Beyond REST?

Building data services with XMPP PubSub

Evan Henshaw-Plath, ENTP.com

Kellan Elliott-McCrea, Flickr.com

We build websites.

we're not XMPP experts, specialty is building really large social sites, rich APIs, Web 2.0 stuff!
we're Jabber outsider, and this talk is about why we're excited about XMPP.

No XEP overload

no xep overload.
and we aren’t here to talk about instant messaging, or chat either.

Beyond REST, the game has changed.

we're huge fans of RESTful APIs. REST won. Its great. We love it.
but recently the game has changed. we're building bigger websites, the latency is lower, the social network effects are huge, and more.

REST is Newtonian physics.

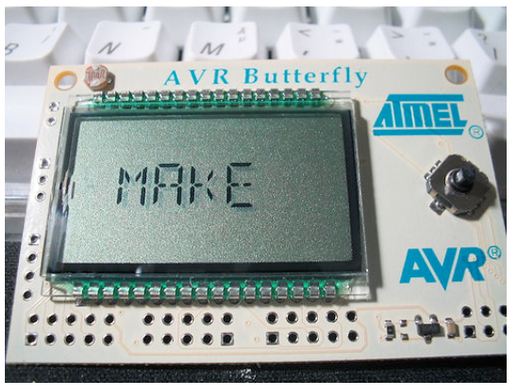
Its like REST is Newtonian physics. For every day problems, its good enough. It makes sense. Its coherent, and its well understood. But it breaks down at scale. Its breaks down when you're talking about really small things, and really fast things, and really really huge things, it doesn't explain quarks and quasars.

XMPP Data Services

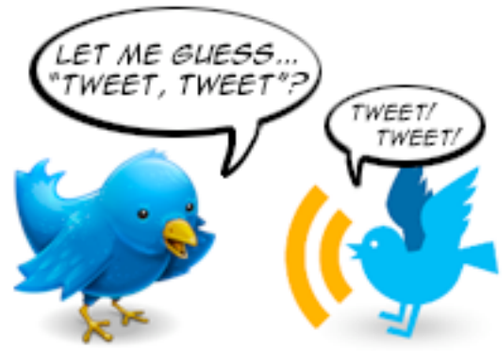
Quantum Mechanics
&
General Relativity

newtonian physics, vs quantum mechanics and relativity.

small and infrequent



fast and furious



flickr LOVES YOU™

facebook

attention streams, twitter tweets, flickr uploads, even sensors on robots.
data streams are everywhere, and cross pollinating between streams.

Data streams

current standard for data streams on the internet is RSS! its chunky streaming protocol.
XMPP PubSub is our solution for those quantum and relativity edge cases.

RPC too.

we won't be talking much about RPC style APIs over XMPP. Vertbra, Engine Yard's cloud automation framework is a great example, but we think data streams are today's problem, and the most bang for your buck. But there is stuff out there, and some of it will be open source soon.

The failure of feeds

feeds are awesome. clearly great. when RSS was young it was cultural that you never ever, ever crawled a feed more then once an hour. once the rate picked up, etags and last-modified made it work, as long as it was blog posts, and podcasts. but they
but then we started putting *new* types of data in feeds.

The success of feeds

high volume, and frequent, change logs, presence, activity logs, attention, click streams, mapping and geo data, weather emergency response systems. hard real time data.

Flickr & Friendfeed

friendfeed is a popular new site, aggregates your data from all over, your flickr photos, your twitter tweets, your del.icio.us links, your youtube favorites into one place. to do that it crawls RSS feeds.

July 21, 2008

on july 21st, 2008, they friendfeed crawled flickr 2.9 million times.
to get the latest photos of 45,754 users
of which 6,721 of that 45,754 visited Flickr in that 24 hour period, and could have
potentially uploaded a photo.

July 21, 2008

2,975,981

on july 21st, 2008, they friendfeed crawled flickr 2.9 million times.
to get the latest photos of 45,754 users
of which 6,721 of that 45,754 visited Flickr in that 24 hour period, and could have
potentially uploaded a photo.

July 21, 2008

2,975,981

45,754

on july 21st, 2008, they friendfeed crawled flickr 2.9 million times.
to get the latest photos of 45,754 users
of which 6,721 of that 45,754 visited Flickr in that 24 hour period, and could have
potentially uploaded a photo.

July 21, 2008

2,975,981

45,754

6,721

on july 21st, 2008, they friendfeed crawled flickr 2.9 million times.
to get the latest photos of 45,754 users
of which 6,721 of that 45,754 visited Flickr in that 24 hour period, and could have
potentially uploaded a photo.

