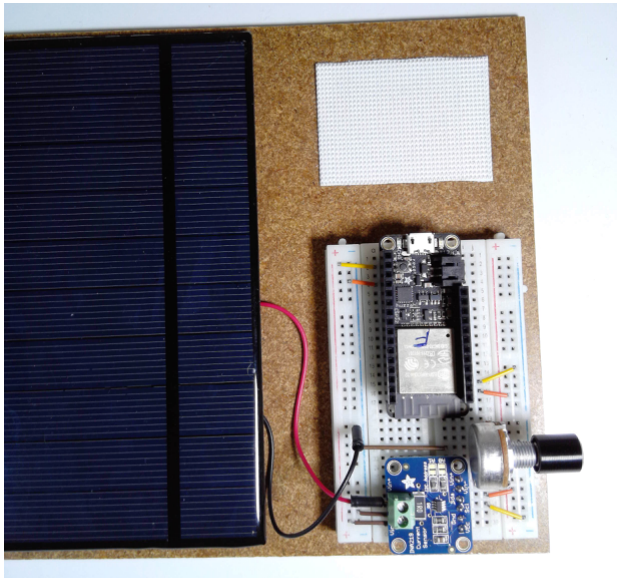
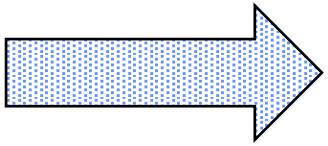
Electronics for IoT

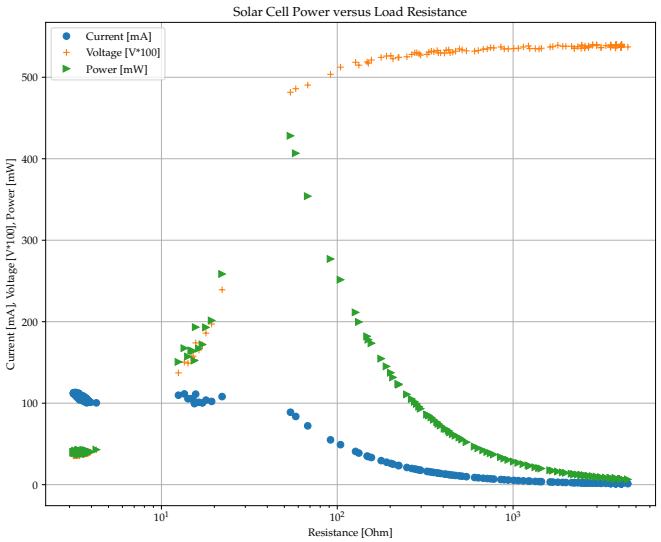
MQTT

Bernhard E. Boser
University of California, Berkeley
boser@eecs.berkeley.edu

Missing Link







Approach

- Get data from ESP32 to host computer (e.g. laptop)
- Then plotting is easy
  - Matlab, Excel, Python, …

