Today's Topics

- Mobile Networking
- HTTP
- HTML
- Webserver!
The computer ... manages the spreading of ninety-four percent of all TV programs, conveys all information about weather, air traffic, special deals... and records every business conversation, every contract...

Computers have changed the world. Computers are the civilisation. If we turn them off, we will fall back to a kind of civilisation, of which we have forgotten how it even works.

The aim of a computer network is to have computers communicate with each other.

Partial map of the Internet based on the January 15, 2005 data found on opte.org. Each line is drawn between two nodes, representing two IP addresses. The length of the lines are indicative of the delay between those two nodes. This graph represents less than 30% of the Class C networks reachable by the data collection program in early 2005.

Wireless Internet

- WLAN = Wireless Local Area Network
- Allows “local” computers to communicate
The Arduino ESP8266 shield allows the Arduino to connect to networks, and also to set up its own network.

## Controlling a Modem

Modems receive:
- *data* to be sent
- *commands* to control them

The modem is controlled by so-called AT commands.

## Connecting the Modem

View from above – connectors are at the bottom.

These are connected to the serial output (TX) of the Arduino.

Details in assignment sheet.
Connecting the Modem

Details in assignment sheet

Controlling a Modem

- Modems are controlled by so-called AT commands (AT = “Attention”)
- AT commands allow to
  - connect to networks,
  - create networks
  - set communication speeds…

The modem is controlled by so-called AT commands
# ESP8266 AT Instruction Set

## Table of Contents

1. Preambles ................................................................. 6
2. Command Description ............................................... 7
3. Basic AT Command Set ............................................... 8
   3.1. Overview ............................................................ 8
   3.2. Commands .......................................................... 9
      1. AT – Test AT startup ............................................. 9
      2. AT+RST – Restart module ...................................... 9
      3. AT+QMR – View version info ................................... 9
      4. AT+GSLP – Enter deep sleep mode .......................... 10
      5. ATE – AT commands echo ...................................... 10
      6. AT+RSTORE – Factory reset .................................... 10
      7. AT+UART – UART configuration ............................... 11
      8. AT+UART_CUR – current UART configuration ............. 12
      9. AT+UART_DEF – default UART configuration .............. 13
     10. AT+GMR – View version info ................................... 14
     11. AT+RFVDD – set RF TX Power according to VDD33 .......... 14
     12. AT+RFPOWER – set maximum value of RF TX Power ........ 15
3.3. WiFi Functions Overview ......................................... 16
   4.1. Commands .......................................................... 18
      1. AT+CWMODE – WiFi mode ..................................... 18
      2. AT+CWMODE_CUR – current WiFi mode ................. 19
      3. AT+CWMODE_DEF – default WiFi mode .................... 20
      4. AT+CNOAP – Connect to AP .................................. 21
      5. AT+CWLAP – List available APs ............................ 24
      6. AT+CWSAP_CUR – Connect to AP, save as default ....... 23
      7. AT+CWSAP – Configuration of SoftAP mode ............ 25
      8. AT+CWSAP_CUR – current config of SoftAP mode .... 26

Source: https://cdn.sparkfun.com/assets/learn_tutorials/4/0/3/4A-ESP8266_AT_Instruction_Set_EN_v0.30.pdf

(Will be linked from Webpage)
Creating a Network

- We want to create a Wifi access point
- Allows other devices to connect

\[ \text{AT+CWMODE=2} \]

Wifi mode

Set up an SSID

- Sets the name by which other devices can identify the network
- Also sets password (8–64 characters), authentication mode, and channel (1–12)

\[ \text{AT+CWSAP=}"PFE","12345678",1,4" \]

SSID  Password  Channel

Use your own SSID and a safe password!
Sending Commands

- The modem is connected to the serial port
- It replies with "OK" if everything is fine

```c
int issueCommand(char *command) {
  // handle request
  Serial.println(command);
  delay(10);
  // The modem replies with "OK"
  // if everything worked well
  if (!Serial.find("OK"))
    return 0; // Error
  return 1;
}
```

Need to signal this to the user (e.g. through display or blinking LED)

