Attested Append-only Memory: Making Adversaries Stick to their Word

Distributed Storage Systems
CS 6464
2-19-09

presented by: Hussam Abu-Libdeh

Motivation

- You want to build a service
  - Easy on a single machine
  - What about failure and reliability?
    - Replicate service on multiple machines
- Replicated services must appear as single server
  - Linearizability
    - Completed client requests appear to have been processed in a **single**, **totally ordered**, **serial schedule** consistent with the order they were submitted

Motivation

- Machines can fail or be hijacked
  - Byzantine failure
    - Can not distinguish if node is non-faulty, faulty, or malicious
- Faulty servers can lie
  - Equivocation
    - Different lies to different people
  - Previously in cs6464, SUNDR & *fork consistency*

Today

Can we use small trusted components to combat equivocation ?

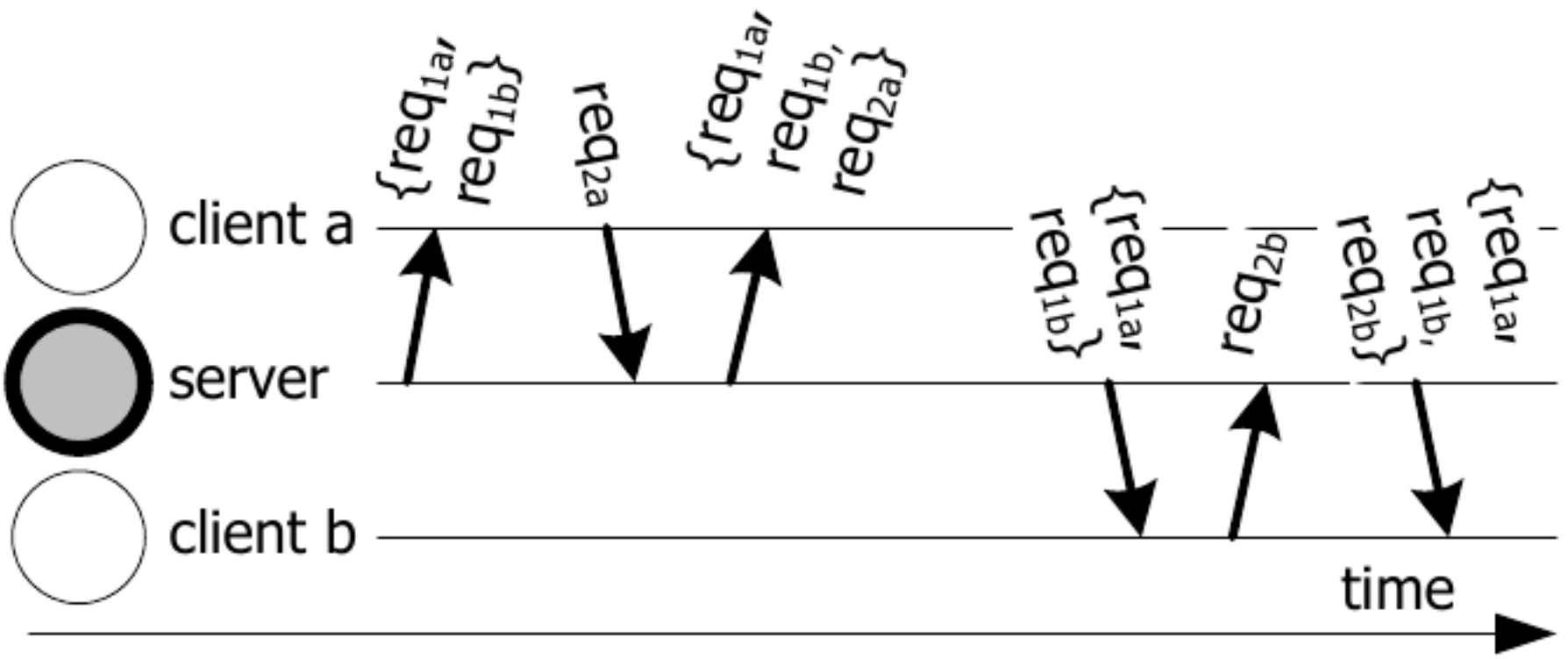
Agenda

- Equivocation “attacks”
- The A2M
- A2M-PBFT-E
- A2M-PBFT-EA
- A2M-Storage
- A2M-PBFT-EAXYZ-FOO-RANDOM-CHARS
  - Ok maybe not
- Discussion

