The University Of Sheffield

Joinme/WiFi, Sigfox, LoraWan and NB-IoT

COM3505, Lecture 6 (wk 7)

Prof Hamish Cunningham

Week 7...

1. provisioning, and connectivity options
2. joinme/wifi
3. choosing projects!

Headless Device Provisioning

Unwrap your shiney new thing and connect it to…?

Either:

- the device has to ship with network credentials, or
- support end-user configuration

Options...

IoT connectivity options

- wifi: Ex09 (below)
- but: wifi is designed for high bandwidth devices, which may be wasteful (our IoT gizmos are probably not going to be streaming HD video…)
- a variety of LPWAN (low power wide area network) options are coming that:
  - exploit IoT potential very low data rates & low power radio options
  - leading examples:
    - LoRa, LoRaWAN
    - Ultra Narrow Band (UNB), e.g. Sigfox
    - LTE-M and NB-IoT (but not widely operational yet)
- (one solution to all this choice is the 'throw in the kitchen sink' approach — e.g. Pycom supporting 5 networks https://www.youtube.com/watch?v=uW8XQxh1h08)

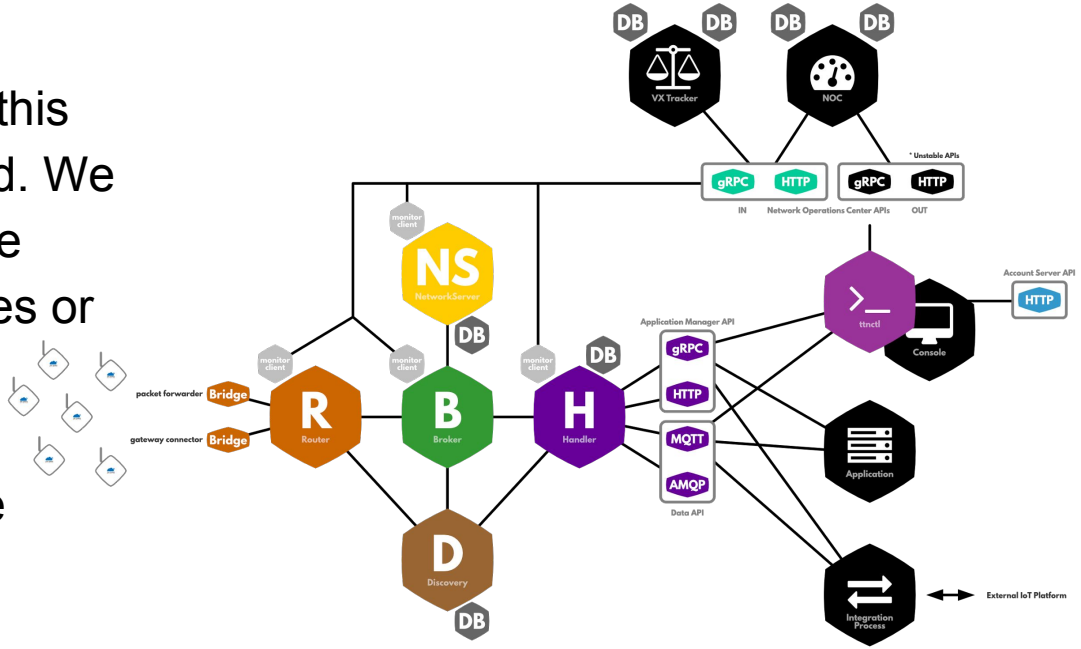
LoRa, LoRaWAN

- LoRa is a radio protocol, spread spectrum (~125 kHz) frequency-modulated chirp
- LoRaWAN is a media access control (MAC) protocol that layers on LoRa
- upload / download are symmetrical
    - LPWAN overview 0:36-6:00
    - LoRa vs LTE 11:51-12:50
    - LoRaWAN 13:10-
    - Commercial vs community 13:44-15:08

The Things Network

The Things Network is a crowdfunded LoRaWAN initiative that enables distributed LoRaWAN. Manifesto: "Everything that carries power will be connected to Internet eventually. Controlling the network that makes this possible means controlling the world. We believe that this power should not be restricted to a few people, companies or nations. Instead this should be distributed over as many people as possible without the possibility to be taken away by anyone."

