USC Viterbi School of Engineering
Center for Cyber-Physical Systems and the Internet of Things
CCI

ANRG

Hands On Workshop: MQTT using MCUs and WiFi with Mbed OS

Jason Tran

Ph.D. Student, Autonomous Networks Research Group
Ming Hsieh Department of Electrical Engineering
Viterbi School of Engineering, University of Southern California
jasontra@usc.edu, http://anrg.usc.edu

Motivation

Covering ISRs, polling loops, etc. is the first step of learning embedded programming. The next step is to equip students with the ability to program richer applications. Embedded operating systems and programming design patterns can empower students to create these applications with cleaner code.

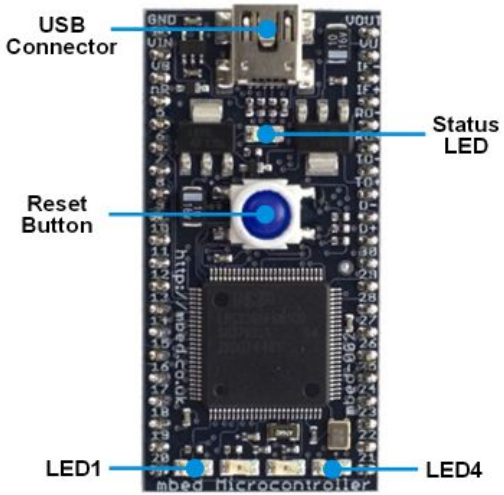
Today, we will cover a very simple design pattern with the use of some typical libraries offered by most OSes. Patterns are hard for students to wrap their head around, and teaching from a book is not very effective. We have found showing examples where a pattern is useful and having students develop using patterns helps students understand the usefulness of patterns and create richer applications (with more organized code!).

One way our undergraduate researchers learned was simulating a work environment. First, we introduce a pattern and have the student read/run more complex sample code. Then, we ask them to use the pattern to develop an idea. This forces them to understand the pattern and keep within a structure we provide (a common skill needed for developing which in turn organizes their program).

Workshop Overview

1. Hands-on: a multi-threaded IoT app built on mbed-OS in action
2. mbed-OS: overview parts of RTOS libraries
3. Dispatcher Programming Pattern
4. Demo code walkthrough
5. Demo: MQTT-SN with Pololu 3pi Robot

Hardware Components



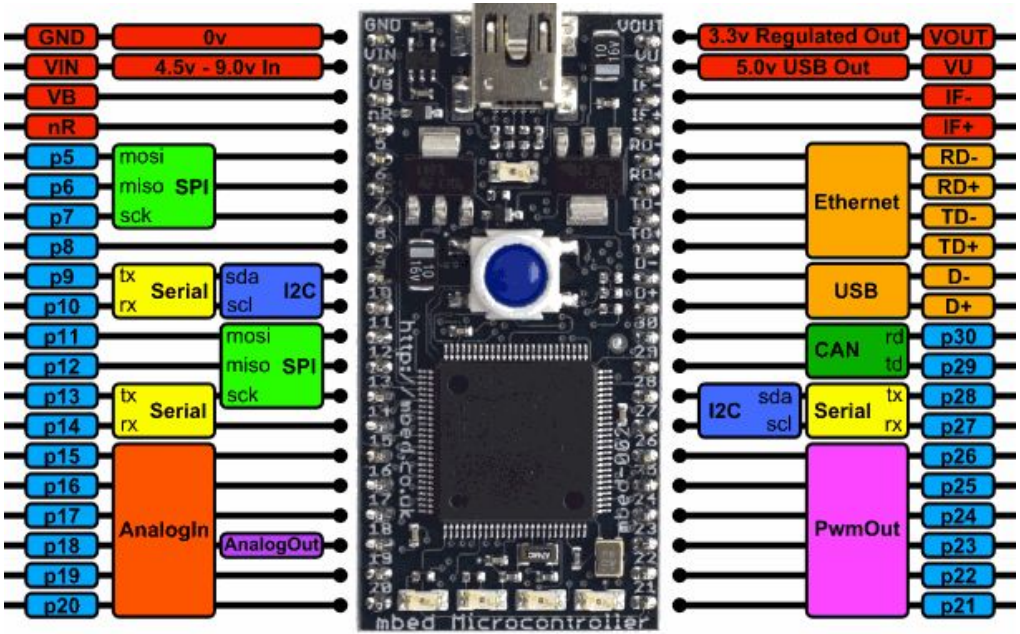
mbed LPC1768 Demo Board
ARM Cortex-M3
96Mhz, 32KB RAM, 512KB Flash

