



Comet: An Active Distributed Key-Value Store

Roxana Geambasu

Amit Levy

Yoshi Kohno

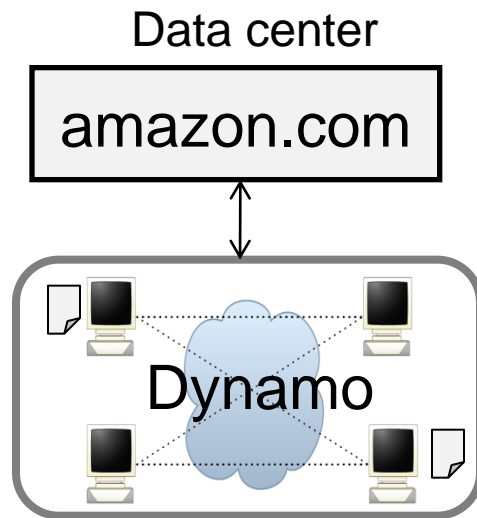
Arvind Krishnamurthy

Hank Levy

University of Washington

Distributed Key/Value Stores

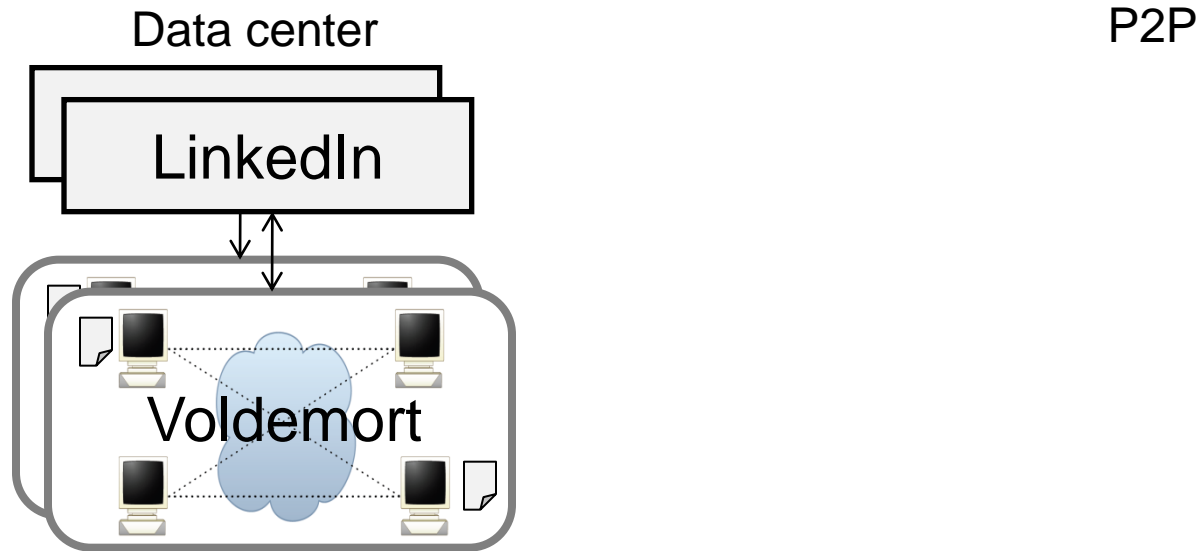
- A simple **put/get** interface
- Great properties: scalability, availability, reliability
- Increasingly popular both within data centers and in P2P



P2P

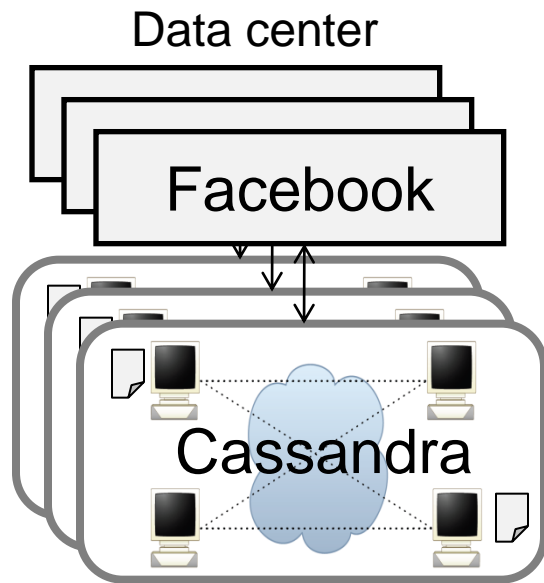
Distributed Key/Value Stores

- A simple **put/get** interface
- Great properties: scalability, availability, reliability
- Increasingly popular both within data centers and in P2P