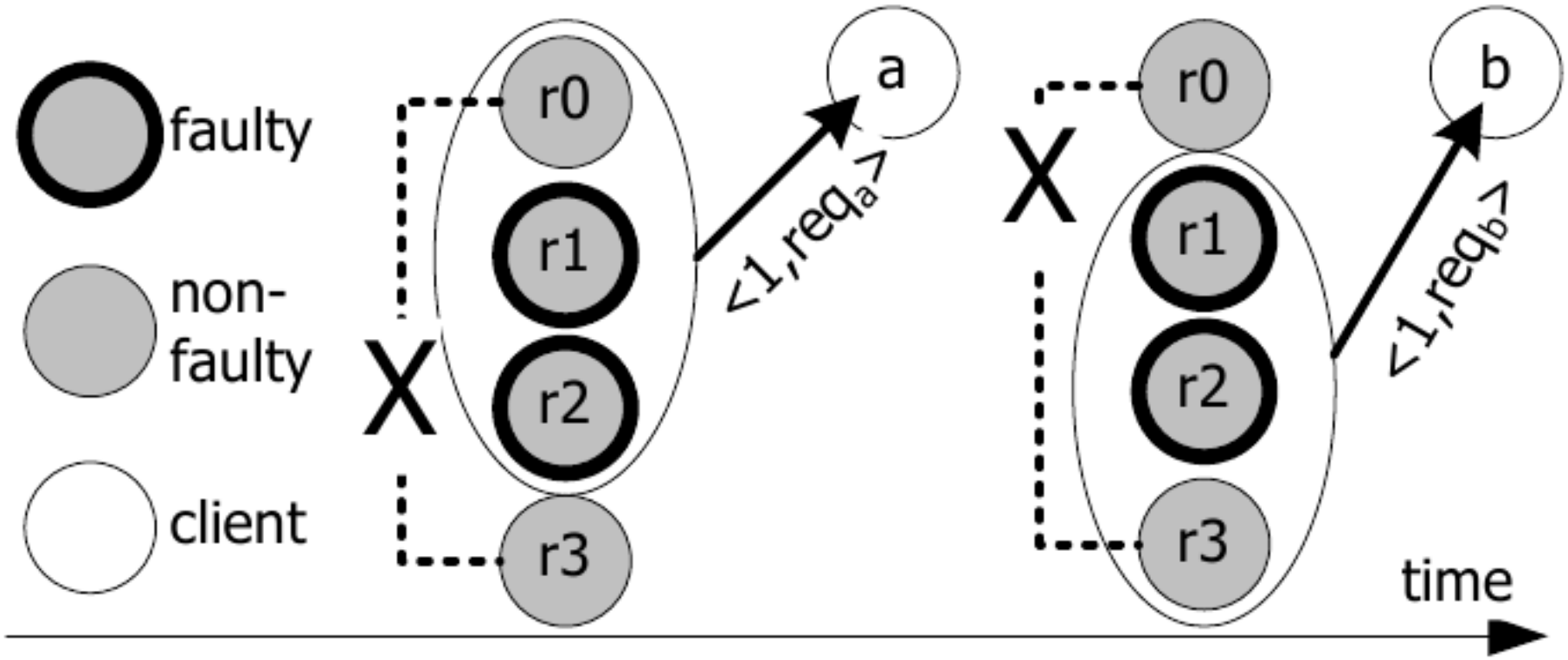
Equivocation

- Servers respond incorrectly and differently to different clients
  - Can be detected if clients were trusted
- Could happen in two places
  - Servers equivocating to clients
  - Servers equivocating to other servers
- Both bad

Equivocating to Clients



Equivocating to Servers



A2M

- Attested Append-only Memory
- A trust abstraction
- Essentially:
  - A chunk of memory
  - You can access it
  - You trust its content
    - You have a reason to trust it
    - Backed up by a TPM, or placed in a trusted VM or VMM or on a separate trusted machine ..etc

