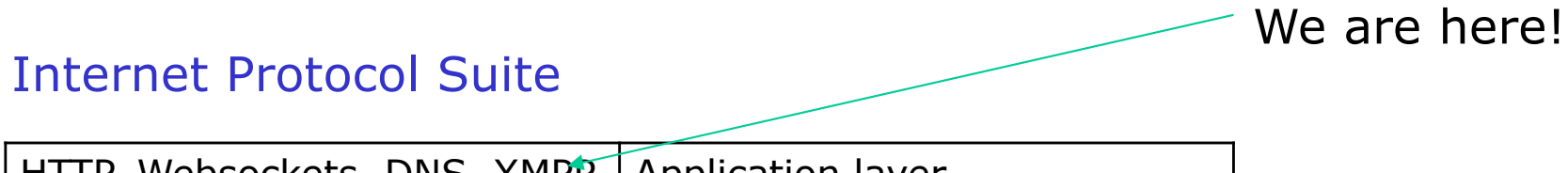
Carnegie Mellon University

# 95-733 Internet of Things
# XMPP Overview

# Where are we?




## Internet Protocol Suite

| | |
|---|---|
| HTTP, Websockets, DNS, XMPP, MQTT, CoAp | Application layer |
| TLS, SSL | Application Layer (Encryption) |
| TCP, UDP | Transport |
| IP(V4, V6), 6LowPAN | Internet Layer |
| Ethernet, 802.11 WiFi, 802.15.4 | Link Layer |

# Who Uses XMPP?

- Note: Short Message Service (SMS) is based on cellular connections. Here we are talking about instant messaging over the internet.
- Cisco Webex
- WhatsApp uses a trimmed down version.
- WhatsApp is one-to-one chat plus multi-user chat plus presence plus contact list management
- Google's Firebase use XMPP Json and Google's Android
- Adopted by Sensor Andrew
- So, what does a messaging system have in common with IoT?
- IoT is all about messaging

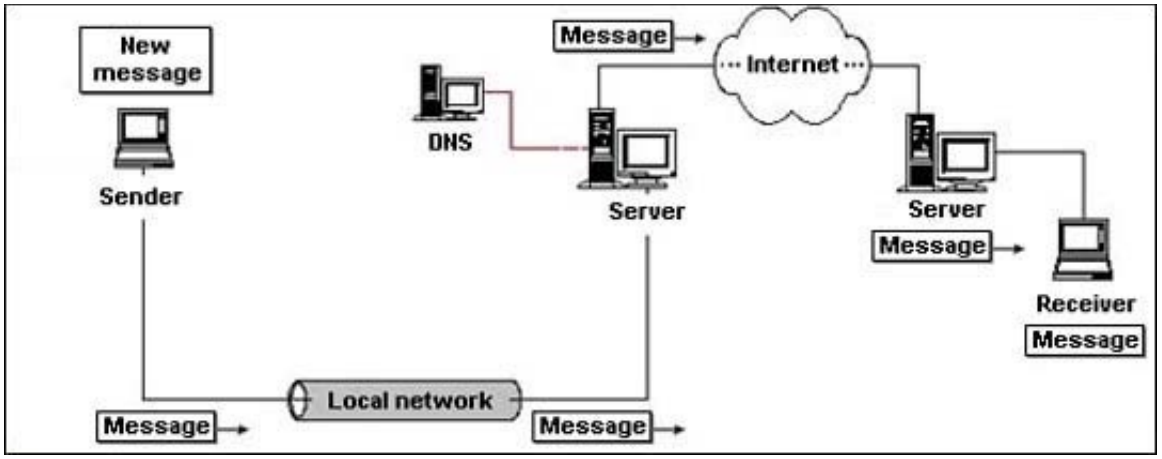
# XMPP was originally named Jabber

- Jabber is best known as “the Linux of instant messaging”. Implemented by ejabberd in Erlang.
- It is an open, secure, ad-free alternative to consumer instant messaging services like AIM, ICQ, MSN, and Yahoo.
- Under the hood, Jabber is a set of streaming XML protocols and technologies that enable any two entities on the internet to exchange messages, presence, and other structured information in close to real time. (Jabber.org)