Coming soon to the top of the Arts tower! (With SigFox too.)



UNB, Sigfox

Ultra narrow band:

- uses less spectrum than LoRa, and experiences lower noise (interference) levels as a result
- upload / download are asymmetrical (download, or network to device, is lower bandwidth than upload, or device to network)
- the tech under Sigfox and NB IoT

Sigfox is a global LPWAN network operator that:

- tries to build adoption at the device (hardware) level by minimising connections costs
- makes money by selling bandwidth
- competes with current mobile telecoms providers

NB IoT, LTE-M, etc.

The mobile operators (working together as 3GPP) have made several responses to LPWAN competition:

- NB IoT is recently standardised, not rolled out as yet; it will deploy “in-band” in spectrum allocated to Long Term Evolution (LTE)
- LTE-M is more mature and “allows IoT devices to connect directly to a 4G network, without a gateway and on batteries”

Other protocols:

- ZigBee, but this is more about PANs (personal area networks) than WANs
- BlueTooth and BlueTooth LE, ditto

These make sense as device-to-gateway protocols, and if we assume that the gateway has mains power, then it is likely to use WiFi or wired ethernet…

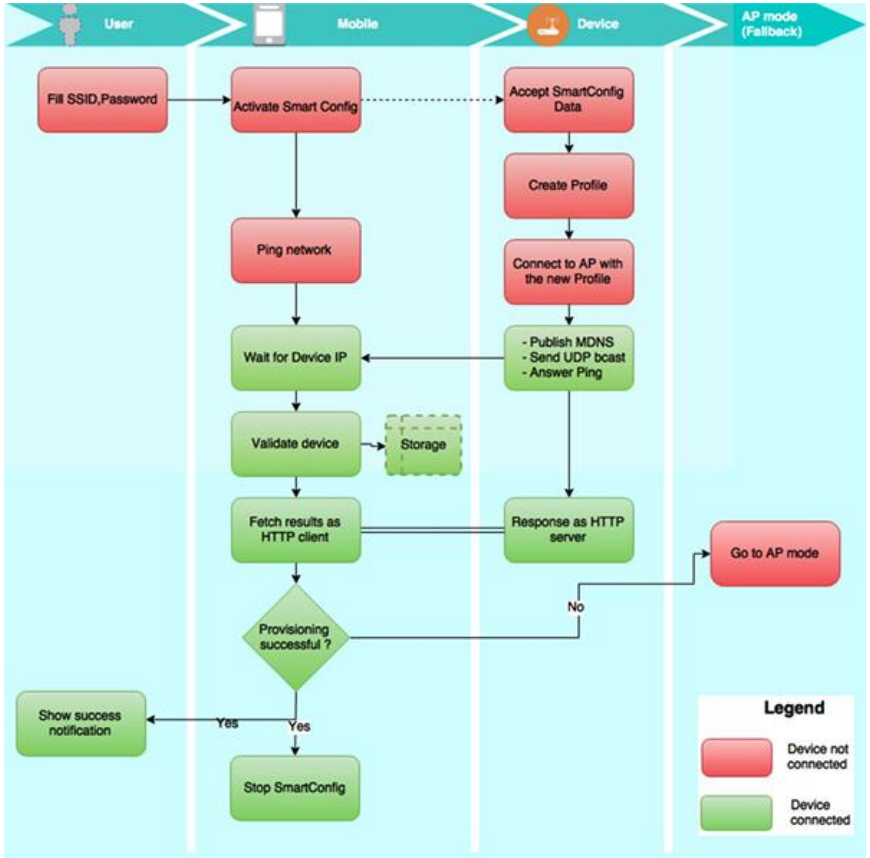
Which brings us back full circle to: ...

