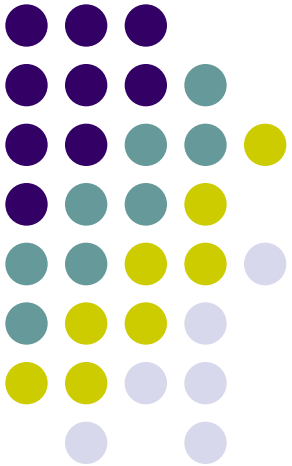
Chapter 11: TCP/IP, WiFi, MQTT

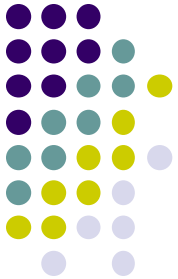
EE2405

嵌入式系統與實驗

Embedded System Lab

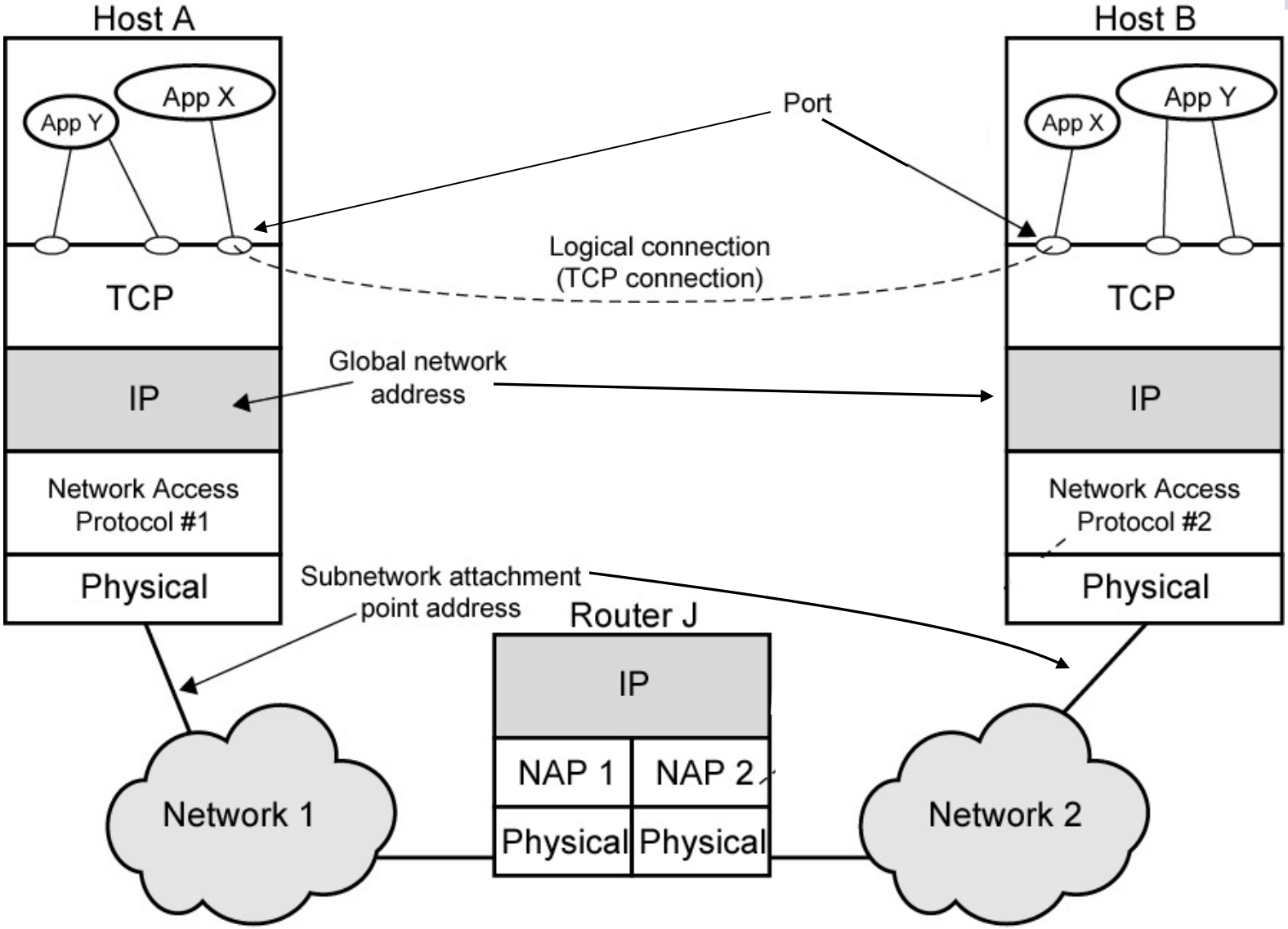


TCP/IP Protocol Suite

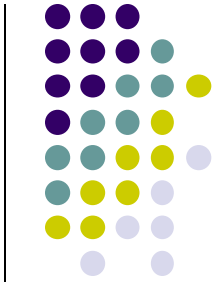


- Most widely used interoperable network protocol architecture
- Specified and extensively used before OSI
  - OSI was slow to take place in the market
- Funded by the US Defense Advanced Research Project Agency (DARPA) for its packet switched network (ARPANET)
  - DoD automatically created an enormous market for TCP/IP
- Used by the Internet and WWW

