Carnegie
Mellon
University

95-733 Internet of Things
Project 1 Key Ideas

Request/Response

- One player makes a request – the client
- One player provides some service and returns a response – the server
- HTTP is based on this simple idea

Publish/Subscribe

- Typically a three party protocol
- One player publishes messages to a topic or event name
- Other players subscribe to the topic or event name
- Messages flow from the publisher to the subscriber(s)

Carnegie
Mellon
University

Project 1 Part 1
Argon to Particle to Node-RED

Microcontroller Logic

Make calls to Particle

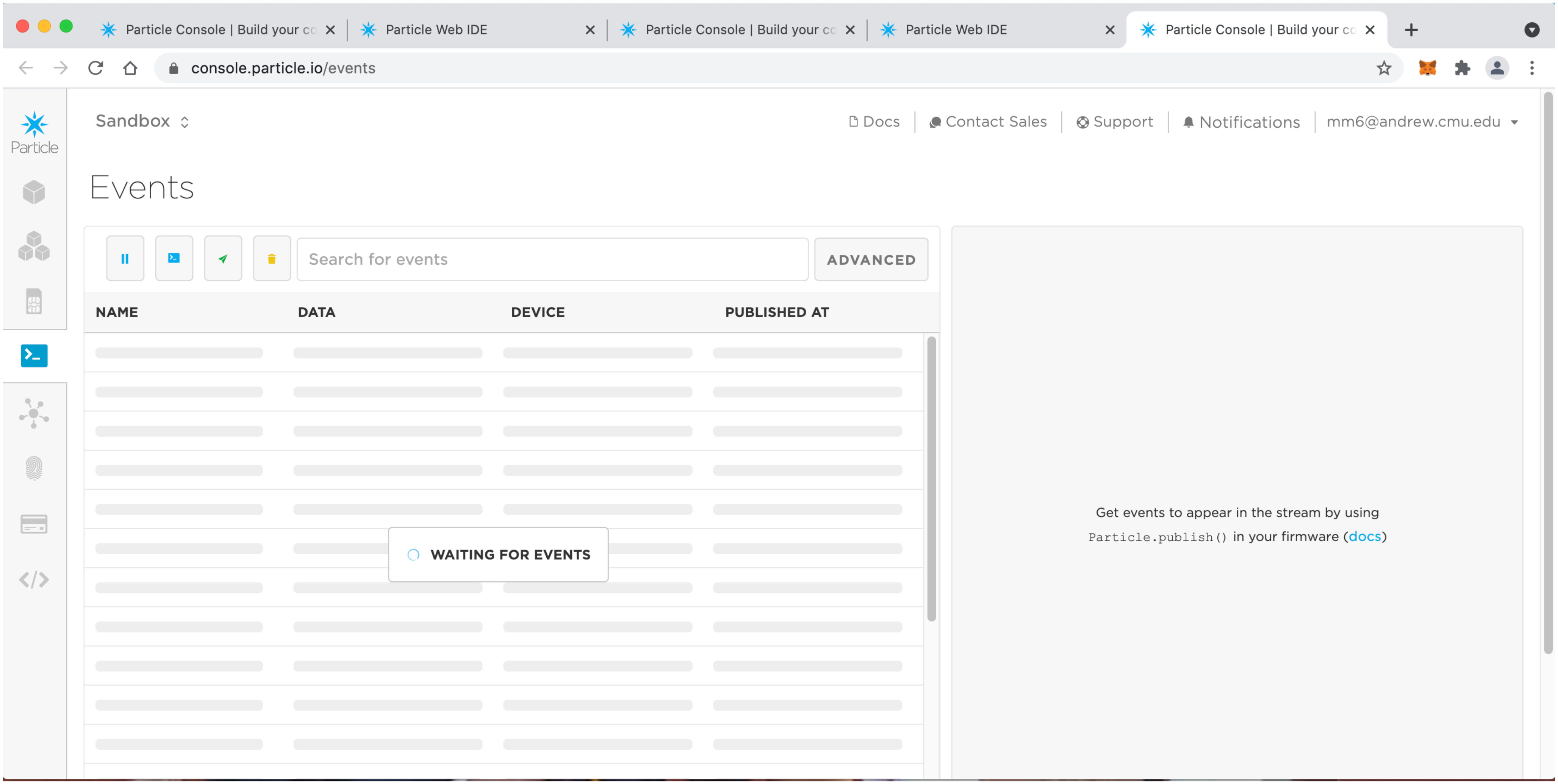
```
// C++ variables defined

int timeCtr = 0;
int NUMSECONDS = 10;

void setup()  {
      // goes once
}

void loop() {
      // called repeatedly by system code
      if (timeCtr <= millis()) {
            // publish to Particle the event name and a JSON string
            Particle.publish(name,String(buf));
            timeCtr = millis() + (NUMSECONDS * 1000);
      }
}
```