Wifi-based provisioning

Options:

- WiFi Protected Setup (**WPS**) — some forms shown to be crackable in 2011, now often disabled
- **SmartConfig** — mostly TI only, and the non-AES version relies on security by obscurity (ouch!), but AES version is secure (ish?!)
- device as access point::
  AP running WPA + HTTP(S) / SSL = **as secure as normal** wifi & web (!)

SmartConfig flow:



An aside: KRACK — patch your machines now!

"KRACK (Key Reinstallation Attack) ... on the WPA ... By repeatedly resetting the nonce transmitted in the third step of the WPA2 handshake, an attacker can gradually match encrypted packets seen before and learn the full keychain used to encrypt the traffic." (Wikipedia)

The linux Arduino.tgz has been updated; Windows: coming; home etc.: reinstall

Ex 9: Joinme/WiFi (1)

Joinme: an ESP library for access point device provisioning (aka **Ex09**):

- the ESP provides an **access point**, called e.g. Thing-MY-MAC-ADDR
- **users connect** to that access point using a phone or laptop or etc., and:
  - either they navigate to a known IP address (192.168.4.1 by default), or
  - the ESP may run a **captive portal** DNS (replies to all requests with the IP of a configuration server, or sign in server)
- the ESP also runs a webserver, which lists **other wifi APs** and allows entry of access keys
- when configured with the correct credentials, the ESP connects to an AP as a client (and can shut down its own AP as required)

Ex 9: Joinme/WiFi (2)

```
// Ex09.ino
// adapting access point and web server to allow connection of a device to an
// arbitrary wifi network

void setup09() { // initialisation
  // in previous exercises I've chained all the setups, but now things are
  // getting more complex I've amalgamated all the setup code so far here...

  Serial.begin(115200); // initialise the serial line
  getMAC(MAC_ADDRESS);  // store the MAC address as a chip identifier
  pinMode(BUILTIN_LED, OUTPUT);       // set up GPIO pin for built-in LED

  dln(startupDBG, "\nsetup09...");
  blink(5);                           // blink the on-board LED to say "hi"

  firstSliceMillis = millis();        // remember when we began
  lastSliceMillis = firstSliceMillis; // an approximation to use in 1st loop

  startAP();             // fire up the AP...
  initWebServer();       // ...and the web server
  WiFi.begin();          // for when credentials are already stored by board
  blink(5);              // blink the on-board LED to say "bye"
}
```

Ex 9: Joinme/WiFi (3)

```
const char *templatePage[] = {    // we'll use Ex07 templating to build pages
  "<html><head><title>",                                                    //  0
  "default title",                                                          //  1
  "</title>\n",                                                             //  2
  "<meta charset='utf-8'>",                                                 //  3
  "<meta name='viewport' content='width=device-width, initial-scale=1.0'>\n"
  "<style>body{background:#FFF; color: #000; font-family: sans-serif;",    //  4
  "font-size: 150%;}</style>\n",                                            //  5
  "</head><body>\n",                                                        //  6
  "<h2>Welcome to Thing!</h2>\n",                                           //  7
  "<!-- page payload goes here... -->\n",                                   //  8
  "<!-- ...and/or here... -->\n",                                           //  9
  "\n<p><a href='/'>Home</a>   </p>\n",                      // 10
  "</body></html>\n\n",                                                     // 11
};

void loop09() { // the workhorse
  webServer.handleClient(); // serve pending web requests every loop
}
```

Ex 9: Joinme/WiFi (4)

```
void initWebServer() { // changed naming conventions to avoid clash with Ex06
  // register callbacks to handle different paths
  webServer.on("/", hndlRoot);               // slash
  webServer.onNotFound(hndlNotFound);        // 404s...
  webServer.on("/generate_204", hndlRoot);   // Android captive portal support
  webServer.on("/L0", hndlRoot);             // TODO is this...
  webServer.on("/L2", hndlRoot);             // ...IoS captive portal...
  webServer.on("/ALL", hndlRoot);            // ...stuff?
  webServer.on("/wifi", hndlWifi);           // page for choosing an AP
  webServer.on("/wifichz", hndlWifichz);     // landing page for AP form submit
  webServer.on("/status", hndlStatus);       // status check, e.g. IP address

  webServer.begin();
  dln(startupDBG, "HTTP server started");
}

// webserver handler callbacks
void hndlNotFound() {
  dbg(netDBG, "URI Not Found: ");
  dln(netDBG, webServer.uri());
  webServer.send(200, "text/plain", "URI Not Found");
}
```