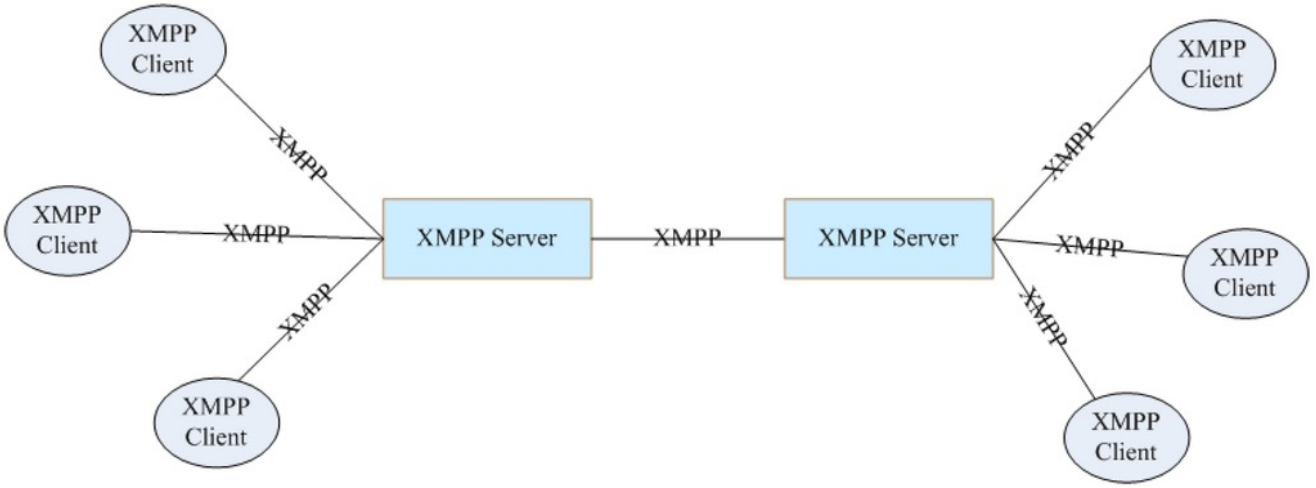
# XMPP From IETF

- In IM, the central point of focus is a list of one's contacts or "buddies" (in XMPP this list is called a "roster").
- Exchange relatively brief text messages with particular contacts in close to real time The catalyst for exchanging messages is "presence" -- i.e., information about the network availability of particular contacts (thus knowing who is online and available for a one-to-one chat session).
- Presence information is provided only to contacts that one has authorized by means of an explicit agreement called a "presence subscription".
- Thus at a high level XMPP needs to be able to complete the following use cases:
  - Manage items in one's contact list (list is maintained on the server)
  - Exchange messages with one's contacts
  - Exchange presence information with one's contacts (send communication status to the server)
  - Manage presence subscriptions to and from one's contacts

# SMTP Architecture



# XMPP Architecture



# XMPP From IBM

- Many useful technologies are often applied in ways their originators never considered.
- For example, HTTP is the de facto standard protocol for serving web pages over the internet, but it is also used as an application-layer transport for other protocols like SOAP and REST.
- XMPP is another useful technology that is finding many new applications beyond simply instant messaging. XMPP has several positive attributes.
- Quiz:

  Is XMPP programming language dependent? No

  Is XMPP OS independent? Yes

  Are the messages defined by a standards body? Yes

  Are MQTT or websocket messages defined by a standards body? No

  Can XMPP be used over websockets? Yes

# XMPP Basic connection

1. Client initiates a TCP connection
2. Client sends presence information to the server
3. The client requests and receives its roster
4. The client interacts with roster members
5. The client disconnects

This is all done with standard XML messages using the XMPP vocabulary and grammar.