Operation of TCP and IP

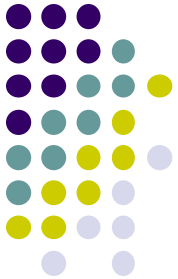


Application Layer



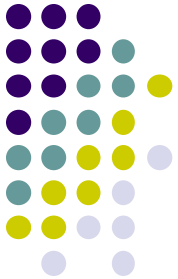
- Widely-known Application layer protocols are those used for the exchange of user information:
  - The Hypertext Transfer Protocol (HTTP) is used to transfer files that make up the Web pages of the World Wide Web.
  - The File Transfer Protocol (FTP) is used for interactive file transfer.
  - The Simple Mail Transfer Protocol (SMTP) is used for the transfer of mail messages and attachments.
  - Telnet, a terminal emulation protocol, is used for logging on remotely to network hosts.

Transport Layer



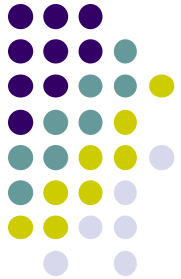
- End-to-end data transfer
- Transmission Control Protocol (TCP)
  - connection oriented
  - reliable delivery of data
  - ordering of delivery
- User Datagram Protocol (UDP)
  - connectionless service
  - delivery is not guaranteed

Internet Layer



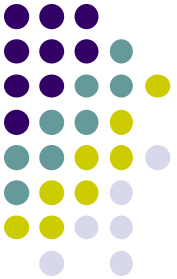
- Connectionless, point to point internetworking protocol (uses the datagram approach)
  - takes care of routing across multiple networks
  - each packet travels in the network independently of each other
    - they may not arrive (if there is a problem in the network)
    - they may arrive out of order
  - a design decision enforced by DoD to make the system more flexible and responsive to loss of some subnet devices
- Implemented in end systems and routers as the Internet Protocol (IP)

IP addresses



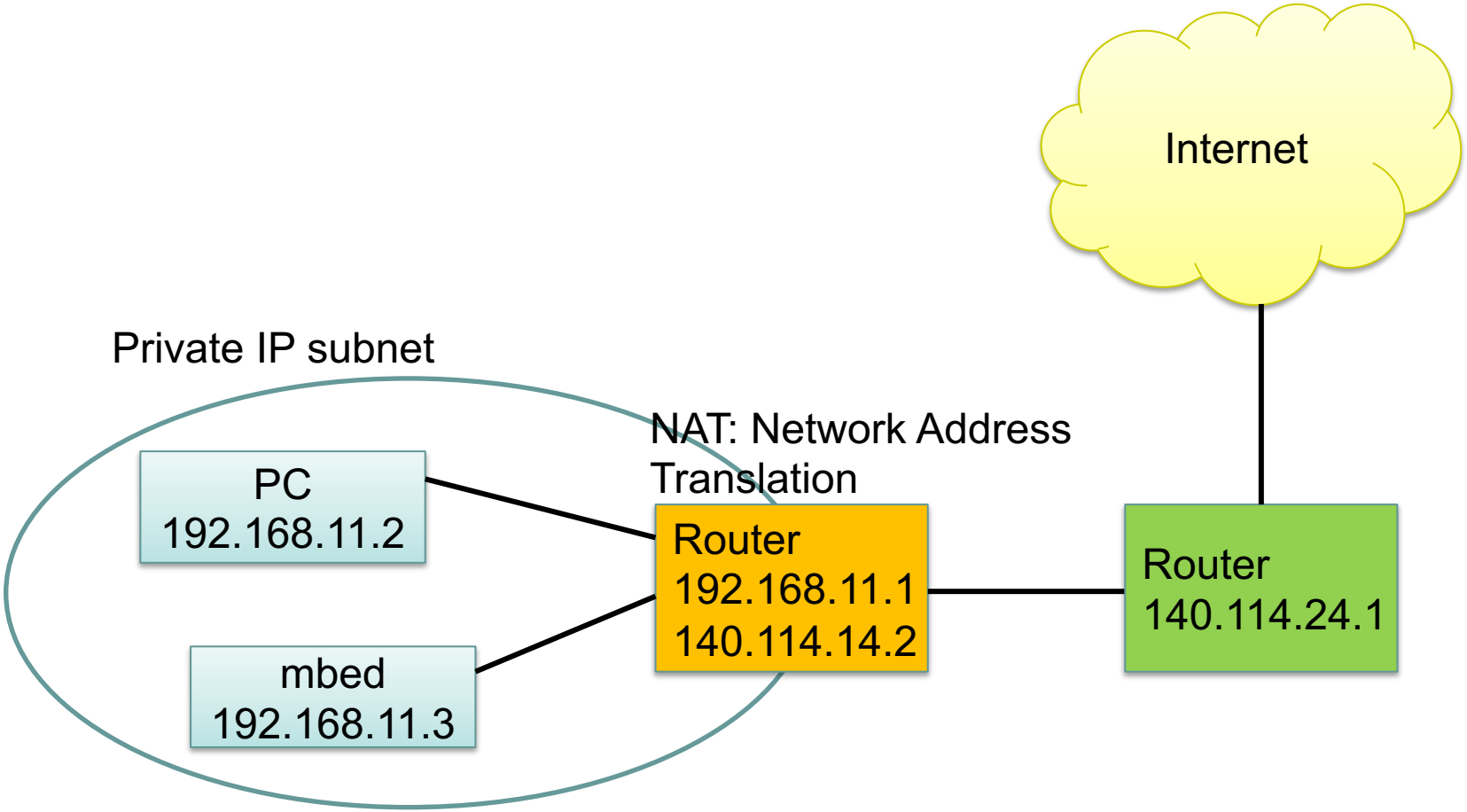
- IPv4 --- 32 bit addresses
  - e.g. 192.168.11.2
  - Each device gets one (or more)
    - Subnet addresses can be reused by NAT with a router
  - In theory there are about 4 billion available
- IPv6 --- 128 bit
- Allocation controlled by ICANN