Data Communication Options

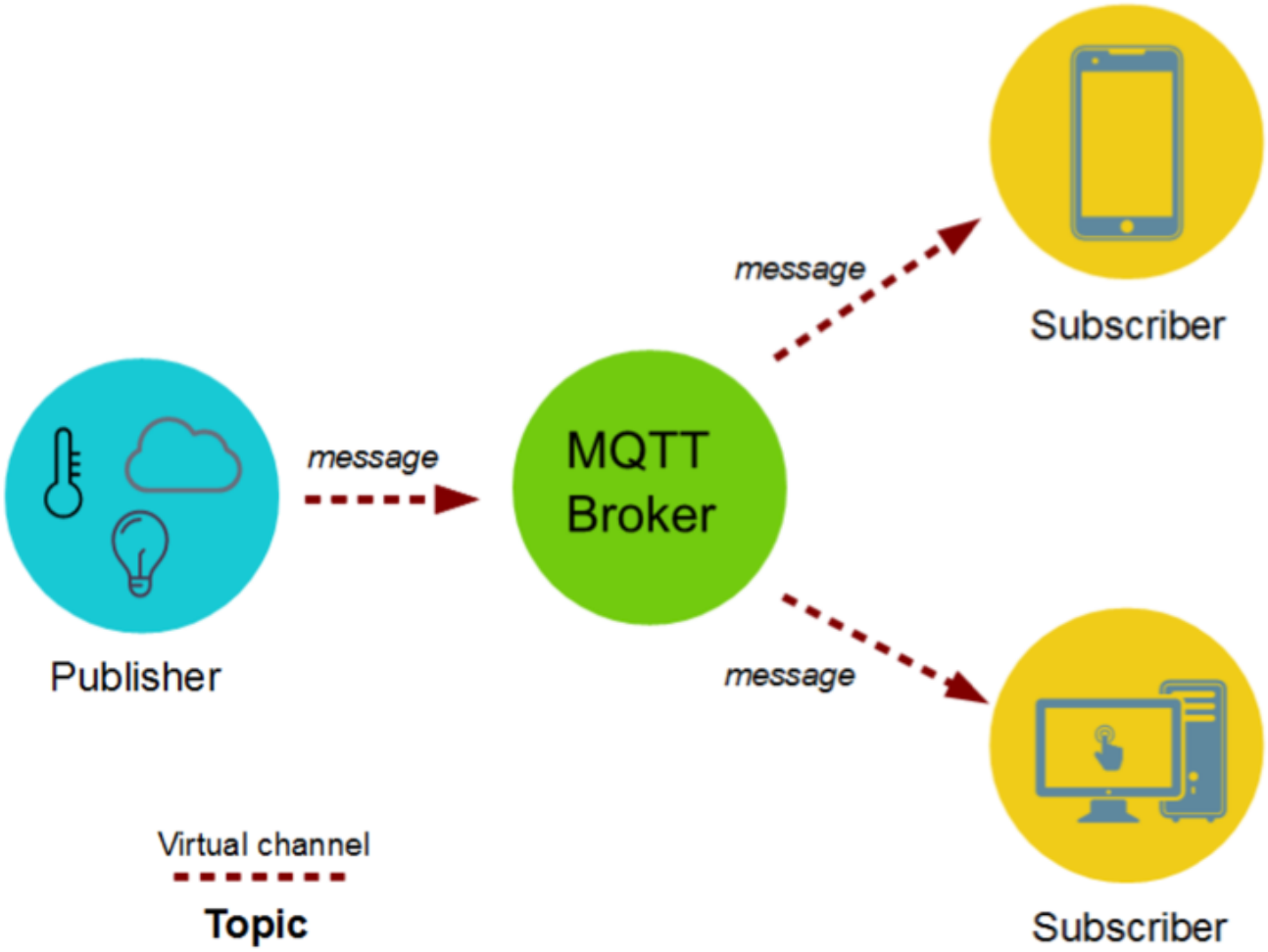
- Html
  - + ubiquitous
  - - verbose
  - - visual output difficult to parse
  - - needs webserver

MQTT

- Message Queuing Telemetry Transport
  - Misnomer: no queuing
  - Lightweight machine-machine messaging protocol
  - Lightweight →
    - low communication bandwidth,
    - suitable for implementation on resource-constrained devices
- Standard
  - ISO/IEC PRF 20922
  - Lot's of support available:
    - Implementations in many computer languages (e.g. Python)
    - Tutorials, documentation …
      - E.g. https://www.hivemq.com/blog/how-to-get-started-with-mqtt
      - Many others