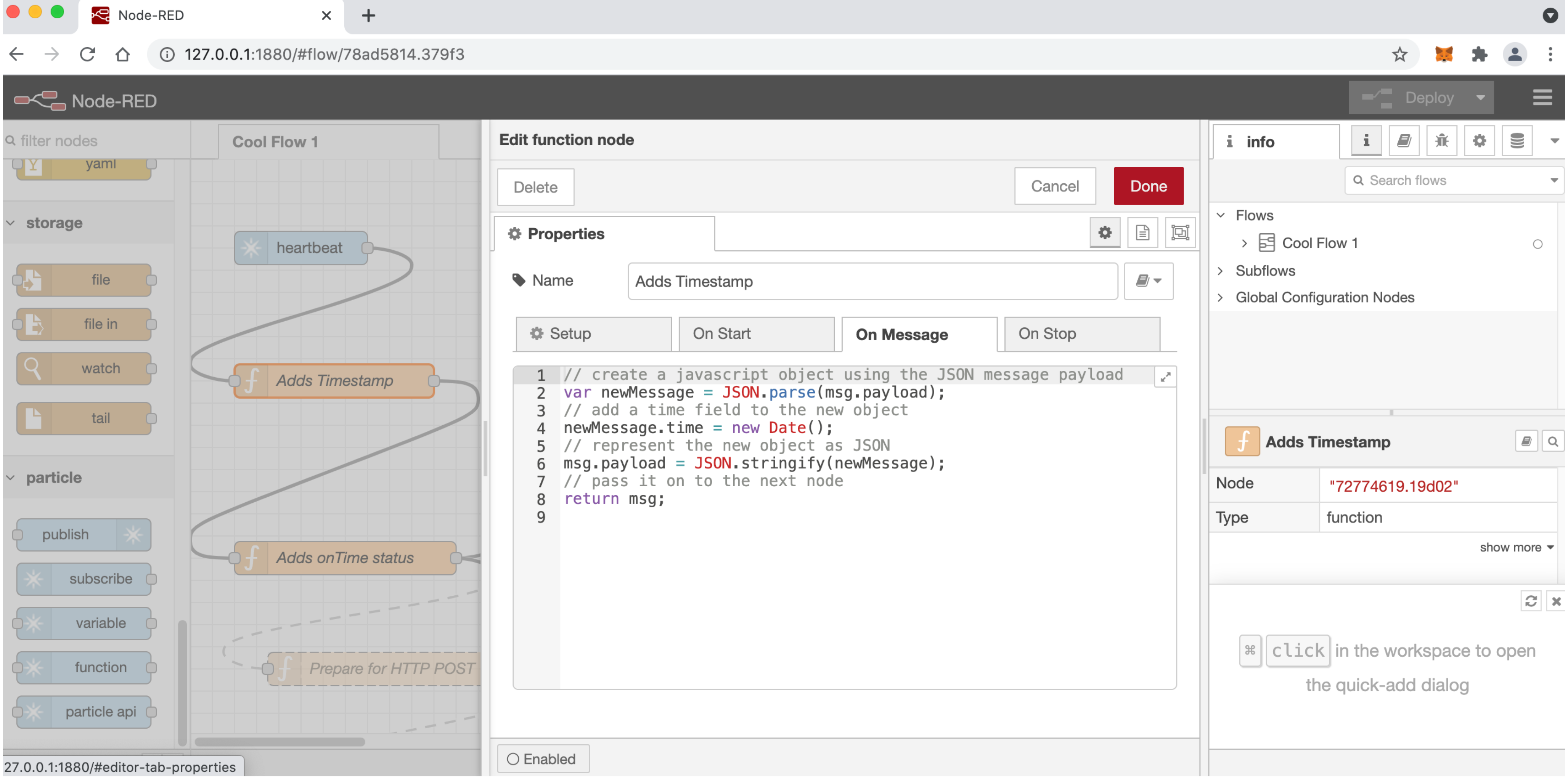
Particle Console



Get events to appear in the stream by using `Particle.publish()` in your firmware (docs)