ESP8266
802.11b/g/n @ 2.4GHz
1MB Flash
Takes 3.3V

First: update LPC1768 firmware

1. Plug in LPC1768 and drag/drop the file mbedmicrontroller_141212.if
2. Unplug/plug USB cable
3. Wait for the update. If the LPC1768's drive mounts and is now empty, continue
4. Drag/drop the file mbed-esp8266-mqtt-example-slave-v3.bin
5. Eject and unplug for now

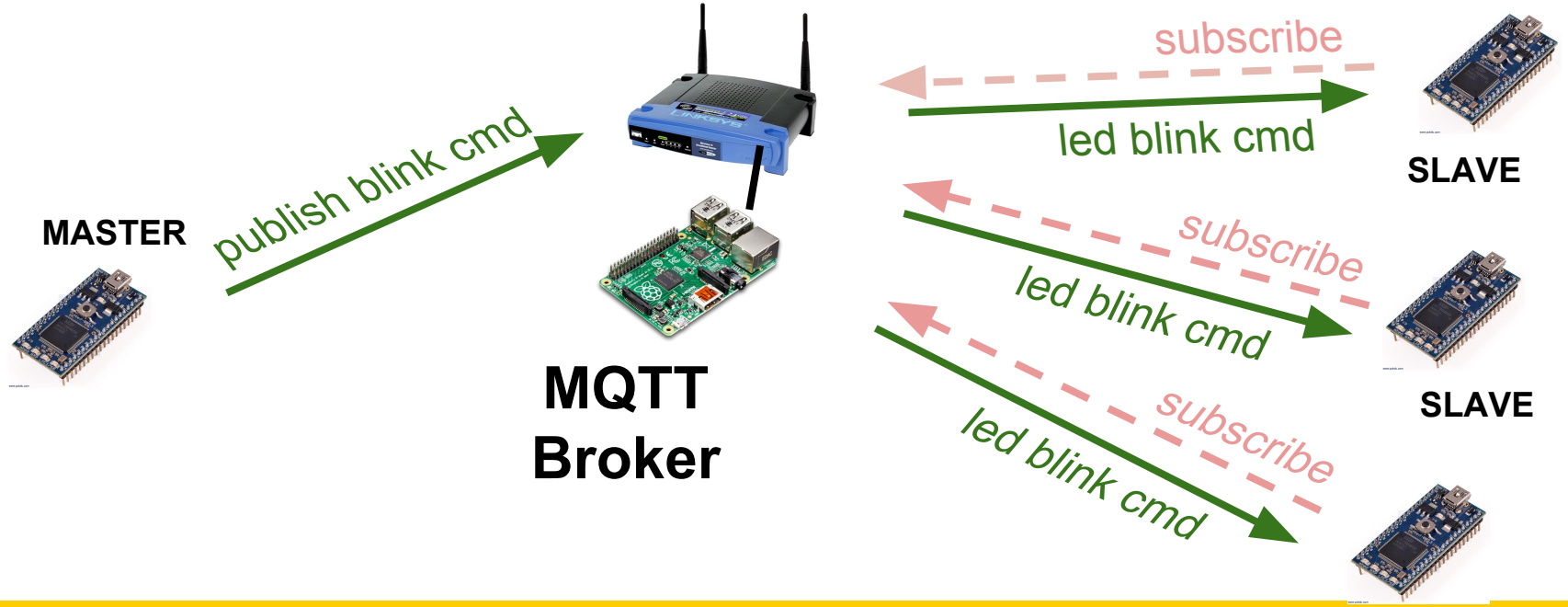