WiFi Setup

```c
void setup() {
  // Initialize serial connection
  Serial.begin(115200);
  Serial.setTimeout(5000);
  // Set mode to wifi access point
  if (!issueCommand("AT+CWMODE=2"))
    return;
  // Enable wifi access point with SSID "PFE"
  if (!issueCommand("AT+CWSAP="PFE","12345678",1,4"))
    return;
}
```

Demo
Ports

- We want to run a service on the device
- Every computer provides ports for network IP connections
- Ports are numbered from 1 to 65535
- Every service has its own port

```
<table>
<thead>
<tr>
<th>Port</th>
<th>Protocol</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ddp</td>
<td>Routing Table Maintenance Protocol</td>
</tr>
<tr>
<td>1</td>
<td>tcp</td>
<td>TCP Port Service Multiplexer</td>
</tr>
<tr>
<td>2</td>
<td>ddp</td>
<td>Name Binding Protocol</td>
</tr>
<tr>
<td>2</td>
<td>udp</td>
<td>Management Utility</td>
</tr>
<tr>
<td>2</td>
<td>tcp</td>
<td>Management Utility</td>
</tr>
<tr>
<td>3</td>
<td>udp</td>
<td>Compression Process</td>
</tr>
<tr>
<td>3</td>
<td>tcp</td>
<td>Compression Process</td>
</tr>
<tr>
<td>4</td>
<td>ddp</td>
<td>AppleTalk Echo Protocol</td>
</tr>
<tr>
<td>4</td>
<td>udp</td>
<td>Unassigned</td>
</tr>
<tr>
<td>4</td>
<td>tcp</td>
<td>Unassigned</td>
</tr>
<tr>
<td>5</td>
<td>udp</td>
<td>Remote Job Entry</td>
</tr>
<tr>
<td>5</td>
<td>tcp</td>
<td>Remote Job Entry</td>
</tr>
<tr>
<td>6</td>
<td>ddp</td>
<td>Zone Information Protocol</td>
</tr>
<tr>
<td>6</td>
<td>tcp</td>
<td>Unassigned</td>
</tr>
<tr>
<td>6</td>
<td>udp</td>
<td>Unassigned</td>
</tr>
<tr>
<td>7</td>
<td>udp</td>
<td>Echo Protocol</td>
</tr>
<tr>
<td>7</td>
<td>tcp</td>
<td>Unassigned</td>
</tr>
<tr>
<td>85</td>
<td>udp</td>
<td>MIT ML Device</td>
</tr>
<tr>
<td>85</td>
<td>tcp</td>
<td>MIT ML Device</td>
</tr>
</tbody>
</table>
```
### File Transfer Protocol (FTP)

- **Protocol:** FTP
- **Port:** 20/udp, 20/tcp
- **Description:** Default Data File Transfer

### Secure Shell (SSH)

- **Protocol:** SSH
- **Ports:** 22/udp, 22/tcp
- **Description:** Remote Login Protocol