The calls will appear on the Particle console

Node-RED



Name: Adds Timestamp

```
// create a javascript object using the JSON message payload
var newMessage = JSON.parse(msg.payload);
// add a time field to the new object
newMessage.time = new Date();
// represent the new object as JSON
msg.payload = JSON.stringify(newMessage);
// pass it on to the next node
return msg;
```

Adds Timestamp

Node	"72774619.19d02"
Type	function

The Particle subscribe node gets the heartbeat JSON string. Here, we add a timestamp.

Project 1 Part 2 Handling Heartbeat Data with Node-RED

Handle timestamps in a Node-RED node

- Get the last visit timestamp from node memory.
- Find time of this visit.
- If this visit minus the last visit > 12 seconds then set onTime to false.
- Otherwise, set onTime to true.
- JSON.stringify the updated object.
- Store time of this visit in node memory
- What if the Argon is fine but the internet is slow?

Project 1 Part 3
A Simple Web Site using Node.js

ViewSimpleMessage.js (1)

```
// ViewSimpleMessage.js
// Display a simple message on a browser
const http = require("http");
const host = 'localhost';
const port = 8000;
// The req variable will hold request information
// from the browser.
// The res variable is used to send results back
// to the browser.
const simpleListener = function (req, res) {
    res.writeHead(200);
    res.end("A simple text message on a browser");
};
```

Global values.
Assignments executed
only once.

Runs on each
HTTP visit.

ViewSimpleMessage.js (2)

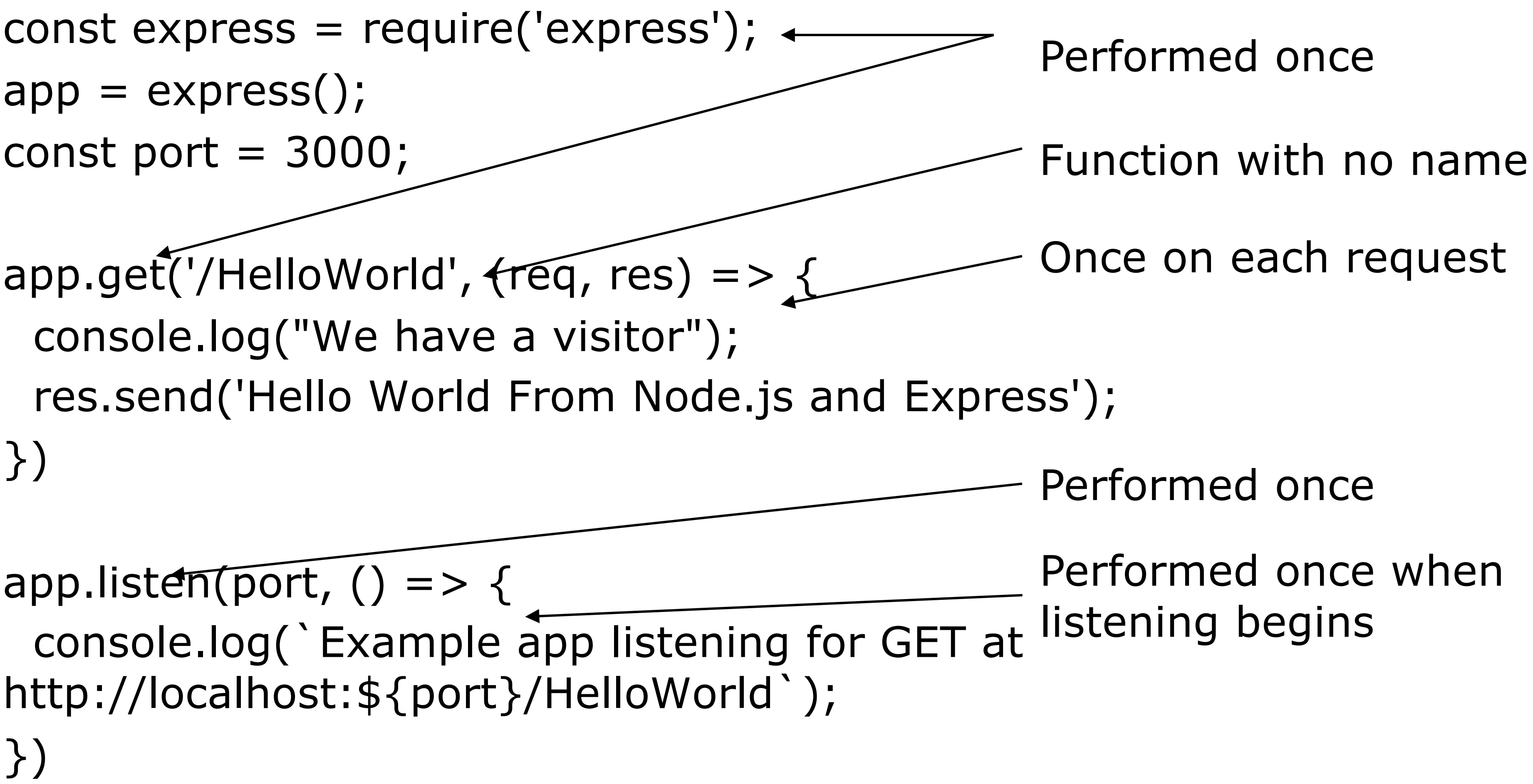
```
// Associate the server with the listener
const server = http.createServer(simpleListener);

// Begin handling browser visits
server.listen(port, host, () => {
    // runs when listening begins
    console.log(`Server is running on
    http://${host}:${port}`);
});
```