Not ideal.

3million requests, maybe 6000 updates. but its worse. if any of those 6000 people uploaded *lots* of photos, friendfeed didn't see that either, because our buffer size on RSS is 20 items. anything more is lost.

1. We're spending a huge amount of resources for a really small number of users, and a single site. Imagining scaling this.
2. Our transport is so noisy we're missing out, and losing a lot of data
3. For what is really a small trickle of data.

We thought about calculating kilowatt hours, and dollars spent on electricity. But we didn't get to it.

Not Friendfeed's fault.

so we're all over here contributing to the heat death of the universe. but its not friendfeeds fault. they're doing everything exactly right, using etags, conditional gets, with the tools that are currently available.

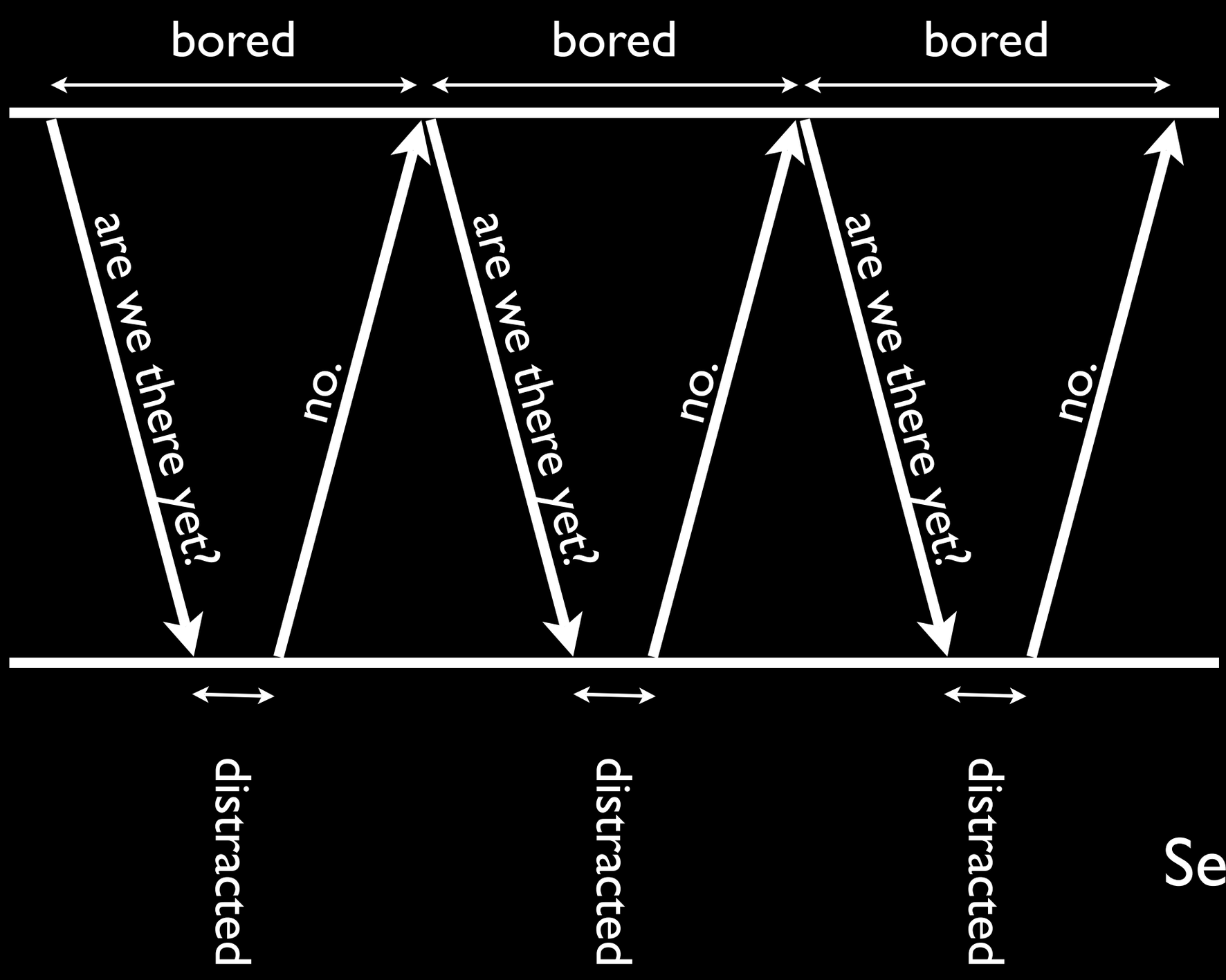
Not going to scale.

this is a small number of users, and a single site. imagine millions of users, across a federated social networks.