Internetworking

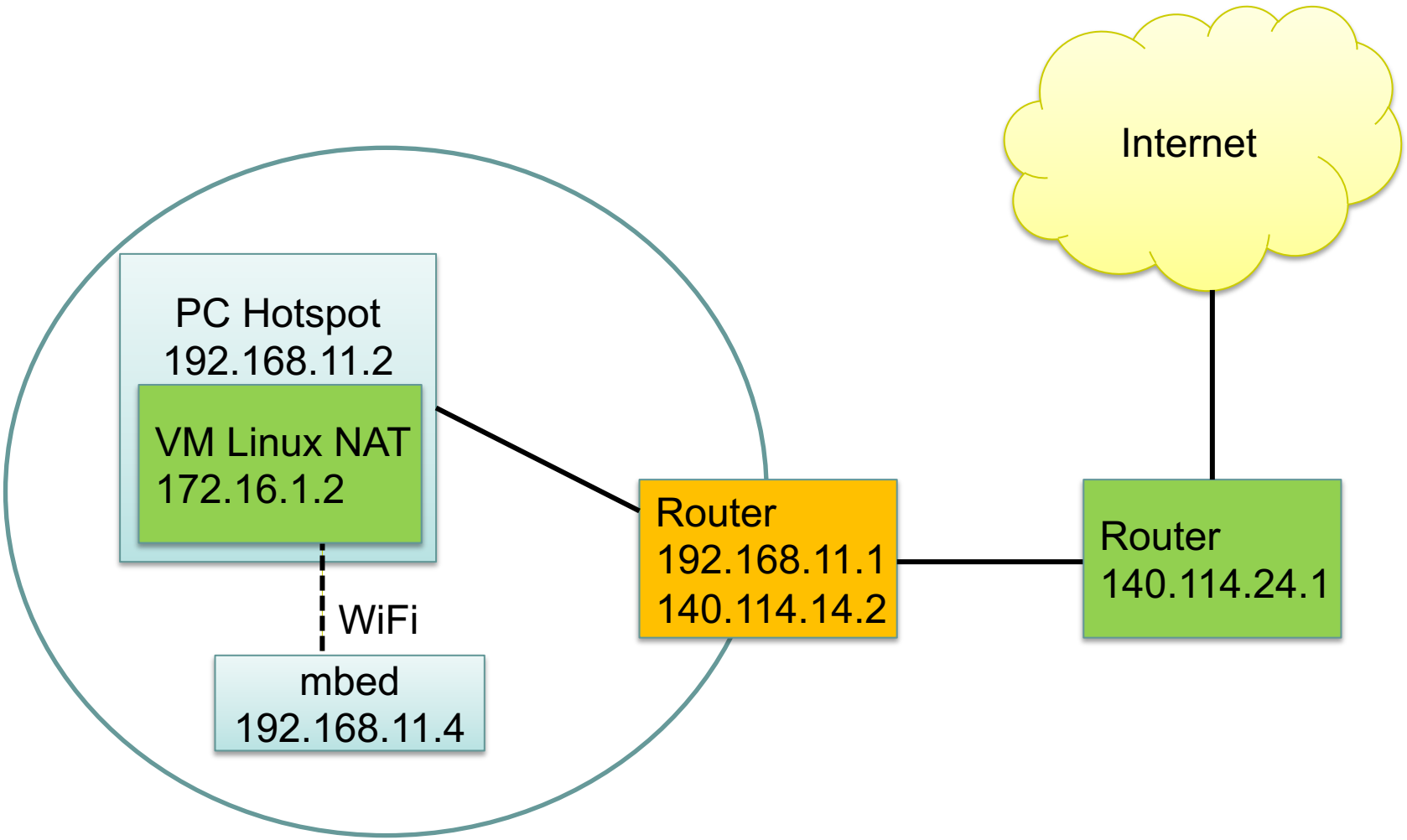


- Interconnected set of networks
  - May be seemed as a large network
- Each constituent network is a *subnetwork*
- Routers interconnect subnetwork
  - To translate addresses
  - To accommodate max packet size