Ex 9: Joinme/WiFi (5)

```
void hndlRoot() {
  dln(netDBG, "serving page notionally at /");
  replacement_t repls[] = { // the elements to replace in the boilerplate
    {  1, apSSID.c_str() },
    {  8, "" },
    {  9, "<p>Choose a <a href=\"wifi\">wifi access point</a>.</p>" },
    { 10, "<p>Check <a href='/status'>wifi status</a>.</p>" },
  };
  String htmlPage = ""; // a String to hold the resultant page
  GET_HTML(htmlPage, templatePage, repls); // GET_HTML sneakily added to Ex07
  webServer.send(200, "text/html", htmlPage);
}
void hndlWifi() {
  dln(netDBG, "serving page at /wifi");
  String form = ""; // a form for choosing an access point and entering key
  apListForm(form);
  replacement_t repls[] = { // the elements to replace in the boilerplate
    { 1, apSSID.c_str() },
    { 7, "<h2>Network configuration</h2>\n" },
    { 8, "" },
    { 9, form.c_str() },
  };
  String htmlPage = ""; // a String to hold the resultant page
  GET_HTML(htmlPage, templatePage, repls); // GET_HTML sneakily added to Ex07
  webServer.send(200, "text/html", htmlPage);
}
```

Ex 9: Joinme/WiFi (6)

```
// Ex09.ino
// adapting access point and web server to allow connection of a device to an
// arbitrary wifi network

void setup09() { // initialisation
  // in previous exercises I've chained all the setups, but now things are
  // getting more complex I've amalgamated all the setup code so far here...

  Serial.begin(115200); // initialise the serial line
  getMAC(MAC_ADDRESS);  // store the MAC address as a chip identifier
  pinMode(BUILTIN_LED, OUTPUT);       // set up GPIO pin for built-in LED

  dln(startupDBG, "\nsetup09...");
  blink(5);                           // blink the on-board LED to say "hi"

  firstSliceMillis = millis();        // remember when we began
  lastSliceMillis = firstSliceMillis; // an approximation to use in 1st loop

  startAP();            // fire up the AP...
  initWebServer();      // ...and the web server
  WiFi.begin();         // for when credentials are already stored by board
  blink(5);             // blink the on-board LED to say "bye"
}

const char *templatePage[] = {    // we'll use Ex07 templating to build pages
  "<html><head><title>",                                                //  0
  "default title",                                                      //  1
```

Ex 9: Joinme/WiFi (7)

```
void hndlWifichz() {
  String title = "<h2>Joining wifi network...</h2>";
  String message = "<p>Check <a href='/status'>wifi status</a>.</p>";
  String ssid = "";          String key = "";
  for(uint8_t i = 0; i < webServer.args(); i++ ) {
    if(webServer.argName(i) == "ssid")
      ssid = webServer.arg(i);
    else if(webServer.argName(i) == "key")
      key = webServer.arg(i);
  }
  if(ssid == "") {
    message = "<h2>Ooops, no SSID...?</h2>\n<p>Looks like a bug :-(</p>";
  } else {
    char ssidchars[ssid.length()+1];
    char keychars[key.length()+1];
    ssid.toCharArray(ssidchars, ssid.length()+1);
    key.toCharArray(keychars, key.length()+1);
    WiFi.begin(ssidchars, keychars);
  }
  replacement_t repls[] = { // the elements to replace in the template
    { 1, apSSID.c_str() },      { 7, title.c_str() },
    { 8, "" },                  { 9, message.c_str() },
  };
  String htmlPage = "";      // a String to hold the resultant page
  GET_HTML(htmlPage, templatePage, repls);
  webServer.send(200, "text/html", htmlPage);
}
```

