ACMS: The Akamai Configuration Management System

A. Sherman, P. H. Lisiecki, A. Berkheimer, and J. Wein

Instructor: Fabian Bustamante

Presented by: Mario Sanchez

The Akamai Platform

- Over 15,000 servers (now 56k!)
- Deployed in 1200+ different ISP networks
- In 60+ countries (over 70+)

Motivation

- Customer profiles:
  - Customers need to maintain close control over the manner in which their web content is served
  - Customers need to configure different options that determine how their content is served by the CDN
- Akamai's internal services and processes:
  - Need for frequent updates or "reconfigurations"

Akamai Configuration Management System (ACMS)

- Supports configuration propagation management
- Accepts and disseminates distributed submissions of configuration information
- Availability
- Reliability
- Asynchrony
- Consistency
- Persistent storage

Problem

- The widely dispersed set of end clients
- At any point in time some servers may be down or have connectivity problems
- Configuration changes are generated from widely dispersed places
- Strong consistency requirements

Assumptions

- The configuration files will vary in size from a few hundred bytes up to 100MB
- Most updates must be distributed to every Akamai node
- There is no particular arrival pattern of submissions
- The Akamai CDN will continue to grow
- Submissions could originate from a number of distinct applications running at distinct locations on the Akamai CDN

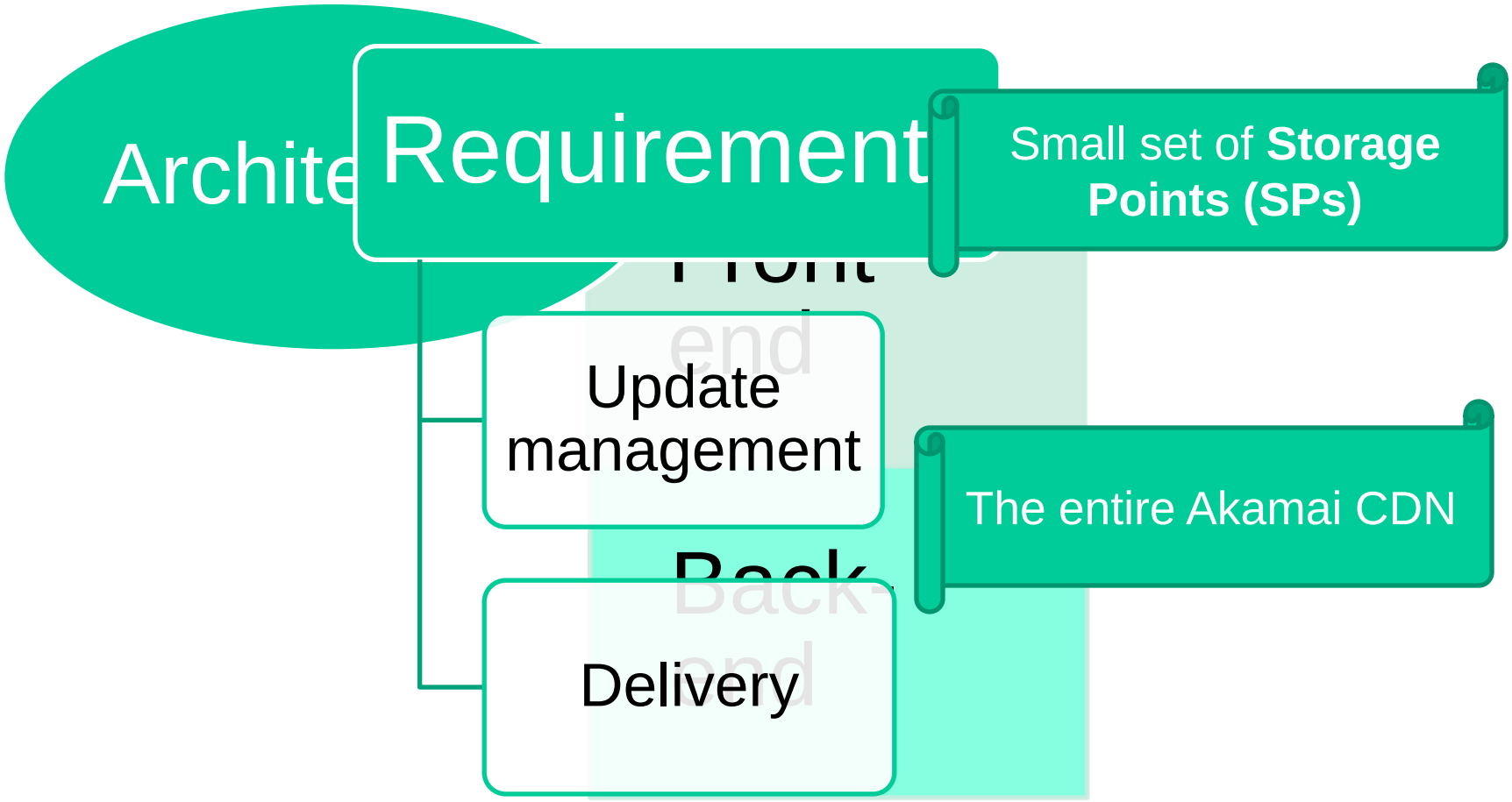
Assumptions Continued

- Each submission of a configuration file *foo completely overwrites the earlier submitted version* of *foo*
- For each configuration file there is either a single writer or multiple idempotent (non-competing) writers