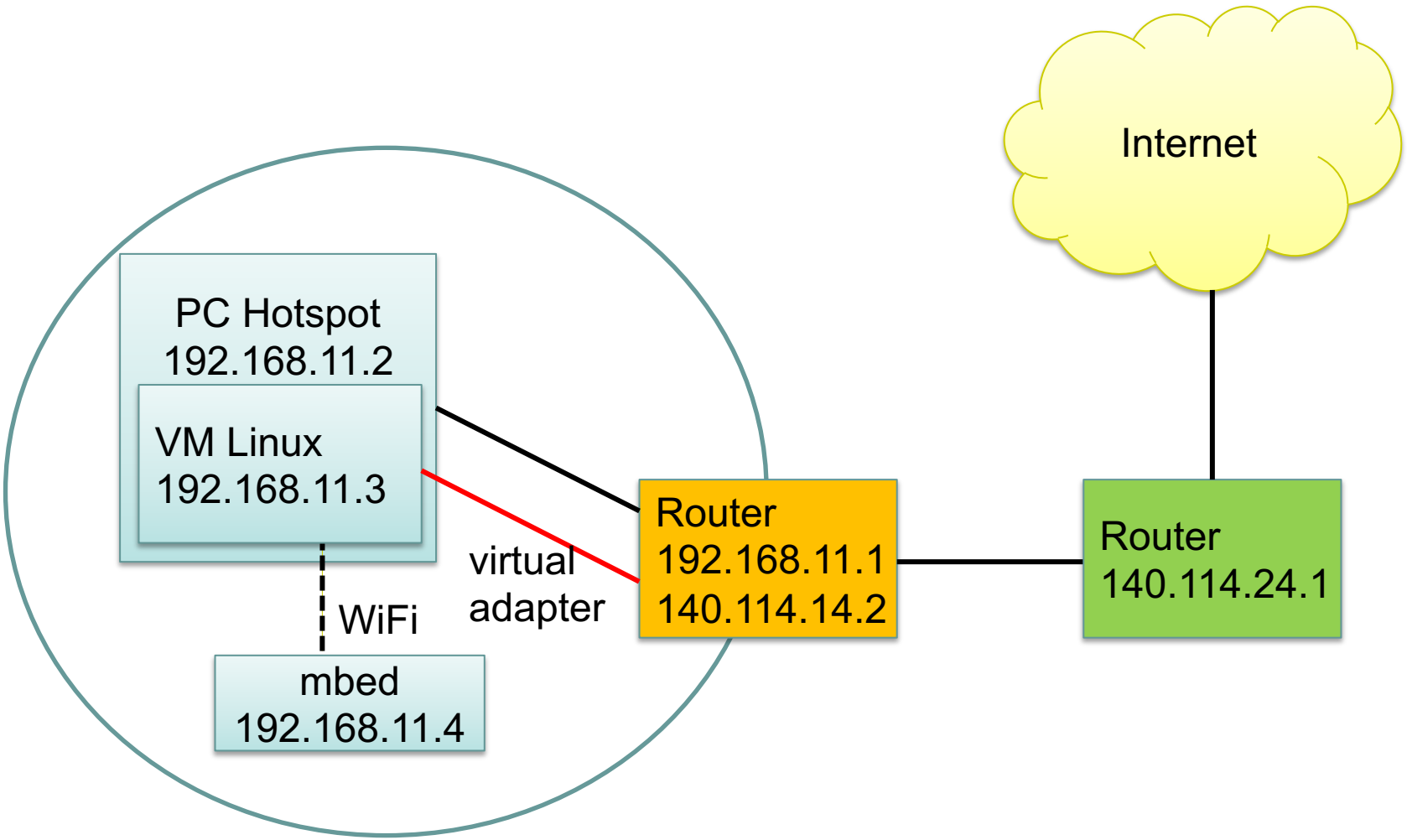
Network Map Example

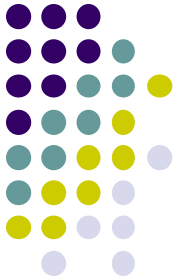


VM NAT Mode Example



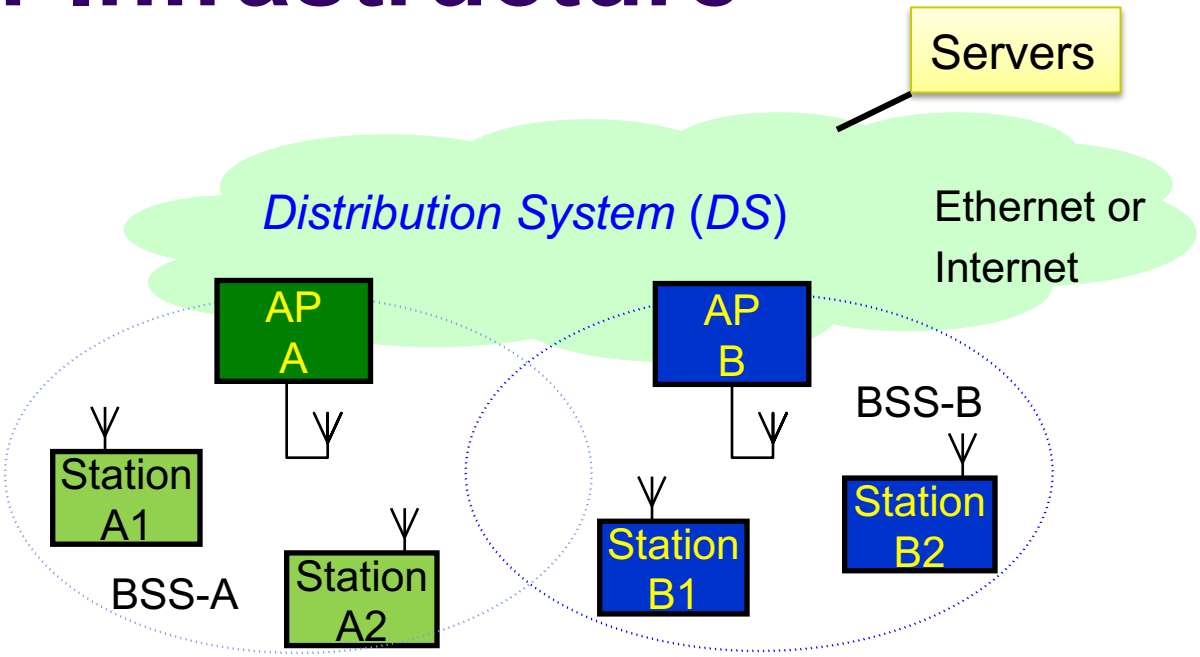
VM Bridge Mode Example





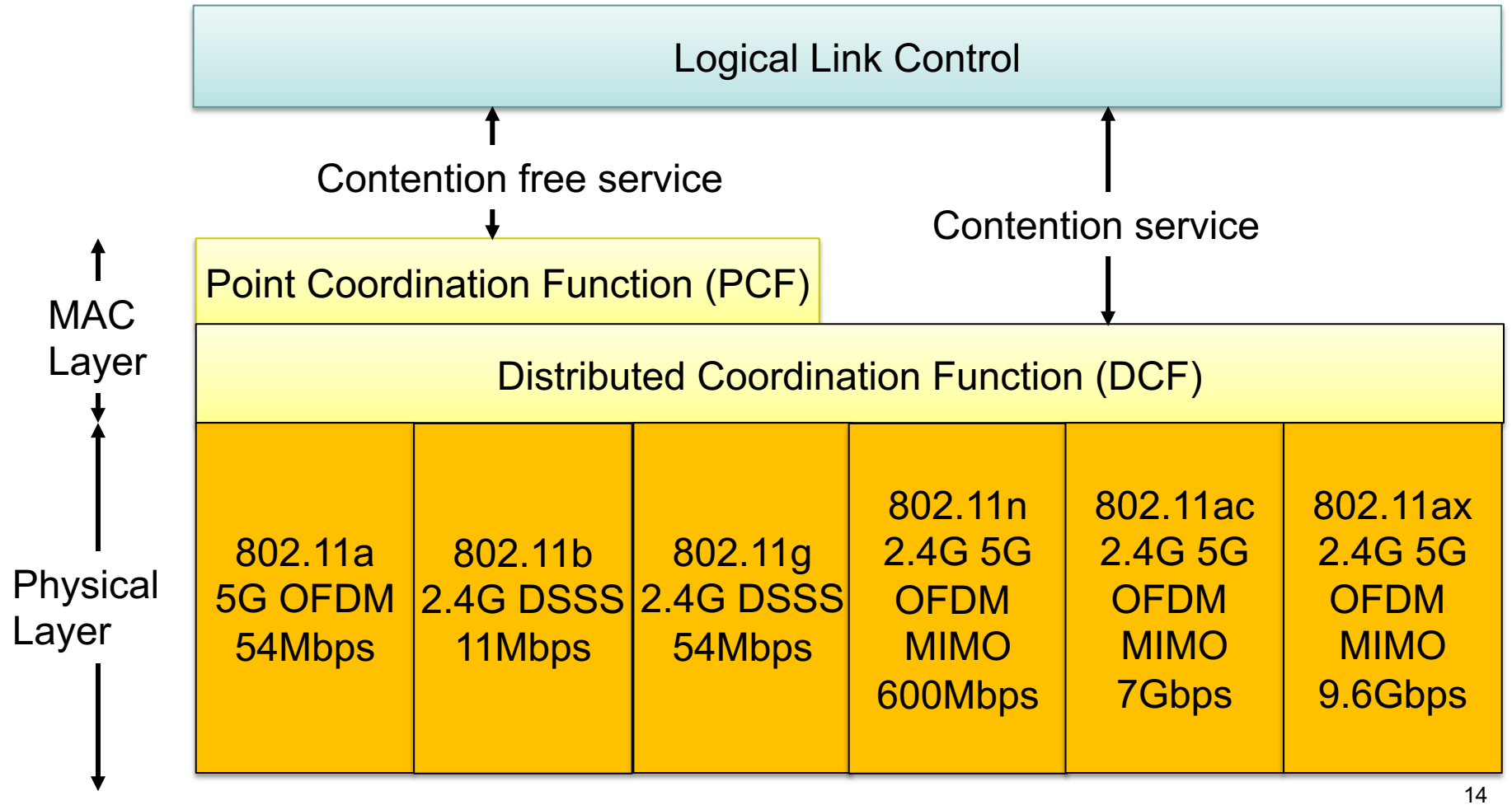
WIFI

802.11 Infrastructure

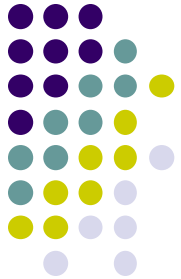


- Infrastructure
  - Access Points (AP) and stations (STA)
  - Basic service set (BSS) --- AP + multiple STAs
  - SSID = service set identifier.
- Distribution System interconnects multiple cells via AP to extends wireless coverage area