Ex 9: Joinme/WiFi (8)

```
void hndlStatus() {          // UI for checking connectivity etc.
  dln(netDBG, "serving page at /status");
  String s = "<ul>\n\n<li>SSID: "; s += WiFi.SSID(); s += "</li>"; s += "\n<li>Status: ";
  switch(WiFi.status()) {
    case WL_IDLE_STATUS:      s += "WL_IDLE_STATUS</li>"; break;
    case WL_NO_SSID_AVAIL:    s += "WL_NO_SSID_AVAIL</li>"; break;
    case WL_SCAN_COMPLETED:   s += "WL_SCAN_COMPLETED</li>"; break;
    case WL_CONNECTED:        s += "WL_CONNECTED</li>"; break;
    case WL_CONNECT_FAILED:   s += "WL_CONNECT_FAILED</li>"; break;
    case WL_CONNECTION_LOST:  s += "WL_CONNECTION_LOST</li>"; break;
    case WL_DISCONNECTED:     s += "WL_DISCONNECTED</li>"; break;
    default:                  s += "unknown</li>";
  }
  s += "\n<li>Local IP: ";     s += ip2str(WiFi.localIP());    s += "</li>\n";
  s += "\n<li>Soft AP IP: ";   s += ip2str(WiFi.softAPIP());   s += "</li>\n";
  s += "\n<li>AP SSID name: "; s += apSSID;                    s += "</li>\n";
  s += "</ul></p>";
  replacement_t repls[] = { // the elements to replace in the boilerplate
    { 1, apSSID.c_str() },    { 7, "<h2>Status</h2>\n" },
    { 8, "" },                { 9, s.c_str() },
  };
  String htmlPage = ""; // a String to hold the resultant page
  GET_HTML(htmlPage, templatePage, repls); // GET_HTML sneakily added to Ex07
  webServer.send(200, "text/html", htmlPage);
}
```

Ex 9: Joinme/WiFi (9)

```
void apListForm(String& f) { // utility to create a form for choosing AP
  const char *checked = " checked";
  int n = WiFi.scanNetworks();
  dbg(netDBG, "scan done: ");

  if(n == 0) {
    dln(netDBG, "no networks found");
    f += "No wifi access points found :-( ";
    f += "<a href='/'>Back</a><br/><a href='/wifi'>Try again?</a></p>\n";
  } else {
    dbg(netDBG, n); dln(netDBG, " networks found");
    f += "<p>Wifi access points available:</p>\n<p><form method='POST' action='wifichz'> ";
    for(int i = 0; i < n; ++i) {
      f.concat("<input type='radio' name='ssid' value='");       f.concat(WiFi.SSID(i));
      f.concat("'");  f.concat(checked);      f.concat(">");     f.concat(WiFi.SSID(i));
      f.concat(" ("); f.concat(WiFi.RSSI(i)); f.concat(" dBm)"); f.concat("<br/>\n");
      checked = "";
    }
    f += "<br/>Pass key: <input type='textarea' name='key'><br/><br/> ";
    f += "<input type='submit' value='Submit'></form></p>";
  }
}
String ip2str(IPAddress address) { // utility for printing IP addresses
  return String(address[0]) + "." + String(address[1]) + "." +
         String(address[2]) + "." + String(address[3]);
}
```

Ex 11: captive portal

```
#include <DNSServer.h>

DNSServer dnsServer;
const byte DNS_PORT = 53;

void setup11() {
  setup09();              // UI for choosing and joining wifi networks
  dln(startupDBG, "\nsetup11...");                  // say hi
  dnsServer.start(DNS_PORT, "*", WiFi.softAPIP()); // DNS * to our IP
}

void loop11() {
  dnsServer.processNextRequest(); // handle DNS  requests
  webServer.handleClient();       // handle HTTP requests
}
```

