IoT Application Protocols

Embedded Interface Design

with **Bruce Montgomery**

Learning Objectives

- Students will be able to…
  - Recognize common IoT Application Protocols
  - Compare and contrast features and use cases for different protocols
  - Apply and develop IoT/Cloud interfaces with the most used protocols

IOT Application Protocols

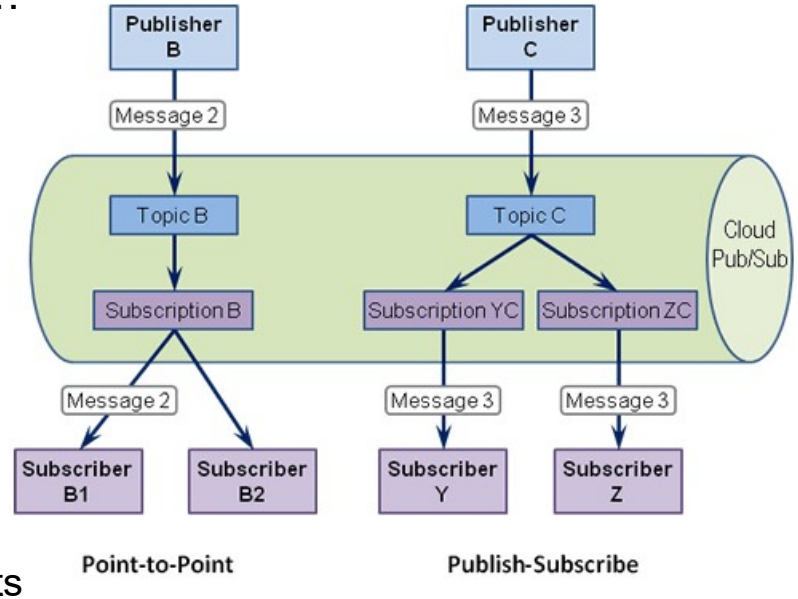
- Defined
- Messaging - briefly
- MQTT
- CoAP
- WebSockets
- Others

IOT Application Protocols

- In OSI Application Layers
  - Below Application Data/Objects
  - Above Transport/Security
- In practice, protocol for passing messages between a device and other devices or services, especially from devices to gateways or to the Cloud
- AWS IoT Framework uses MQTT, HTTP, or MQTT over WebSockets [1]
- MS Azure IoT Hub uses MQTT, AMQP, HTTP, MQTT or AMQP over WebSockets [2]
- General messaging concerns:
  - Messaging patterns or models
  - Quality of service approaches
  - Message commands – RESTful/RESTless
- Protocol selection is based on assessment of specific characteristics of protocols, devices, and transactions (more later)

Messaging Patterns

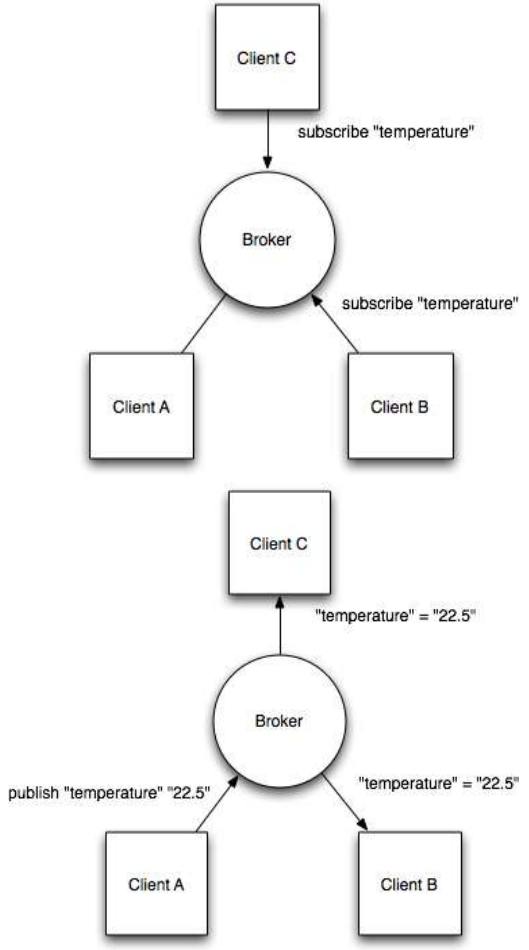
- Messaging Patterns describe the model messages follow to flow data between message producers and consumers…
- Typical Patterns
  - Publish/Subscribe (MQTT, AMQP, SMQ, STOMP, XMPP PubSub)
  - Request/Response (CoAP, HTTP, WebSocket)
  - Point to Point (aka Peer to Peer) (XMPP)
  - ACTive (Availability for Concurrent Transactions) – used by XMPP
- Other
  - Pipeline/Databus (aggregation, load-balancing)
  - Survey (1 request, multiple responses)
- Brokers – Tools to allow multi-protocol messaging
  - RabbitMQ – Primarily AMQP, supports MQTT, WebSockets
  - Others: ZeroMQ, ActiveMQ, Nanomsg, etc.
  - More later in the class...