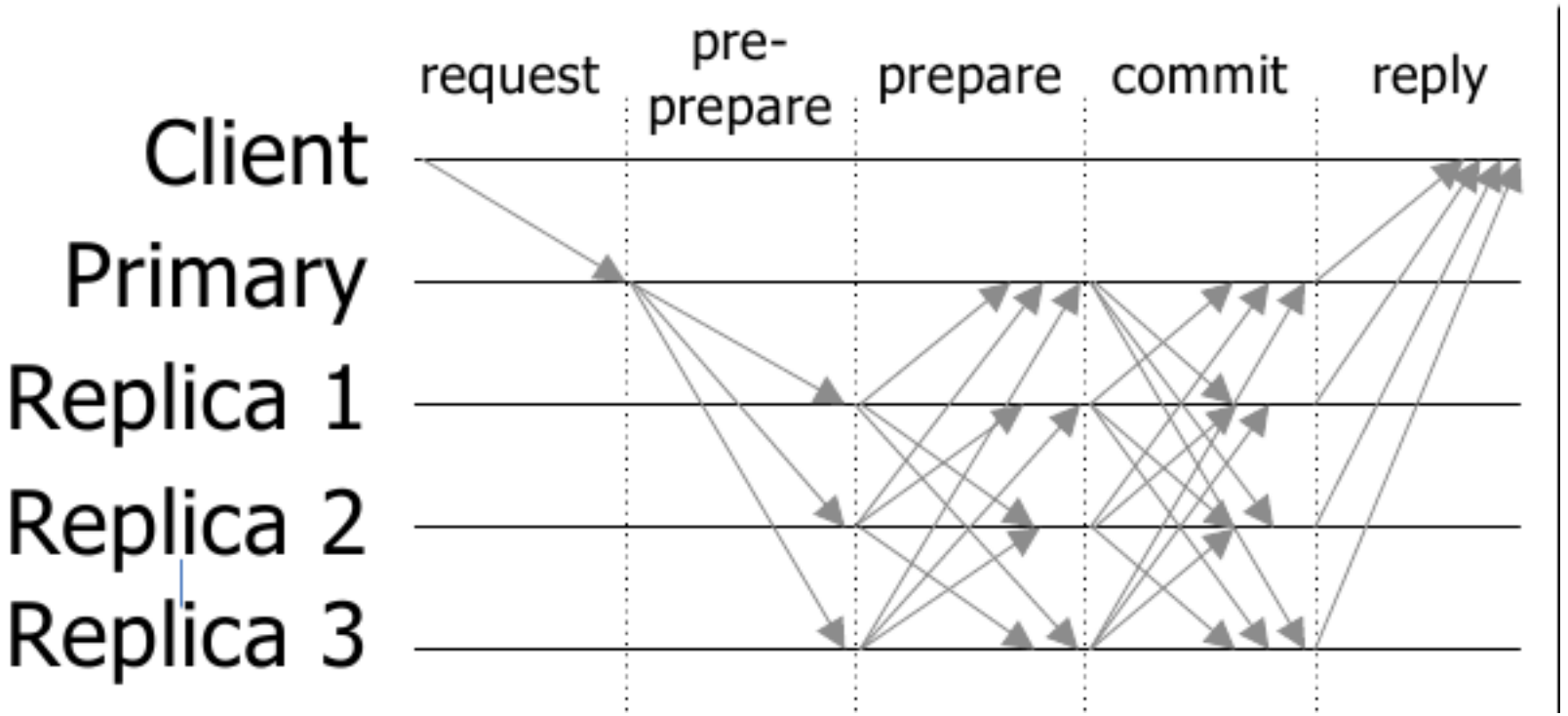
A2M Interface

- Supports basic operations
  - append(q,x)
    - Add value to the tail of the list
  - lookup(q,n,z)
    - Look up value at position n
  - end(q,z)
    - Look up last entry in list
  - truncate(q,n)
    - Remove all entries below n
  - advance(q,n,d,x)
    - Skip a few positions (n-current position) in the list

PBFT

- Practical Byzantine Fault-Tolerance
- Client sends request, later a reply is accepted if received from more than 1/3 of the servers
- Internally works in 3 phases
  - Primary multicasts pre-prepare to all replicas
  - If a server receives pre-prepares from > 2/3 of the servers, it multicasts a prepare message
  - If a server receives prepares from > 2/3 of the servers, it multicasts a commit message

PBFT


request
pre-prepare
prepare
commit
reply
Client
Primary
Replica 1
Replica 2
Replica 3