Powering your ESP8266



use 3.3v

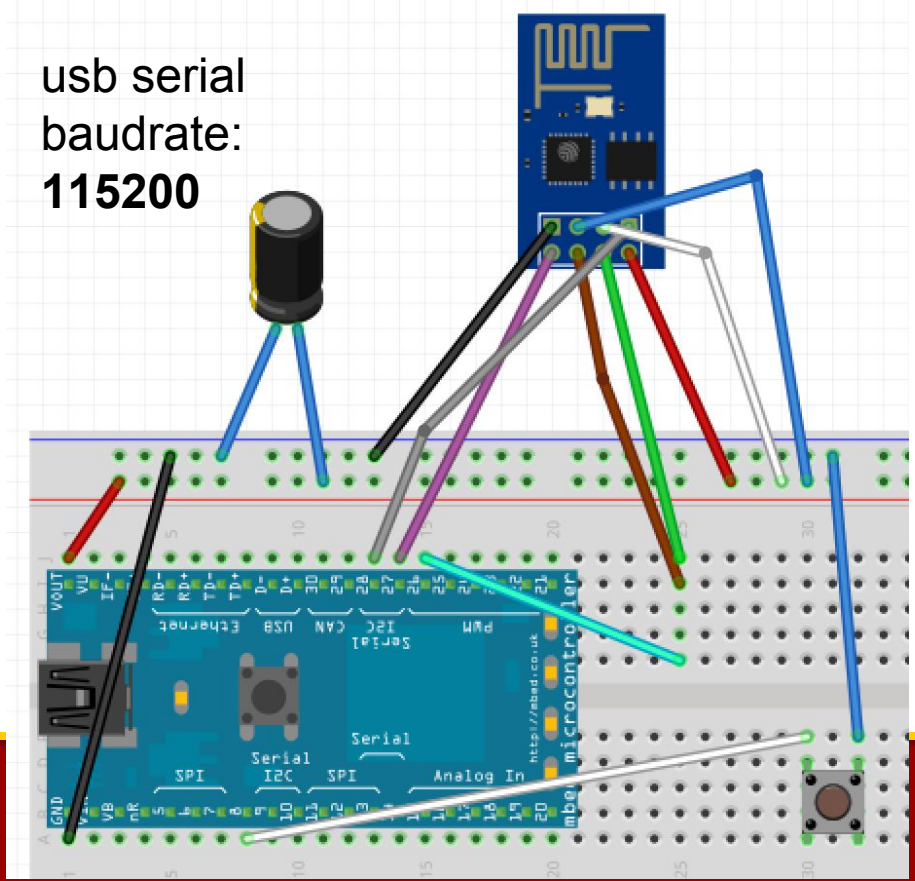
do NOT use 5v (the ESP8266 may fry)

MQTT Demo Diagram

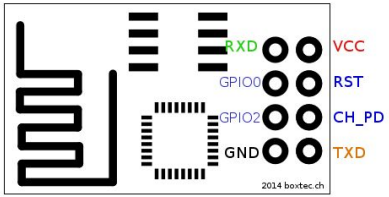


Assemble Breadboards

usb serial
baudrate:
115200

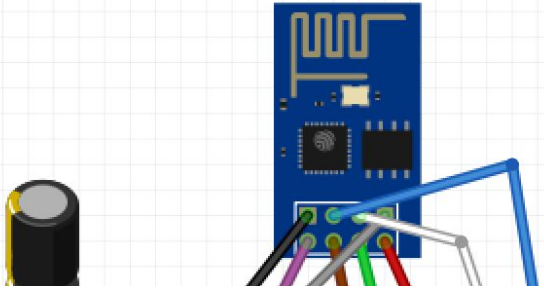


ESP8266		LPC1768
3v3/VCC	<->	3.3v
RST	<->	p26
EN/CH_PD	<->	p26
TX	<->	p27
RX	<->	p28
GPIO0	<->	3.3v
GPIO2	<->	3.3v
GND	<->	GND