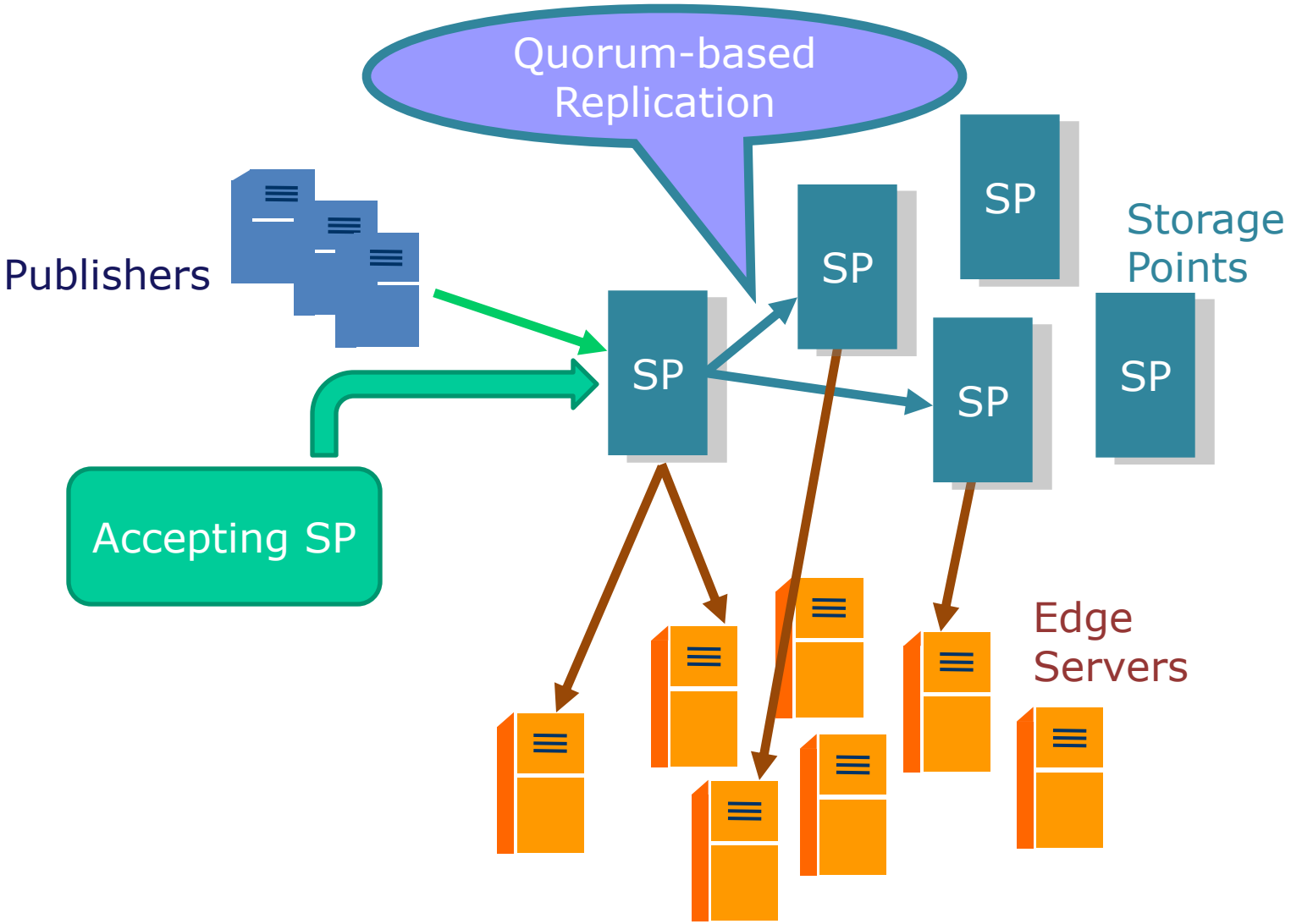
Requirements

- **High Fault-Tolerance and Availability** - multiple entry points
- **Correctness** - unique ordering of versions
- **Acceptance Guarantee** - quorum of entry points
- **Persistent Fault-Tolerant Storage** - asynchronous delivery avoids downtime
- **Efficiency and Scalability** - quick server synchronization
- **Security**

Approach


Archite
Requirement
Small set of **Storage Points (SPs)**
Front end
Update management
The entire Akamai CDN
Back end
Delivery

