IoT Programming
CS144r/244r

Robert Cohn
Intel
Feb 24, 2014
IOT in the News

• Qleek: How the internet of things could make media physical again
Making a Product

• Concept
• Design: user experience
• Engineering
• Manufacture
Internet of Things

• Originally internet in IOT was just a metaphor
  – RFID tags so we can track assets in a warehouse
• Embedded devices have had micro-controllers and sensors for 25 years
• What has changed
  – More compute power
  – Better sensors
  – Reduced cost
  – Algorithms
  – **Networking**
  – **Software components**
Case Studies

• Use some examples to understand architecture of an application

• WeggUp - A sleeping cycle and light alarm clock

• Web-controlled Twittering Roomba

• www.instructables.com for project ideas
WeggUp - A sleeping cycle and light alarm clock

- Gradual wakeup with sunrise simulation & sounds
- Detects sleep cycle to find best wakeup time – motion detector
- Shuts off audio & light when asleep
- Web interface to control wakeup time and monitoring
WeggUp

Alarm logic

Web server

Bed

Light

Web server

Cloud

Web Interface

Telegram

browser

browser
Web-controlled Twittering Roomba

- Web interface to control
- Tweets status
Web-controlled Twittering Roomba
Application Architecture Ingredients

- **Components**: microcontroller, sensors, web server, phone
- **Function**: monitor, interface
- **Topology**: microcontroller talks to cloud
- **Protocols**: HTTP, TCP
- **Interfaces**: REST
- **Software Stack**: HTML, Nodejs, PHP, ...
**Application Architecture**

**Components & function**
- **Device:** connect to physical world
  - Microcontroller: behavior
  - Sensors: switch, knob, temp, video, mic
  - Actuators: led, speaker, servo, stepper motor
- **PC/Tablet/phone:** user interface
  - Observe, control
- **Server**
  - Proxy, connectivity
  - Aggregation: collect sensor data
  - Computation: speech/vision
  - Storage: persistent logging
  - Security: authentication
  - Services: geocoding

**Topology**

- **Cloud**
- **services**
- **hosting**
- **Thing**
- **Human**
Application Architecture
Physical Connections

Server
• Data center

PC/Tablet/Phone
• Wifi

Device
• Cost & energy are concern
• BLE: low energy bluetooth
  – PAN: personal area network
  – Star topology
  – $2.95
• Zigbee
  – LAN
  – Mesh topology
  – $3.20
• Wifi
  – LAN
  – Star topology
  – $4.50
  – Most energy efficient/bit
## Application Architecture: Communication Protocols

<table>
<thead>
<tr>
<th>Layer</th>
<th>Web</th>
<th>IoT-focused</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>HTTP</td>
<td>CoAP</td>
</tr>
<tr>
<td>Transport</td>
<td>TCP</td>
<td>UDP</td>
</tr>
<tr>
<td>Internet/Network</td>
<td>IP</td>
<td>6LoWPAN</td>
</tr>
<tr>
<td>Data Link</td>
<td>Ethernet</td>
<td>Ethernet</td>
</tr>
<tr>
<td>Physical</td>
<td>RJ45, DSL, Docsis, Wifi</td>
<td>BLE, WiFi</td>
</tr>
</tbody>
</table>

- HTTP is protocol of web
  - Browsers, applications, web servers, proxies, highly interoperable
  - Processing and bandwidth overhead, 500 bytes
- CoAP for IoT
  - Efficiency
  - Gateways so non IOT endpoint can be HTTP
Application Architecture

Application Stack

**Client**
- iOS: ObjectiveC/Cocoa Touch
- Mac: ObjectiveC/Cocoa
- Windows desktop: C#/Win32
- Win8: */winrt
- Browser: HTML5 (Javascript/HTML/CSS)

