Building it

From mock-up to prototype to production

- Rapid prototyping is important
- Mock-up vs prototype
  - Start with a mock-up, or
  - Start working on the target hardware
- Using the same components when prototyping as in production is good
  - Makes production easier

Mock-up: Raspberry Pi

Mock-up: Arduino Yun

Prototyping: CC2538

IoT hardware – closer look

IoT hardware

- Sensors and actuators
  - Connectors to the physical world
- Microprocessor
  - To be able to do something with the sensors/actuators
- Communication device
  - To communicate with the world
- Power source

Typical IoT hardware properties

- Microcontroller
  - Memory
    - Flash ROM on the order of 64-512 kilobytes
    - RAM on the order of 8-32 kilobytes
  - I/O to connect sensors and actuators
    - Serial ports, ADCs, SPI, etc
- Radio
  - External transceivers
  - Systems-on-a-Chip
    - Radio integrated with microcontroller

Typical IoT hardware properties

- Power source
  - Battery
  - Energy harvesting
    - Solar cells
    - Ambient RF (like RFID)
    - Piezo electric
    - Temperature differences

Radio hardware

- Microcontroller + transceiver
  - CC1120 – sub-GHz
  - CC2520 – IEEE 802.15.4
  - nRF8001 – Bluetooth Smart
  - CC3300 – WiFi
- Connected via SPI
- System-on-a-Chip
  - CC2538 – IEEE 802.15.4 + ARM Cortex M3
  - Broadcom Wiced – WiFi + ARM Cortex M3

Power

- Power consumers
  - Sensors, actuators
  - Microprocessor
  - Communication
- Radio must be completely off
  - Consumes power even when idle
- Radio duty cycling mechanisms

OS vs rolling your own

- Rolling your own non-OS firmware
  - Write everything, flash the chip
- An OS helps you with
  - Network stacks
  - Memory managament
  - Drivers
  - Power management
  - ... and much, much more

IoT software challenges

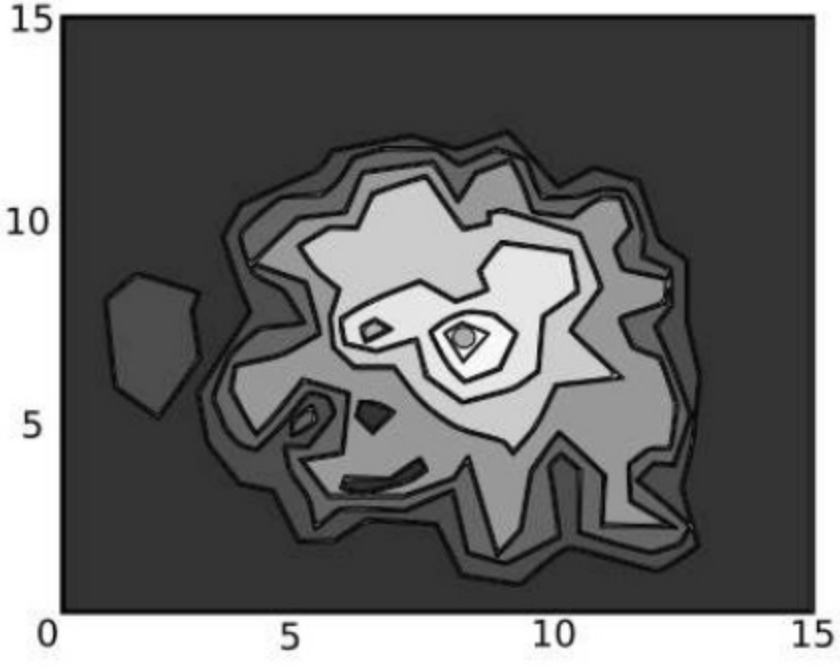
- Severe limitations in every dimension
  - Memory
  - Processing power
  - Communication bandwidth
  - Energy, power consumption
- The number of different platforms is large
  - Different microcontrollers, different radios, different compilers

IoT system challenges

- Communication-bound, distributed systems
  - Exceptionally low visibility
    - Hard to understand what is going on
  - Application development difficult
  - Debugging difficult
- Large-scale
- Wireless communication