Distributed Key/Value Stores

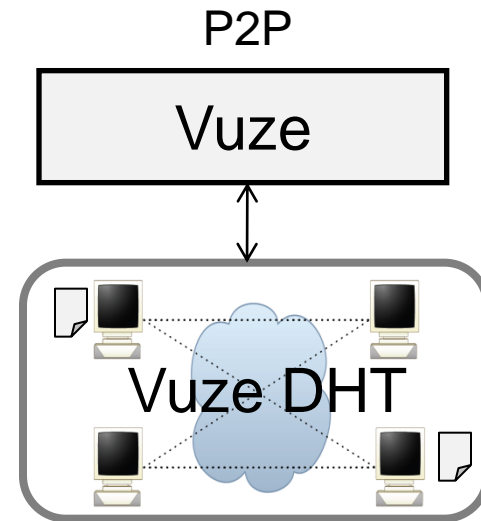
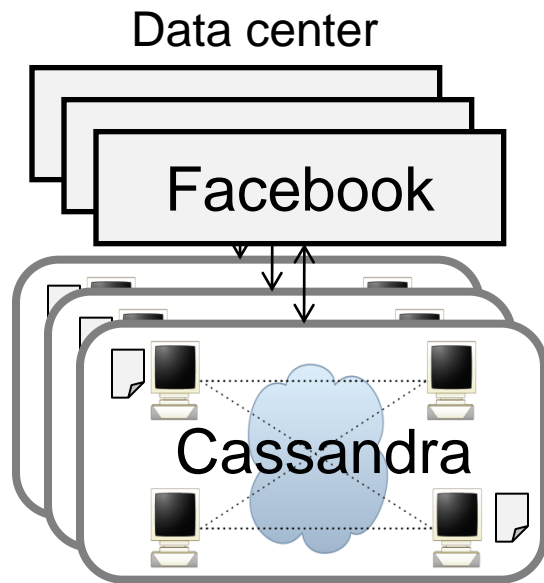
- A simple **put/get** interface
- Great properties: scalability, availability, reliability
- Increasingly popular both within data centers and in P2P



P2P

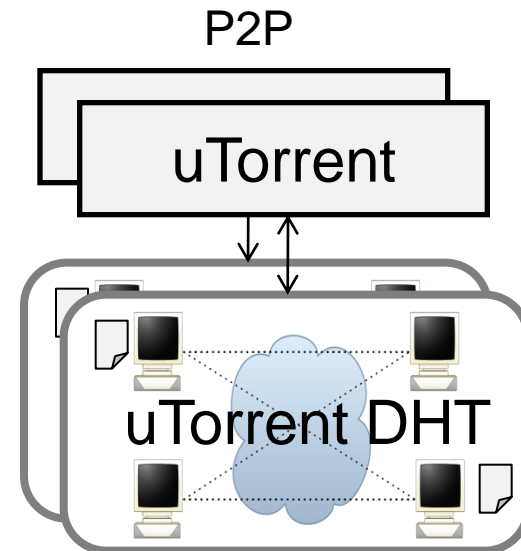
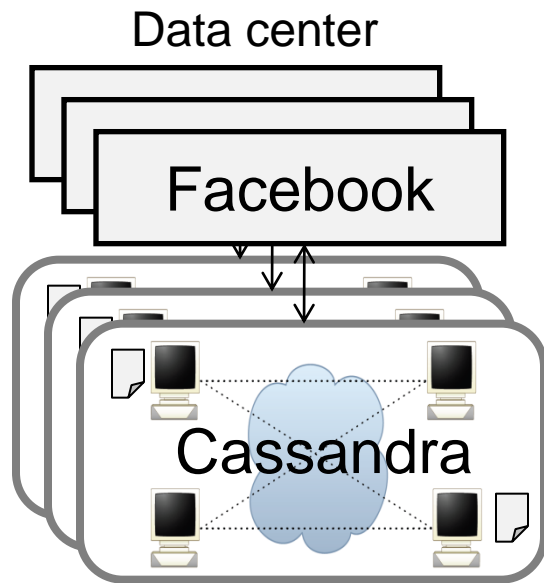
Distributed Key/Value Stores

- A simple **put/get** interface
- Great properties: scalability, availability, reliability
- Increasingly popular both within data centers and in P2P