Polling sucks.

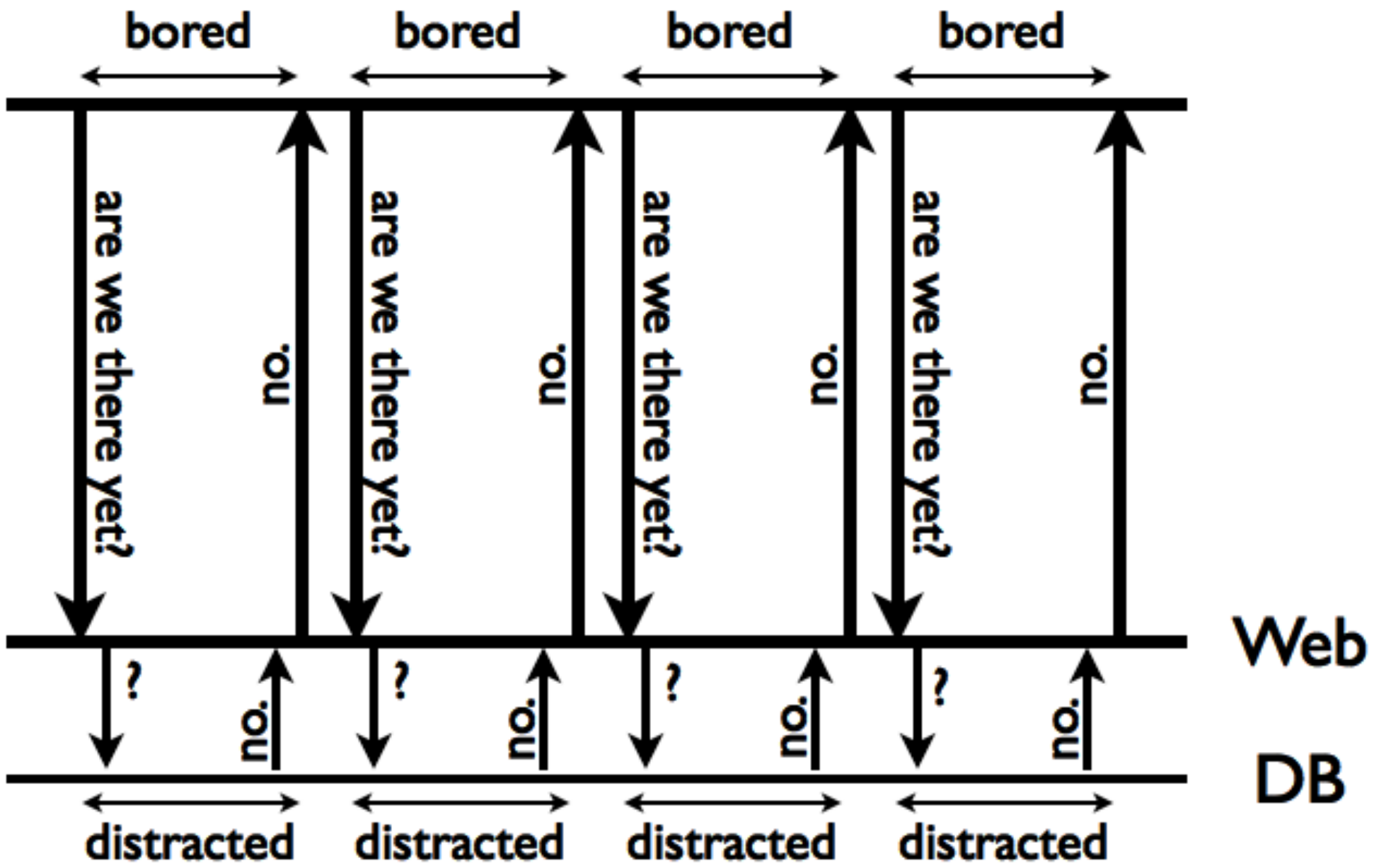
to inevitable conclusion. polling sucks.

Client



this is the way the web streams updates. this is what polling looks like. ideally. long and boring car trip. Consumer is the kid in the back seat. "Are we there yet? Are we there yet?"

Client



And that was ideal. Under real world circumstances its even worse. And both the consumer and the server are burning cycles waiting.