References [3], [4]

MQTT

- Message Queue Telemetry Transport [5], [6]
- Open standard from IBM & Eurotech, now owned by Eclipse “Paho” project for M2M/IoT
- Focused on lightweight M2M/IoT communication
- Lightweight: smallest packet size 2 bytes (header)
- Small client footprint – C# M2Mqtt library is 30 KB
- Reliable – 3 QoS options to avoid packet loss on client disconnect
- Simple: TCP based, Asychronous, Pub/Sub with a Broker, Few verbs, Payload agnostic
- Security: SSL/TLS, can encrypt payloads
- MQTT-SN defines a UDP mapping, adds indexing of long topic names
- OASIS Standard (MQTT v3.1.1)
- Port 1883 – Plain Text, Port 8883 - TLS



Reference [7]

MQTT Topics

- Topics - Label for grouping of application messages, matched against subscriptions to forward the messages
- Publish messages under specific topics
  - publish(topic, message)
- Subscribe/Unsubscribe to/from Topic filters
  - subscribe(topic_filter)
  - unsubscribe(topic_filter)

- From reference [11]

- Topic names beginning with '$' character are used for implementation internal purposes
- $SYS/ is usually used for topics that contain server information and control the API

MQTT Topic Filters

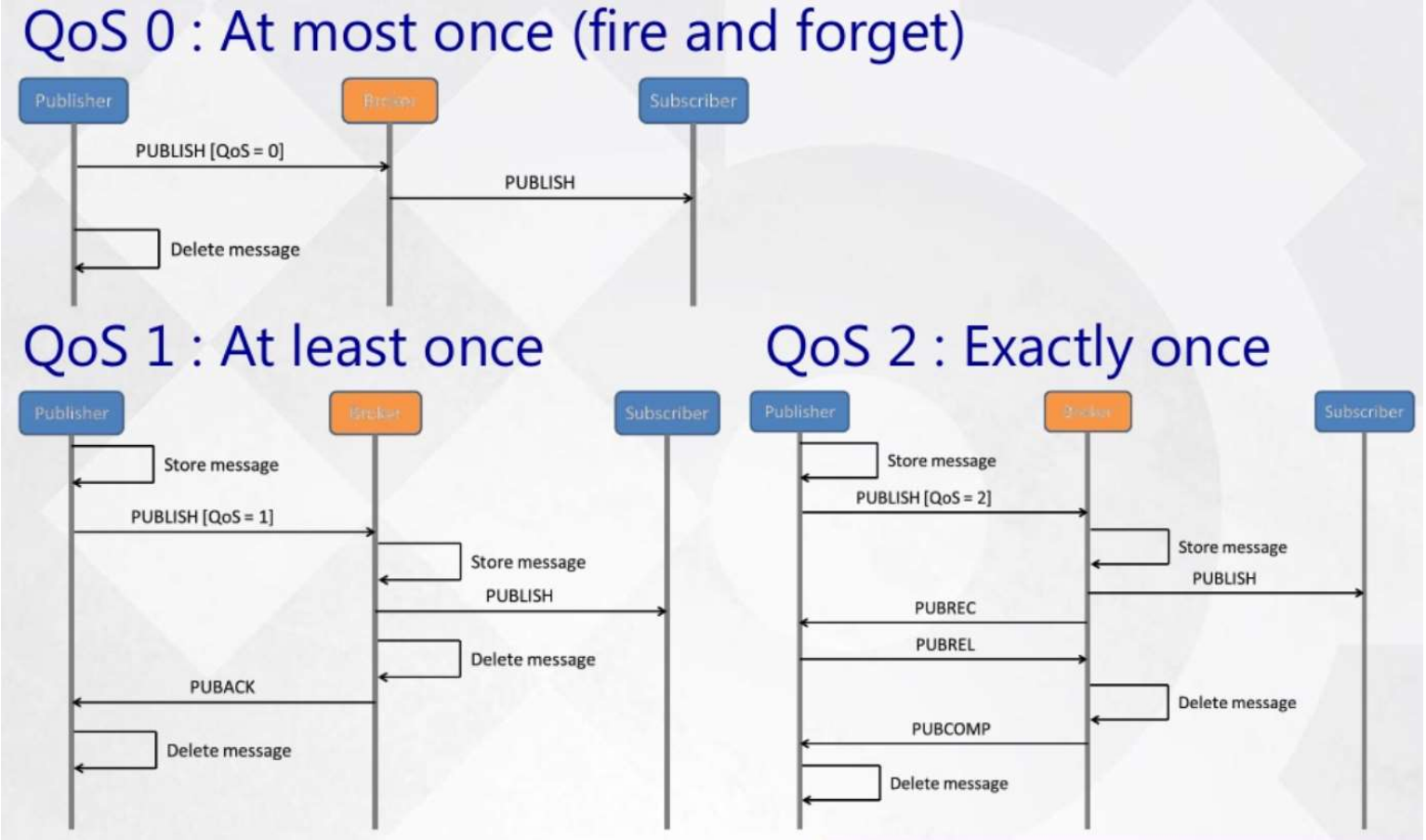
- Consider an MQTT topic: company/building_1/main_hall/temp
- Topic filters - An expression indicating one or more topic names in a subscription; May use wild cards
  - Can use wildcards to specify more than one Topic of interest
  - Multilevel wildcard - '#'
    - Used to represent parent and any number of child levels
    - Can be used on its own or followed by a topic level separator
    - Must be the last character in a topic filter
    - Examples
      - # - matches all topics
      - company/building_1/main_hall/#
        - matches main_hall and all topics under that
      - company/building_1/#/temp
        - Invalid filter
  - Single level wildcard – '+'
    - Matches only one topic level
    - Must occupy an entire level of a filter
    - Can be used in conjunction with Multilevel wildcard
    - Examples
      - +
        - Is Valid
      - company/building_1/+/temp
        - is valid
      - company/+/main_hall/#
        - is valid
  - From reference [11]