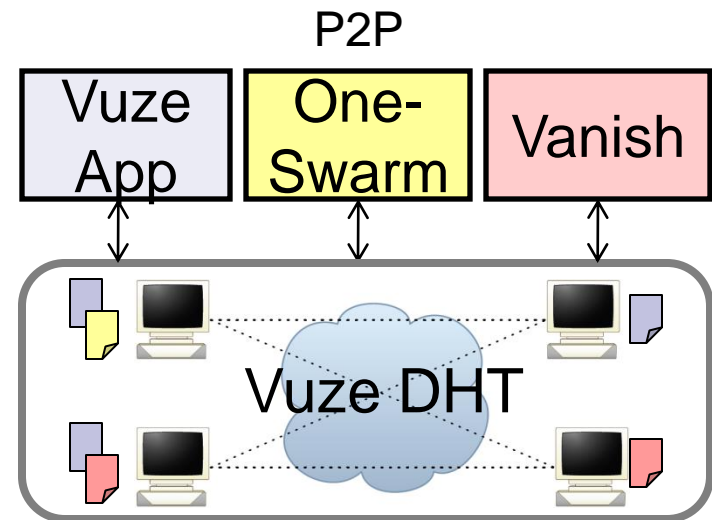
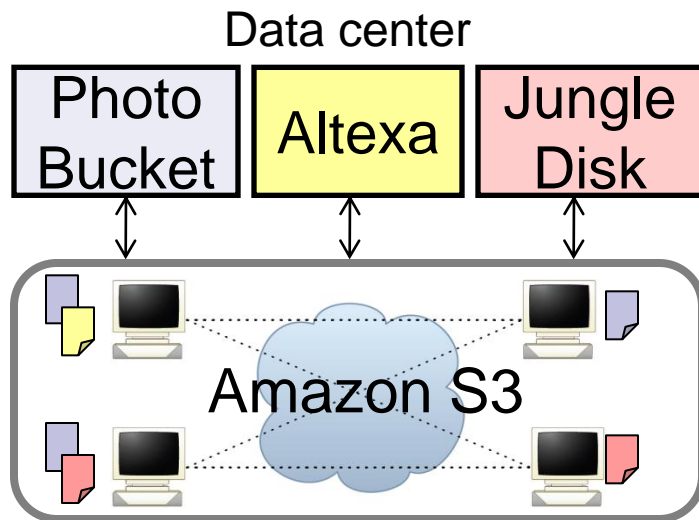
Distributed Key/Value Stores

- A simple **put/get** interface
- Great properties: scalability, availability, reliability
- Increasingly popular both within data centers and in P2P



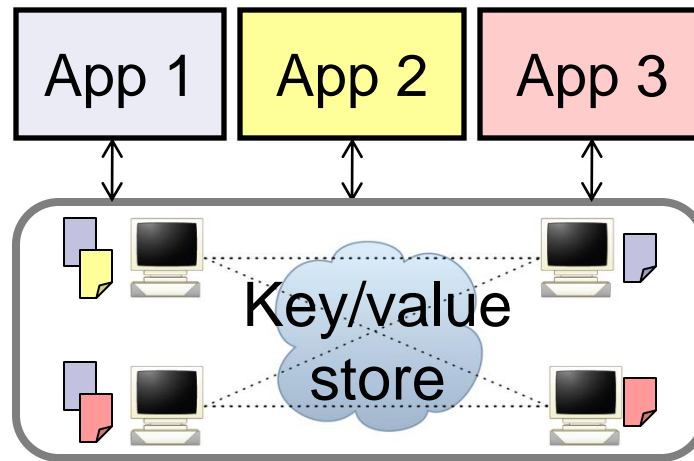
Distributed Key/Value Storage Services

- Increasingly, key/value stores are **shared** by many apps
 - Avoids per-app storage system deployment
- However, building apps atop today's stores is challenging



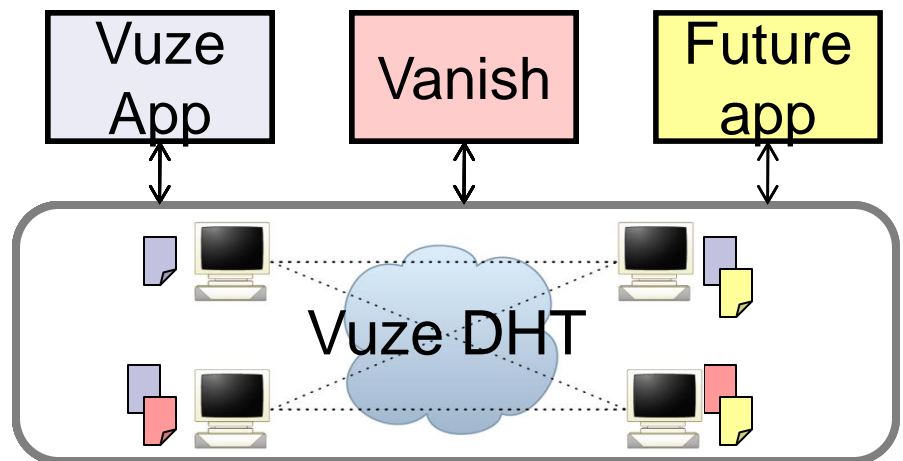
Challenge: Inflexible Key/Value Stores

- Applications have different (even **conflicting**) needs:
 - Availability, security, performance, functionality
- But today's key/value stores are **one-size-fits-all**
- Motivating example: our Vanish experience



Motivating Example: Vanish [USENIX Security '09]