Let's be clear

Message passing means many things. We're talking specifically about:

* asynchronous, but real-time communication
* non-blocking event loop driven processing
* share nothing architectures

Web meet the event loop!

The Switch

Response/Response
Send/Recieve

Message Passing!

a message system lets get out of that constant polling nightmare. we register interest, go about our business, and when an event happens, we're notified. (revolutionary new 20 year old technology)

Hijacking XMPP

how are we going to do web scale message passing? we're hijacking XMPP.

Why XMPP?

- persistent connections

this is so weird if you're from the web world.

Why XMPP?

- stateful

you don’t have to handshake on ever message.

Why XMPP?

- designed to be an event stream protocol

it was *built* to do this shit. not like HTTP

Why XMPP?

- natively federated and asynchronous

i can haz routing! server to server was assumed not a hack we added in

Why XMPP?

- identity, security, and presence built in.

always nice to have, and you're going to have to build it if you're building social software.

Why XMPP?

- Jabber servers are built to do this stuff!

Handling 80k concurrent connections, with apache that's like doing 6.4 billion page views on a single box per day.

Why XMPP?

- persistent connections
- stateful
- designed to be an event stream protocol
- natively federated and asynchronous
- identity, security, and presence built in.
- Jabber servers are built to do this stuff.

it's just xml

```
<message from='bigbrother@megacorp.gov/work'
  to='winston@example.net'>
  <body>WAR IS PEACE
  FREEDOM IS SLAVERY
  IGNORANCE IS STRENGTH</body>
</message>
```

jids. they look like email addresses. they work like email address to.

it’s just xml

```
<message from='winston@example.net'
  to='bigbrother@megacorp.gov/work'>
  <body>double plus ungood</body>
</message>
```

PubSub?

just means publish subscribe. its data streams vs chat. this is the message passing we were talking about. “let me know when something changes, kthxbye.”

let me know when something changes, kthxbye?

XMPP PubSub

you might have heard of it? its nothing special, just some conventions for XMPP data streams.

xmpp pubsub stanzas

```
<iq type='set'
    from='winston@homeland.gov/blogbot'
    to='pubsub.24hournews.com'
    id='pub1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <publish node='/news/inspiration/quotes'>
      <item>
        <entry xmlns='http://www.w3.org/2005/Atom'>
          <title>the war on terrorism</title>
          <summary>
            WAR IS PEACE
            FREEDOM IS SLAVERY
            IGNORANCE IS STRENGTH
          </summary>
          <link rel='alternate' type='text/html'
                href='http://homeland.gov/news'/>
          <id>tag:homeland.gov,1984:entry-32397</id>
          <published>1984-12-13T18:30:02Z</published>
          <updated>1984-12-13T18:30:02Z</updated>
        </entry>
      </item>
    </publish>
  </pubsub>
</iq>
```

This is what an xmpp pubsub stanza looks like

the iq - addressing

```
<iq type='set'
    from='winston@homeland.gov/blogbot'
    to='pubsub.24hournews.com'
    id='pub1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <publish node='/news/inspiration/quotes'>
      <item>
        <entry xmlns='http://www.w3.org/2005/Atom'>
          <title>the war on terrorism</title>
          <summary>
            WAR IS PEACE
            FREEDOM IS SLAVERY
            IGNORANCE IS STRENGTH
          </summary>
          <link rel='alternate' type='text/html'
                href='http://homeland.gov/news'/>
          <id>tag:homeland.gov,1984:entry-32397</id>
          <published>1984-12-13T18:30:02Z</published>
          <updated>1984-12-13T18:30:02Z</updated>
        </entry>
      </item>
    </publish>
  </pubsub>
</iq>
```

First we have the iq, it tells us who published the stanza and where the message should be delivered.

using pubsub

```
<iq type='set'
    from='winston@homeland.gov/blogbot'
    to='pubsub.24hournews.com'
    id='pub1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <publish node='/news/inspiration/quotes'>
      <item>
        <entry xmlns='http://www.w3.org/2005/Atom'>
          <title>the war on terrorism</title>
          <summary>
            WAR IS PEACE
            FREEDOM IS SLAVERY
            IGNORANCE IS STRENGTH
          </summary>
          <link rel='alternate' type='text/html'
                href='http://homeland.gov/news'/>
          <id>tag:homeland.gov,1984:entry-32397</id>
          <published>1984-12-13T18:30:02Z</published>
          <updated>1984-12-13T18:30:02Z</updated>
        </entry>
      </item>
    </publish>
  </pubsub>
</iq>
```