Architecture



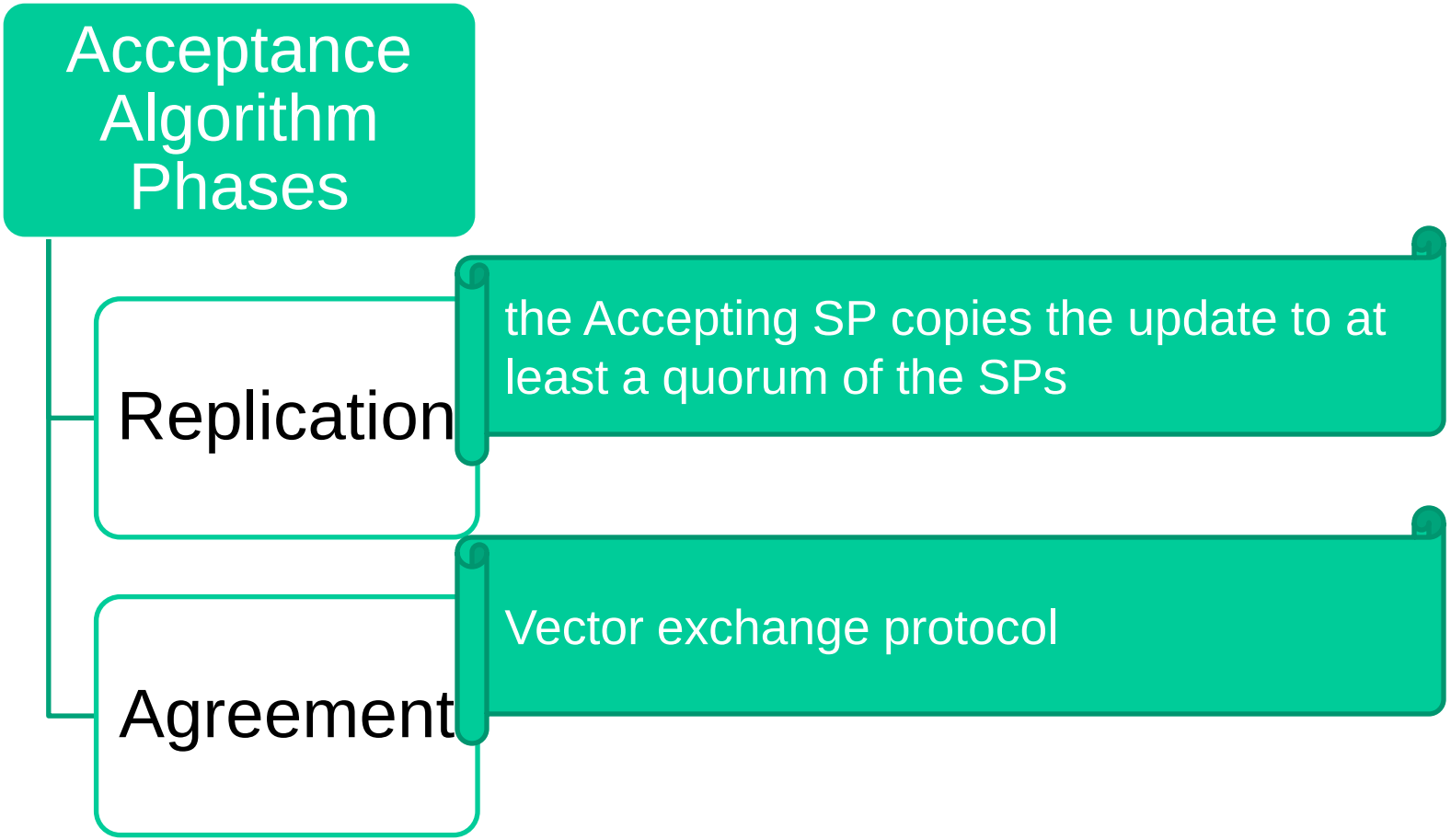
Quorum-based Replication

- A quorum is defined as a majority of the ACMS SPs
- Any update submission should be replicated and agreed upon by the quorum
- A majority of operational and connected SPs should be maintained
- Every future majority overlaps with the earlier majority that agreed on a file