# Naming Things (1)

- A lot like an email address, XMPP uses Jabber ID's (JIDs)
- All of these JIDs could be logged on at the same time.

mm6@andrew.cmu.edu — Called a "bare JID"

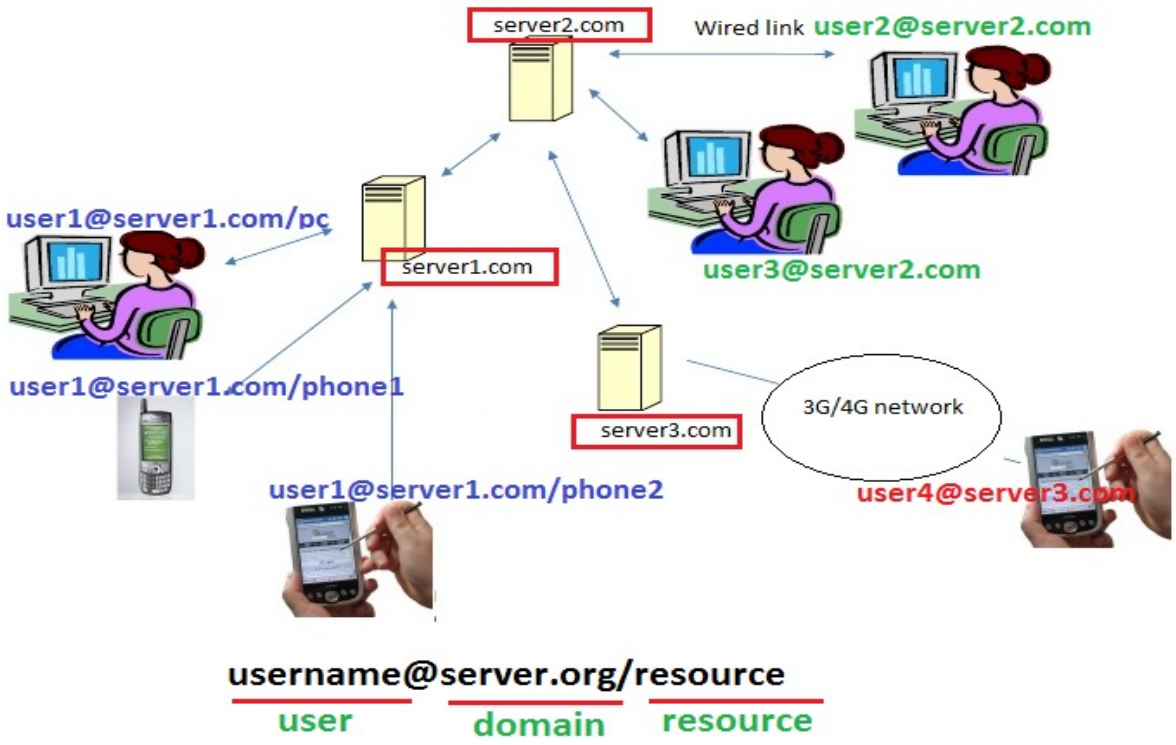
mm6@andrew.cmu.edu/mobile — A full JID includes a resource.