MQTT Publish/Subscribe

MQTT Publish/Subscribe



MQTT Messages

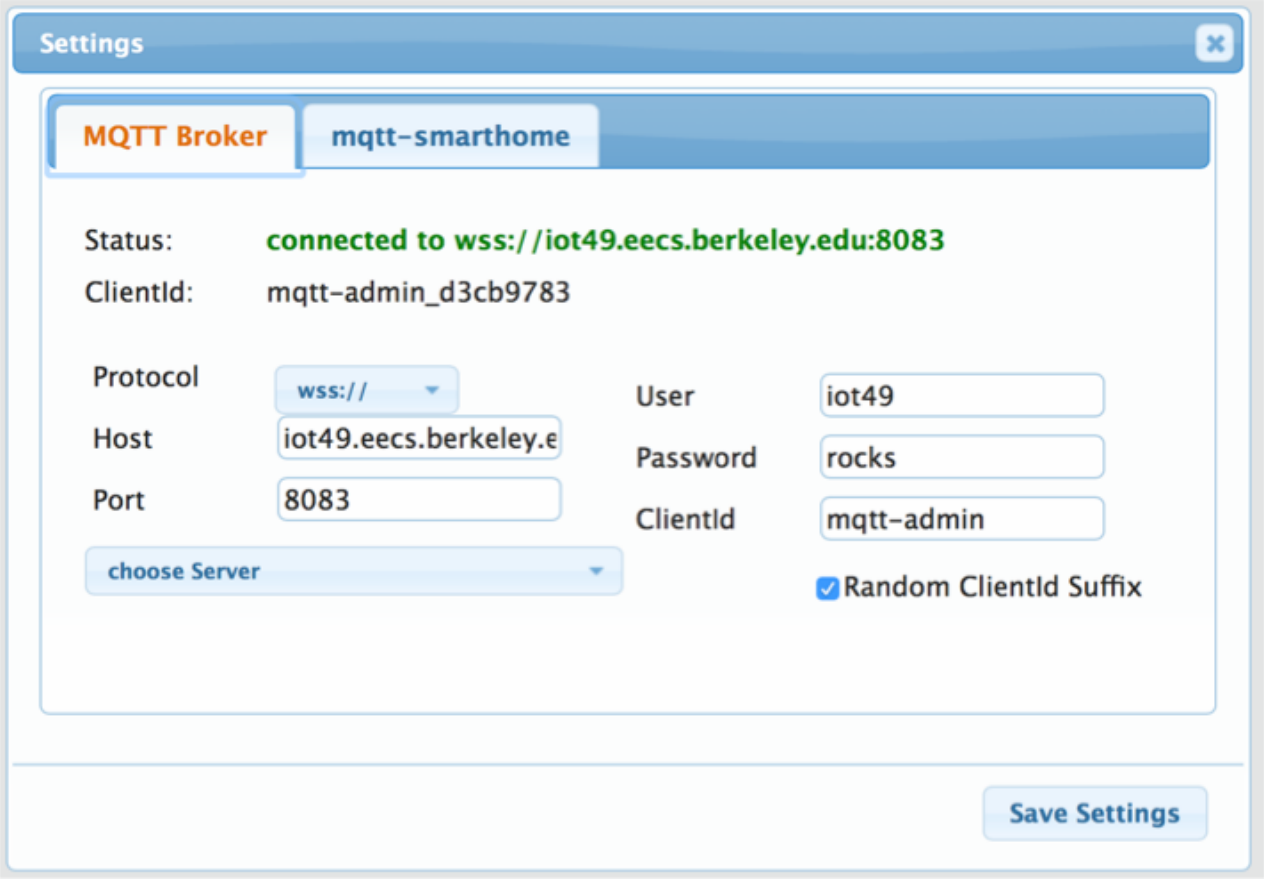
- Topic
  - Hierarchical, separated by /
  - E.g.
    - Kitchen/temperature
    - IBM/stockprice
    - …
- Message
  - Arbitrary text