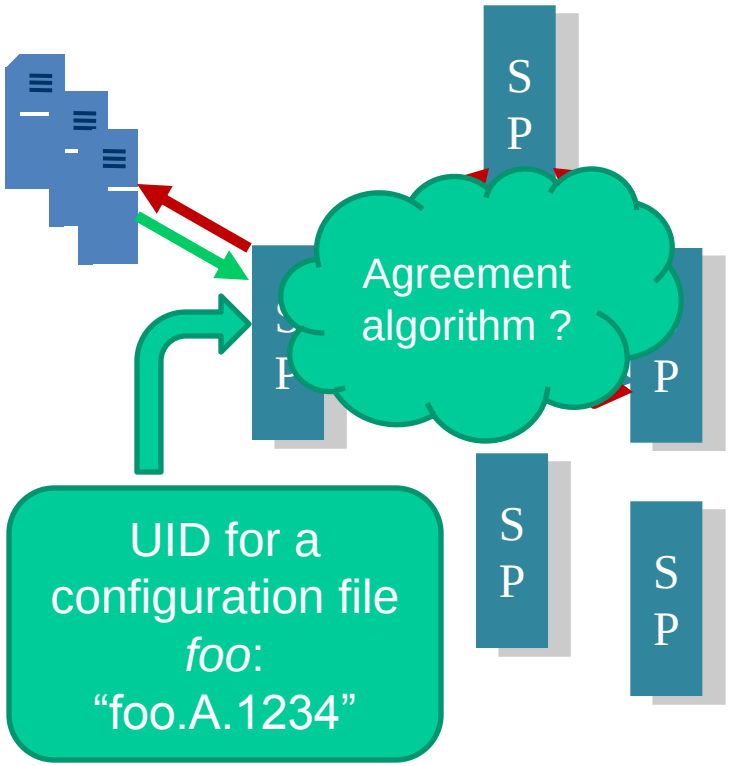
Quorum-based Replication

- Each SP maintains connectivity by exchanging **liveness messages** with its peers.
- If there is no quorum of SPs, the system halts and refuses to accept updates
- Monitored by the NOCC
- SPs are located in distinct ISPs networks to reduce the probability of an outage