- Vanish is a self-destructing data system built on Vuze
- Vuze problems for Vanish:
 - Fixed 8-hour data timeout
 - Overly aggressive replication, which hurts security
- Changes were simple, but **deploying** them was difficult:
 - Need Vuze engineer
 - Long deployment cycle
 - Hard to evaluate before deployment

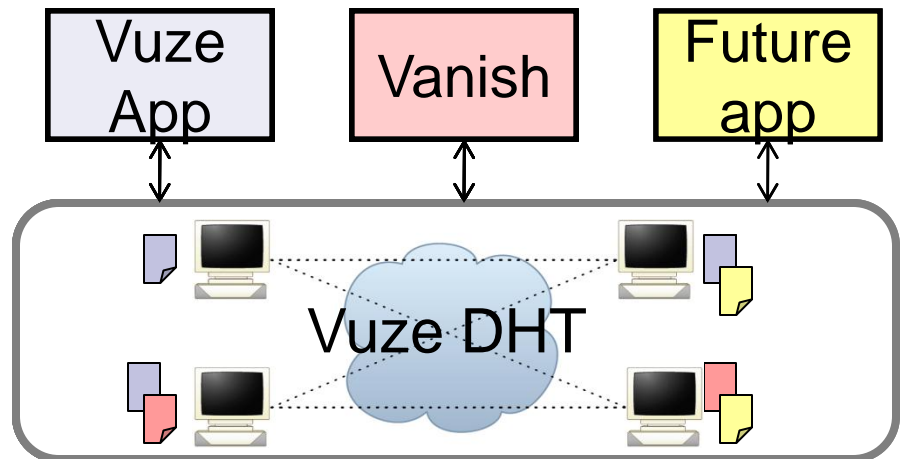


Motivating Example: Vanish [USENIX Security '09]

- Vanish is a self-destructing data system built on Vuze
- Vuze
 - Fixed
 - Over
- Changes were simple, but **deploying** them was difficult:
 - Need Vuze engineer
 - Long deployment cycle
 - Hard to evaluate before deployment

Question:

How can a key/value store support many applications with different needs?



Extensible Key/Value Stores

- Allow apps to customize store's functions
 - Different data lifetimes
 - Different numbers of replicas
 - Different replication intervals
- Allow apps to define **new** functions
 - Tracking popularity: data item counts the number of reads
 - Access logging: data item logs readers' IPs
 - Adapting to context: data item returns different values to different requestors

Design Philosophy

- We want an extensible key/value store
- But we want to keep it **simple!**
 - Allow apps to inject **tiny** code fragments (10s of lines of code)
 - Adding even a tiny amount of programmability into key/value stores can be extremely powerful
- This paper shows how to build extensible **P2P DHTs**
 - We leverage our DHT experience to drive our design

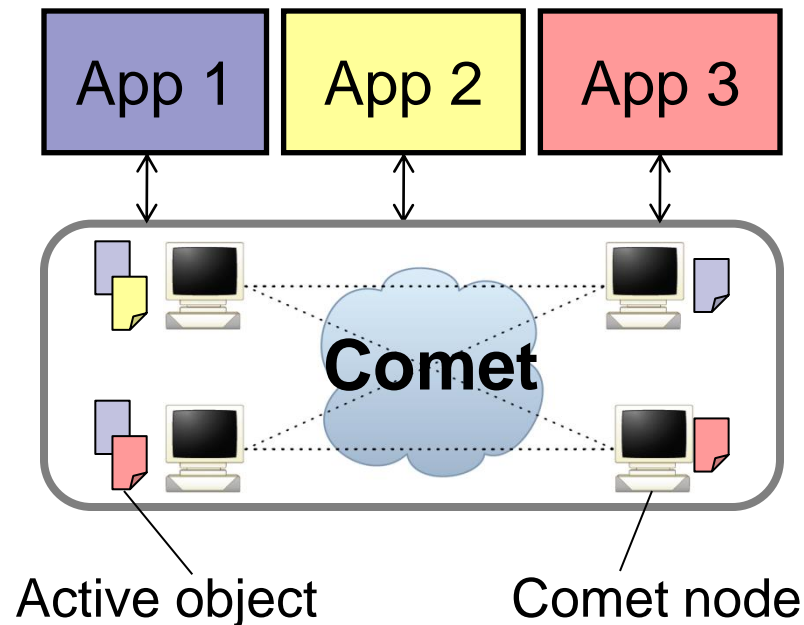


Outline

- Motivation
- **Architecture**
- Applications
- Conclusions

Comet

- DHT that supports application-specific customizations
- Applications store **active objects** instead of passive values
 - Active objects contain **small code snippets** that control their behavior in the DHT