Then we have the pubsub element which contains all the information we want to pass on to the subscriber

the node to publish to

```
<iq type='set'
    from='winston@homeland.gov/blogbot'
    to='pubsub.24hournews.com'
    id='pub1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <publish node='/news/inspiration/quotes'>
      <item>
        <entry xmlns='http://www.w3.org/2005/Atom'>
          <title>the war on terrorism</title>
          <summary>
            WAR IS PEACE
            FREEDOM IS SLAVERY
            IGNORANCE IS STRENGTH
          </summary>
          <link rel='alternate' type='text/html'
               href='http://homeland.gov/news'/>
          <id>tag:homeland.gov,1984:entry-32397</id>
          <published>1984-12-13T18:30:02Z</published>
          <updated>1984-12-13T18:30:02Z</updated>
        </entry>
      </item>
    </publish>
  </pubsub>
</iq>
```

The node, the thing you are publishing to / subscribing to. It’s an opaque identifier, but we recommend you use the same uri path as your parallel REST api for the same content

an atom payload

```
<iq type='set'
    from='winston@homeland.gov/blogbot'
    to='pubsub.24hournews.com'
    id='pub1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <publish node='/news/inspiration/quotes'>
      <item>
        <entry xmlns='http://www.w3.org/2005/Atom'>
          <title>the war on terrorism</title>
          <summary>
            WAR IS PEACE
            FREEDOM IS SLAVERY
            IGNORANCE IS STRENGTH
          </summary>
          <link rel='alternate' type='text/html'
                href='http://homeland.gov/news'/>
          <id>tag:homeland.gov,1984:entry-32397</id>
          <published>1984-12-13T18:30:02Z</published>
          <updated>1984-12-13T18:30:02Z</updated>
        </entry>
      </item>
    </publish>
  </pubsub>
</iq>
```

Then the playload is just an Atom item.

arbitrary payloads

```
<iq type='set'
    from='winston@homeland.gov/blogbot'
    to='pubsub.24hournews.com'
    id='pub1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <publish node='/random/rails/controller.xml'>
      <nameofmodelclass>
        <field1>field 1's value</field1>
        <field2>field 2's value</field2>
        <onetomanyfield1 href='http://example.comm/url1' />
        <onetomanyfield1 href='http://example.comm/url2' />
      </nameofmodelclass>
    </publish>
  </pubsub>
</iq>
```

The payload you are publishing is arbitrary, Using atom is good, but for many apps you can include custom information you are passing around, as long as your clients grok it.

some code!

```
while true
  event = queue.get_next_event()

  #loop
  Subscriptions.find_by_node(:all,
        event.pubsub_nodes  ).each do |subscriber|

    #send new message
    subscriber.send_xmpp_message(event.to_xmpp)

  end
end
```

