LOYOLA
UNIVERSITY CHICAGO
Department of Computer Science

GCASR ‘15: Middleware for Collaborative Distributed/Mobile Applications: XMPP or Reactive HTTP?

Brian Gathright, Konstantin Läufer, Azizullah Parsa, and George K. Thiruvathukal

Research Areas

Distributed Systems, Programming Languages, Software Architecture, Software Design Patterns

Objectives

Comparison of communication and coordination middleware implementation choices in collaborative distributed/mobile applications.

Primary criteria: Software Quality Attributes

- Static: Modularity/Testability, Maintainability, Extensibility
- Dynamic: Reliability, Performance, Scalability

Other criteria:

- Availability, quality, documentation and learning curve of Libraries/frameworks for multiple clients: browser/JavaScript, iOS, command line
- Support for likely future requirements (in the context of collaborative apps)

Representative Use Case

Distributed click counter for soccer stadium with multiple entrance doors

- Each client, one per entrance door, sees the same shared state and can increment or decrement it.
- Clients publish events as well as subscribe to them.
- Simplistic but requires substantial domain expertise.

Reactive HTTP

- RESTful web service (HTTP/1.1) implemented in Scala + spray
- JSON as the data (payload) format and response streaming (server-sent events)
- communication end-to-end reactive (asynchronous/nonblocking)
- scredis reactive/nonblocking driver for the Redis key-value store

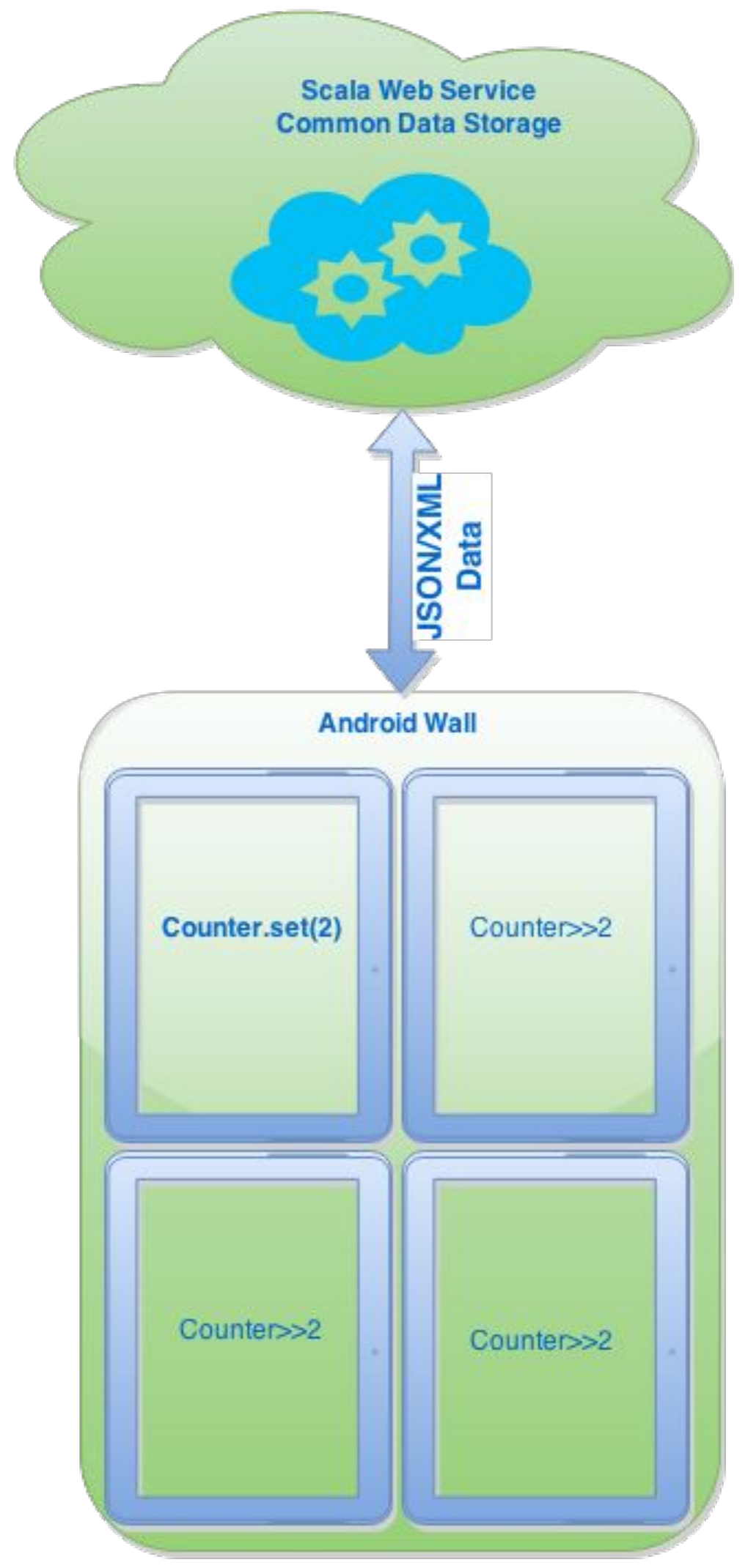
https://github.com/LoyolaChicagoCode/clickcounter-spray-scala
http://laufer-dev.cs.luc.edu:8080 <- *running instance*

```
pathPrefix("counters" / Segment) { id =>
 …
 path(”increment”) {
  post { // whole computation runs in future!
   onComplete(repository.update(id, _.v + 1)) {
    case Success(Some(true)) =>
     complete(StatusCodes.NoContent)
    case Success(Some(false)) =>
     complete(StatusCodes.Conflict, errorMsg)
   }
```