Acceptance Algorithm



Acceptance Algorithm Continued



A publisher contacts an *accepting* SP

The Accepting SP first creates a temporary file with a unique filename (UID)

The accepting SP sends this file to a number of SPs

If replication succeeds the accepting SP initiates an agreement algorithm called ***Vector Exchange***

Upon success the accepting SP "accepts" and all SPs upload the new file

Acceptance Algorithm Continued

- The VE vector is just a bit vector with a bit corresponding to each Storage Point
- A 1-bit indicates that the corresponding Storage Point *knows of a given update*
- When a majority of bits are set to 1, we say that an *agreement* occurs and it is safe for any SP to upload this latest update

Acceptance Algorithm Continued

- The Accepting SP sets its own bit to 1 and the rest to 0,
- broadcasts the vector along with the UID of the update to the other SPs

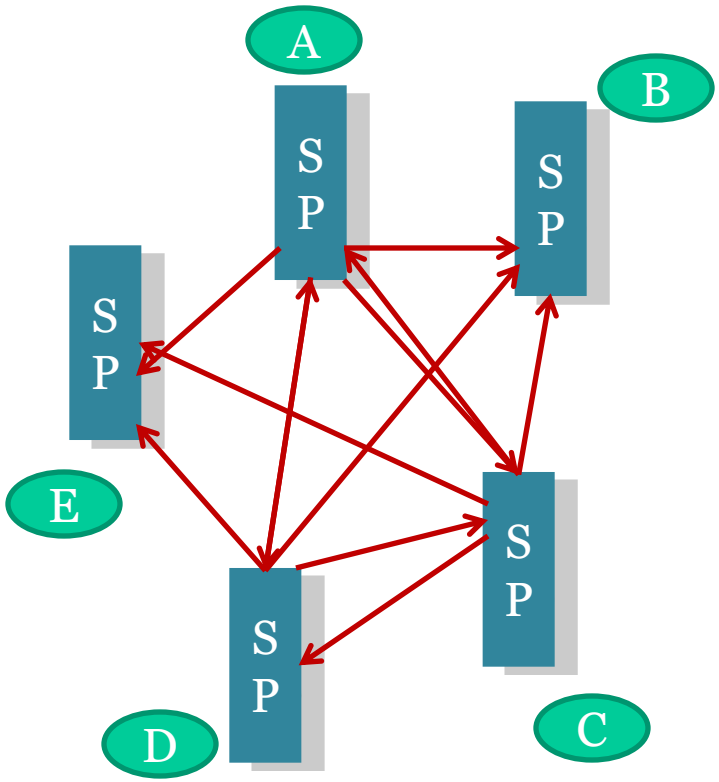
- Any SP that sees the vector sets its corresponding bit to 1,
- re-broadcasts the modified vector to the rest of the SPs

- Each SP learns of the agreement independently when it sees a quorum of bits set

- When the Accepting SP that initiated the VE instance learns of the agreement it accepts the submission of the publishing application