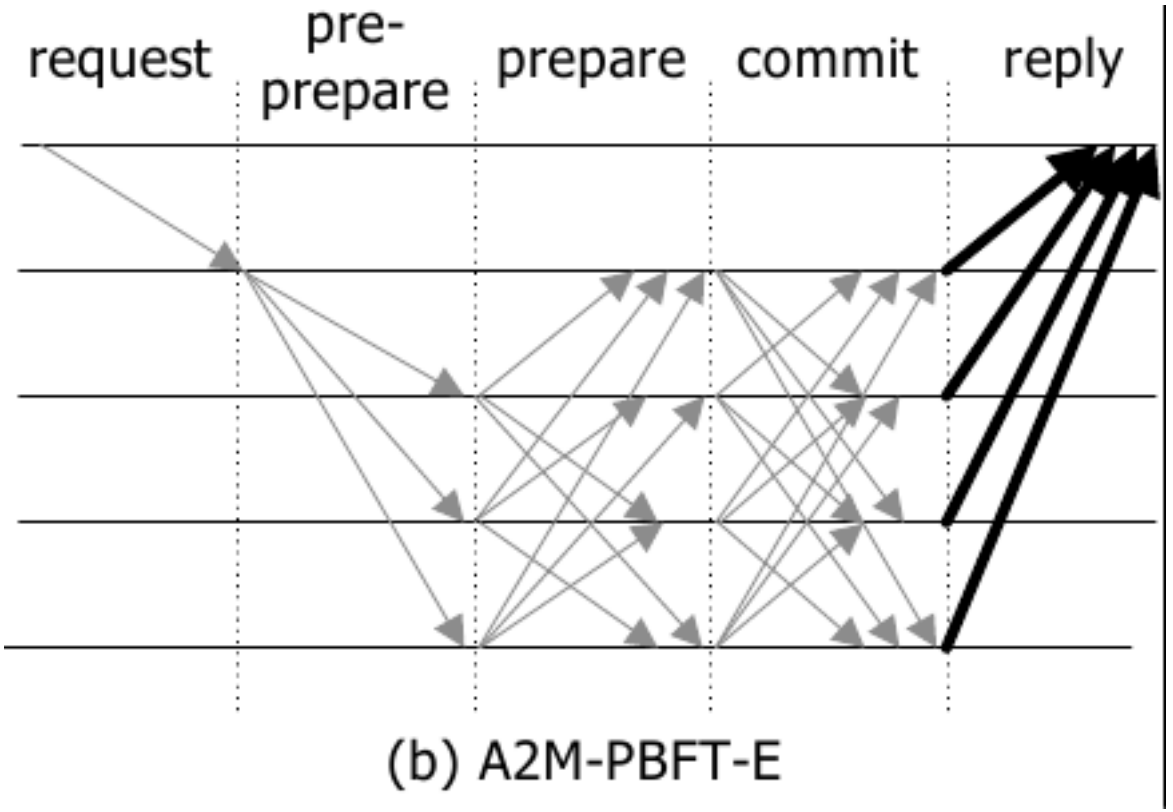
PBFT

- Two steps of PBFT
  - Agreement
    - pre-prepare, prepare, and commit messages
  - Execution
    - communication between replicas and client
- Other parts of PBFT
  - Checkpointing, changing views ..etc
  - Not central to our discussion today

A2M-PBFT-E

- PBFT with trusted Execution step (A2M)
  - Replicas can equivocate to each other
  - Equivocation to clients will be detected
- Clients accept reply quorums if all agree in A2M entry for the reply sequence number
  - Requires > 2/3 replicas be non-faulty (like PBFT)
  - If 1/3 < # faulty < 2/3
    - Clients won't commit faulty sequence #s because at least one replica will have correct A2M entry

