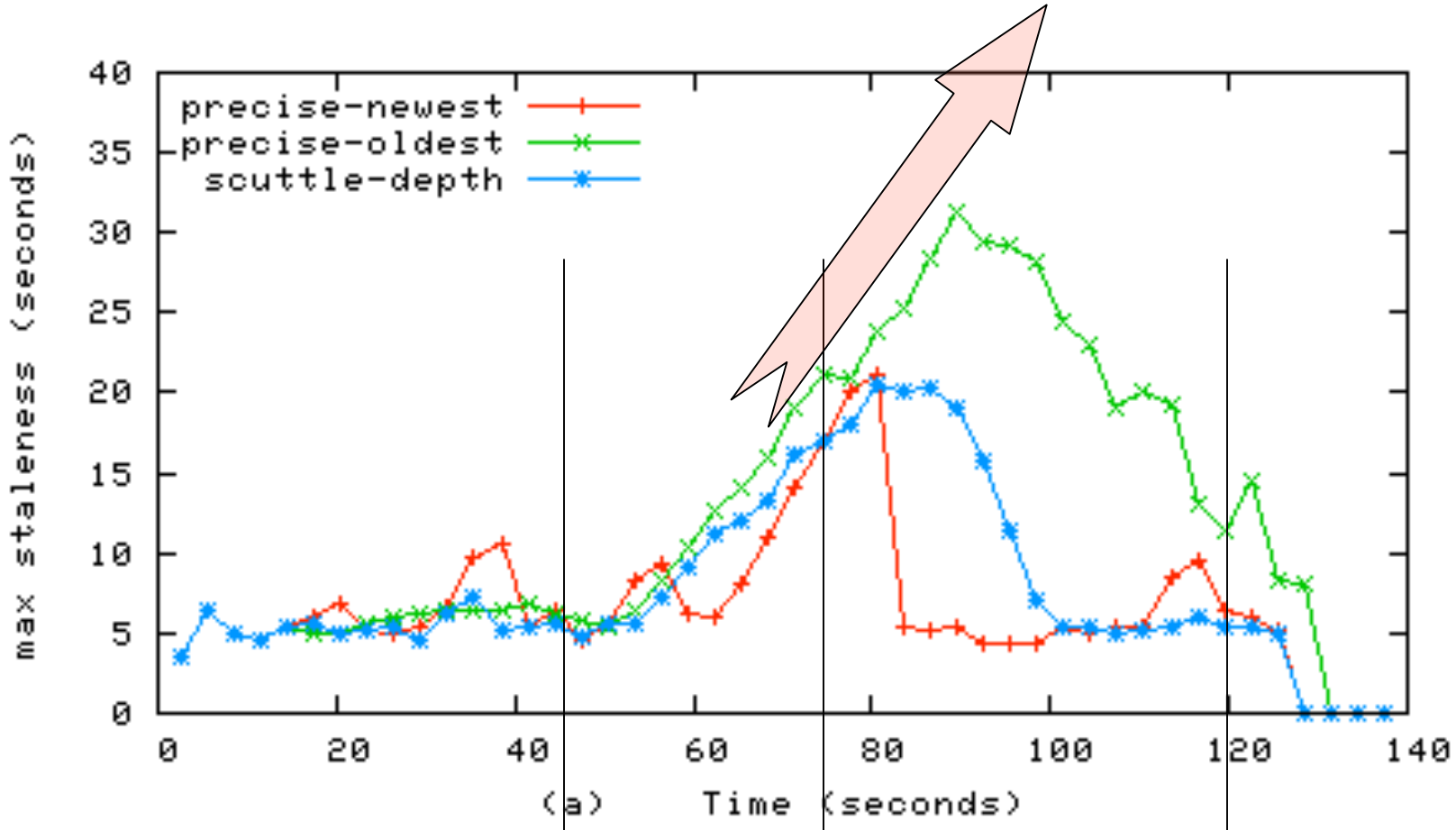
Scuttlebutt convergence

- *May not eliminate all diffs in single exchange*
- But it *does* converge to consistent state, even when only a subset of updates are exchanged