- When any SP learns of the agreement it uploads the file

Acceptance Algorithm Continued



"A" initiates and broadcasts a vector:
A:1 B:0 C:0 D:0 E:0

"C" sets its own bit and re-broadcasts:
A:1 B:0 C:1 D:0 E:0

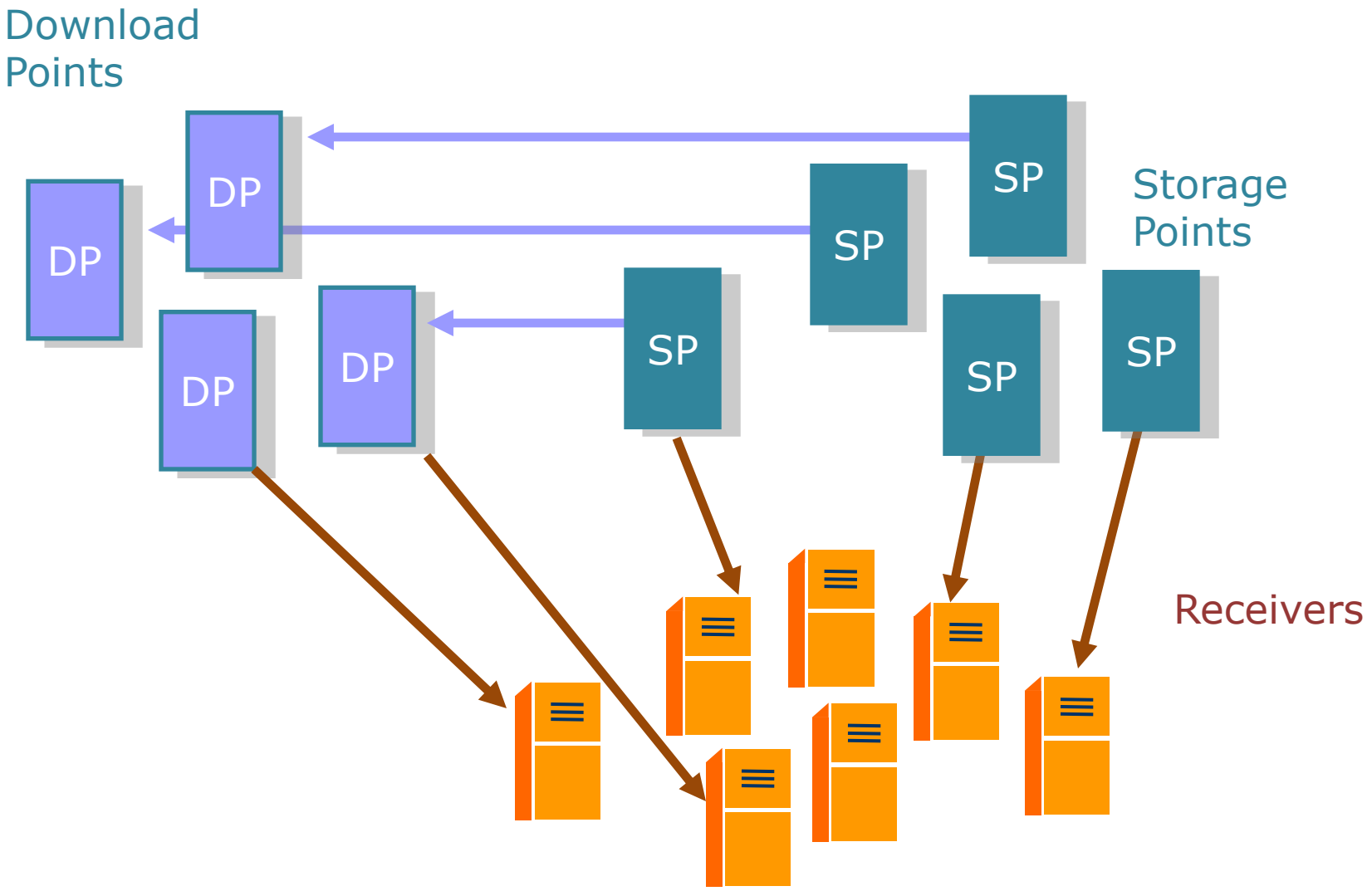
"D" sets its bit and rebroadcasts
A:1 B:0 C:1 D:1 E:0

Any SP learns of the "agreement" when it sees a majority of bits set.