MQTT Message Format [8]

bit	7	6	5	4	3	2	1	0
byte 1	Message Type				DUP	QoS		Retain
byte 2	Remaining length (i.e. length of option + payload)							
byte 3	Variable Header Component							
byte n								
byte m	Payload							
byte n								

Message	Code	Description
CONNECT	1	Client request to connect to Server
PUBLISH	3	Publish message
SUBSCRIBE	8	Client subscribe request
UNSUBSCRIBE	10	Client Unsubscribe
DISCONNECT	14	Client is disconnecting

MQTT QoS Patterns [7]



MQTT QoS (Quality of Service) Patterns

- QoS 0 – Fire and Forget
  - Messages are not stored or redelivered by sender, or acknowledged by receiver
  - In AWS, this QoS level means a message will be delivered 0 or more times
- QoS 1 – At Least Once
  - Sender stores message until acknowledged by the receiver
  - Possible that a message can be delivered by the sender more than once
- QoS 2 – Exactly Once
  - Sender sends message, receiver acks, sender sends intent to release message, receiver acks
  - Not supported by AWS IoT Message Broker/MQTT implementation
- Reference [9]

MQTT QoS (Quality of Service) Patterns

- Use QoS 0 when …
  - You have a complete or almost stable connection between sender and receiver. A classic use case is when connecting a test client or a front end application to a MQTT broker over a wired connection.
  - You don't care if one or more messages are lost once a while. That is sometimes the case if the data is not that important or will be send at short intervals, where it is okay that messages might get lost.
  - You don't need any message queuing. Messages are only queued for disconnected clients if they have QoS 1 or 2 and a persistent session.
- Use QoS 1 when …
  - You need to get every message and your use case can handle duplicates. The most often used QoS is level 1, because it guarantees the message arrives at least once. Of course your application must be tolerating duplicates and process them accordingly.
  - You can't bear the overhead of QoS 2. Of course QoS 1 is a lot faster in delivering messages without the guarantee of level 2.
- Use QoS 2 when …
  - It is critical to your application to receive all messages exactly once. This is often the case if a duplicate delivery would do harm to application users or subscribing clients. You should be aware of the overhead and that it takes a bit longer to complete the QoS 2 flow.