mm6@andrew.cmu.edu/tablet — A resource is used for message delivery

mm6@andrew.cmu.edu/auto — Useful if logged in from several devices

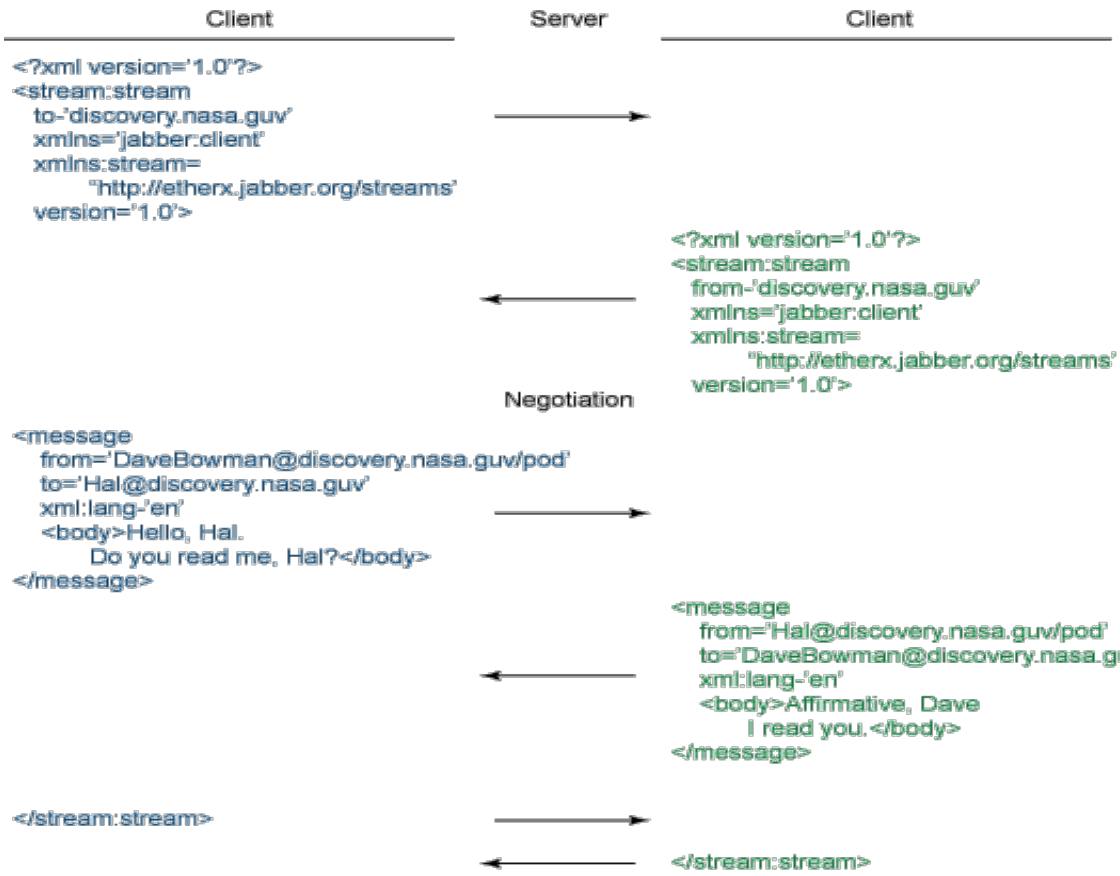
On Whatsapp: 412-776-1212@s.whatsapp.net

# Naming Things (3)




From: https://www.blikoontech.com/xmpp/xmpp-a-soft-friendly-introduction

# An Example (from IBM)




The XML is being Transferred in pieces. The TCP connection only closes at the end.

There are two XML documents involved.

Would this work over websockets?

Sure. It involves a bidirectional conversation.

# XMPP From the perspective of the application developer

- We do not want to work at the level of XML or JSON.
- We want middleware to provide support.
- Middleware separates concerns. It hides the details associated with messaging.
- Details include marshalling and un-marshaling of parameters and addressing.
- Details include generating the correct XMPP message to send.
- Details include reading and writing messages to the TCP layer.
- At the application programmer level, WE WANT NONE OF THAT!
- Use middleware to hide all of that!