Simulation Experiments

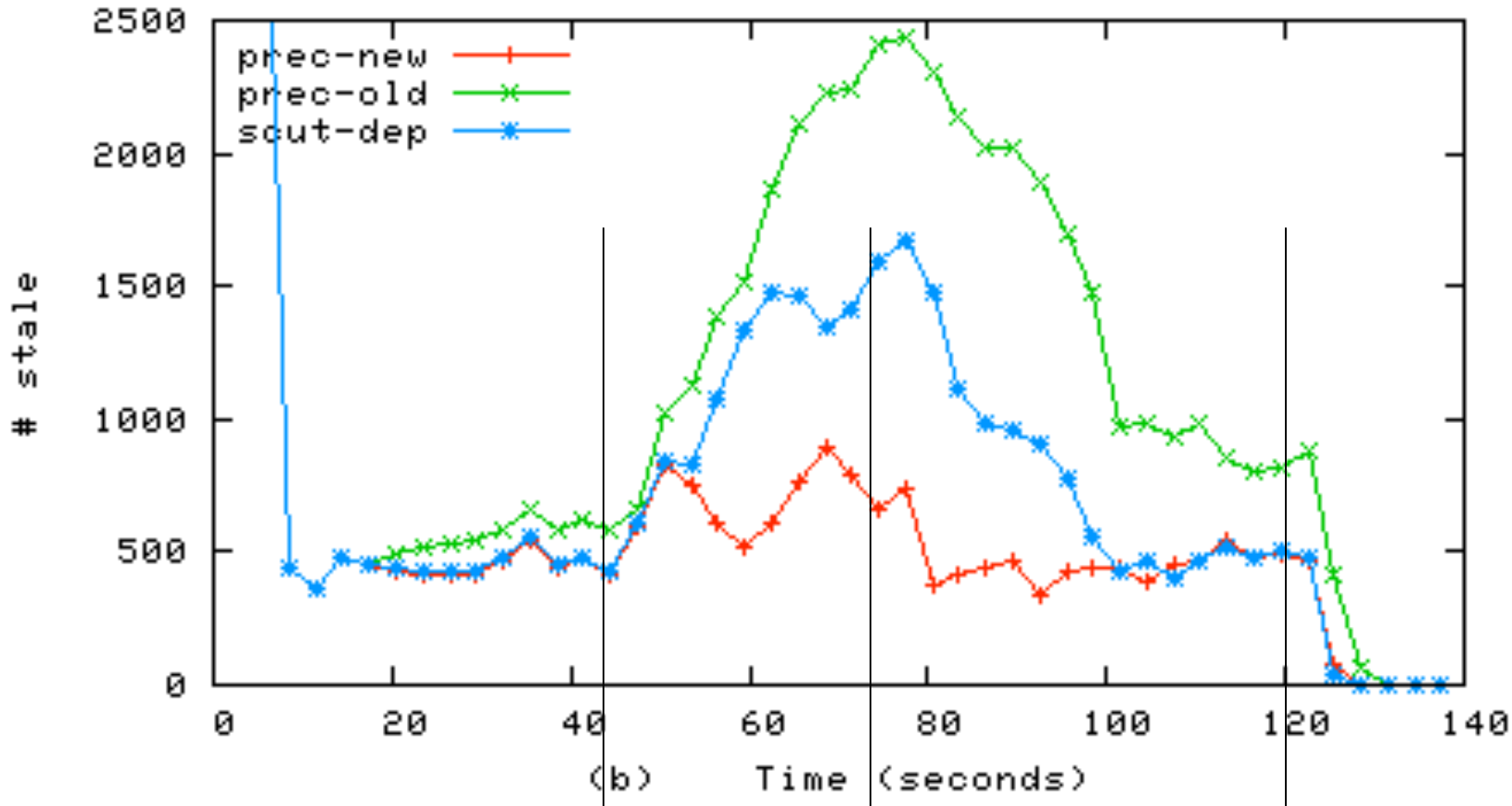
- 128 gossipers
- 64 attributes / gossipers (total: 8192 attrs)
- Updates: uniform (similar results with Zipf)
- Gossip once a second
- MTU: 100 diffs