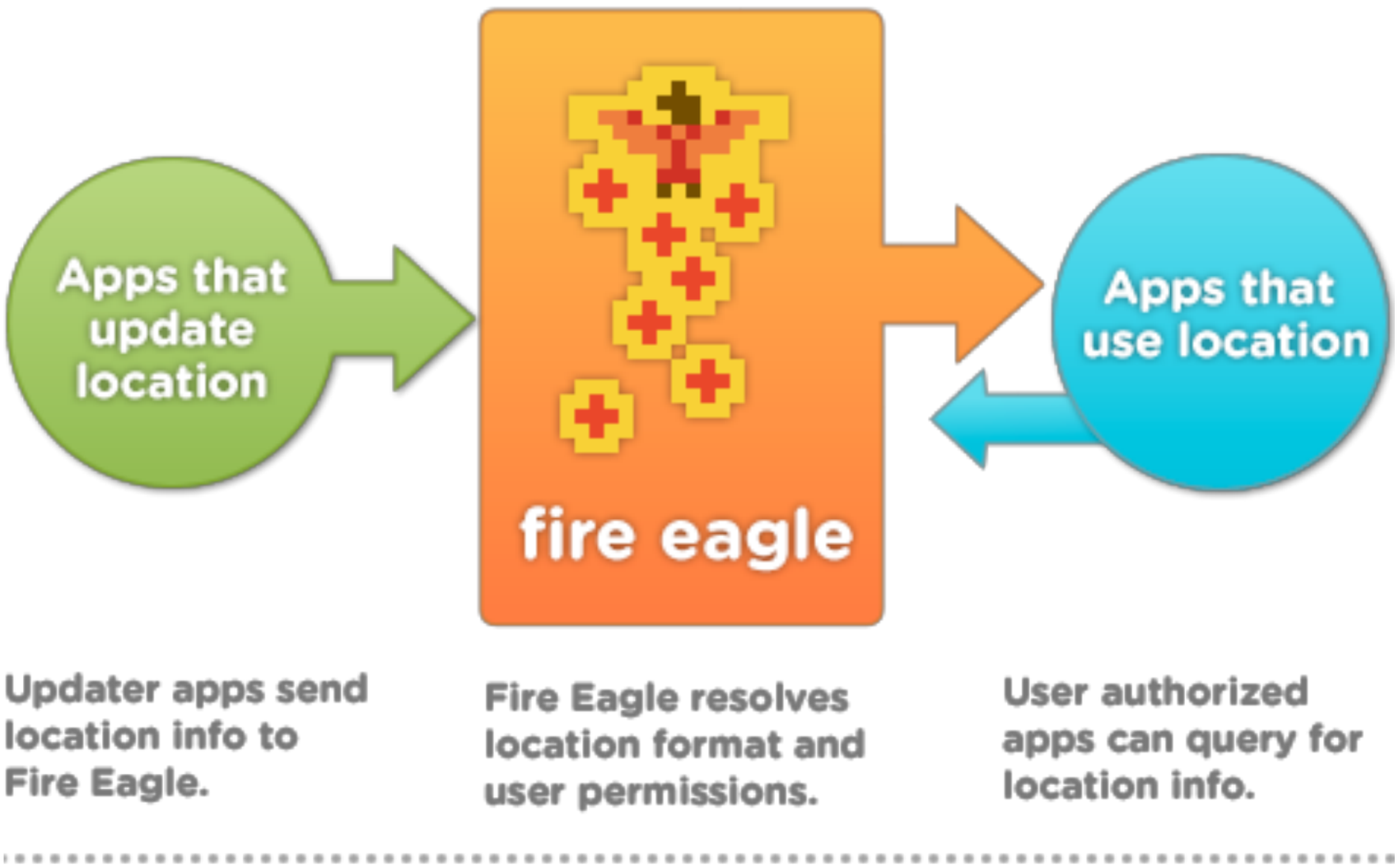
how to handle pubsub. recieve, look up, send, receive, look up, send

Applied XMPP

so now lets look at what we, the pragmatic, api developing website building hackers are excited about it, what we've been doing with it.

Case study #1: FireEagle

FireEagle



Fire Eagle is a location broker, a user's location goes in, and other applications can query to get it. Uses OAuth for private, signed and encrypted for every user. Feed scaling tricks don't help.

pathological use pattern

updates tend to be infrequent, but timeliness is very important to consumer. consumers are wanting to poll use every second for something that might change once a week. (they poll us every 11 seconds, because they get blocked at every 10 seconds) every 10 seconds, per user. even in private beta we're noticing the effects of polling architecture.
alternates look like webhooks - webhooks try to do messaging but pay the cost of statelessness, socket tear down, improperly tuned servers, etc.

paginating a stream

explain the problem with pagination of recent updates, response buffer of 20 items at a time, rate limited to one request every 10 seconds, at some point you can’t keep up (2 updates per second)

Case study #2: Flickr

we've already establish why a XMPP data services might be useful for flickr in the friendfeed example. but what would it look like?

firehose?

one example is the Twitter “firehose” approach. an XMPP feed of every photo uploaded. at peak, thats 60 photos per second, or roughly 10 times as many per second as people are born on earth.

a firehose.

- 60/sec
- Atom enriched XMPP packets, ~2k each
- public photos only
- xmpp://photos@flickr.com

an XMPP feed of every photo uploaded. at peak, thats 60 photos per second, or roughly 10 times as many per second as people are born on earth.

a firehose.