# XMPP Client in Ruby

**Listing 1. Simple XMPP agent for word definitions (IBM)**

```
require 'xmpp4r/client'
# Create a *very* simple dictionary using a hash
hash = {}
hash['ruby'] = 'Great object oriented scripting language'
hash['xmpp4r'] = 'Simple XMPP library for ruby'
hash['xmpp'] = 'Extensible Messaging and Presence Protocol'
# Connect to the server and authenticate
jid = Jabber::JID::new('bot@default.rs/Home')
cl = Jabber::Client::new(jid)
cl.connect
cl.auth('password')
```

# XMPP Client in Ruby

```
# Indicate our presence to the server
cl.send Jabber::Presence::new

# Send a salutation to a given user that we're ready
salutation = Jabber::Message::new( 'hal@default.rs', 'DictBot
ready' )
salutation.set_type(:chat).set_id('1')
cl.send salutation
```

# XMPP Client in Ruby

```
# Add a message callback to respond to peer requests
cl.add_message_callback do |inmsg|
    # Lookup the word in the dictionary
    resp = hash[inmsg.body]
    if resp == nil
      resp = "don't know about " + inmsg.body
    end
    # Send the response
    outmsg = Jabber::Message::new( inmsg.from, resp )
    outmsg.set_type(:chat).set_id('1')
    cl.send outmsg
end
```

# Java uses the Smack API

In order to test the client, we'll need an XMPP server. To do so, create an account on jabber.hot-chilli.net – a free Jabber/XMPP service.

```
import org.jivesoftware.smack.Chat;
import org.jivesoftware.smack.ConnectionConfiguration;
import org.jivesoftware.smack.MessageListener;
import org.jivesoftware.smack.Roster;
import org.jivesoftware.smack.RosterEntry;
import org.jivesoftware.smack.XMPPConnection;
import org.jivesoftware.smack.XMPPException;
import org.jivesoftware.smack.packet.Message;
// Works with Android
```

# Java uses the Smack API

```
private XMPPConnection connection;
public void login(String userName, String password) throws
                        XMPPException {
    // Use a local XMPP server
    ConnectionConfiguration config = new
                    ConnectionConfiguration("localhost", 5222);
    connection = new XMPPConnection(config);
    connection.connect();
    connection.login(userName, password);
}
```


95-733 Internet of Things

# Java uses the Smack API

```
public void displayBuddyList() {
     Roster roster = connection.getRoster();
     Collection<RosterEntry> entries = roster.getEntries();
     System.out.println("\n\n" + entries.size() + " buddy(ies):");
      for(RosterEntry r:entries) {
          System.out.println(r.getUser());
       }
 }
```

Many XMPP Javascript libraries exist for real time chat within a browser over websockets.

# XMPP and Thing Registration

```
<iq type='set'
      from='thing@example.org/imc'
      to='discovery.example.org'
      id='1'>
     <register xmlns='urn:xmpp:iot:discovery'>
        <str name='SN' value='39487234732948723'/>
        <str name='MAN' value='www.ktc.se'/>
        <str name='MODEL' value='IMC'/>
        <num name='V' value='1.2'/>
        <str name='KEY' value='4857402340298342'/>
     </register>
  </iq>
```

# Suppose a sensor is registered. How do we read from it?

# Reading sensor data

- Request to a Thing for an Automatic Meter Reading

```
<iq type='get' from='client@clayster.com/amr'
                    to='device@clayster.com' id='S0001'>
      <req xmlns='urn:xmpp:iot:sensordata' seqnr='1’
           momentary='true'/>
</iq>
```

- Response from the Thing – I got your request

```
<iq type='result' from='device@clayster.com'
                        to='client@clayster.com/amr' id='S0001'>
      <accepted xmlns='urn:xmpp:iot:sensordata' seqnr='1'/>
</iq>
```

# Data arrives from a sensor

```
<message from='device@clayster.com'
              to='client@clayster.com/amr'>
    <fields xmlns='urn:xmpp:iot:sensordata' seqnr='1' done='true'>
        <node nodeId='Device01'>
            <timestamp value='2013-03-07T16:24:30'>
                <numeric name='Temperature' momentary='true'
                        automaticReadout='true' value='23.4' unit='°  C'/>
                <numeric name='load level' momentary='true'
                        automaticReadout='true'
                        value='75' unit='%'/>
            </timestamp>
        </node>
  </fields>
</message>

<!– Note how the units are stated and the lack of ambiguity -->
```