802.11 Protocol Architecture

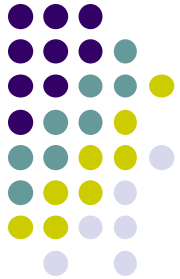


802.11 Protocol Layers



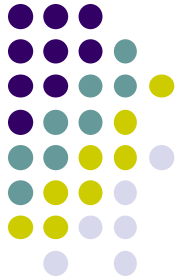
- Physical layer defines frequency band, data rate and actual radio transmission.
- MAC layer regulates access to the shared radio frequency to avoid conflict.
  - Distributed coordination function (Ethernet like)
  - Point coordination function (Polling)
- Logical Link Layer provide interface functions such as error control

Mbed WiFi Connect to Socket



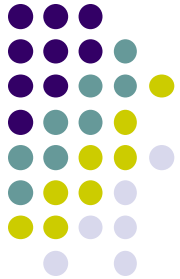
```
NetworkInterface *_net;
TCPSocket _socket;
void run() {
          WiFiInterface *wifi = WiFiInterface::get_default_instance();
          int ret = wifi->connect(MBED_CONF_APP_WIFI_SSID,
MBED_CONF_APP_WIFI_PASSWORD, NSAPI_SECURITY_WPA_WPA2);
           _net=wifi;

          print_network_info();

          result = _socket.open(_net);

          SocketAddress address;

          resolve_hostname(address))
          address.set_port(REMOTE_PORT);

          result = _socket.connect(address);

          if (!send_http_request() || !receive_http_response())
              return;
}
```