2*60 = ~ 1 megabit

whats the bandwidth look like? thats really interesting. that means you can build a friendfeed style aggregator for *ALL* flickr uploads (not just 45k people) on a single box, hosted on your DSL line. this is what is going to make real, diverse federated social networks possible.

granularity is better

- privacy concerns, even for public data
- jabber servers work better with smaller rosters
- more small data services easier to scale sideways.
- 1Mbps * 1000 developers (<1%) = 1 Gbps (can compress up to 70%)

except we aren't going to give it to you.
even for public data, context is important
also this isn't what the Jabber servers were written to do. remember we're hijacking this stuff.
more smaller feeds are easier to shard, and split across a cluster.
even for flickr, we notice a couple of extra gigabits a second.
though it turns out zlib is really good at compressing XML

jid: userid@flickr.com

one data feed per user, allow recompsing. protected by oauth-over-xmpp

prototyped geotagged photos data service
jid: geotagged@research01

```
<iq type='set'
    from='winston@homeland.gov/blogbot'
    to='pubsub.24hournews.com'
    id='pub1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <publish node='/news/inspiration/quotes'>
      <item>
       <entry>
         <title>Atom-Powered Robots Run Amok</title>
         <link href="http://example.org/2003/12/13/atom03"/>
         <id>urn:uuid:1225c695-cfb8-4ebb-aaaa-80da344efa6a</id>
         <updated>2003-12-13T18:30:02Z</updated>
         <summary>Some text.</summary>
         <geo:point>45.256 -71.92</geo:point>
       </entry>
      </item>
    </publish>
  </pubsub>
</iq>
```

Building an XMPP data service in 4 easy steps.

1. Hello, world?

1. Hello, world?

Get a client library

- Jabber::Simple for Ruby
- XMPPHP for PHP
- Smack for Java
- lots more in *every* language

1. Hello, world?

Get a Jabber account

- Jabber.org
- GTalk
- you don’t need your own

server for “Hello word”

1. Hello, world?

```
# Send a message to a friend, asking for authorization if necessary:
  im = Jabber::Simple.new("user@example.com", "password")
  im.deliver("friend@example.com", "Hey there friend!")
```

2. Install a Jabber server

- ejabberd 2.x
- djabberd
- OpenFire, WildFire
- Tigase

lots great open source alternative
ejabberd works for me, is popular. a bit memory hungry w/ lots of connections. its written in Erlang, which I don't speak. djabberd is written in Perl, used by 6A, and Jaiku. OpenFire and WildFire are good alternatives. Tigase is an interesting new Java with pluggable service model.

2. Install a Jabber server

lots of features you aren't going to use

all of these servers are going to come w/ features you don't want. like user registration.
you're going to turn of most of them. remember we're hijacking this stuff.

3. Build a component.

- "Hello, world?" won't scale
- Use a component. XEP-0114
- Component persistent, talks over a local socket.
- YAGNI: rosters, presence, etc.
- Load balance between components built-in

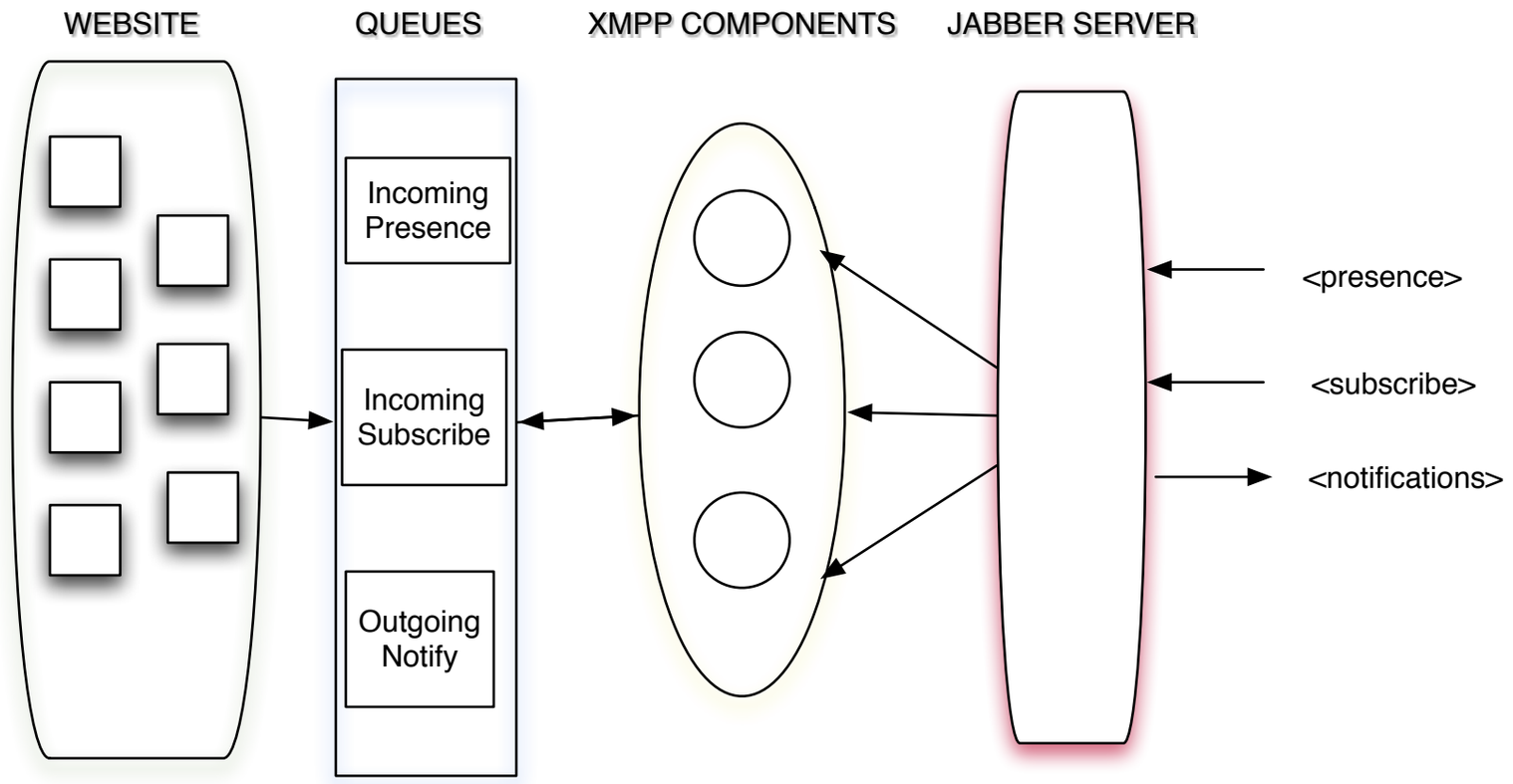
our example of just connecting as a client is great. but if you're just connecting as a client then you can't scale sideways, and presence packets are going to drown you.
components are persistent daemons that talk on a local socket.