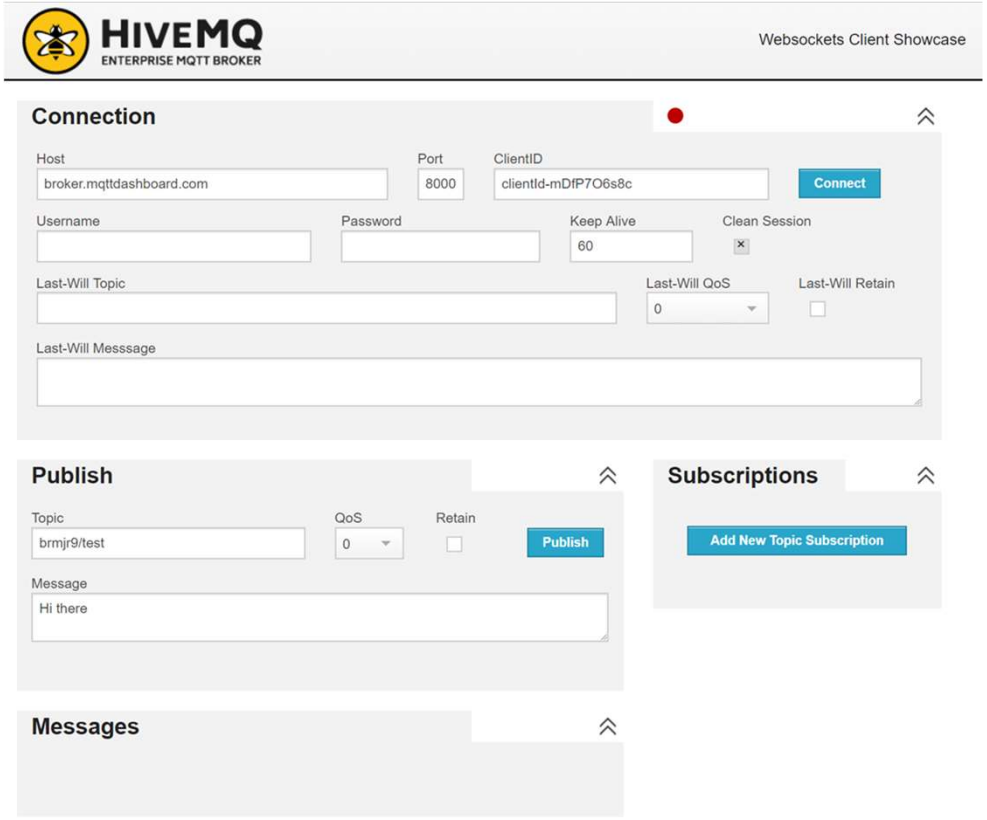
MQTT for Python

- Paho Python library [10] does not support
  - message persistence
    - Persist messages if application crashes
  - high availability modes
    - Automatic failover if client cannot connect to server

```
import paho.mqtt.client as mqtt

# The callback for when the client receives a CONNACK response from the server.
def on_connect(client, userdata, rc):
    print("Connected with result code "+str(rc))
        # Subscribing in on_connect() means that if we lose the connection and
        # reconnect then subscriptions will be renewed.
        client.subscribe("$SYS/#")

# The callback for when a PUBLISH message is received from the server.
def on_message(client, userdata, msg):
        print(msg.topic+" "+str(msg.payload))

client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message

client.connect("iot.eclipse.org", 1883, 60)

# Blocking call that processes network traffic, dispatches callbacks and
# handles reconnecting.
# Other loop*() functions are available that give a threaded interface and a
# manual interface.
client.loop_forever()
```

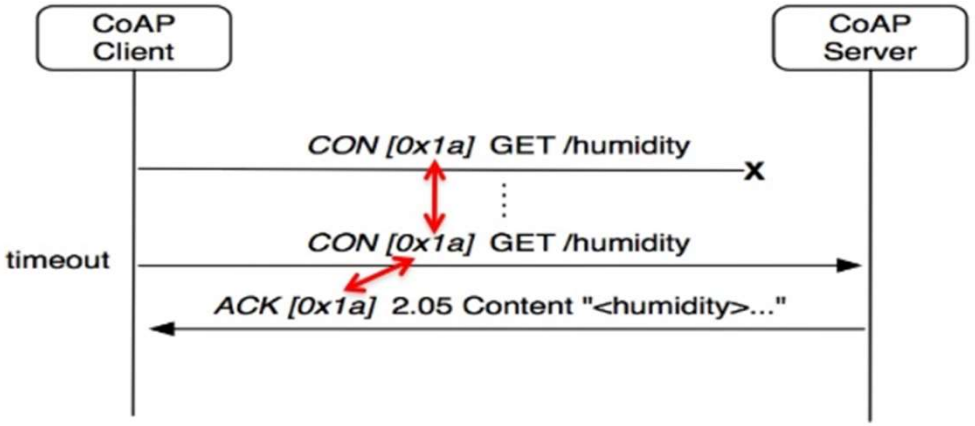
Exercise: MQTT Test

- MQTT test client at: http://www.hivemq.com/demos/websocket-client/
- Connect to the default broker and port: broker.mqttdashboard.com:8000
- Use a clientid you'll recognize
- You can subscribe to a topic
- You can publish to a topic
- You can subscribe to a neighbor's topic and they can publish to you
- Try the MQTT wildcards and $SYS/ topic as well