**Server**
- PHP
- .net (asp)
- Ruby
- Java
- Python
- Javascript (nodejs)

**Device**
- Arduino/Spark
  - Low power/cost
  - No OS, slow, small memory
  - Sketches in wiring, C-like
  - Used in lab
- Netduino
  - Like arduino
  - C# + visual studio
- Raspberry Pi, Galileo, beaglebone
  - Powerful processors
  - Linux
  - Files, networking, software
  - C, python, javascript (nodejs)
  - Galileo also uses Wiring
  - Extensive runtime
  - Not well suited for hard real-time
    - Predictability over speed
    - Glitches in game or video
My Preference

• Javascript across the stack
  – HTML5 for client
  – Nodejs for server
  – Nodejs for device

• Benefits
  – 1 language, but not 1 program
  – Great support for communication
  – Big community committed to open source
Javascript

• Started as scripting language for browser
  – Now for server, applications
• Similar syntax to C & Java
  – if (a > 3) b = 2;
• Dynamic typing
  > var a = 1
  > a + 1
  2
  > a = '1'
  > a + 1
  '11'
Javascript

• Objects are collection of key value pairs
  
  ```javascript
  a = {b: 1, c: 2}
  a.b = 3
  a.d = 4
  ```
  
  – JSON: string format for objects
    • Convenient for sending over network

• Functions are 1\textsuperscript{st} class objects
  
  ```javascript
  foo(1, function(a) {return a + 1})
  ```
Javascript

• Single threaded
  – Simplifies programming
  – If you are doing heavy computation, UI may not respond

• Event driven, asynchronous
  – Driven by events & handlers
    • instead of `processButtonPress(getButtonPress());`
    • `onButtonPress(processButtonPress)`
More Structured Javascript

- Browsers try to handle malformed html
- Jslint, jshint for error checking
- Coffeescript, typescript transcompile to javascript
Environments for Javascript

• Browser: runs on everything with a display
  – PC, phone, tablet, ...
• Nodejs: does not require display
  – Laptop, server, beaglebone, raspberry pi, galileo
In Browser, JS is part of HTML

Browser’s language for web pages: HTML + CSS + JS

**HTML**: Content
<button>Hello!</button>

**CSS**: Style
button { color: blue }

**Javascript**: Behavior
```
$(‘button’).click(function() {
    $(this).css(‘color’, ’red’);
})
```
Demo: Javascript in the browser

**HTML**

```html
<html>
  <body>
    <button>Hello!</button>
  </body>
</html>
```

**Interactive with JS**

```html
<script src="http://code.jquery.com/jquery-1.11.0.min.js"></script>
<script>
  $(function() {
    $('button').click(function() {
      $(this).css('color','red');
    });
  });
</script>
```
Evolution of Web Pages

1997: Simple Text and Images

2004: Interactivity enabled by JS

2013: Full Applications, canvas, WebGL, video
What Makes HTML5 New and Exciting?

• Deploy Applications without plugins
  – Graphics: canvas and SVG
  – Audio & Video
  – Offline
  – Use mobile device sensors: location & orientation

• HTML has become the most broadly supported cross-platform runtime for Mobile Devices
And More is Coming

**W3C APIs Make HTML5 a Complete Application Platform**

- **W3C HTML WG**
  - HTML5 core, elements, layouts, DOM.

- **W3C Device API WG**
  - Enable web application interaction with device hardware, services, and applications.
  - *Intel participates actively*

- **W3C Application Platform WG**
  - Improve client-side application development on the web.

- **W3C WebApp Security WG**
  - Security and policy mechanisms to improve security of web applications.

- **W3C CSS WG**
  - Develop and maintain cascading style sheets.

- **W3C Audio WG**
  - Add more advanced audio capabilities, such as the processing and synthesis of audio streams.

- **W3C Web Cryptography WG**
  - Define cryptography APIs to provide common functionality to web applications.

- **W3C System Applications WG**
  - Define runtime environment, security model and associated APIs to build web apps similar to native.
  - *Intel participates heavily*

- **W3C Web Performance WG**
  - Enhance application performance of user agent features and APIs.

- **W3C Tracking Protection WG**
  - Improve privacy and user control.
  - *Intel co-chairs*

- **W3C NFC WG**
  - Develop ways for web pages to interact with NFC devices.
  - *Intel chairs*

- **Browser Testing and Tools**
  - Develop technologies for use in testing, debugging, and troubleshooting of web applications.