Maximum Staleness



Updates / sec: 128 256 128 0

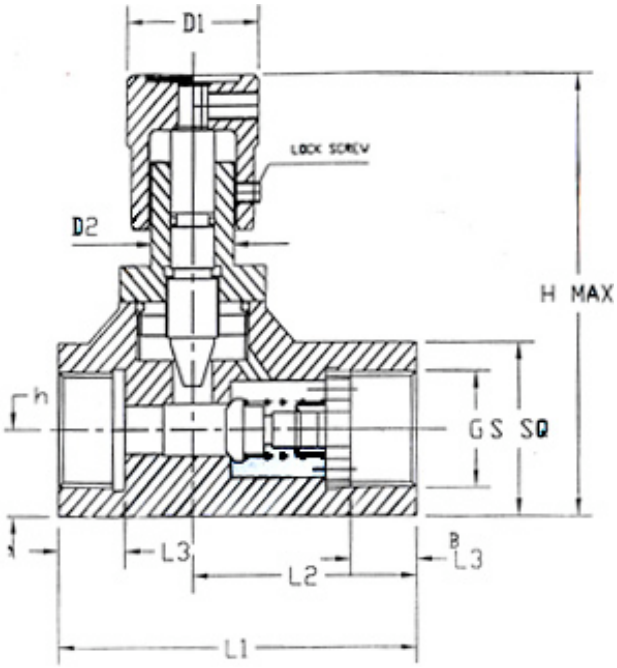
stale attributes



Updates / sec: 128 256 128 0

Flow Control

- Merge alone cannot solve overload problem
- Flow Control: determine the maximum rate at which a peer can submit updates
- Requirements:
  - Optimal
  - Fair
  - Adaptive



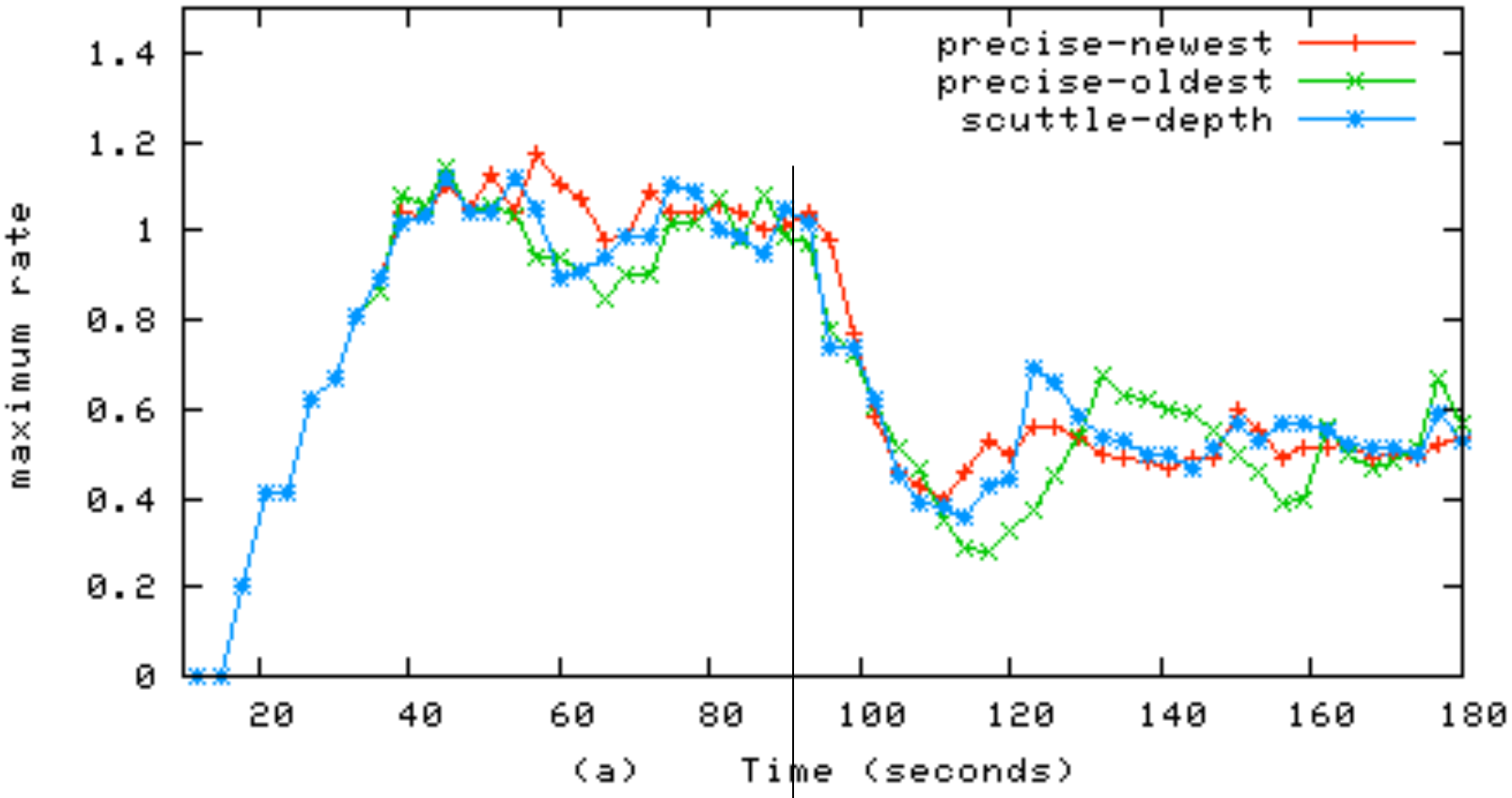
Fairness

- Accomplished through gossip itself
- Each participant maintains a maximum update rate at which it will submit updates
- When participants gossip, they split the difference between max. rates