### Simple Mail Transfer Protocol (SMTP)

- **Protocol:** SMTP
- **Ports:** 25/udp, 25/tcp
- **Description:** Liefert E-Mail aus
<table>
<thead>
<tr>
<th>Service</th>
<th>Port</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>netrjs-4</td>
<td>74/udp</td>
<td>Remote Job Service</td>
</tr>
<tr>
<td>netrjs-4</td>
<td>74/tcp</td>
<td>Remote Job Service</td>
</tr>
<tr>
<td></td>
<td>75/udp</td>
<td>Any private dial out service</td>
</tr>
<tr>
<td></td>
<td>75/tcp</td>
<td>Any private dial out service</td>
</tr>
<tr>
<td>does</td>
<td>76/udp</td>
<td>Distributed External Object Store</td>
</tr>
<tr>
<td>does</td>
<td>76/tcp</td>
<td>Distributed External Object Store</td>
</tr>
<tr>
<td></td>
<td>77/udp</td>
<td>Any private RJE service</td>
</tr>
<tr>
<td></td>
<td>77/tcp</td>
<td>Any private RJE service</td>
</tr>
<tr>
<td>vettcp</td>
<td>78/udp</td>
<td>vettcp</td>
</tr>
<tr>
<td>vettcp</td>
<td>78/tcp</td>
<td>vettcp</td>
</tr>
<tr>
<td>finger</td>
<td>79/udp</td>
<td>Finger</td>
</tr>
<tr>
<td>finger</td>
<td>79/tcp</td>
<td>Finger</td>
</tr>
<tr>
<td>http</td>
<td>80/udp</td>
<td>World Wide Web HTTP</td>
</tr>
<tr>
<td>http</td>
<td>80/tcp</td>
<td>World Wide Web HTTP</td>
</tr>
<tr>
<td>hosts2-ns</td>
<td>81/udp</td>
<td>Name Server</td>
</tr>
<tr>
<td>hosts2-ns</td>
<td>81/tcp</td>
<td>Name Server</td>
</tr>
<tr>
<td>xfer</td>
<td>82/udp</td>
<td>XFER Utility</td>
</tr>
<tr>
<td>xfer</td>
<td>82/tcp</td>
<td>XFER Utility</td>
</tr>
<tr>
<td>mit-ml-dev</td>
<td>83/udp</td>
<td>MIT ML Device</td>
</tr>
<tr>
<td>mit-ml-dev</td>
<td>83/tcp</td>
<td>MIT ML Device</td>
</tr>
<tr>
<td></td>
<td>84/udp</td>
<td>Common Trace Facility</td>
</tr>
</tbody>
</table>

**HTTP = HyperText Transfer Protocol → Liefert website aus**

### Use your own SSID and a safe password!

- Allow multiple connections
  
  ```
  AT+CIPMUX=1
  ```

- Enable a server on a given port
  
  ```
  AT+CIPSERVER=1,80
  ```

### Web Setup

```cpp
void setup() {
  Serial.begin(115200);
  Serial.setTimeout(5000);

  // Set mode to wifi access point
  if (!issueCommand(AT+CWMODE=2))
    return;

  // Enable wifi access point with SSID "PFE"
  if (!issueCommand(AT+CWSAP="PFE","12345678",1,4))
    return;

  // Enable multiple TCP/UDP connections
  if (!issueCommand(AT+CIPMUX=1))
    return;

  // Enable TCP server on port 80
  if (!issueCommand(AT+CIPSERVER=1,80))
    return;
}
```
Receiving Data

```c
char *read_data(int *id) {
    // If a client connects, the modem sends a string
    // +IPD,<ID>,<len>[,<remote IP>,<remote port>]:<data>
    // Wait for connection from a client
    if (!Serial.findUntil("+IPD,", "\r"))
        return NULL;
    // read ID
    *id = Serial.parseInt();
    if (!Serial.findUntil("," , "\r"))
        return NULL;
    // read length
    int len = Serial.parseInt();
    // ignore until colon
    if (!Serial.findUntil( ":", "\r"))
        return NULL;
    // allocate data
    char *data = (char *)malloc(len + 1);
    if (data == NULL)
        return NULL;
    // Fill it
    Serial.readBytes(data,len);
    // And we're done
    data[len] = '\0';
    return data;
}
```

Sending Data

```c
void send_data(char *data, int id) {
    // To send data, use "AT+CIPSEND=<id>,<len>\r\n", followed by data
    int len = strlen(data);
    Serial.print("AT+CIPSEND=");
    Serial.print(id);
    Serial.print(",\r\n");
    Serial.print(data);
    delay(200);
    Serial.print("\r\n");
    Serial.print(len);
    delay(1000);
    Serial.print("\r\n");
    delay(200);
    Serial.print("\r\n");
    delay(1000);
    Serial.print("\r\n");
    delay(200);
    Serial.print("\r\n");
    delay(1000);
    Serial.print("\r\n");
}
A Web server (= a computer) waits on port 80 for a Web client (= another computer) to initiate a connection.

The client sends a request for a specific website.

The server then delivers this website.
Waiting for Empty Line

- Aside from the GET command the browser also sends information about itself
- We read until we see an empty line
- An empty line consists of two consecutive `'\n'` (newline symbol)
- There can also be `'\r'` (carriage return) characters in-between

---

Processing HTML