---

*Intel actively participates in W3C Web Specs*†
Node.js

- Created to support asynchronous (non-blocking) IO
  - Network and file requests return immediately, data comes later in a callback
    - If an IO request blocks, then another thread or process is needed to be responsive. Non-blocking IO can use a single thread
  - JS in browser already had support for asynchronous
- Many (me included) use it as a JS scripting environment, using both asynchronous and synchronous IO
- Uses V8, JS engine in chrome + asynchronous IO library (files, network, ...)
- Built-in support to create web servers
- No sandbox, full access
Demo: nodejs

• Create server for cloud or device

```javascript
var http = require('http');
http.createServer(function (req, res) {
    res.writeHead(200, {'Content-Type': 'text/plain'});
    res.end('Hello World\n');
}).listen(1337, '127.0.0.1');
console.log('Server running at http://127.0.0.1:1337/');
```
**Nodejs**

```javascript
var b = require('bonescript');
var led = "USR3";
var state = 0;

b.pinMode(led, 'out');
toggleLED = function() {
    state = state ? 0 : 1;
    b.digitalWrite(led, state);
};

setInterval(toggleLED, 1000);
```

**Galileo**

```javascript
var arduiNode = require('../arduiNode');
arduiNode.initBoard();
arduiNode.pinMode(13, 1);
var toggle = 0;
setInterval(function() {
    state = state ? 0 : 1;
    b.digitalWrite(led, state);
}, 1000);
```

**Raspberry Pi**

```javascript
var gpio = require('rpi-gpio');
gpio.setup(7, gpio.DIR_OUT, write);

function write() {
    gpio.write(7, true, function(err) {
        if (err) throw err;
        console.log('Written to pin');
    });
}
```
Javascript Runtime

• Minimal built-in
• Usually need to include 3\textsuperscript{rd} party libraries
• Browser
  – bower for package management
  – Jquery, underscorejs
• Nodejs
  – NPM for package management
  – Expressjs: web server middleware
  – Fs-extra: more complete filesystem
JS Debugging

• Debugging Chrome Dev Tools
  – very powerful html, css, & JS debugger
    • use for nodejs & browser apps

• safari, IE have similar tools
Back to our Application
Application Architecture Ingredients

- Components
- Function
- Topology
- Protocols
- Software Stack
- Interfaces
Interfaces

- **REST**
  - Client/server
  - Make a request, receive a response
  - Browser can make requests, but cannot receive a request
  - Server can send and receive requests
  - Heavyweight

- **Web socket**
  - Bidirectional channel between server or browser
  - One endpoint must be server
  - Persistent connection
  - Lightweight communication

- **WebRTC data**
  - Bidirectional
  - Peer to Peer
    - No restrictions on endpoints
    - No restrictions on network
Service Oriented Architecture (SOA)

• Service Oriented Architecture decomposes application into services that are available over web with a well defined protocol
• makes it easier to develop and test
• Services typically use REST API
• Create a service:
  – arduino device provides temperature service
• Use a service:
  – twitter
  – facebook
  – weather
  – maps
  – geocoding
  – news
  – translation
• [Google Services](https://developers.google.com)
REST

- Representational state transfer
- Request: a URI: http://ip.jsontest.com/
- Response: some data: HTML, XML, JSON, plain text
- Style to support interoperability, performance, and scalability for web interfaces
- Constraints
  - client-server
    - client makes request, server responds
    - separation of concerns, can be changed independently
  - stateless
    - no client context on server between requests; each request stands on its own, server can store state in database
  - cacheable
    - improve response time, reduce bandwidth
    - some data marked as non-cacheable
  - layered
    - client cannot tell if there are intermediaries like proxies
  - code on demand
    - download code into client: java or JavaScript
  - uniform interfaces
    - resources (URI) http://ip.jsontest.com/
    - self description: is it cacheable, text, img
REST