Comet's Goals

- **Flexibility**

- Support a wide variety of small, lightweight customizations

- **Isolation and safety**

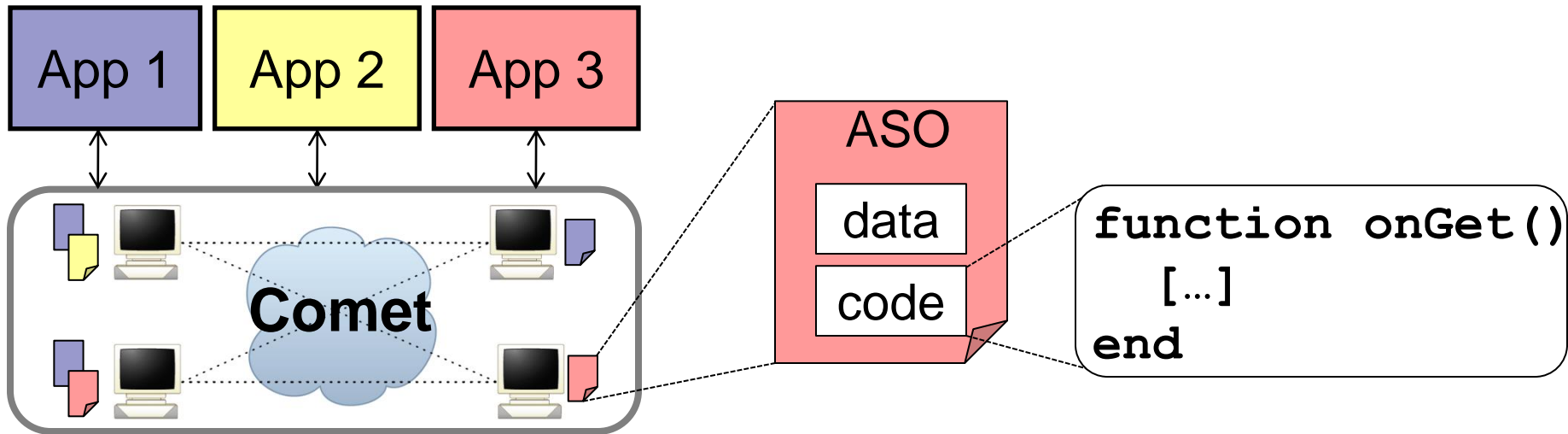
- Limited knowledge, resource consumption, communication

- **Lightweight**

- Low overhead for hosting nodes

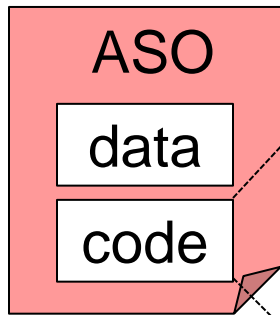
Active Storage Objects (ASOs)

- The ASO consists of data and code
 - The data is the value
 - The code is a set of **handlers** that are called on **put/get**