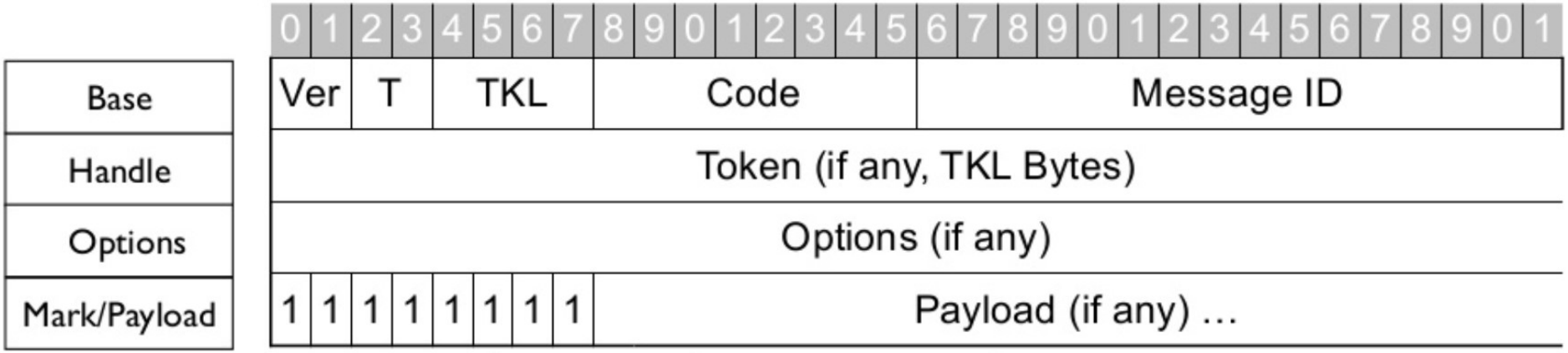


CoAP

- RFC 7252 – Constrained Application Protocol
- CoAP [12] is RESTful – often described as REST for small devices
  - GET, PUT, POST, DELETE operations like HTTP
  - More interoperable with Web systems and development
  - Embedded web transfer protocol (coap://)
  - URI support
- CoAP can use XML, JSON, and other payloads
- Async Request/Response Architecture
- Built-in Discovery
- Uses UDP on IP with a 4 byte fixed header and compact option encoding
- Security defaults to DTLS



CoAP Message Format

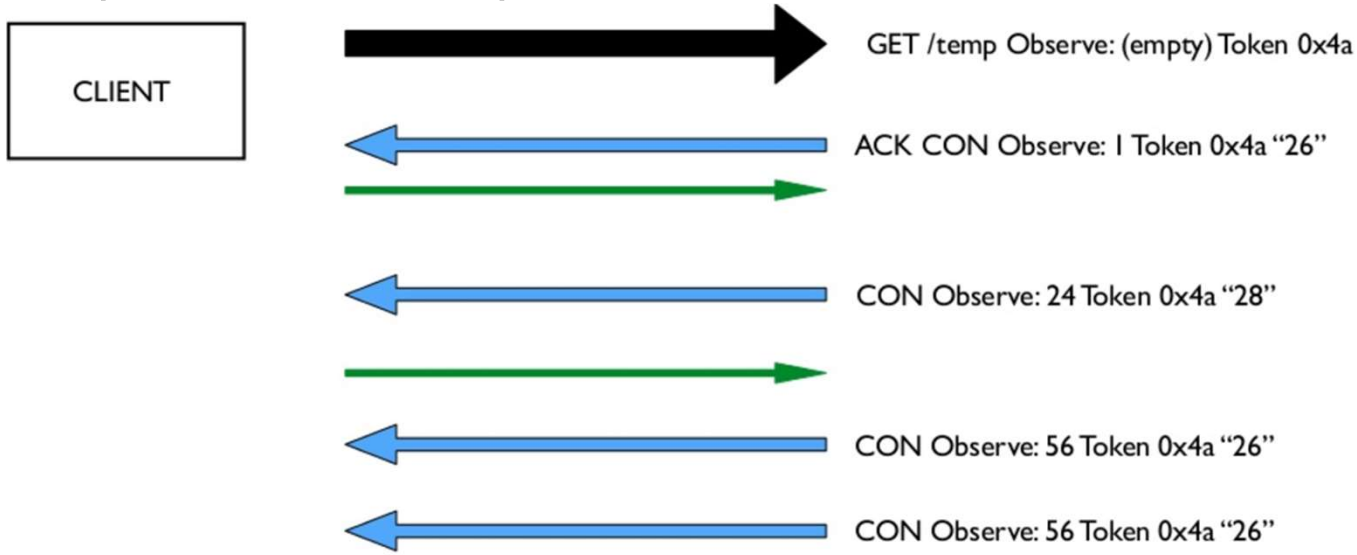


Ver	Version
T	Transaction Type
TKL	Token Length
Code	Request Code
Message ID	Identifier

Reference [8]