A2M-PBFT-E

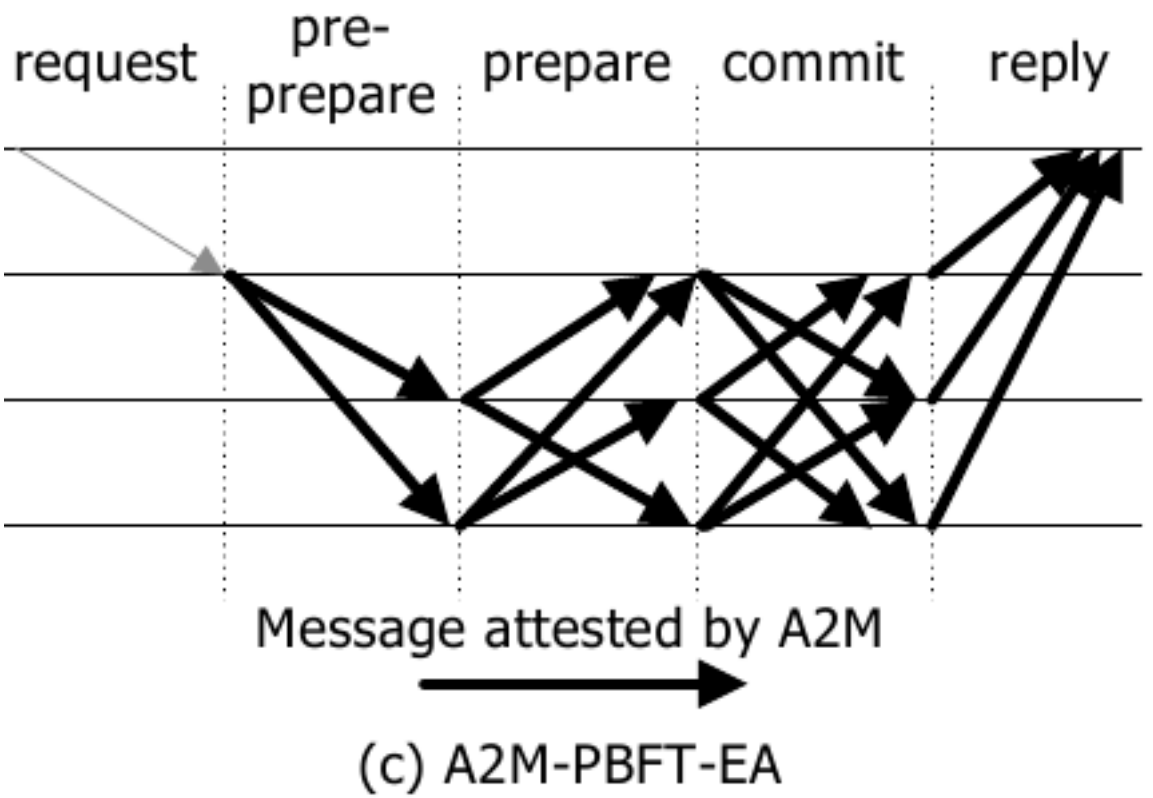


(b) A2M-PBFT-E

A2M-PBFT-EA

- PBFT w/ trusted Execution & Agreement steps
  - Equivocation to clients will be detected
  - Equivocation to servers will be detected
- At each step, replicas attest msgs with A2M
  - Just need a majority (>1/2) of replicas to agree
  - Thus can tolerate <1/2 of faulty servers