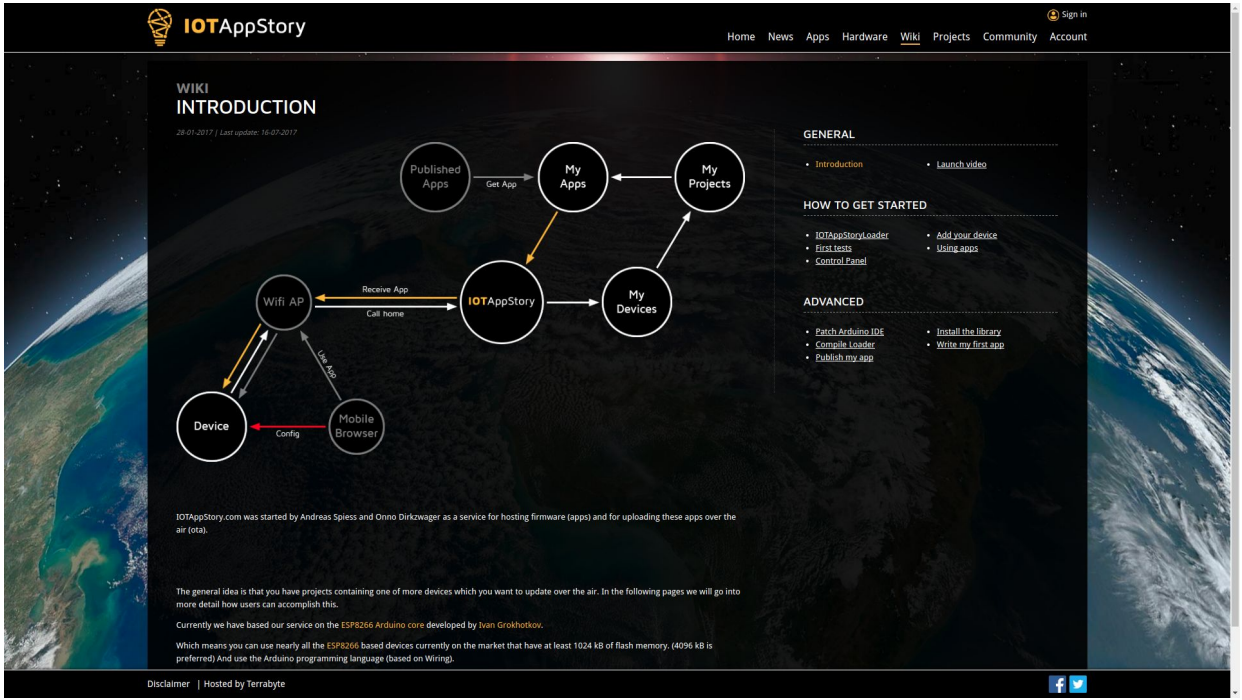
Summary: Joinme/WiFi

	ESP32		USER
1.	advertise wifi AP	→	join wifi
		⋮	
2.	DNS * to AP IP	←	request any website
		⋮	
3.	HTTP 200	→	list of available wifi APs
		⋮	
4.	restart wifi as client of AP	←	choose AP, enter key, POST
		⋮	
5.	(optionally) shut down AP		switch to cloud or other UI

AP-based config + OTA (Ex10) = ….?

IoTAppStory.com from our friend Andreas Spiess:

- device polls firmware repo
- when updates available triggers OTA update
- possible IoT app store model… (though currently mainly ESP8266)



Next up….

- MOLE quiz 2 tomorrow
- **Ex12** order the project options according to your 1st/2nd/3rd choices and send to our cloud server (and **push your code**!)
  1. air quality monitor
  2. campus panic button
  3. RoboThing
  4. WaterElf: sustainable food tech
  5. Peer-to-Peer voting systems
  6. other
- Parameter names: 1st, 2nd, 3rd, e.g. “`1st=4&2nd=5&3rd=1`”
- Next week: distribute project hardware… (first push first served!)

Your **TODO** list for week 7:

- Update your git repository clone as usual
- Read and digest Notes/Week07.mkd
- Do exercise 12
- Do the reading
- Make sure you understood the lecture, and review the slides if needed
- **Lab this week**:
  - two hours in NC_PC on **Tues at 10am**

NOTE: Friday 10th Nov meeting postponed!