CoAP "Observe" Messaging

- Observe request can be cancelled with an RST
- CON = Confirmable request, should be ACKed by recipient; ACKs can contain results
- NON = Non-comfirmed request, no ACK required
- Note the "Server" is the device providing data
- Reference [8]



CoAP for Python

- Several libraries available [13]
- Aiocoap: based on Python 3.4 asyncio [14]
- txThings: based on Twisted [15]
  - recommended to use if not in Python 3
- CoAPthon: has a branch that uses Twisted [16]
- Also, there is a CoAP client for Firefox and Chrome, called Copper [24]

Testing CoAP connectivity

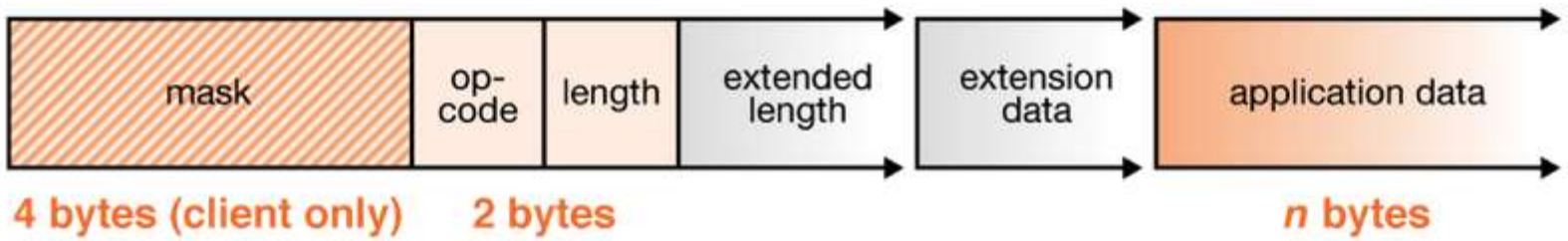
Start a server in a terminal:
$ ./server.py

Send a GET to the server from another terminal:
$ ./aiocoap-client coap://localhost/.well-known/core </time>; obs, </.well-known/core>; ct=40, </other/separate>, </other/block>

(full session example [17])