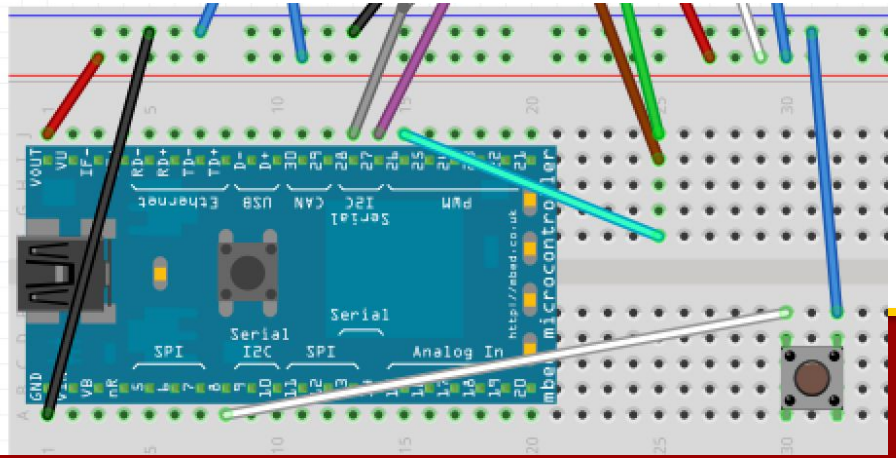


Assemble Breadboards

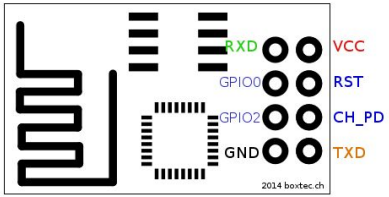
usb serial
baudrate:
115200



ESP8266 needs a very stable voltage. Placing a capacitor will stabilize the 3.3v coming from the LPC1768



ESP8266		LPC1768
3v3/VCC	<->	3.3v
RST	<->	p26
EN/CH_PD	<->	p26
TX	<->	p27
RX	<->	p28
GPIO0	<->	3.3v
GPIO2	<->	3.3v
GND	<->	GND



Mbed OS

ARM's mbed OS 5 release integrated a Real-Time Operating System at its core

RTOS API:

Thread	***Mutex***	Semaphore
Signals	Queue	MemoryPool
Mail	RtosTimer	ISRs

today's demo code uses these libraries

Embedded IoT Challenges

Popularity of IoT surged interest for richer applications on embedded devices (solution: threads)

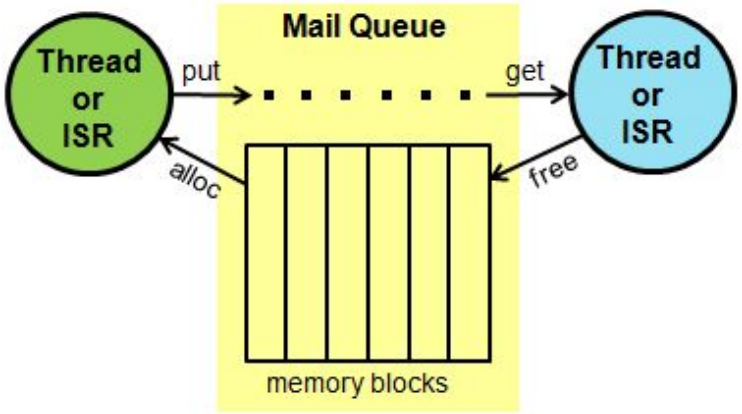
Running network stacks make software complicated (solution: threads)

Need to manage memory for incoming and outgoing packets (solution: Mail)

need an operating system!

mbed-OS: **Mail**

Why **Mail** instead of **Queue**? **Mail** works like a **Queue** with the added benefit of providing a memory pool for allocating messages (not only pointers). *Useful for packets!*



Dispatcher Pattern

```
1  while(1) {
2
3      evt = LEDMailbox.get();
4
5      if(evt.status == osEventMail) {
6          msg = (MailMsg *)evt.value.p;
7
8          switch (msg->content[1]) {
9              case LED_PUBLISH_BLINK_FAST:
10                 printf("LEDThread: pushbutton pressed. publishing blink"
11                        " fast command to master node\n");
12                 break;
13             case LED_ON_ONE_SEC:
14                 printf("LEDThread: turning LED2 on for one second...\n");
15                 break;
16             case LED_BLINK_FAST:
17                 printf("LEDThread: blinking LED2 fast for one second...\n");
18                 break;
19             default:
20                 printf("LEDThread: invalid message\n");
21                 break;
22         }
23
24         LEDMailbox.free(msg);
25     }
26 } /* while */
```