A2M-PBFT-EA



(c) A2M-PBFT-EA

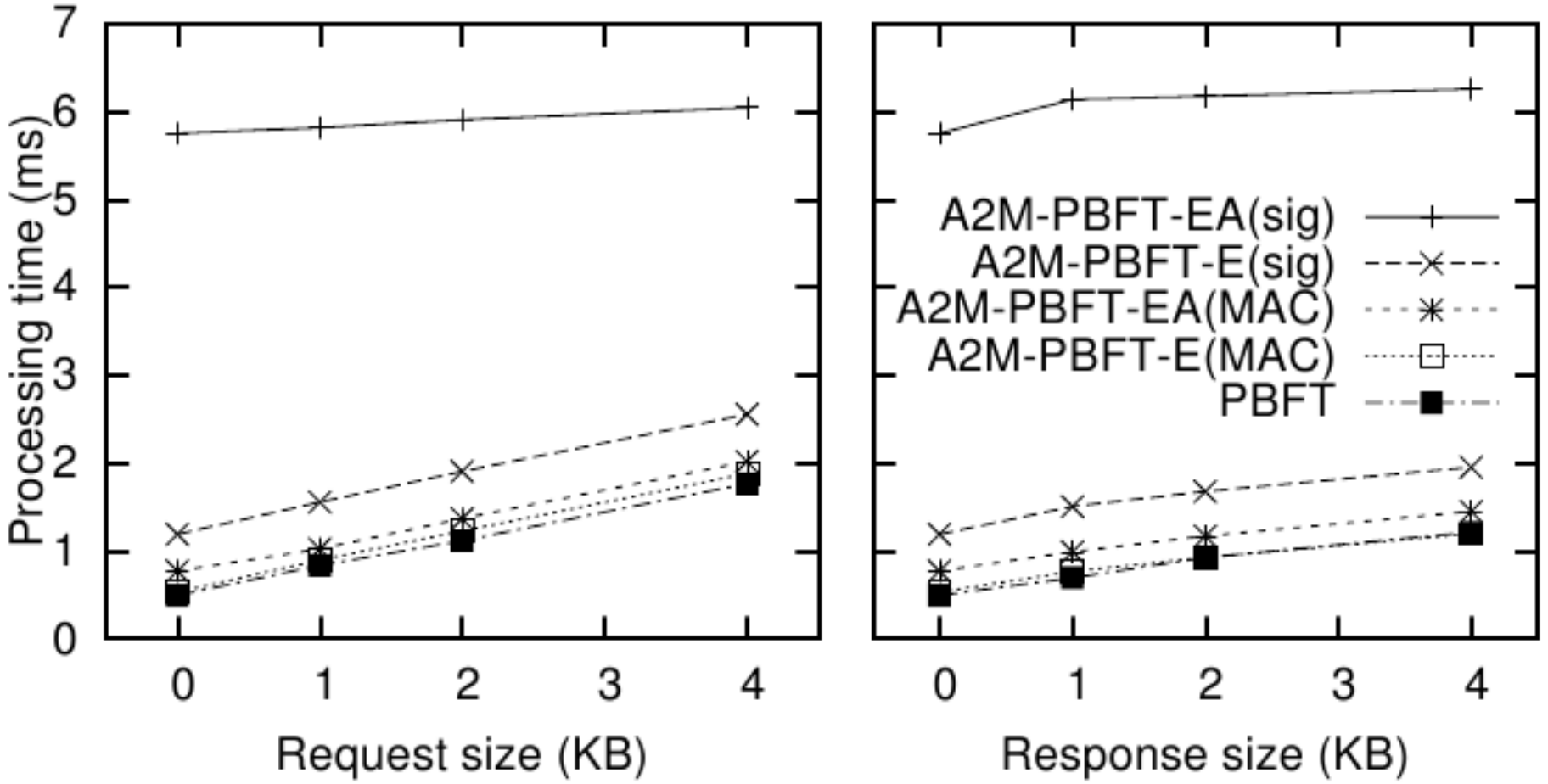
A2M-Storage

- Server maintains two A2M logs
  - One for operation digest (like SUNDR) (log s)
  - One for latest write sequence number (log h)
- Client use timestamps with read/write
  - Timestamp = (req_seq, att_seq_h, att_seq_s)
    - i.e. Client operations attest if current sequence # is latest
  - Clients store their last timestamp
- Read/Write operations use timestamp
  - If latest, proceed; otherwise, refresh

Evaluation

- Emulated A2M in a C++ module
- Ran agreement protocol w/ 4 replicas & 1 client
- Microbenchmark
  - requests/replies of various size
- Macrobenchmark
  - NFS front-end with PBFT backend
  - Compile a relatively small package
- Results not surprising

Evaluation - Microbenchmarks



Evaluation - Macrobenchmarks

NFS Phase	-S	-PBFT	-A2M-PBFT-E (sig)	-A2M-PBFT-E (MAC)	-A2M-PBFT-EA (sig)	-A2M-PBFT-EA (MAC)
Copy	0.219	0.709	1.026	0.728	2.141	0.763
Uncompress	1.015	3.027	4.378	3.103	8.601	3.236
Untar	2.322	4.448	6.826	4.553	12.896	4.669
Configure	12.748	12.412	19.173	12.659	26.181	13.040
Make	7.241	7.461	9.778	7.500	11.379	7.510
Clean	0.180	0.298	0.640	0.312	0.742	0.311
Total	23.725	28.355	41.821	28.854	61.940	29.528

Table 1: Mean time to complete the six macrobenchmark phases in seconds.

Evaluation – Varying delay time

Additional NFS-latency (μs)	A2M-PBFT-E (MAC)	A2M-PBFT-E (MAC) with batching	A2M-PBFT-EA (MAC)	A2M-PBFT-EA (MAC) with batching
1	28.854	28.763	29.528	29.505
10	29.598	29.025	31.299	30.188
50	32.735	30.232	36.242	32.214
250	48.784	37.237	66.441	45.199
1000	117.59	65.813	192.53	101.62

Table 2: Mean time to complete the six macrobenchmark phases in seconds for different A2M additional latency costs.

Thank You