- Android client using RxScala and homegrown server-sent event client

https://github.com/LoyolaChicagoCode/clickcounter-android-rxscala-http <- *download from here*

```
/** The observable for the server-sent events. */
lazy val eventSource =
 HttpEventSourceObservable.getObservable[(Int,
ModelState)]
 (url + "/counters/" + counterId + "/stream”)
lazy val postObserver = new HttpPostObserver
 (serviceUrl + "/counters/" + counterId + "/”)
```



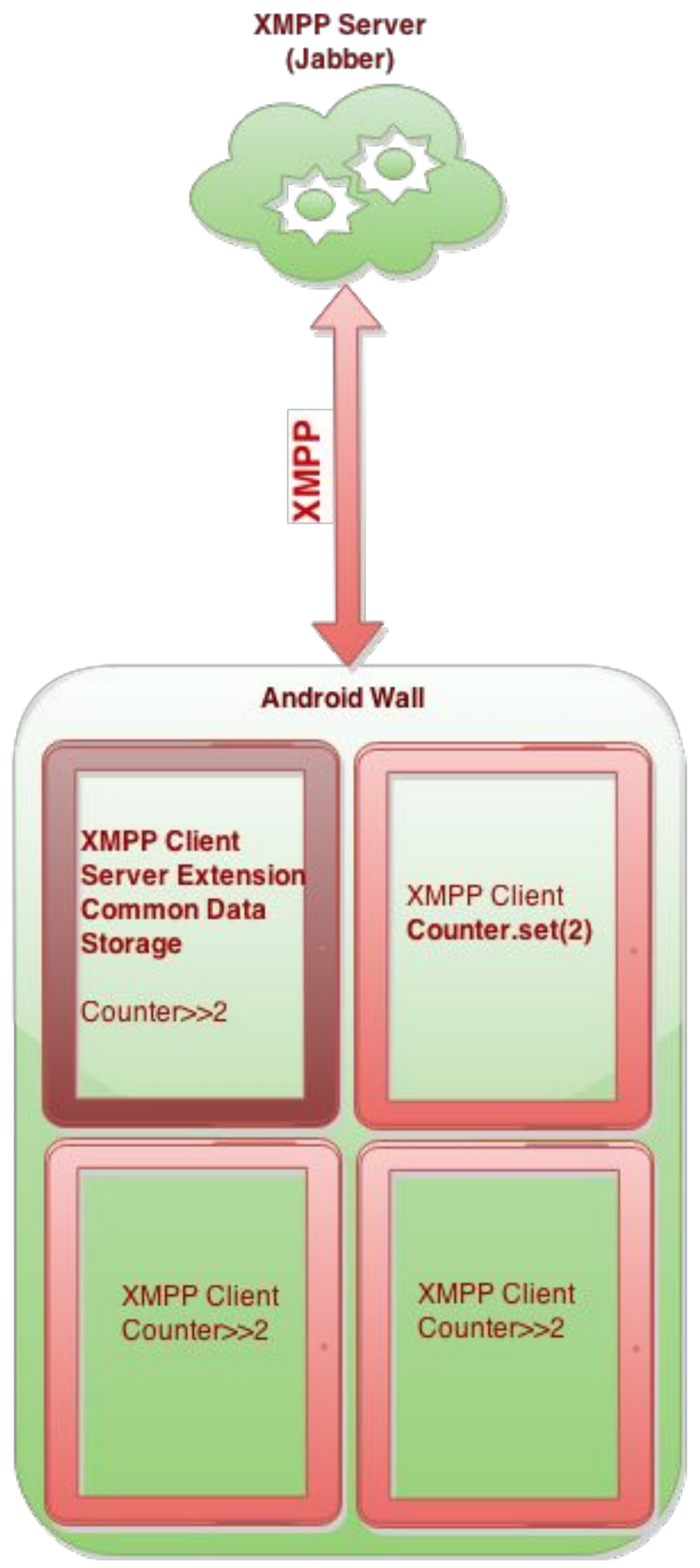
XMPP (WIP)