Recovery

- The recovery protocol is called Index Merging
- SPs continuously run the background recovery protocol with one another
- The downloadable configuration files are represented on the SPs in the form of an index tree
- The SPs “merge” their index trees to pick up any missed updates from one another
- The **Download Points** also need to sync up state

Architecture Optimization



The Index Tree

Configuration files are organized into a tree and split into groups:

- **Group Index file:** lists the UIDs of the latest agreed upon updates for each file in the group
- **Root Index files:** lists all Group Index files together with the latest modification timestamps of those indexes

The Index Merging Algorithm

- Snapshot is a hierarchical index structure that describes latest versions of all accepted files
- Each SP updates its own snapshot when it learns of a quorum agreement
- For full recovery each SP needs only to merge in a snapshot from *majority*-1 other SPs
- SPs merge their index stamp, not just the data listed in those indexes

Timestamping Rules

1. When a Storage Point includes new information inside an index file, its next iteration must increase the file's timestamp
2. An SP should always set its index's timestamp to the highest timestamp for that index among its peers (even if it does not include new info)

(Snapshots are also used by the edge servers to detect changes)

Correctness Requirement

- T is the maximum allowed clock skew between any two communicating SPs
- The communication protocol enforces this bound by rejecting liveness messages from SPs that are at least T seconds apart
- No two SPs that accepts updates for the same file can have a clock skew more than 2T seconds
- Updates for the same file submitted at least 2T+1 seconds apart will be ordered correctly

Data Delivery

- Processes on edge servers subscribe to specific configurations via their local Receiver process
- A Receiver checks for updates to the subscription tree by making HTTP IMS requests recursively
- If the updates match any subscriptions the Receivers download the files

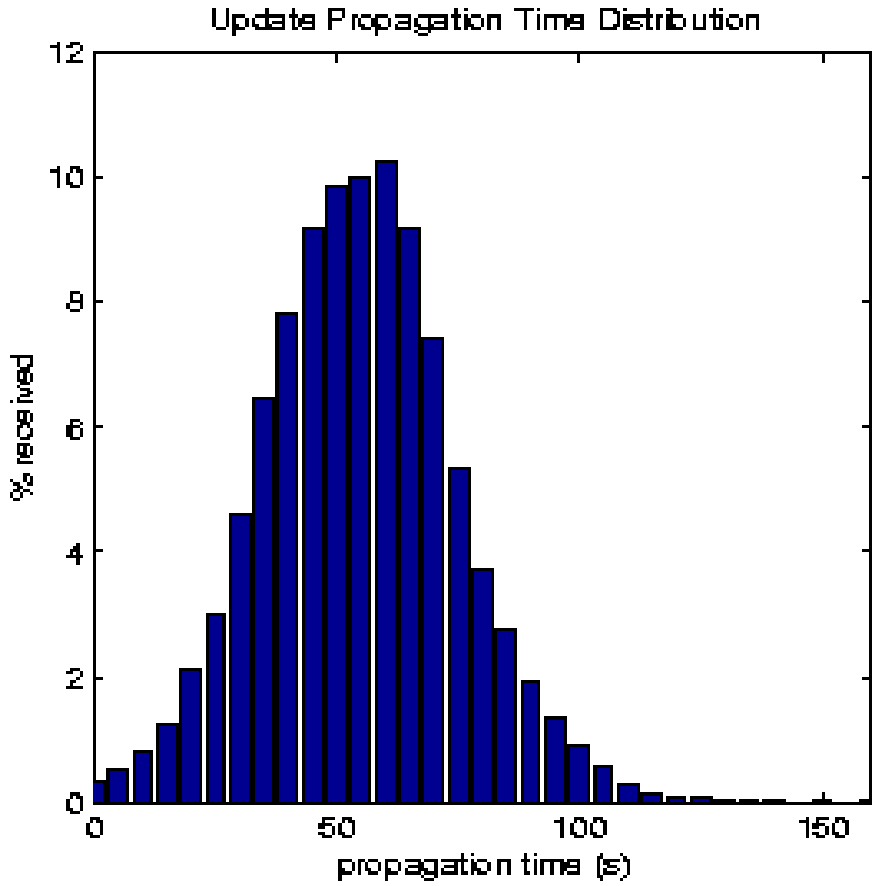
Evaluation

- Workload of the ACMS front-
  the middle of a work we
- 14,276 total file submi
- Five operating Storage P