Simple ASO Example

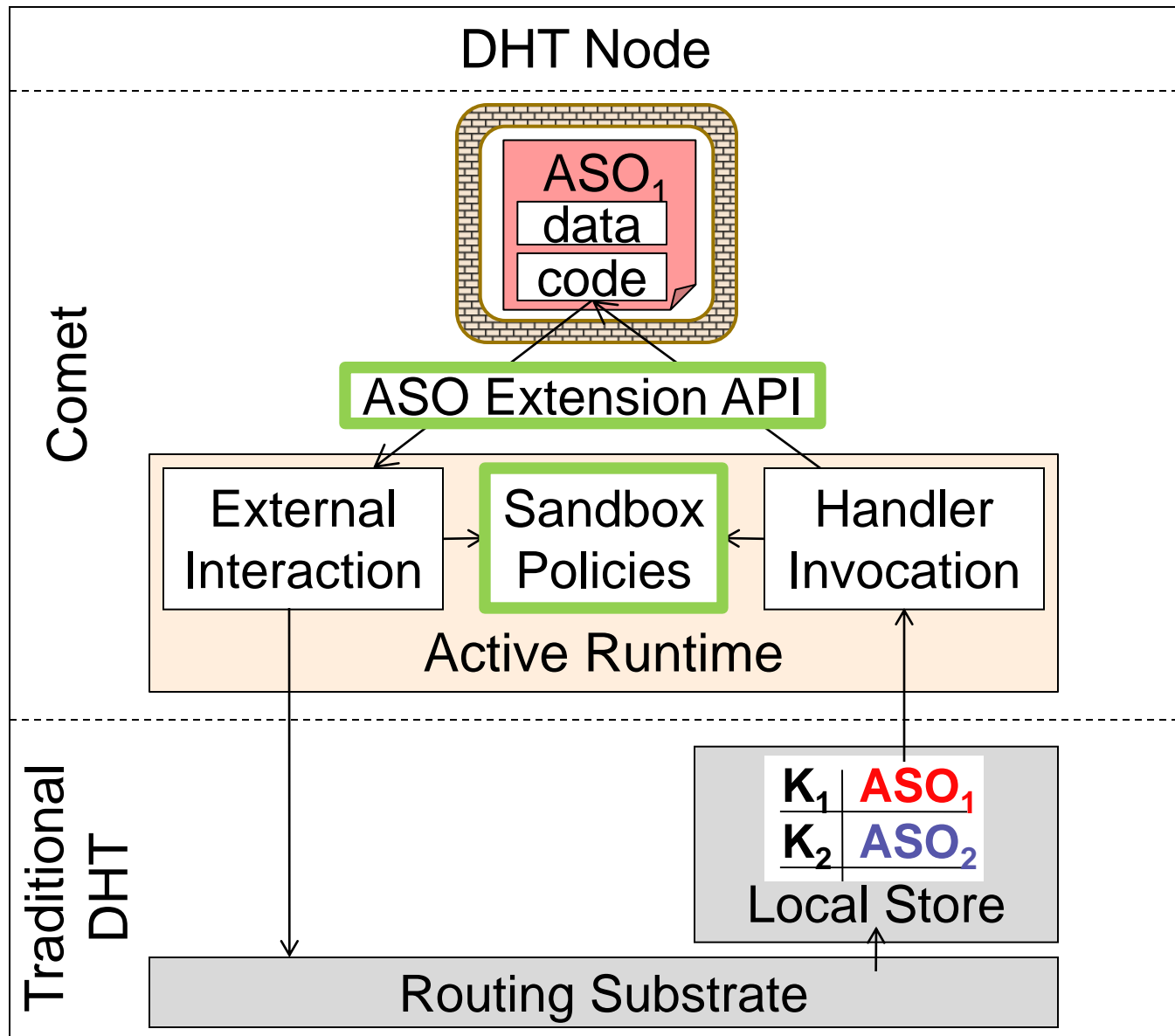
- Each replica keeps track of number of **gets** on an object



```
aso.value = "Hello world!"
aso.getCount = 0
function onGet()
  self.getCount = self.getCount + 1
  return {self.value, self.getCount}
end
```

- The effect is powerful:
 - **Difficult** to track object popularity in today's DHTs
 - **Trivial** to do so in Comet without DHT modifications

Comet Architecture



The ASO Extension API

Applications	Customizations
Vanish	Replication
	Timeout
	One-time values
Adeona	Password access
	Access logging
P2P File Sharing	Smart tracker
	Recursive gets
P2P Twitter	Publish / subscribe
	Hierarchical pub/sub
Measurement	Node lifetimes
	Replica monitoring

The ASO Extension API

Intercept accesses	Periodic Tasks	Host Interaction	DHT Interaction
<code>onPut(<i>caller</i>)</code>	<code>onTimer()</code>	<code>getSystemTime()</code>	<code>get(<i>key</i>, <i>nodes</i>)</code>
<code>onGet(<i>caller</i>)</code>		<code>getNodeIP()</code>	<code>put(<i>key</i>, <i>data</i>, <i>nodes</i>)</code>
<code>onUpdate(<i>caller</i>)</code>		<code>getNodeID()</code>	<code>lookup(<i>key</i>)</code>
		<code>getASOKey()</code>	
		<code>deleteSelf()</code>	

- Small yet **powerful** API for a wide variety of applications
 - We built over a dozen application customizations
- We have explicitly chosen **not** to support:
 - Sending arbitrary messages on the Internet
 - Doing I/O operations
 - Customizing routing ...

The ASO Sandbox

1. Limit ASO's knowledge and access
 - Use a standard language-based sandbox
 - Make the sandbox **as small as possible** (<5,000 LOC)
 - Start with tiny Lua language and remove unneeded functions
2. Limit ASO's resource consumption
 - Limit per-handler bytecode instructions and memory
 - Rate-limit incoming and outgoing ASO requests
3. Restrict ASO's DHT interaction
 - Prevent traffic amplification and DDoS attacks
 - ASOs can talk only to their neighbors, no recursive requests



Comet Prototype

- We built Comet on top of Vuze and Lua
 - We deployed experimental nodes on PlanetLab
- In the future, we hope to deploy at a large scale
 - Vuze engineer is particularly interested in Comet for **debugging** and **experimentation** purposes



Outline

- Motivation
- Architecture
- Applications
- Conclusions

Comet Applications

Applications	Customization	Lines of Code
Vanish	Security-enhanced replication	41
	Flexible timeout	15
	One-time values	15
Adeona	Password-based access	11
	Access logging	22
P2P File Sharing	Smart Bittorrent tracker	43
	Recursive gets*	9
P2P Twitter	Publish/subscribe	14
	Hierarchical pub/sub*	20
Measurement	DHT-internal node lifetimes	41
	Replica monitoring	21