Wireless connectivity

- Counter-intuitive
- Not a simple replacement to cables



The magic behind it all: Contiki

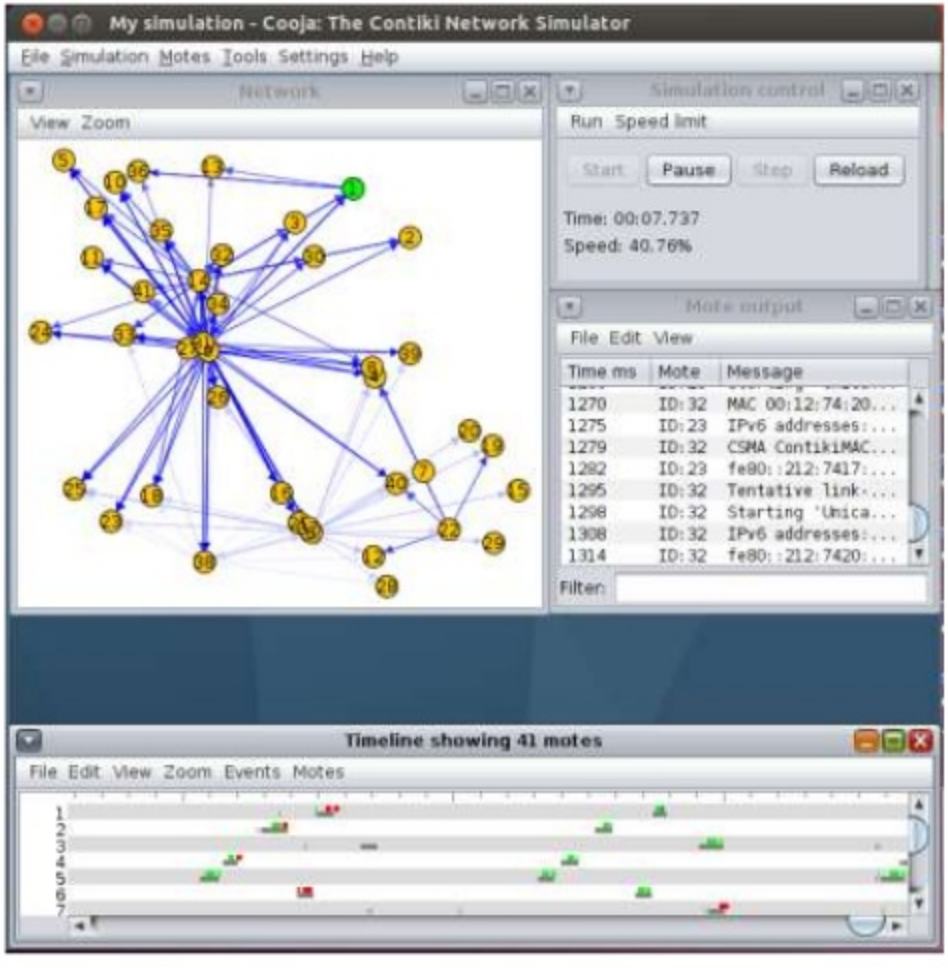
Contiki: an IoT OS

- Runs on tiny microcontrollers and SoCs
- Handles communication over a number of different radios
- Provides processes, timers, libraries, memory allocation, network stacks, etc

Contiki

- Dealing with limited memory
  - Memory block allocation strategies
  - Protothreads
- Portability
  - Fully written in C
- Application development and debugging
  - The Cooja network simulator

The Cooja network simulator



A Brief History of Contiki

Contiki: 10+ years old

- Based on the uIP TCP/IP stack
- First release March 10, 2003
  - Ports to a range of ~~odd~~ lovely platforms
    - Most 6502-based (Apple II, Atari 2600)
    - Commodore C64
- The Contiki idea: put things on the Internet
  - First system to do IP-based wireless sensor networks


(C) Copyright 2003 Pumpkin, Inc.


Radio Tagging Hardware
Door Proxi
Sensors
GE Security
Light Sensor
Expansion Serial Port


YA07 JYW

2008: IPv6 for Contiki

- uIPv6 was contributed by Cisco in 2008
- World's smallest fully certified IPv6 stack

TIME magazine award

TIME IN PARTNERSHIP WITH CNN

TIME's Best Inventions of 2008

49 Best Inventions

Previous Next

30. The Internet Of Things

ARTICLE TOOLS

Print
Email
Sphere
AddThis
RSS
Yahoo! Buzz

In September, a group of high-tech companies that includes Cisco and Sun formed the IP for Smart Objects Alliance. Simply put, the organization intends to create a new kind of network that will allow sensor-enabled physical objects — appliances in your home, products in a factory, cars in a city — to talk to one another, the same way people communicate over the Internet.