Runs once

Project 1 Part 4
A Simple Web Site using Node.js and Express

Web Server using Node & Express



```
const express = require('express');
app = express();
const port = 3000;

app.get('/HelloWorld', (req, res) => {
  console.log("We have a visitor");
  res.send('Hello World From Node.js and Express');
})

app.listen(port, () => {
  console.log(`Example app listening for GET at
http://localhost:${port}/HelloWorld`);
})
```

Performed once

Function with no name

Once on each request

Performed once

Performed once when listening begins

Project 1 Part 5
Node-RED to Node.js and Express

viewLastHeartBeat.js (1)

```
const express = require('express')
const port = 3000
app = express();

// initialize lastVisit
var lastVisit = 0;

// We need to parse the body of the post request
// from Node-RED
var bodyParser = require('body-parser')
// and we need to parse JSON data
app.use(bodyParser.json() );
```

Handle two different vistors

Executed once. lastVisit is available in function bodies

viewLastHeartBeat.js (2)

```
// Handle a visit from a browser calling with GET.
// return the last visit of Node-RED.
app.get('/ViewLastHeartBeat', (req, res) => {
  console.log('Browser visit for last heartbeat');
  // respond to browser
  res.send('Last time Argon visited via Node-RED ' + lastVisit);
})
```

When will this not be 0?

viewLastHeartBeat.js (3)

```
// This function is called with an HTTP POST by Node-RED.
// The HTTP request has a content-type header set to
// application/json.
// The JSON data has deviceID, time, and onTime values.
app.post('/SetNewHeartBeat', function (req, res) {
  console.log('Visit from Argon ');
  console.log(req.body);
  console.log(req.body.deviceID)
  lastVisit = req.body.time;
  // respond to Node-RED
  res.send('Argon update received');
})

app.listen(port, () => {
  console.log(`Browser views last heartbeat at
http://localhost:${port}/ViewLastHeartBeat`)
})
```

Let NR tell us the time?