MQTT Example – ESP32 Client

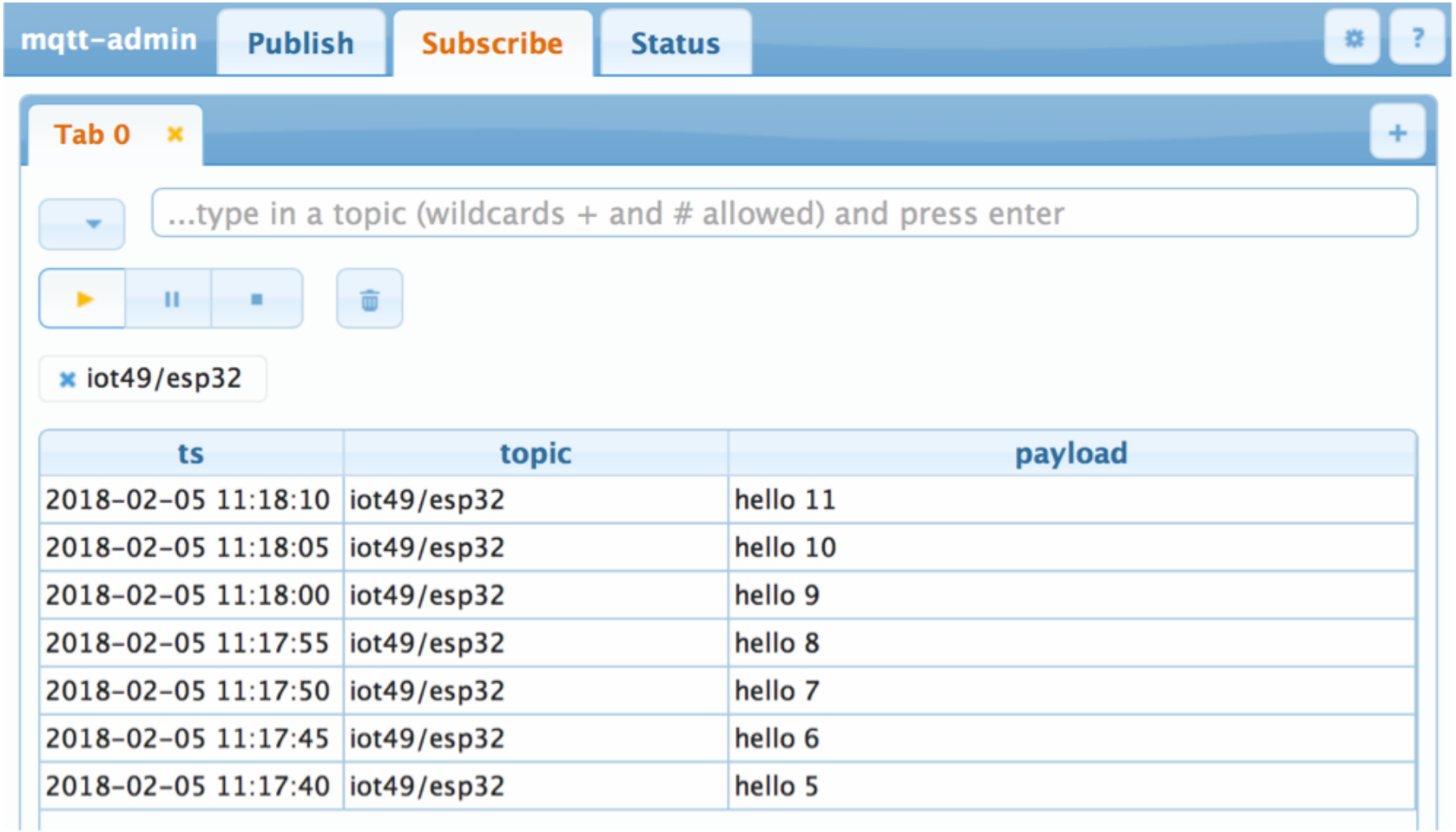
```
from mqttclient import MQTTClient
from time import sleep

BROKER = "iot49.eecs.berkeley.edu"
USER = "iot49"
PWD = "rocks"

print("Connecting to broker", BROKER, "...")
mqtt = MQTTClient(BROKER, user=USER, password=PWD, ssl=True, )

print("Connected!")

def mqtt_callback(topic, msg):
    print("RECEIVE topic = {}, msg = {}".format(topic, msg))

mqtt.set_callback(mqtt_callback)
mqtt.subscribe("iot49/a")
mqtt.subscribe("iot49/b")

for i in range(100):
    topic = "iot49/esp32"
    message = "hello " + str(i)
    print("PUBLISH topic = {} message = {}".format(topic, message))
    mqtt.publish(topic, message)
    for _ in range(10):
        mqtt.check_msg()
        sleep(0.5)
```

MQTT Example – Web Client (Testing)

- https://hobbyquaker.github.io/mqtt-admin/



Subscribe …



ts	topic	payload
2018-02-05 11:18:10	iot49/esp32	hello 11
2018-02-05 11:18:05	iot49/esp32	hello 10
2018-02-05 11:18:00	iot49/esp32	hello 9
2018-02-05 11:17:55	iot49/esp32	hello 8
2018-02-05 11:17:50	iot49/esp32	hello 7
2018-02-05 11:17:45	iot49/esp32	hello 6
2018-02-05 11:17:40	iot49/esp32	hello 5

Publish


```
PUBLISH topic = iot49/esp32 message = hello 19
PUBLISH topic = iot49/esp32 message = hello 20
PUBLISH topic = iot49/esp32 message = hello 21
PUBLISH topic = iot49/esp32 message = hello 22
RECEIVE topic = b'iot49/a', msg = b'hi there'
PUBLISH topic = iot49/esp32 message = hello 23
PUBLISH topic = iot49/esp32 message = hello 24
PUBLISH topic = iot49/esp32 message = hello 25
```