3. Build a component.

```
while true
  event = queue.get_next_event()

  #loop
  Subscriptions.find_by_node(:all,
        event.pubsub_nodes  ).each do |subscriber|

    #send new message
    subscriber.send_xmpp_message(event.to_xmpp)

  end
end
```

that code snipper we showed you earlier. that was a component

4. Architecture



rabbitmq, beanstalk

iPhone might use it?

```
<message from="pubsub.aosnotify.mac.com" to="samnsofi@aosnotify.mac.com/
5e60ad2e47da9fca36de59244f25c9b1cd8e0cb8" id="/protected/com/apple/
mobileme/samnsofi/mail/Inbox__samnsofi@aosnotify.mac.com__3gK4m">
<event xmlns="http://jabber.org/protocol/pubsub#event">
<items node="/protected/com/apple/mobileme/samnsofi/mail/Inbox">
<item id="5WE7I82L5bdNGm2">
<plistfrag xmlns="plist-apple">
<key>maild</key>
<string>E1B537</string>
</plistfrag>
</item>
</items>
</event>
<x xmlns="jabber:x:delay" stamp="2008-07-18T01:11:11.447Z"/>
</message>
```

OAuth over XMPP

OAuth, delegated authorization. Let xmpp bots act on your behalf to access protected resources. Don’t give over your login and password.

OAuth over HTTP

```
Authorization: OAuth realm="http://sp.example.com/",
oauth_consumer_key="0685bd9184jfhq22",
oauth_token="ad180jjd733klru7",
oauth_signature_method="HMAC-SHA1",
oauth_signature="wOJIO9A2W5mFwDgiDvZbTSMK%2FPY%3D",
oauth_timestamp="137131200",
oauth_nonce="4572616e48616d6d65724c61686176",
oauth_version="1.0"
```

Here is what OAuth over http using request headers....

OAuth over XMPP

```
<iq type='set'
    from='random-id@twhirl.org'
    to='last.fm'
    id='sub1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <subscribe jid='random-id@twhirl.org'
                node='/music/Kellan+Elliott-McCrea'/>
    <oauth xmlns='urn:xmpp:oauth'>
      <oauth_consumer_key>0685bd9184jfhq22</oauth_consumer_key>
      <oauth_token>ad180jjd733klru7</oauth_token>
      <oauth_signature_method>PLAINTEXT+HMAC-SHA1>/oauth_signature_method>
      <oauth_signature>wOJIO9A2W5mFwDgiDvZbTSMK%2FPY%3D</oauth_signature>
    </oauth>
  </pubsub>
</iq>
```

And here is the same thing with the authorization being passed along with the xmpp stanza.

thank you!

questions?