Carnegie
Mellon
University

Project 1 Part 6
A browser visits with AJAX

Asynchronous JS And XML (AJAX)

index.html

```
<script type=“text/javascript” src=“Ajax.js”></script>
 <script type =”text/javascript” src =“ArgonStatus.js”></script>
  :
 <button onClick=“getStatus()”>Get Device Update</button>
  :
   <div id = "deviceID"></div>
   <div id = "lastVisit"></div>
   <div id = "onTime"></div>
  :
  :
 <script>getStatus()</script>
```

Load HTML and JS in a browser
and then make a call to the local JS

ArgonStatus.js (1)

```
// A call on getStatus causes an HTTP GET request back to the
// server.
// The response data is available to the updateStatus() function.
function getStatus() {
   var req = newXMLHttpRequest();
   req.onreadystatechange = getReadyStateHandler(req,
                                                 updateStatus);
   req.open("GET", "getStatusInJSON", true);
   req.setRequestHeader("Content-Type",
                        "application/x-www-form-urlencoded");
   req.send();
}
```

ArgonStatus.js (2)

```
// Call back handler to update the HTML
// when a response arrives
function updateStatus(statusJSON) {
    // create an object from the JSON string
    var statusObj =JSON.parse(statusJSON);
    var time = statusObj.lastVisit;
    var deviceID = statusObj.deviceID
    var onTime = statusObj.onTime;
    // place the response data in the HTML
    document.getElementById("lastVisit").innerHTML = time;
    document.getElementById("deviceID").innerHTML = deviceID;
    document.getElementById("onTime").innerHTML = onTime;
}
```

Ajax.js (1)

```
// Ajax.js
// Returns a new XMLHttpRequest object, or false if the browser
// doesn't support it

function newXMLHttpRequest() {

    var xmlreq = false;

    // Create XMLHttpRequest object in non-Microsoft browsers
    if (window.XMLHttpRequest) {
        xmlreq = new XMLHttpRequest();

    } else if (window.ActiveXObject) { … handle Windows case…
    return xmlreq;
}
```

Ajax.js (2)

```
function getReadyStateHandler(req, responseXmlHandler) {
        return function () {
        // If the request's status is "complete"
        if (req.readyState == 4) {
             if (req.status == 200) {
                // Pass the payload of the response to the handler
                // function.
                responseXmlHandler(req.response);
            } else {  … handle errors …
```

What function actually gets called?

index.js on server side

// uses public directory to hold index.html, and javascript files
// lastVisitDate has the current date and time

app.get function
// handles HTTP GET /getStatusInJSON for AJAX visits and
// returns the last visit..you need to modify

app.post function
// handles HTTP POST /SetNewHeartBeat for Node-RED visits and
// returns an acknowlegement

Carnegie
Mellon
University

Project 1 Part 7
Using Websockets

Serving a browser (1)

One directory responding to the browser on full page visits

```
index.html
   <script src = “websocket.js”></script>
   <script src = “microcontrollerstatus.js”></script>
    :
   <ul id=“output”>
```

server.js

provides three URL’s for index.html, websocket.js, and microcontroller.js

Accessing index.html causes the others to be fetched.

Serving a browser (2)

websockets.js

```
    var wsUri = 'ws://localhost:6969’;
    var websocket = new WebSocket(wsUri);
    // Client-initiated send text to the websocket
    function sendText(msg) {
        console.log("sending text: " + msg);
        websocket.send(msg);
    }
    // A callback function invoked for each new message from
    // the server
    websocket.onmessage = function(evt) { onMessage(evt) };
    function onMessage(evt) {
        console.log("received: " + evt.data);
        updateStatus(evt.data);      // next slide
    }
```

Send text over the socket

Receive text from peer

Serving a browser (3)

microcontrollerstatus.js

```
  function updateStatus(msg) {
     var msgObj = JSON.parse(msg);
     var id = msgObj.deviceID;
     var time = msgObj.time;
     var contents = document.getElementById("output");
                                          contents.innerHTML = "";
      var listItem = document.createElement("li");
      listItem.appendChild(
      document.createTextNode(id+" arrived at "+time));
      contents.appendChild(listItem);
  }
```

Websocket Service(1)

Another directory servicing web sockets to browsers and Node-RED

```
// server.js
const express = require('express');
const http = require('http');
const WebSocket = require('ws');

const port = 6969;
const server = http.createServer(express);
const wss = new WebSocket.Server({ server })
```

Websocket Service (2)

```
wss.on('connection', function connection(ws) {
  console.log("Connection established");
  ws.on('message', function incoming(data) {
    wss.clients.forEach(function each(client) {
      if (client !== ws && client.readyState ===
          WebSocket.OPEN) {
                client.send(data);
          }
    })
  })
})
server.listen(port, function() {
  console.log(`Server is listening on ${port}!`)
})
```