MQTT Features

- Clients need only know broker, not each other
  - No “what’s your IP address”
  - Asynchronous:
    - No connection issues (“turn this on first, then …”)
- Text messages
  - Lightweight – good for low bandwidth situations
  - Easy parsing (you choose format)
- Topics
  - Organization
  - Hierarchical, separated by /, e.g.
    - solar/current, solar/voltage, kitchen/temperature, stocks/DJIA
    - alice/solar/current, fred/solar/current

Putting it all together …

- Broker
- Client
  - Python library “MQTTClient”
  - Topics
  - Messages
  - QoS
- Security

MQTT Broker

- "Hub" of the service
- Many service offerings:
  - Amazon AWS, Microsoft Azure, IBM Watson, …
  - Free brokers for testing (no security)
    - iot.eclipse.org
    - iot49.eecs.berkeley.edu
  - Roll your own …
    - https://mosquitto.org

Sharing Brokers

- Typically, brokers are shared among many users
  - Thousands or millions on commercial servers, e.g. Amazon AWS
- That's great for sharing data and collaboration, e.g.
  - `boston/temperature`
  - `berkeley/temperature`
  - …
- But if several users send unrelated information to the same topic, e.g. everybody in EE49
  - `solar_panel_current`
- … the result is a mess!

Cooperatively sharing MQTT Brokers

- Prefix all topics with unique identifier, e.g.
  - `bernhardboser/current`
  - `aliceguyon/current`
- Commercial brokers enforce this
- Others, e.g.
  - `iot.eclipse.org`
  - `iot49.eecs.berkeley.edu`
- … rely on users following an agreed convention
- EE49:
  - Prefix all topics with your name
  - Beware of multiple participants with same name … (UID?)

MQTTClient - connect

```
BROKER = "iot49.eecs.berkeley.edu"
USER = "iot49"
PWD = "rocks"

print("Connecting to broker", BROKER, "...")
mqtt = MQTTClient(BROKER, user=USER, password=PWD, ssl=True, )

print("Connected!")
```

subscribe

```
def mqtt_callback(topic, msg):
    print("RECEIVE topic = {}, msg = {}".format(topic, msg))

mqtt.set_callback(mqtt_callback)
mqtt.subscribe("iot49/a")
mqtt.subscribe("iot49/b")

while True:
    mqtt.check_msg()
```

What's wrong with these topics?

publish

```
topic = "iot49/esp32"
message = "hi there!"
mqtt.publish(topic, message)
```

publish – subscribe loop

```
for i in range(1000):
    topic = "iot49/esp32"
    message = "hello " + str(i)
    print("PUBLISH topic = {} message = {}".format(topic, message))
    mqtt.publish(topic, message)
    for _ in range(10):
        mqtt.check_msg()
        sleep(0.5)
```

Python strings / byte arrays

- byte:
  - 8-Bits of data
  - Can hold $2^8 = 256$ values
- char:
  - ~ 127 (ASCII) latin, decimals, and punctuation
  - Thousands with other alphabets (Greek, …)
- Python 3 treats bytes and chars different
  - Python 2 “blurs the lines”
  - Code that works in Python 2 won’t necessarily in Python 3
- Literals:
  - String: `'this is a string literal'`
  - Byte array: `b'this is a byte array literal'`

string / byte array conversions

```
>>> b'byte array'.decode('utf-8')
'byte array'
>>> 'some string'.encode('utf-8')
b'some string'
```

- Encoding:
  - E.g. utf-8 (many others)
  - For only latin characters, this rarely matters
  - For others, get funny symbols if incorrect …
- MQTT library uses byte arrays (not strings) …

Putting it all together …

- Broker
- Client
  - Python library “MQTTClient”
  - Topics
  - Messages
  - QoS
- Security

MQTT QoS

- QoS
  - 1: deliver at most one time
  - 2: deliver at least one time
  - 3: deliver exactly one time
- Optional arguments to publish and subscribe:
  - mqtt.publish(topic, message, qos=0)
  - mqtt.subscribe(topic, qos=0)
- Not all brokers and clients support all QoS levels
- MQTT has a few other features
  - E.g. last will
  - Check the online documentation