```c
void process_data(char *data, int id) {
    // We ignore all requests except for GET
    if (strncmp(data, "GET", strlen("GET")))
        return;
    // This is where extra processing of data
    // may take place
    send_html("<h1>Hello, world</h1>", id);
}
```

---

Sending HTML

```c
void send_html(char *data, int id) {
    // We always send the same page
    char output[2048];
    sprintf(output,
            "HTTP/1.1 200 OK\r\n\nContent-Type: text/html\r\nContent-Length: %d\r\n\n%s",
            strlen(data), data);
    send_data(output, id);
}
```
Demo

Inputs

• Idea: control LED via the website
• turn on with http://192.168.4.1/on/
• turn off with http://192.168.4.1/off/

```c
void process_data(char *data, int id) {
    // We ignore all requests except for GET
    if (strncmp(data, "GET", strlen("GET")))
        return;

    if (strncmp(data, "GET /on", strlen("GET /on")) == 0)
        { turnLedOn();
            send_html("<h1>LED is on</h1>", id);
        } else if (strncmp(data, "GET /off", strlen("GET /off")) == 0)
        { turnLedOff();
            send_html("<h1>LED is off</h1>", id);
        } else {
            send_html("<h1>Hello, world</h1>", id);
        }
}
```
Links

- In HTML by using
  `<a href="URL">text</a>`
  one can link to other websites
- URLs without a host name (www.foo.com)
  link to the same host

Outputting Links

cient.println("<p>");
cient.println("LED <a href="/on">turn on</a> ");
cient.println(" | ");
cient.println("<a href="/off">turn off</a> ");
cient.println("</p>");

produces

<p>
LED  <a href="/on">turn on</a>
|  
<a href="/off">turn off</a>
</p>
void process_data(char *data, int id) {
  // We ignore all requests except for GET
  if (strncmp(data, "GET", strlen("GET")))
    return;
  if (strncmp(data, "GET /on", strlen("GET /on")) == 0)
    { turn_led_on();
      send_html("<h1>LED is on</h1><a href="/off">turn off</a>", id);
    } else if (strncmp(data, "GET /off", strlen("GET /off")) == 0)
    { turn_led_off();
      send_html("<h1>LED is off</h1><a href="/on">turn on</a>", id);
    } else
    { send_html("<h1>Hello, world</h1><a href="/on">turn on</a>", id);
    }
}

Inputs with Links

Demo

Access Control

• Behind a router or computer your Arduino is invisible to the internet
• On the internet anyone can access your device and record “secret” URLs
• Before you put your program on the Internet, please contact your friendly computer scientist
Creating a Network
- We want to create a Wi-Fi access point
- Allows other devices to connect

```
AT+CWMODE=2
```

Web Setup

```cpp
void setup()
{
  // Initialize serial connection
  Serial.begin(115200);
  Serial.setTimeout(5000);

  // Set mode to wifi access point
  if (!issueCommand("AT+CWMODE=2"))
    return;

  // Enable wifi access point with SSID "PFE"
  if (!issueCommand("AT+CWSAP="PFE","12345678",1,4"))
    return;

  // Enable multiple TCP/UDP connections
  if (!issueCommand("AT+CIPMUX=1"))
    return;

  // Enable TCP server on port 80
  if (!issueCommand("AT+CIPSERVER=1,80"))
    return;
}
```

Receiving Data

```cpp
char *read_data(int *id)
{
  // If a client connects, the modem sends a string
  // +IPD,<ID>,<len>[,<remote IP>,<remote port>]:<data>
  // Wait for connection from a client
  if (!Serial.findUntil("+IPD,"))
    return NULL;

  // read ID
  *id = Serial.parseInt();
  if (!Serial.findUntil(",", ","))
    return NULL;

  // read length
  int len = Serial.parseInt();
  if (!Serial.findUntil(":", ","))
    return NULL;
```