Socket Send

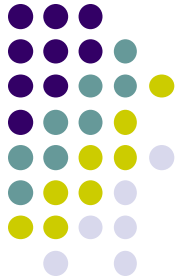


```
 bool send_http_request()
      {
          /* loop until whole request sent */
          const char buffer[] = "GET / HTTP/1.1\r\n" "Host: HOST_IP\r\n"
"Connection: close\r\n" "\r\n";

          nsapi_size_t bytes_to_send = strlen(buffer);
          nsapi_size_or_error_t bytes_sent = 0;

          printf("\r\nSending message: \r\n%s", buffer);

          while (bytes_to_send) {
              bytes_sent = _socket.send(buffer + bytes_sent, bytes_to_send);

              bytes_to_send -= bytes_sent;
          }

          printf("Complete message sent\r\n");

          return true;
      }
```

Socket Receive



```
 bool receive_http_response()
      {
          char buffer[MAX_MESSAGE_RECEIVED_LENGTH];
          int remaining_bytes = MAX_MESSAGE_RECEIVED_LENGTH;
          int received_bytes = 0;

          /* loop until there is nothing received or we've ran out of
buffer space */
          nsapi_size_or_error_t result = remaining_bytes;
          while (result > 0 && remaining_bytes > 0) {
              nsapi_size_or_error_t result = _socket.recv(buffer +
received_bytes, remaining_bytes);

              received_bytes += result;
              remaining_bytes -= result;
          }

          return true;
      }
```

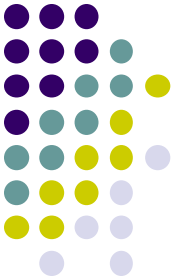


MQTT

MQTT

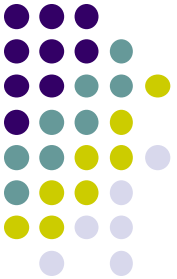
- Message Queuing Telemetry Transport
- Machine-to-machine (M2M)/Internet of Things connectivity protocol
- OASIS standard and ISO standard (ISO/IEC PRF 20922)
- Public and royalty-free license
- Amazon Web Service IoT, IBM WebSphere MQ, Microsoft Azure IoT, Adafruit, Facebook Messenger

Features



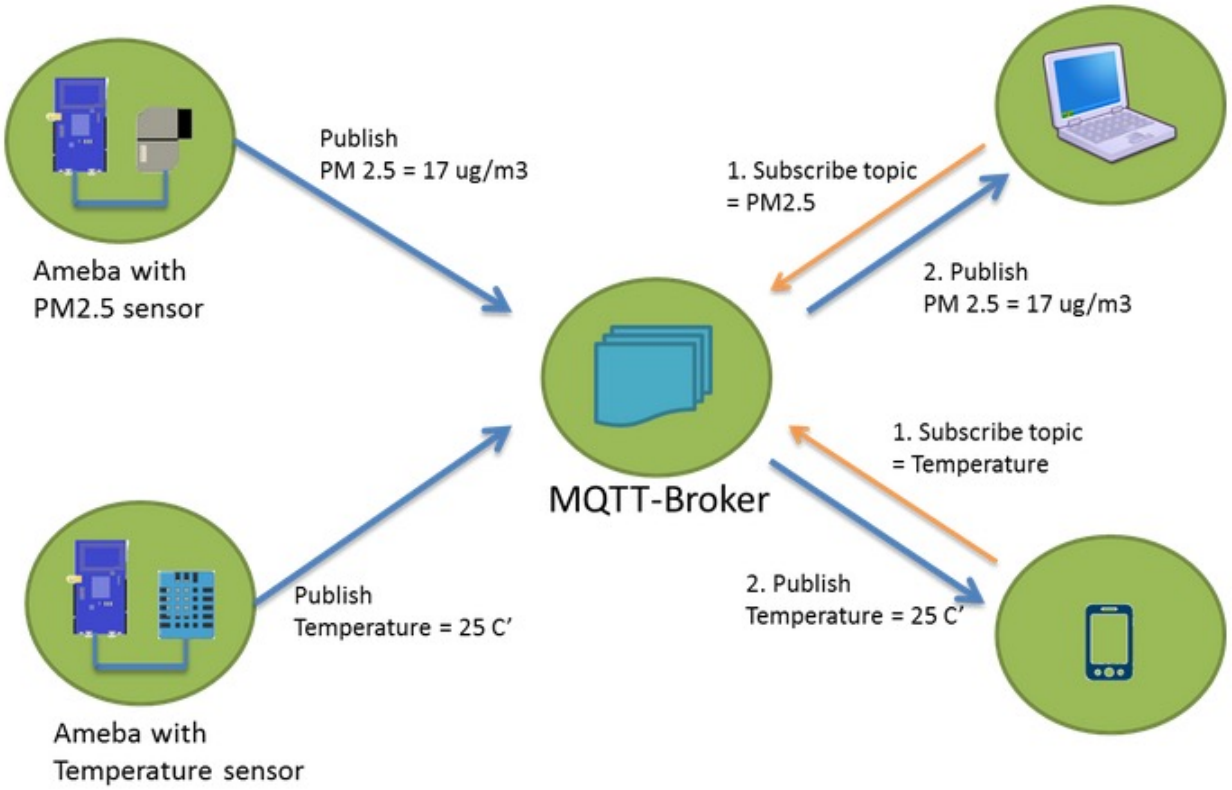
- Small code footprint
  - Ideal if processor or memory resources are limited
  - Ideal if bandwidth is low or network is unreliable
- Publish/subscribe message exchange pattern
- Works on top of TCP/IP
- Quality of service: at most once, at least once, exactly once
- Client libraries for Android, Arduino, C, C++, C#, Java, JavaScript, .NET
- Security: authentication using user name and password, encryption using SSL/TLS
- Persistence: MQTT has support for persistent messages stored on the broker.
- MQTT-SN (protocol for sensor network) works on non-TCP/IP networks (e.g. Zigbee)