- Extensible Messaging and Presence Protocol (XMPP): an open, extensible protocol for device discovery and XML data interchange in near real-time
- initially designed for inst. messaging
- handles broad range of middleware needs, rich plugin ecosystem
- hosted or locally installed Jabber server such as ejabberd
- Android client written in Scala and using Smack, an open-source Java client library for XMPP (work in progress)

https://github.com/briangathright/hello-xmpp-app <- *WIP*

```
muc.addMessageListener(new MessageListener {
 override def processMessage(m: Message) = {
  if (message.getBody != null) {
   val from = XmppStringUtils.
    parseResource(message.getFrom)
   runOnUiThread {
    updateUi(message.getBody)
```

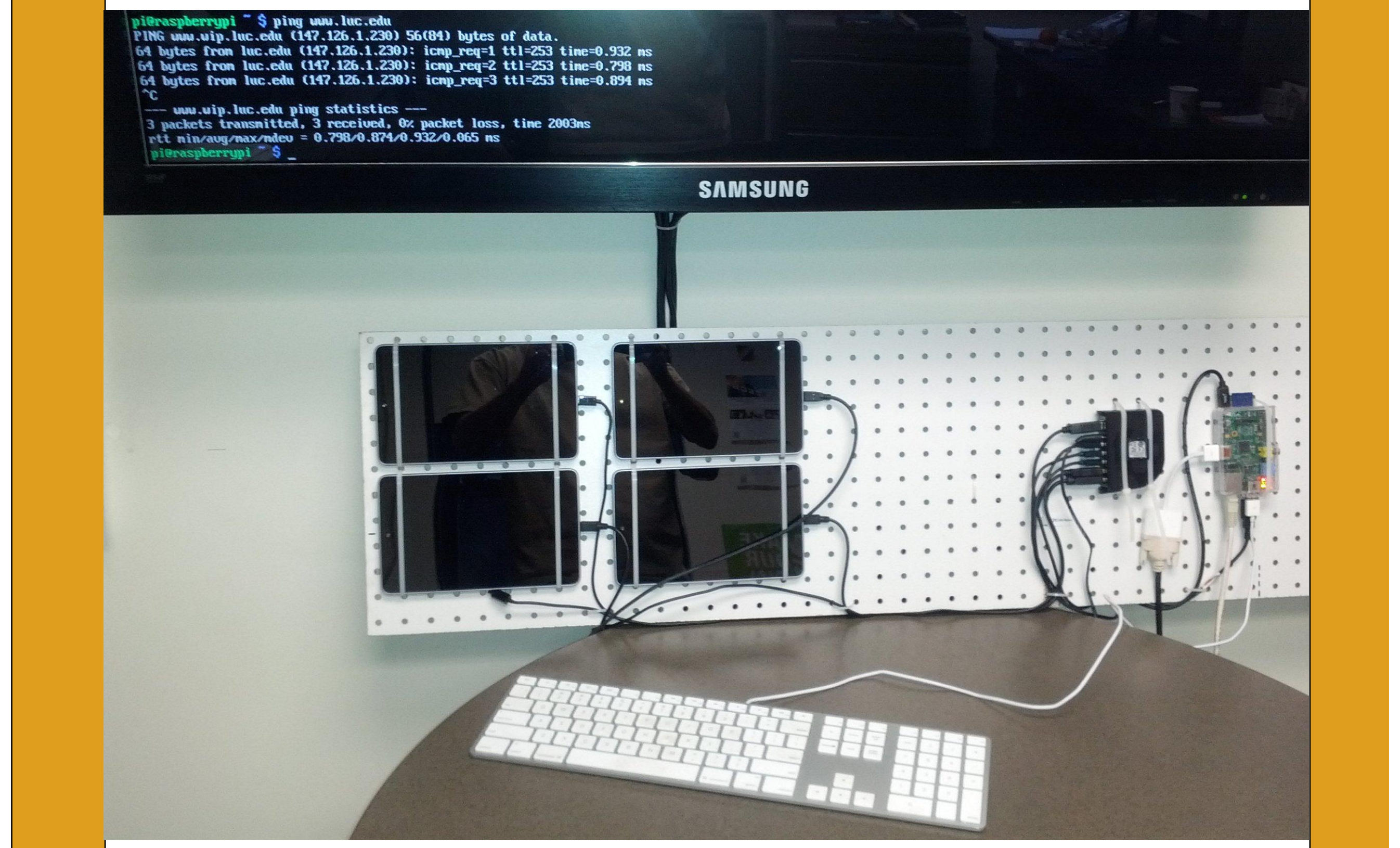
- requires additional, highly available headless XMPP client to maintain counter values in Redis key-value store



Context

This class of use cases has arisen in the ongoing Android Wall project, where we combine multiple low-cost, commodity tablet devices to a cluster addressing a “trilogy” of concerns: *visualization, sensing, and computation.*

4x N7 Android Tablet Wall prototype with Raspberry Pi as server



Other Approaches Considered

- WiFi Direct: unlikely to scale beyond three or four peers
- Client connects directly to storage: problem is driver availability for different clients
- Client connects to message queue?

Preliminary findings

- HTTP-based approach very scalable and extensible to different clients
- Many libraries have heavyweight dependencies for use in Android client
- Support for SSE still emerging
- XMPP-based approach has higher learning curve but likelier to support future reqs.