ILLUSTRATION FOR TIME BY CHRISTOPH NIEMANN

Top Stories

- Readying Bu
- More Allegati Governor Cas
- Why Nicarag
- Where the Re
- Europe's Hop Being Dashed

Recent history

- July 2012: Thingsquare founded, Contiki 2.6 released
- October 2012: Moved to github
- November 2012: automated regression tests with Travis
- November 2012: IPv6/RPL updates
- December 2012: New testing framework
- July 2013: Significant RPL updates
- November 2013: Contiki 2.7

Next steps: Contiki 3.x

- Push the Thingsquare firmware features towards Contiki
- UDP, TCP socket APIs
- HTTP socket API
- Websocket API

Contiki features

- Three network stacks
  - IPv4/IPv6 stacks
    - RPL, 6lowpan, CoAP, HTTP, ...
  - Rime stack
- Instant Contiki development environment
- The Cooja network simulator
- Sleepy meshing (ContikiMAC)
- Interactive shell
- Flash-based file system
- Powertracing
- Protothreads
- Multithreading
- Event timers, real-time timers
- Libraries: linked list, ringbuf, crc16, etc

Additional Thingsquare features: Contiki 3.x

- IPv4/IPv6 routing
  - NAT64 and DNS64
- AES encryption
- Network sniffer
- Frequency hopping protocol
- Portable duty cycling (Drowsie, CSL)
- WebSocket protocol
- UDP, TCP, HTTP socket APIs
- More hardware platforms

Back to the Big Red Button

What the Big Red Button did

- When the button switch went from 1 to 0
  - Do HTTP POST request to http://requestb.in
- Several steps:
  - Ask the DNS server for requestb.in
  - Get the answer, but converted to an IPv6 address – converted by the router
  - Set up a TCP connection to the IPv6 address
  - The router translates to IPv4
  - Send the HTTP POST request

HTTP POST request

```
POST /abcdefghij HTTP/1.1
Host: requestb.in
Content-Type: application/x-www-form-urlencoded
Content-Length: 15

button=big%20red
```

The Contiki code

```
#define URL "http://requestb.in/abcdefghij"
#define DATA "button=big%20red"
static struct http_socket s;

http_socket_post(&s, URL, DATA, strlen(DATA),
"application/x-www-form-urlencoded", callback,
NULL);
```

RESTful APIs

- Leverage the way that HTTP works
- Representational state transfer
- Coined by Roy Fielding in 2000
- Resources, representation
- State
- Actions, requests

Example RESTful API

- Collection resource
  - http://api.example.com/devices
- Element resource
  - http://api.example.com/device/23
- Typical actions:
  - GET collection
    - Return a list of devices
  - POST collection
    - Create a new device
  - GET element
    - Return the device
  - PUT/POST element
    - Update the device information