* Require signed ASOs (see paper)



Three Examples

1. Application-specific DHT customization
2. Context-aware storage object
3. Self-monitoring DHT

1. Application-Specific DHT Customization

- Example: customize the replication scheme

```
function aso:selectReplicas(neighbors)
    [...]
end

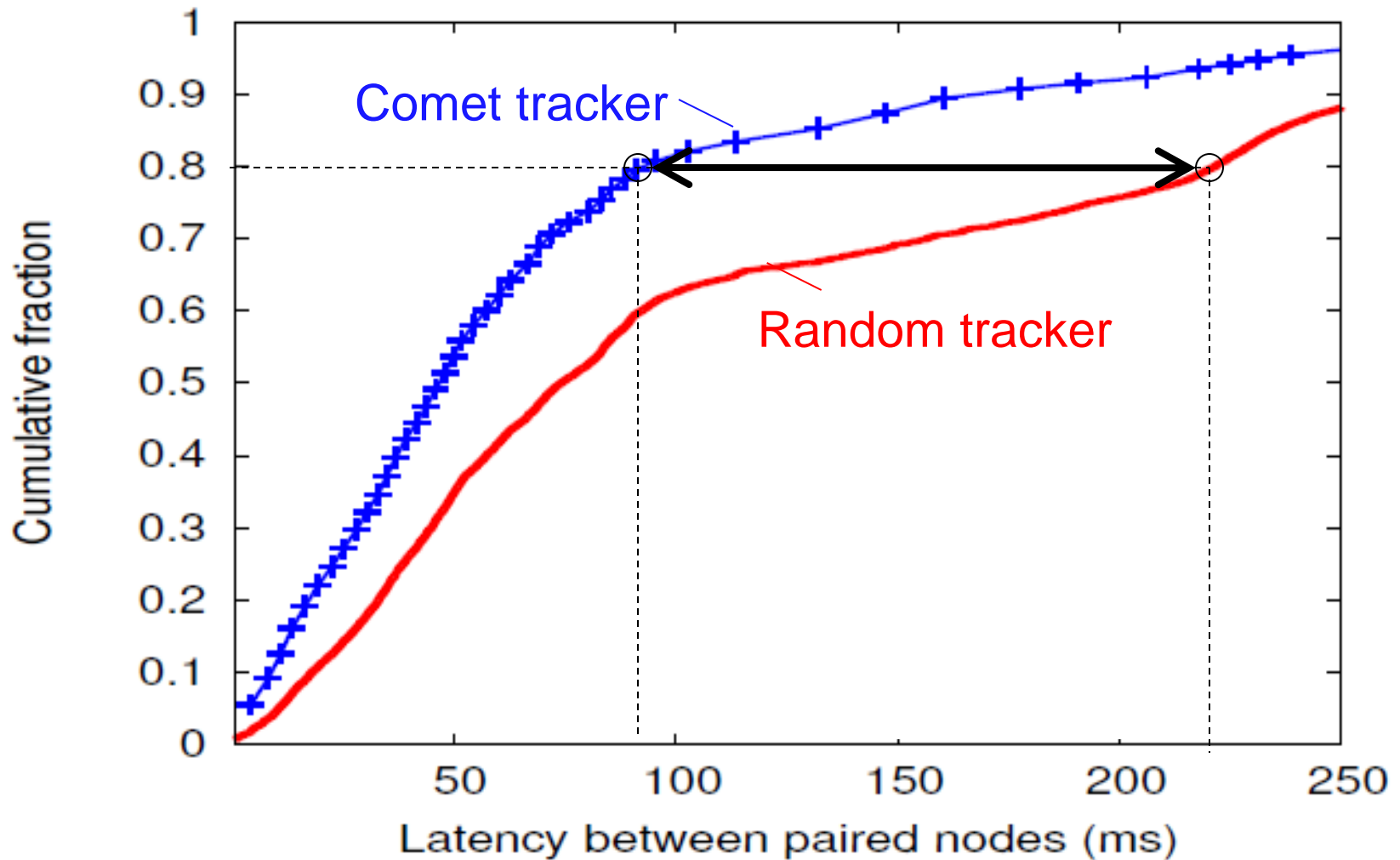
function aso:onTimer()
    neighbors = comet.lookup()
    replicas = self.selectReplicas(neighbors)
    comet.put(self, replicas)
end
```

- We have implemented the Vanish-specific replication
 - Code is 41 lines in Lua

2. Context-Aware Storage Object

- Traditional distributed trackers return a **randomized** subset of the nodes
- Comet: a proximity-based distributed tracker
 - Peers **put** their IPs and **Vivaldi coordinates** at **torrentID**
 - On **get**, the ASO computes and returns the set of **closest peers** to the requestor
- ASO has 37 lines of Lua code