# Sensor Andrew Based on XMPP (2007)

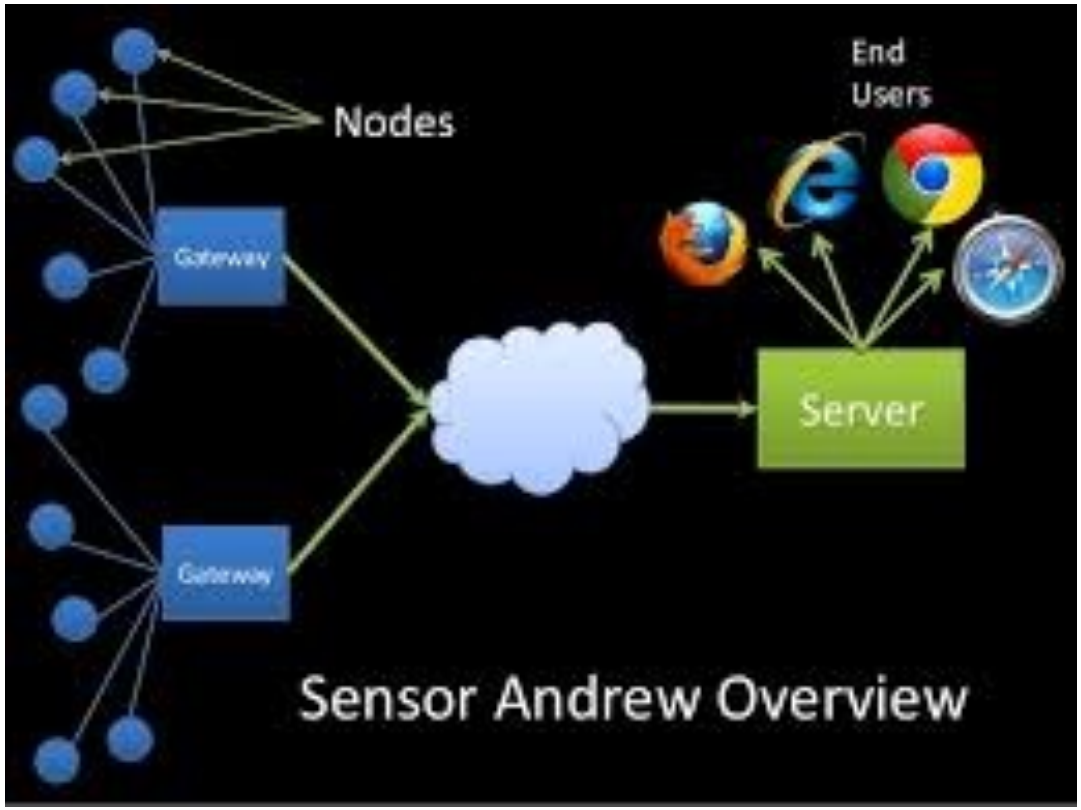
Non-functional characteristics:
Open (XMPP)
- Standards based
- Standard message formats

Heterogeneous sensors
Security, Privacy Challenges
Reliable (ejabberd – Erlang open source)
Fault tolerant (ejabberd, Erlang)
- In ejabberd all information can be stored on more than one node, nodes can be added or replaced "on the fly". Erlang is big on handling failures

Performance (speed) XML is typically far slower than compact binary messages
Extensible
Manageable
Cost


95-733 Internet of Things

# Sensor Andrew




A good architecture survives change. What could change?

Price of things
Variety of things (sensors)
Applications
Ubiquity of networks
Speed of networks
Battery life
Speed of processors
Effects of failure
Government regulations?

We do not allow cars on the road without seatbelts. We may need governments to regulate IOT devices for security. New California IoT law goes into effect January 1, 2020