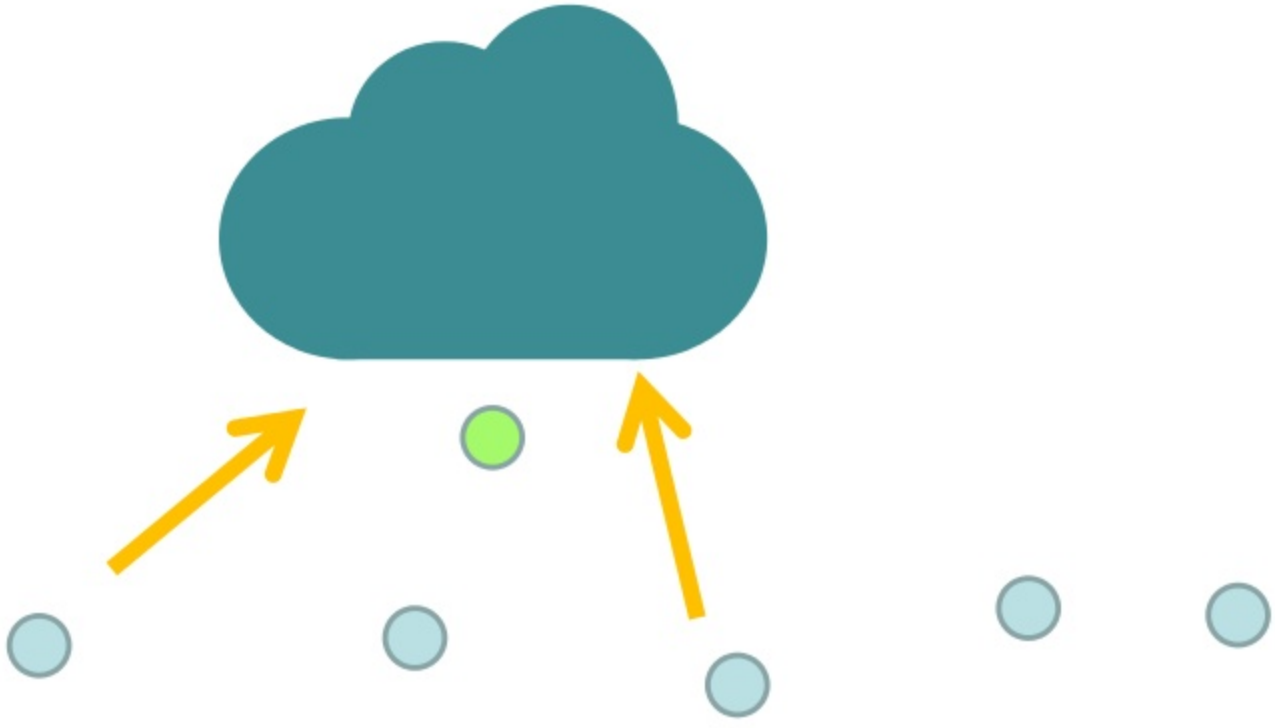
Representation: JSON

- JavaScript Object Notation
- Very simple
  - Compared to stuff like XML
- A representation of a Javascript object
  - Can easily be converted back and forth
  - var str = JSON.stringify(obj);
  - var obj = JSON.parse(str);

JSON Example

```
{
 "id": "12345acb",
 "name": "Adam",
 "age": 392131145,
 "sensors": [ "light", "button" ]
}
```

Lab 2: Build our own cloud service



- Device connect to the cloud server with a websocket
- Server receives message from device, broadcasts to all connected devices

Material

- Cloud software
  - Node.js
  - Running on our laptop
- The Thingsquare kit
- Device software
  - websocket-example.c

Cloud software: node.js

- Javascript framework for scalable high-concurrency servers
- Non-blocking, event-driven programming style
- Lots of libraries
- npm – the node package manager

Node.js crash course

- Javascript programs are run from the command prompt
  - node program.js
- Unlike server software like Apache, node.js is extremely raw
  - You program everything from the ground up
  - Libraries help you do almost everything
- Use HTTP library to build web server
- Use Websocket library to build web socket server

Install node

- http://nodejs.org/
- Unzip websocket-server.zip
- Open a command prompt, go to the websocket-server directory
- In the websocket-server directory, install the websocket module
  - npm install websocket
- node example-server.js

The cloud server code

```
var serverPort = 8080;

var http = require('http');

var server = http.createServer(function(request, response) {
    response.writeHead(200, {'Content-Type': 'text/plain'});
    response.write('This is a websocket-only server\n');
    response.end();
});

server.listen(serverPort, function() {
    console.log('Server is listening on port ' + serverPort);
});
```

The cloud server code

```
var websocket = require('websocket').server;
var wsServer = new websocket({
    httpServer: server
});

var connections = [];

function broadcastMessage(message) {
    for (var i = 0; i < connections.length; i++) {
        connections[i].sendUTF(message);
    }
}
```

The cloud server code

```
wsServer.on('request', function(request) {
    var connection = request.accept(null, request.origin);

    var connectionIndex = connections.push(connection) - 1;

    console.log('Connection from ' + connection.remoteAddress + '.');

    connection.on('message', function(message) {
        if (message.type === 'utf8') {
            console.log('Message: ' + message.utf8Data);
            broadcastMessage(message.utf8Data);
        }
    });

    connection.on('close', function(connection) {
        connections.splice(connectionIndex, 1);
    });

});
```

The device code

```
PROCESS_THREAD(websocket_example_process, ev, data)
{
    static struct etimer et;
    PROCESS_EXITHANDLER(websocket_close(&s));
    PROCESS_BEGIN();

    websocket_open(&s, "ws://192.168.1.65:8080/",
                   "thingsquare", callback);

    while(1) {
        etimer_set(&et, CLOCK_SECOND * 10);
        PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));
        websocket_send_str(&s, "hello");
    }

    PROCESS_END();
}
```

The device code

```
static void
callback(struct websocket *s, websocket_result r,
         uint8_t *data, uint16_t datalen)
{
    if(r == WEBSOCKET_CONNECTED) {
        websocket_send_str(s, "Connected");
    } else if(r == WEBSOCKET_DATA) {
        char buf[100];
        snprintf(buf, sizeof(buf), "%.*s", datalen, data);
        thsq_client_set_str("data", buf);
    }
    thsq_client_push();
}
```

Update the code

- Obtain the IPv4 address of your laptop
  - On Windows, run “ipconfig” command
  - On Linux, run “ifconfig” command
  - On Mac, run “ifconfig” command
- Insert your own IPv4 address in the **`websocket_open()`** call

Running the code

- Run the websocket server on your laptop
  - node websocket-server.js
- Compile and upload the code to your device
- Enter “data” in the variable inspection field
- You should see
  - The server should say something like
    - `Connection from 192.168.1.98`
  - The device should say “Connected” then “hello” in the variable inspection field

What we have done

- We have built our own IoT cloud service
  - On our laptop
- We have connected an IoT device to our cloud service
- We have seen two-way communication, from the device to the cloud and back

More like this

http://thingsquare.com