MQTT Security

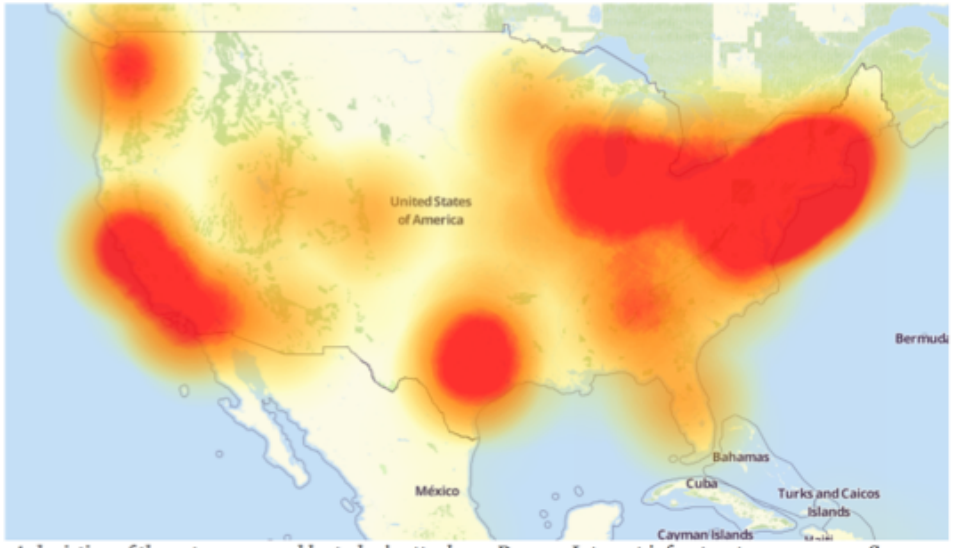
- Is security important in an experiment with solar cells?

Oct 21, 2016

21 OCT 16 Hacked Cameras, DVRs Powered Today's Massive Internet Outage

A massive and sustained Internet attack that has caused outages and network congestion today for a large number of Web sites was launched with the help of hacked "Internet of Things" (IoT) devices, such as CCTV video cameras and digital video recorders, new data suggests.

Earlier today cyber criminals began training their attack cannons on **Dyn**, an Internet infrastructure company that provides critical technology services to some of the Internet's top destinations. The attack began creating problems for Internet users reaching an array of sites, including Twitter, Amazon, Tumblr, Reddit, Spotify and Netflix.



A depiction of the outages caused by today's attacks on Dyn, an Internet infrastructure company. Source: Downdetector.com.

MQTT Security Fundamentals

- https://www.hivemq.com/mqtt-security-fundamentals/
- Basic security:
  1. Authentication
  2. Authorization
  3. TLS (Transport Layer Security)

MQTT Authentication

```
BROKER = "iot49.eecs.berkeley.edu"
USER = "iot49"
PWD = "rocks"

print("Connecting to broker", BROKER, "...")
mqtt = MQTTClient(BROKER, user=USER, password=PWD, ssl=True)
```

MQTT Authorization

MQTT TLS

```
BROKER = "iot49.eecs.berkeley.edu"
USER = "iot49"
PWD = "rocks"

print("Connecting to broker", BROKER, "...")
mqtt = MQTTClient(BROKER, user=USER, password=PWD, ssl=True)
```

Approach

- Get data from ESP32 to host computer (e.g. laptop)
- Then plotting is easy
  - Matlab, Excel, Python, …

Python Plotting Library

- https://matplotlib.org
- Similar to matlab

home | examples | gallery | pyplot | docs » The Matplotlib API »

pyplot

matplotlib.pyplot

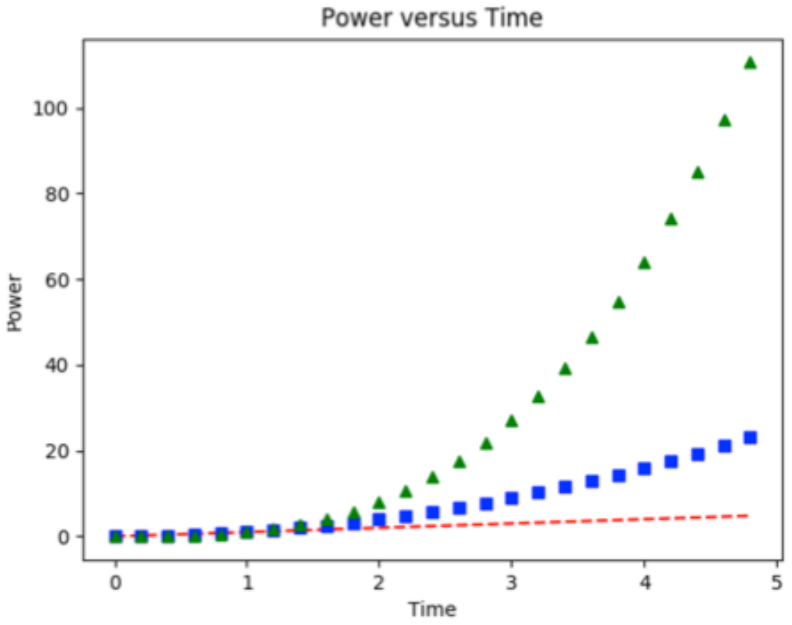
Provides a MATLAB-like plotting framework.