Local Adaptation

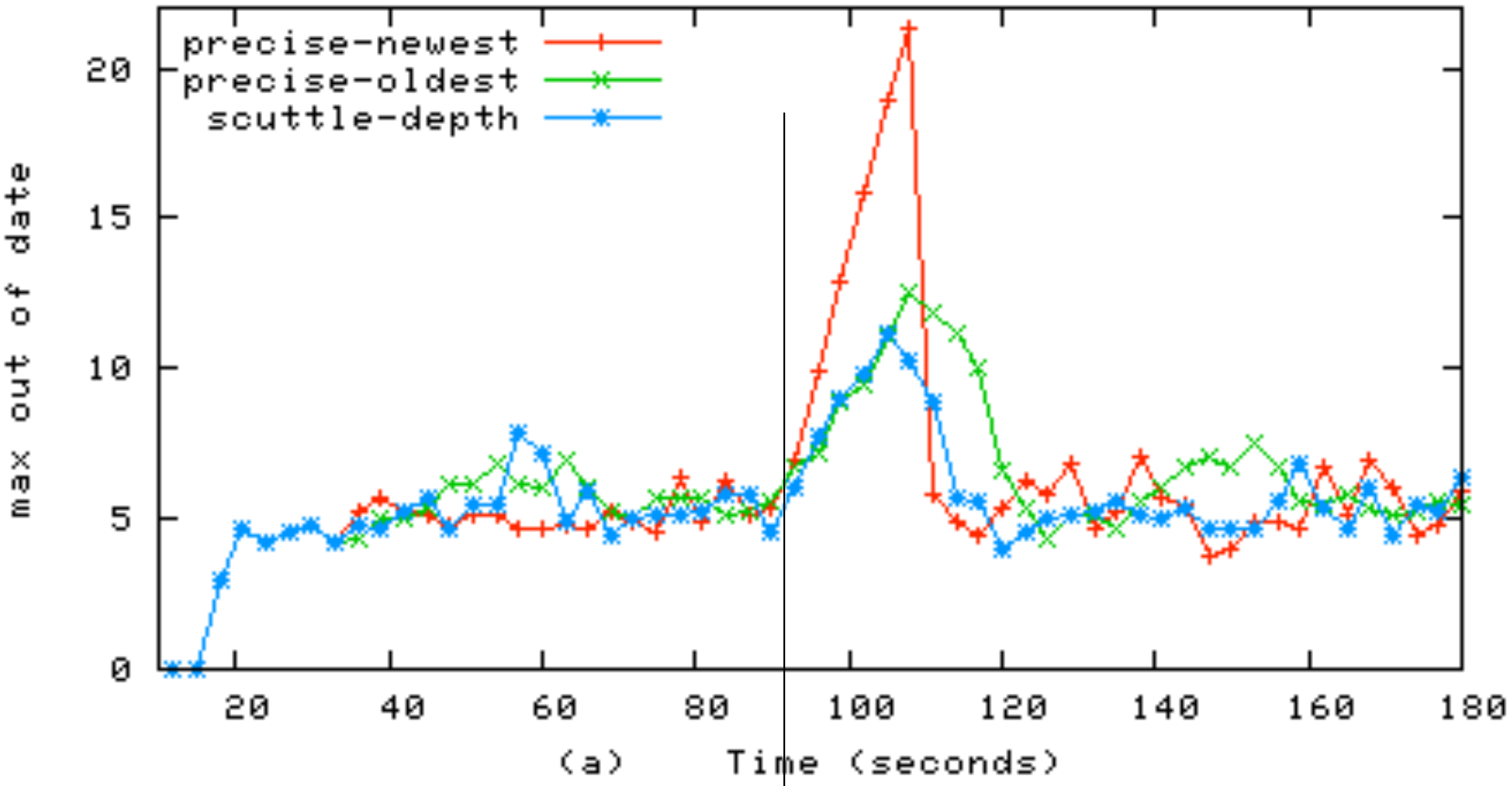
- AIMD approach, a la TCP
- If gossip message overflows, then reduce maximum rate by a percentage
- If gossip message underflows, then increase rate additively

Maximum Update Rate



MTU: 100 | MTU: 50

Maximum Staleness



MTU: 100 MTU: 50

Conclusion

- In overload situation, gossip does *not* provide predictable performance
- We contributed
  - A low overhead reconciliation mechanism
  - Flow Control for anti-entropy protocols