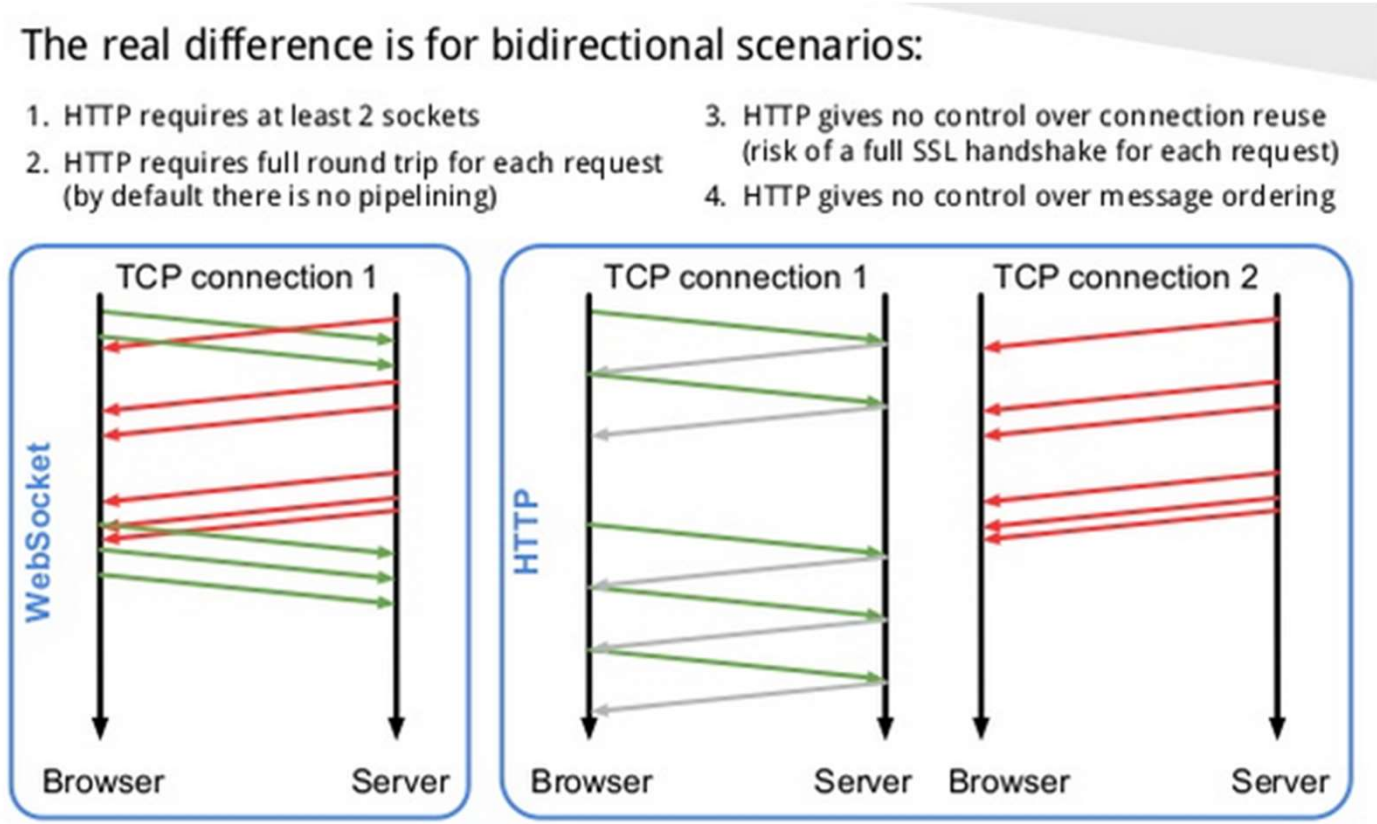
WebSockets

- WebSockets – RFC 6455 [18]
- Bi-directional, full-duplex, persistent connection from a web browser (or client) to a server
- Connection stays open until client or server closes connection
- Single servers can speak to many open clients
- HTTP(S) over TCP/IP until WebSocket connection is established
- Still uses HTTP Port 80 or HTTPS Port 443 by default



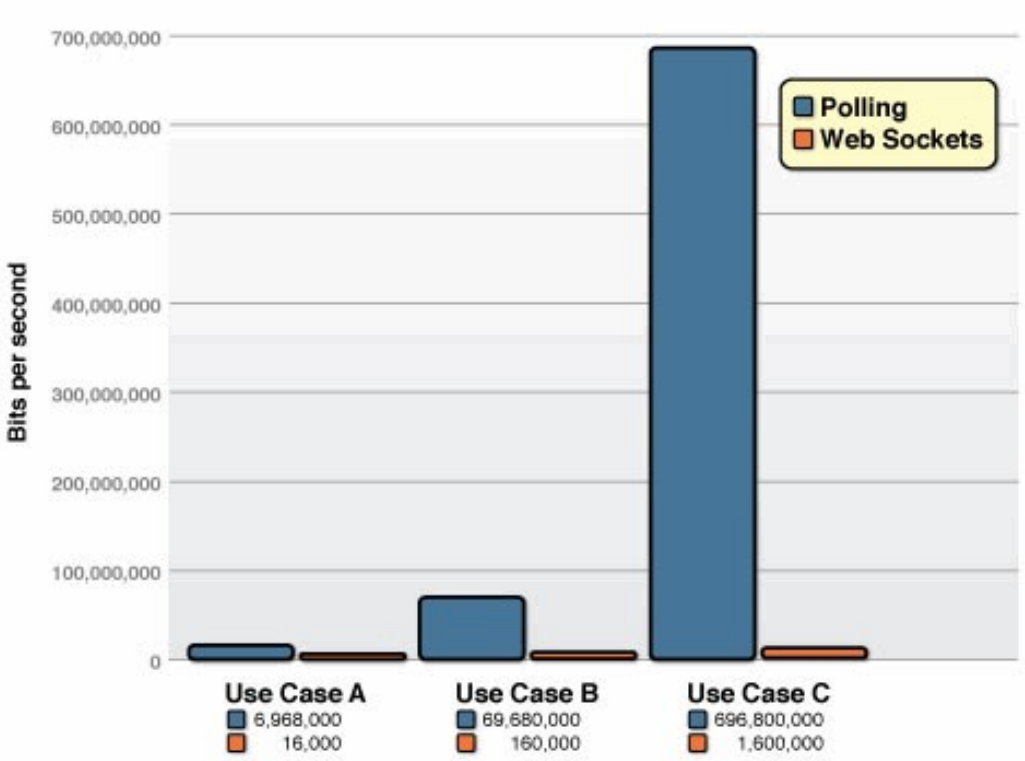
WebSockets vs. HTTP

- Alternative to HTTP transactions that have to reestablish with each message or use cookies for identification [19]



WebSockets vs. HTTP

- **Use case A:** 1,000 clients receive 1 message per second: Network throughput is (2 x 1,000) = 2,000 bytes = 16,000 bits per second (0.015 Mbps)
- **Use case B:** 10,000 clients receive 1 message per second: Network throughput is (2 x 10,000) = 20,000 bytes = 160,000 bits per second (0.153 Mbps)
- **Use case C:** 100,000 clients receive 1 message per second: Network throughput is (2 x 100,000) = 200,000 bytes = 1,600,000 bits per second (1.526 Mbps)
- Reference [20]