`pylab` combines pyplot with numpy into a single namespace. This is convenient for interactive work, but for programming it is recommended that the namespaces be kept separate, e.g.:

```
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(0, 5, 0.1);
y = np.sin(x)
plt.plot(x, y)
```

Pyplot Example

```
import numpy as np
import matplotlib.pyplot as plt

# evenly sampled time at 200ms intervals
t = np.arange(0., 5., 0.2)

# red dashes, blue squares and green triangles
plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
plt.xlabel('Time')
plt.ylabel('Power')
plt.title('Power versus Time')
plt.show()
```



Getting Data from ESP32 to Host

```
import json

# on ESP32 ...
def data2string(**args):
    string = json.dumps(args)
    return string

# on host ...
def string2data(string):
    data = json.loads(string)
    return data

msg = data2string(t=1.5, v=3, i=2)

# mqtt.publish('data', msg)

data   = string2data(msg)
```

- ** converts args to dict
- json.dumps converts dict to string
- Send string from ESP32 to host with MQTT
- Convert back to dict with json.loads

Remote Plotting Library

- https://github.com/bboser/iot-plot

iot-plot

Remote plotting library for resource constrained devices. Uses MQTT for communication. `plotclient` is compatible with MicroPython, enabling small microcontrollers to create high quality plots of measurement data.

Installation

```
pip install iot-plot
```

This also installs `matplotlib`, if not installed already. `iot-plot` requires Python 3.

To upgrade to a new version, run

```
pip install iot-plot --upgrade
```

Usage

Start the server on a host (e.g.\ laptop):

```
$ plotserver
```

PlotClient Example

```python
from plotclient import PlotClient
from mqttclient import MQTTClient
from math import sin, cos, exp, pi

mqtt = MQTTClient("iot.eclipse.org")
mp = PlotClient(mqtt, session="hopla")

# give the series a unique name (in case you create multiple plots)
SERIES = "sinusoid"

# data column names
mp.new_series(SERIES, 'time', 'cos', 'sin', 'sin*cos')

# generate the data
def f1(t): return cos(2 * pi * t) * exp(-t)
def f2(t): return sin(2 * pi * t) * exp(-t)
def f3(t): return sin(2 * pi * t) * cos(2 * pi * t) * exp(-t)
for t in range(200):
    t *= 0.025
    # submit each datapoint to the plot server
    mp.data(SERIES, t, f1(t), f2(t), f3(t))

# save data as pkl document
# see plot_load_pkl.py for an example of loading it back into python
mp.save_series(SERIES)

# create a plot, default dir is $IoT49
mp.plot_series(SERIES,
    filename="mqtt_plotter_example.pdf",
    xlabel="Time [s]",
    ylabel="Voltage [mV]",
    title=r"Damped exponential decay $e^{-t} \cos(2\pi t)$")
```