Dispatcher Pattern

```
1  while(1) {
2
3      evt = LEDMailbox.get();
4
5      if(evt.status == osEventMail) {
6          msg = (MailMsg *)evt.value.p;
7
8          switch (msg->content[1]) {
9              case LED_PUBLISH_BLINK_FAST:
10                 printf("LEDThread: pushbutton pressed. publishing blink"
11                        " fast command to master node\n");
12                 break;
13             case LED_ON_ONE_SEC:
14                 printf("LEDThread: turning LED2 on for one second...\n");
15                 break;
16             case LED_BLINK_FAST:
17                 printf("LEDThread: blinking LED2 fast for one second...\n");
18                 break;
19             default:
20                 printf("LEDThread: invalid message\n");
21                 break;
22         }
23
24         LEDMailbox.free(msg);
25     }
26 } /* while */
```

Threads sleep at this blocking call until something is put() into the mailbox. You can also call this with a timeout (RtosTimer used underneath).

Dispatcher Pattern

```
1  while(1) {
2
3      evt = LEDMailbox.get();
4
5      if(evt.status == osEventMail) {
6          msg = (MailMsg *)evt.value.p;
7
8          switch (msg->content[1]) {
9              case LED_PUBLISH_BLINK_FAST:
10                 printf("LEDThread: pushbutton pressed. publishing blink"
11                        " fast command to master node\n");
12                 break;
13             case LED_ON_ONE_SEC:
14                 printf("LEDThread: turning LED2 on for one second...\n");
15                 break;
16             case LED_BLINK_FAST:
17                 printf("LEDThread: blinking LED2 fast for one second...\n");
18                 break;
19             default:
20                 printf("LEDThread: invalid message\n");
21                 break;
22         }
23
24         LEDMailbox.free(msg);
25     }
26 } /* while */
```

This switch() filters the message using some header type byte and "dispatches" the task to be done

Dispatcher Pattern

```
1  while(1) {
2
3      evt = LEDMailbox.get();
4
5      if(evt.status == osEventMail) {
6          msg = (MailMsg *)evt.value.p;
7
8          switch (msg->content[1]) {
9              case LED_PUBLISH_BLINK_FAST:
10                 printf("LEDThread: pushbutton pressed. publishing blink"
11                        " fast command to master node\n");
12                 break;
13             case LED_ON_ONE_SEC:
14                 printf("LEDThread: turning LED2 on for one second...\n");
15                 break;
16             case LED_BLINK_FAST:
17                 printf("LEDThread: blinking LED2 fast for one second...\n");
18                 break;
19             default:
20                 printf("LEDThread: invalid message\n");
21                 break;
22         }
23
24         LEDMailbox.free(msg);
25     }
26 } /* while */
```

Frees the previously allocated Mail message

Code Walkthrough

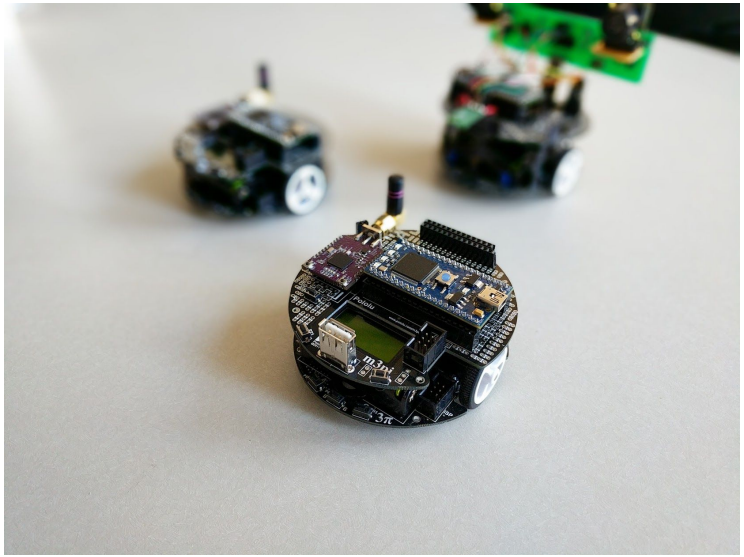
Fast paced walk through the code:

- MQTT library how to
- Code for Threads and Mail

Code and slides will be updated and documented on github for later exploration. Questions welcome after the session!