WebSockets for Python

- Many options: Autobahn, Crossbar.io, Nginx, Twisted, Flask…
- Simpler options:
  - Tornado (server set up shown here ->)
  - Pywebsocket
- Reference [21]

```
import tornado.httpserver
import tornado.websocket
import tornado.ioloop
import tornado.web

class WSHandler(tornado.websocket.WebSocketHandler):

  def open(self):
    print 'user is connected.\n'

  def on_message(self, message):
    print 'received message: %s\n' %message
    self.write_message(message + ' OK')

  def on_close(self):
    print 'connection closed\n'

application = tornado.web.Application([(r'/ws', WSHandler),])

if __name__ == "__main__":
  http_server = tornado.httpserver.HTTPServer(application)
  http_server.listen(8888)
  tornado.ioloop.IOLoop.instance().start()
```

Other Protocols

- AMQP – Advanced Message Queuing Protocol
- STOMP – Simple/ Streaming Text Oriented Message Protocol
- DDS – Data Distribution Service
- XMPP – Extensible Messaging and Presence Protocol
- SMQ – Simple Message Queue
- HTTP(S) – Hypertext Transfer Protocol (Secure)
- HTTP/2 – reduced headers from HTTP 1,1
- Reference [22]

Protocol	Sponsor	Blessing	Message Pattern	QoS	Security	"Addressing"	REST?	Constrained Devices?
MQTT	MQTT.org	OASIS	P/S	3 levels**	Best practices	Topic only		Y
CoAp	IETF	IETF	R/R	Optional	DTLS	URL	x	Y
XMPP	XMPP Standards Foundation	IETF	P-P P/S by extension	None (could be done by extension)	TLS/SSL XEP-0198	JID		I think so
AMQP	OASIS	OASIS	P/S	Sophisticated	TLS; SASL	Y		N
DDS	OMG	OMG	P/S	Sophisticated	In beta	Topic/key		Y (no optional features)
SMQ	Real Time Logic	Proprietary (might be opened)	P/S	Limited; mostly via TCP	SSL	Y		Y
HTTP/2	IETF	IETF	R/R	TCP	TLS/SSL	URL	x	Y (HPACK)
AllJoyn	Qualcomm	AllSeen Alliance	RPC	No	Done by app	Y		Thin client option
STOMP	Community		P/S	Simple Server-specific	N	N Server-specific		N

Next Steps

- Hardware Handout today – tracking sheet (view only) at
  - https://docs.google.com/spreadsheets/d/1mKPs_GmbZGsNRROy2-BztipEiR1NFhsgTi9uQJ-Lerk/edit?usp=sharing
  - Shipped to distance students Saturday, see your e-mail
  - You should have a RPi3, a power supply, an SD card, and a DHT22 sensor
    - You will source your own USB cables and resistor/wiring for DHT22
- Review Project 1 – submission links are up on Canvas
- Background survey is up – please complete by Sunday 9/15 – participation grade element
- Wednesday: M2M, Low Level Protocols, IoT Protocols
- Next week: M2M Protocols, LPWAN Protocols, Cloud Architectures, Cloud for IoT, AWS, AWS for IoT

References

[1] https://docs.aws.amazon.com/iot/latest/developerguide/protocols.html
[2] https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-devguide-protocols
[3] http://www.eejournal.com/archives/articles/20150420-protocols/
[4] http://www.enterpriseintegrationpatterns.com/patterns/messaging/PublishSubscribeChannel.html
[5] http://mqtt.org/
[6] http://www.eclipse.org/paho/
[7] https://www.slideshare.net/paolopat/mqtt-iot-protocols-comparison
[8] https://www.slideshare.net/vgholkar/io-t-protocolsoscon2014
[9] http://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels
[10] https://eclipse.org/paho/clients/python/
[11] Payatu class on Practical IoT Device Hacking - MQTT
[12] http://coap.technology/
[13] http://coap.technology/impls.html
[14] https://github.com/chrysn/aiocoap
[15] https://github.com/mwasilak/txThings
[16] https://github.com/Tanganelli/CoAPthon
[17] http://aiocoap.readthedocs.io/en/latest/guidedtour.html
[18] https://websocket.org/aboutwebsocket.html
[19] https://www.slideshare.net/alinone/from-push-technology-to-the-realtime-web
[20] https://websocket.org/quantum.html
[21] http://iot-projects.com/index.php?id=websocket-a-simple-example
[22] http://www.eejournal.com/archives/articles/20150420-protocols/
[23] https://www.slideshare.net/RealTimeInnovations/io-34485340
[24] https://addons.mozilla.org/en-US/firefox/addon/copper-270430/