Fields of Application



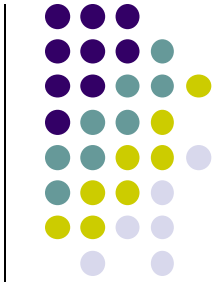
- Home automation (e.g. lightening, smart meter)
- Healthcare
- Mobile phone apps (e.g. messaging, monitoring)
- Industrial automation
- Automotive
- IoT applications in general

Publish and Subscribe Model



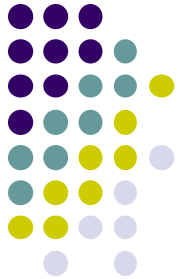
Messages in MQTT are published on topics.

Topics



- Topics are treated as a hierarchy, using a slash (/) as a separator.
  - Like a file system
  - For example: sensors/computer1/temperature/HD1
  - Wildcards (+) --- match one single level
    - sensors/+/temperature/+
  - Wildcards (#) --- match all remaining levels
    - sensors/ computer1/#
- Clients subscribe to topics to receive messages

Quality of Service



- QoS levels (0→1→2 more reliable)
- 0
  - The broker/client will deliver the message once, with no confirmation.
- 1
  - The broker/client will deliver the message at least once, with confirmation required.
- 2
  - The broker/client will deliver the message exactly once by using a four step handshake.

Example

- “mosquito” is an open source message broker

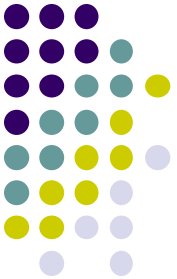
Subscribe client

```
$ mosquitto_sub -h localhost -t test

Hello
```

Publish client

```
$ mosquitto_pub -h localhost -t test -m "Hello”
```

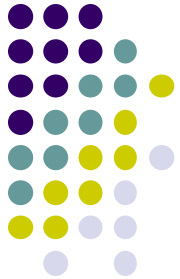
mbed MQTT API



- `connect` to the broker, based on a filled `MQTTPacket_connectData` structure
- `diconnect` from the broker
- `subscribe` to a topic and register a callback function to be called whenever a new message arrives
- `unsubscribe` from a topic
- `publish` messages defined with `MQTT::Message` structure to a topic

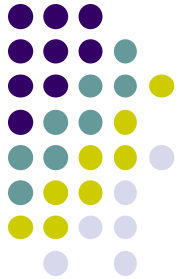
https://github.com/ARMmbed/mbed-mqtt

mbed MQTT Example



```
SocketAddress sockAddr(host, 1883);
int rc = mqttNetwork.connect(sockAddr);//(host, 1883);

MQTTPacket_connectData data = MQTTPacket_connectData_initializer;
data.MQTTVersion = 3;
data.clientID.cstring = "Mbed";

rc = client.connect(data);
client.subscribe(topic, MQTT::QOS0, messageArrived);

mqtt_thread.start(callback(&mqtt_queue, &EventQueue::dispatch_forever));
btn2.rise(mqtt_queue.event(&publish_message, &client));
int num = 0;
while (num != 5) {
        client.yield(100);
        ++num;
}

printf("Ready to close MQTT Network......\n");

rc = client.unsubscribe(topic);
rc = client.disconnect();
mqttNetwork.disconnect();
```

mbed Publish Messages



```
void publish_message(MQTT::Client<MQTTNetwork, Countdown>*
client) {
        message_num++;
        MQTT::Message message;
        char buff[100];
        sprintf(buff, "QoS0 Hello, Python! #%d",
message_num);
        message.qos = MQTT::QOS0;
        message.retained = false;
        message.dup = false;
        message.payload = (void*) buff;
        message.payloadlen = strlen(buff) + 1;
        int rc = client->publish(topic, message);

        printf("rc:  %d\r\n", rc);
        printf("Puslish message: %s\r\n", buff);
    }
```