The period from the time an Accepting SP is first contacted by a publishing application, until it replies with "Accept"

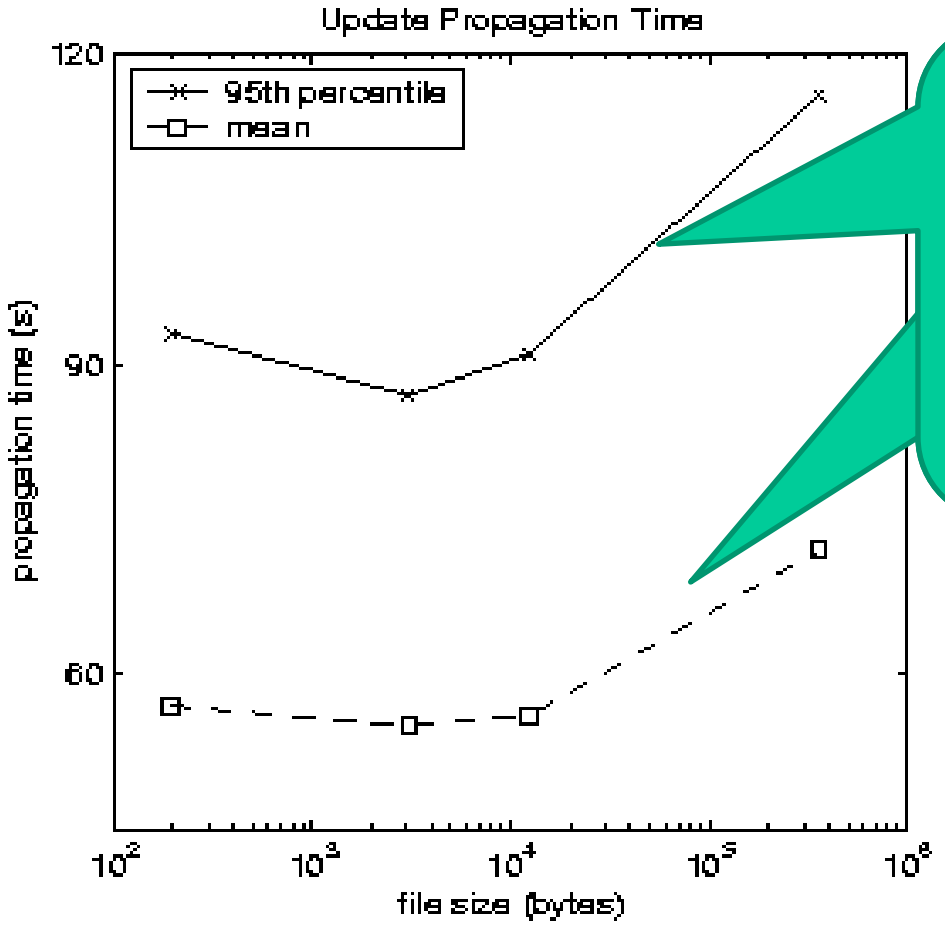
Size range	Avg file sz	Distribution	Avg time(s)
0K-1K	290	40%	0.61
1K-10K	3K	26%	0.63
10K-100K	22K	23%	0.72
100K-1M	167K	7%	2.23
1M-10M	1.8M	1%	13.63
10M-100M	51M	3%	199.87

Propagation Time Distribution



- A random sampling of 250 Akamai nodes
- The average propagation time is approximately 55 seconds

Propagation times for various size files



The average propagation time

were delayed due to temporary network connectivity issues

Discussion

- Push-based vs. pull-based update
- The effect of increasing the number of SP on the efficiency
- The effect of having less number on nodes in the quorum
- The effect of having a variable sized quorum
- Consistency vs. availability trade-offs in quorum selection
- The trade-off of having great cacheability

Impressions

- Just a cool implementation, nothing more?
- Anything new? Sheer size?
- Solves a specific problem

Thank you!

Credit: This presentation was based on the work of Parya Moinzadeh, UIUC…