• 4 Verbs
  – GET: retrieve a resource, should not modify state: get a pdf file, or a web page, or a list of items matching a query
  – POST: add something to a collection. upload a file, creating an account
  – PUT: replace a resource with something new
  – DELETE: remove a resource

• Nouns follow the hostname, and parameters, too
  – Example from browser: http://ip.jsonstest.com/
  – Example from https://github.com/rscohn2/iotns
  – add a name
    curl -X PUT 'iotns.herokuapp.com://addname?name=example.com&value=192.168.1.1'
  – Get name
    curl -X GET 'iotns.herokuapp.com://getname?name=example.com'
  – Delete
    curl -X DELETE 'iotns.herokuapp.com://clearnames'

• Google & Mashery have tools to explore API
  – https://cloud.google.com/console/project/apps~universal-rex-451/apiui/api/calendar/method/calendar.calendarList.list
REST Server

- Demo: Make our own API with express
- Node has minimal http server, express adds middleware to make REST API
  - routing: dispatch based verbs and nouns
  - Serves static html
  - templating
  - exports.foo = function(req, res){
    var url = require('url');
    var url_parts = url.parse(req.url, true);
    var query = url_parts.query;
    res.send("foo: " + query.bar);
  }
• Ruby & Sinatra

```ruby
#!/usr/bin/env ruby
require 'sinatra'

get '/' do
  redirect to('/hello/World')
end

get '/hello/:name' do
  "Hello #{params[:name]}!"
end
```
REST client calls

Nodejs, npm install request –save
https://github.com/mikeal/request

```javascript
var request = require('request');
request('http://www.google.com',
    function (error, response, body) {
        if (!error && response.statusCode == 200) {
            // Print the google web page.
            console.log(body)
        }
    })
```

Browser with jquery

```javascript
$.ajax({
    type: "POST",
    url: "some.php",
    data: { name: "John", location: "Boston" }
}).done(function( msg ){
    alert( "Data Saved: " + msg );
});
```
Hosting

- Easy to run nodejs on laptop, but networking may make sharing difficult
- PaaS like Heroku make it easy to deploy your service
- Get the program running locally, deploy to their server in the cloud
- Free
- add Procfile:
  web: node app.js
- git init
- git add .
- git commit -m 'initial version'
- heroku create cs133r
- git push heroku master
Web Sockets

Socket.io layers on top of web sockets
Persistent connection, bidirectional & low cost messaging

Client

```javascript
var socket = io.connect('http:');

socket.on('news', function (data) {
  console.log(data);
  socket.emit('my other event', { my: 'data' });
});
```

Server

```javascript
var app = require('express')()
, server = require('http').createServer(app)
, io = require('socket.io').listen(server);

server.listen(3000);

app.get('/', function (req, res) {
  res.sendfile(__dirname + '/index.html');
});

io.sockets.on('connection', function (socket) {
  socket.emit('news', { hello: 'world' });
  socket.on('my other event', function (data) {
    console.log(data);
  });
});
```
Putting it all Together

• Nodejs program to control the pins

• REST API
  – Direct browser to device
    • Wrap pin code in a REST API
    • Browser or command line to make REST API calls
  – Host controller in cloud
    • Browser and device talk to controller
    • Controller integrates other services (e.g. calendar)

• Web socket
  – Browser sends events to device
Shortcuts

• Mocking: software placeholder
  – Device
  – Service
  – Interface
• Phone as ‘thing’:
  – Accelerometer
  – Geolocation
  – Camera
  – HTML has API support for the sensors
• MVP: Minimum viable product
Resources

• stackoverflow.com for questions
• chrome
• github.com for sharing code with team
• Nodejs & npm
  – Express http://expressjs.com/
  – REST calls: https://github.com/mikeal/request
  – Service authentication: passportjs
• heroku.com for hosting app: Quickstart
• Google & mashery for services to integrate into your app
• Adobe Brackets for editing JS: http://brackets.io/
Summary

• Application Architecture
• JavaScript across the stack
  – Browser & nodejs
• Interfaces
• Structure application as services, use existing services