http://github.com/ANRGUSC/mbed-esp8266-mqtt-example

Demo: ANRG Robots



Autonomous Networks
Research Group
http://anrg.usc.edu/

ROMANO Protocol

A protocol built on top of MQTT to build robot control applications.

https://www.youtube.com/playlist?list=PLhYQAfTaJ0ANCYDoyiGNsThtuFwBbwKLs

Quick MQTT Overview

- Publish-subscribe pattern based lightweight messaging protocol (TCP/IP or UDP/IP)
- Messages are dispatched by a centralized **MQTT Broker**
- Nodes subscribe to a topic and receive any messages published to the topic

Quick MQTT Overview

Topic example: “myHome/livingRoom/temp”

(subtopics separated by “/”, can filter via subtopics)

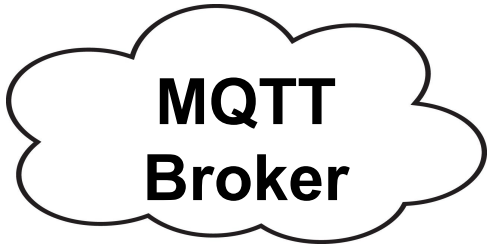
Advantages:

- Loose coupling from publishers and subscribers
- Scalable
- Made today’s demo easier to prepare
- Simplicity makes it more student friendly

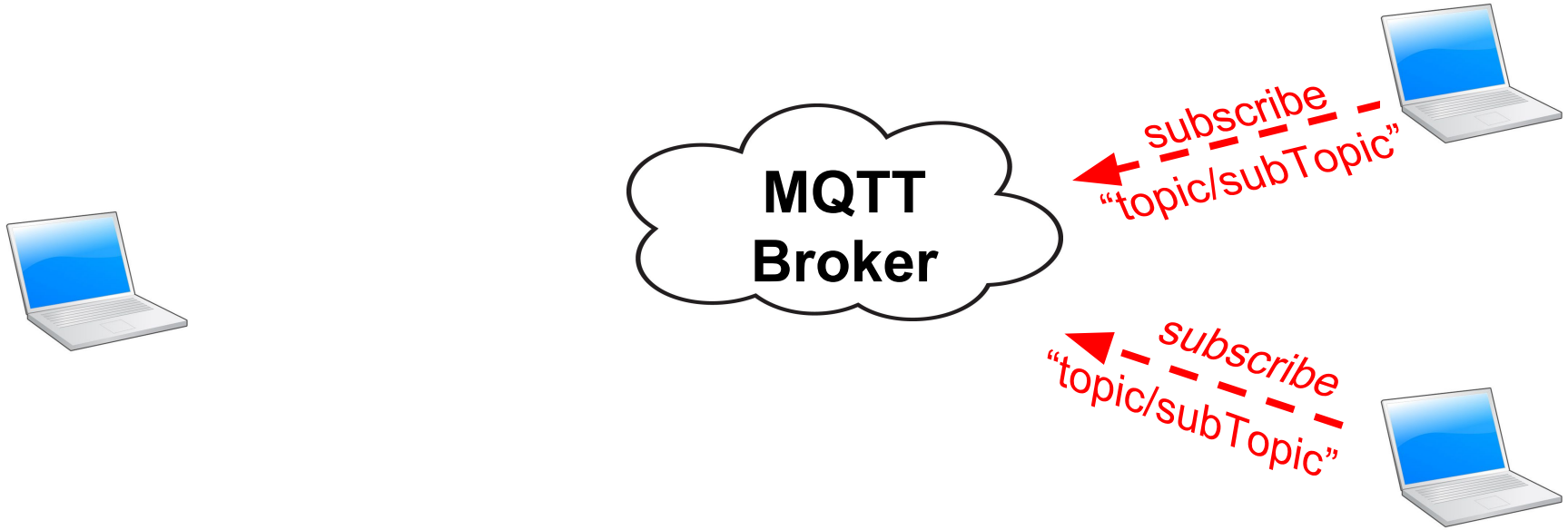
Disadvantage:

- Broker needs to be highly available (cannot go down)
- Students find embedded libraries slightly difficult

MQTT: Message Queue Telemetry Transport



MQTT: Message Queue Telemetry Transport



MQTT: Message Queue Telemetry Transport