Proximity-Based Distributed Tracker



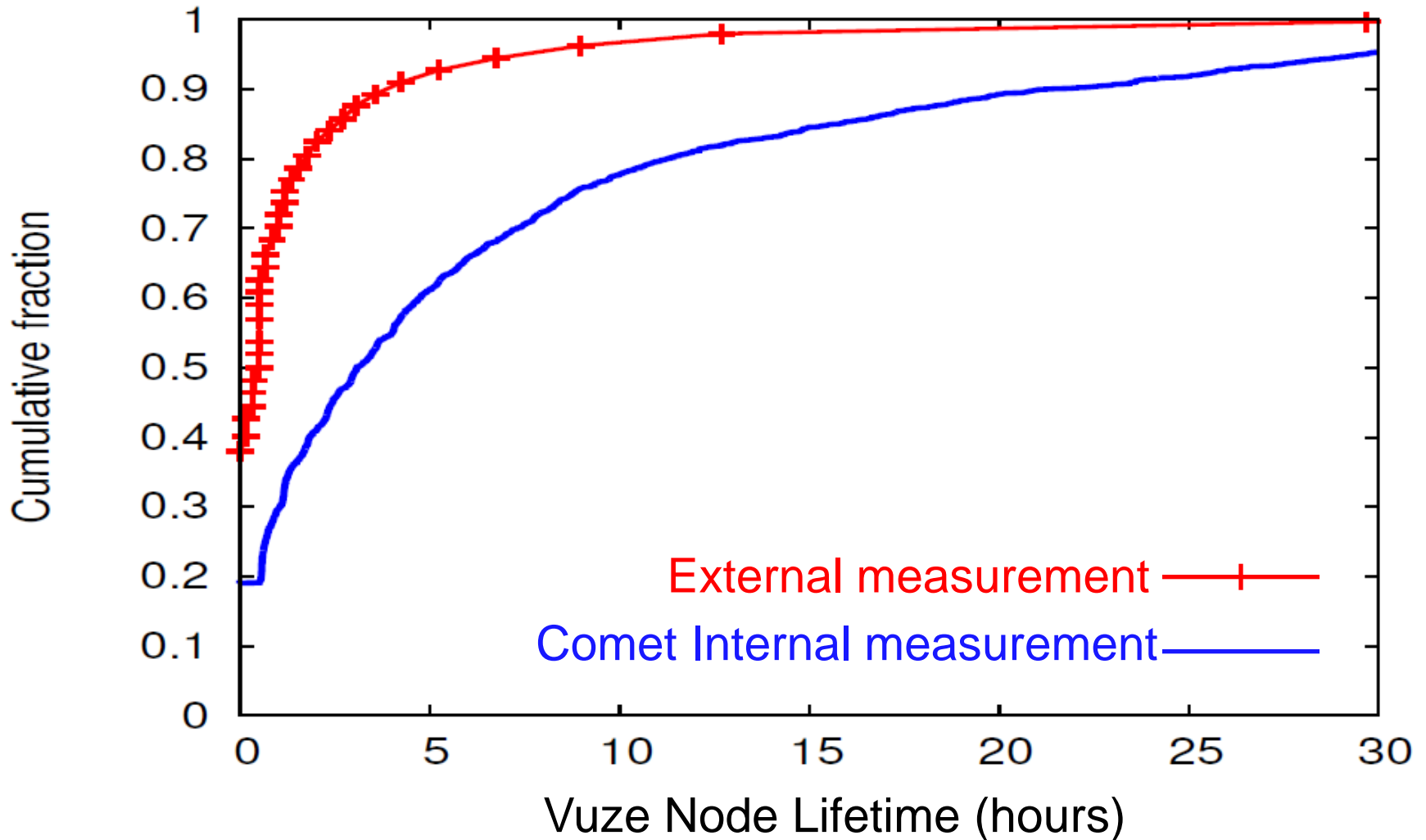
3. Self-Monitoring DHT

- Example: monitor a **remote** node's neighbors
 - Put a monitoring ASO that "pings" its neighbors periodically

```
aso.neighbors = {}  
  
function aso:onTimer()  
  neighbors = comet.lookup()  
  self.neighbors[comet.systemTime()] = neighbors  
end
```

- Useful for **internal** measurements of DHTs
 - Provides additional visibility over **external** measurement (e.g., NAT/firewall traversal)

Example Measurement: Vuze Node Lifetimes





Outline

- Motivation
- Architecture
- Evaluation
- **Conclusions**

Conclusions

- Extensibility allows a shared storage system to support applications with different needs
- Comet is an **extensible DHT** that allows per-application customizations
 - Limited interfaces, language sandboxing, and resource and communication limits
 - Opens DHTs to a new set of stronger applications
- Extensibility is likely useful in data centers (e.g., S3):
 - Assured delete
 - Storage location awareness
 - Logging and forensics
 - Popularity