`iot.eclipse.org` is PlotServer default

Same session in PlotServer and PlotClient!

```
$ plotserver –session hopla
```

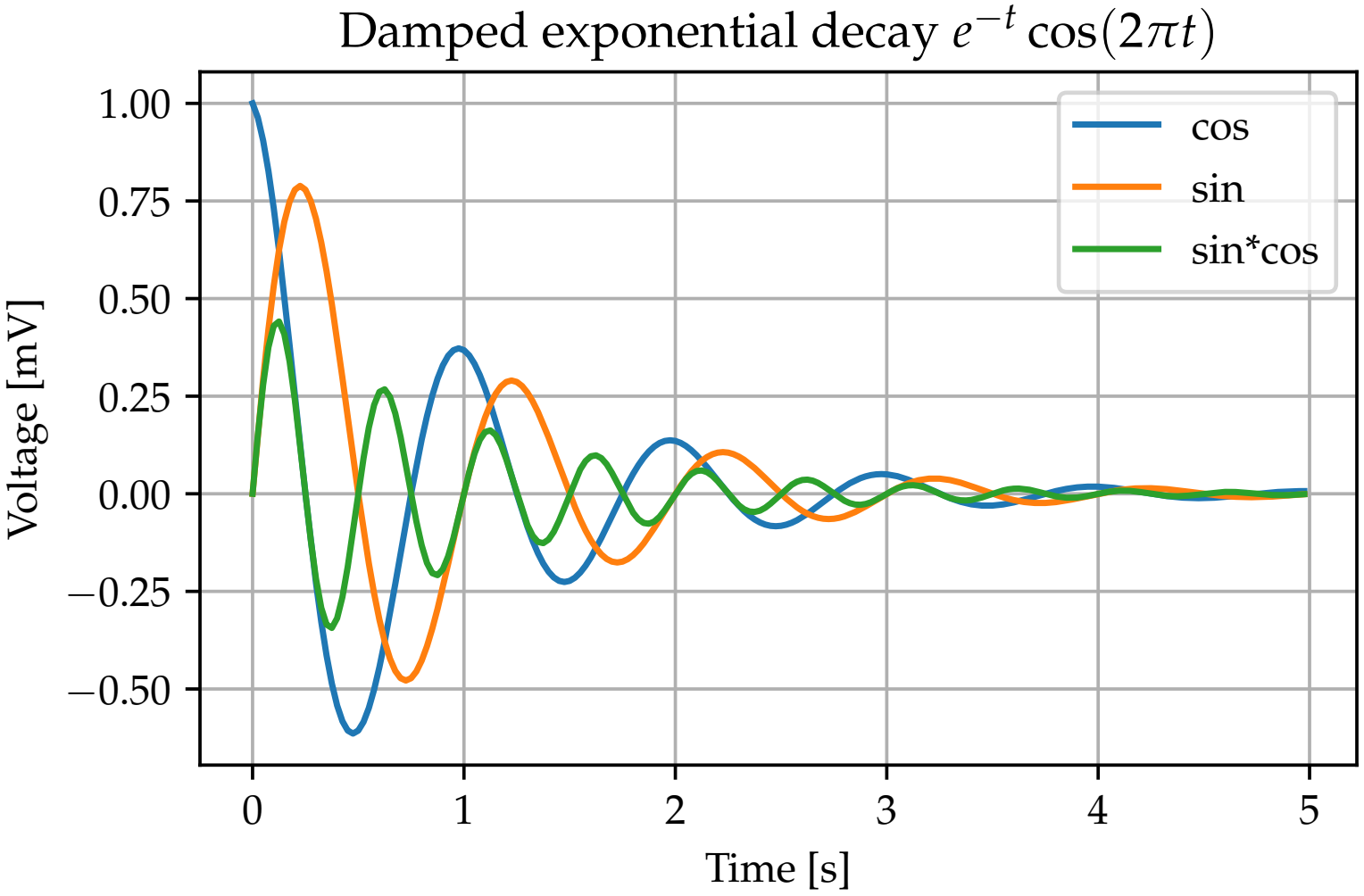
Define data column headings

Send data to PlotServer

Save data to file (optional)

Plot (to pdf file)

Example …


Damped exponential decay $e^{-t}\cos(2\pi t)$
Voltage [mV]
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
0
1
2
3
4
5
Time [s]
cos
sin
sin*cos

Skeleton of IoT App

1. Establish Internet connection
   - E.g. `boot.py`
2. Initializations
   - MQTT client
   - I2C and INA219
3. On host:
   - Start MQTT plot server
4. Collect data
   - Measure solar cell I/V characteristic and send to cloud
5. Instruct plot server to create plot

MQTT Summary

- MQTT
  - Machine-to-machine communication
  - Lightweight
- Many `clients` share same `broker`, e.g.
  - Broker iot.eclipse.org
  - Clients
    - ESP32 (sends solar power measurements)
    - Laptop (plots results)
- Messages
  - Topic
    - Prefix to disambiguate
  - Message content (“array of bytes”)

What's Next

- Lab:
  - Put all this together
- Lecture …
  - Low power applications that run "indefinitely"
  - Energy harvesting (e.g. solar cell)
  - Energy …