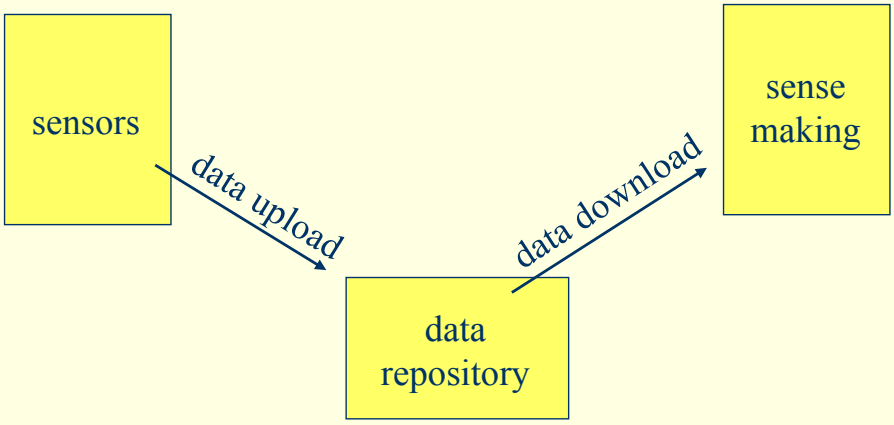
SE-IOT: Internet of Things

Networking Communications

Derek Kiong
dkiong@nus.edu.sg

Distributed systems/communication

- Network socket library/frameworks allow data transfer



Sockets

- Network TCP sockets are bi-directional asymmetric network data streams
  - Server sockets listen and accept connections from client sockets on pre-selected port
  - Client sockets attempt connections on known port (not necessarily implemented in Python)
  - Following network connection, read/writes are symmetric and similar to file I/O

Basic Server Socket

```
import socket
host = ''
port = 50000
backlog = 5
soc = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
soc.bind((host,port))
soc.listen(backlog)
while 1:
    client, address = soc.accept()
    f = client.makefile(mode="rw")
    input = f.readline().strip()
    print('%s: read [%s]' % (address, input))
    f.write('echo [%s]' % (input))
    f.close()
    client.close()
```


Basic Client Socket

```
import socket
from datetime import datetime
host = 'localhost'
port = 50000
soc = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
ip = socket.gethostbyname(host)
soc.connect((ip, port))

f = soc.makefile(mode="rw")
f.write('Hello there. It is now %s.\n' %
(datetime.now(),))
f.flush()
reply = f.readline()
print(reply)
f.close()
soc.close()
```


Application protocol on network socket eg HTTP

```
import socket
hostname = 'www.iss.nus.edu.sg'
path = '/'
soc = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
ip = socket.gethostbyname(hostname)
soc.connect((ip, 80))
f = soc.makefile(mode="rw")
f.write('GET %s HTTP/1.0\r\n' % (path,))
f.write('Host: %s\r\n' % (hostname,))
f.write('\r\n')
f.flush()
reply = f.read()
print(reply)
f.close()
soc.close()
```


Libraries for popular Internet protocols

- Python modules **`urllib.request`** (Python3) and **`urllib2`** (Python2) implement **`urlopen()`**

```
import urllib2

target = "http://www.iss.nus.edu.sg/"
f = urllib2.urlopen(target)
content = f.read().decode("utf-8")
f.close()
print content
```


Internet protocols

- Python libraries **`ftplib`**, **`poplib`**, **`imaplib`**, **`smtplib`**, **`nntplib`** implement well-known protocols **`ftp`**, **`pop`**, **`imap`**, **`smtp`** and **`nntp`**
- See **`https://docs.python.org/2/library/internet.html`**

Multi-Threading

- Threading in Python may be achieved by passing a callable object (may be function) to the **`Thread`** class or subclassing from it

```
from threading import Thread

def thread1(left, right):
    for i in range(12):
        time.sleep(1)
        print "%s%d%s " % (left,i,right),
        sys.stdout.flush()

t1 = Thread(target=thread1, args=('[',']'))
t1.start()
thread1('<','>')
```


Multi-Threading 2

- A callable object may **`__call__()`** method

```
from threading import Thread

class thread2(object):
    def __call__(self, left, right):
        for i in range(12):
            time.sleep(1)
            print "%s%d%s " % (left,i,right),
            sys.stdout.flush()

t2 = Thread(target=thread2(), args=('[',']'))
t2.start()
t = thread2()
t('<','>')
```


Multi-Threading 3

- A subclass of Thread with overriding **`run()`** method

```
from threading import Thread

class thread3(Thread):
  def run(self):
    for i in range(12):
      time.sleep(1)
      print "%s%d%s "%(self._Thread__args[0],i,self._Thread__args[1]),
      sys.stdout.flush()

t3 = thread3(args=('{','}'))
t3.start()
thread1('<','>')
```


Multi-threaded Server Socket

```
def serviceto(client):
    f = client.makefile(mode="rw")
    input = f.readline().strip()
    print('%s: read [%s]' % (address, input))
    f.write('echo [%s]' % (input))
    f.close()
    client.close()

soc = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
soc.bind((host,port))
soc.listen(backlog)
while 1:
    client, address = soc.accept()
    thread1 = Thread(target=serviceto, args=(client,))
    thread1.start()
```


Multi-threading safety

- Multi-threading requires proper synchronization for thread-safe operations
  - Short term synchronization
    - Block until exclusive access is guaranteed
    - eg race condition when multiple threads increment counter
  - Long term synchronization
    - Block until resource available
    - eg delays when writing to full buffer

Http Server

- Handler methods may be overriden

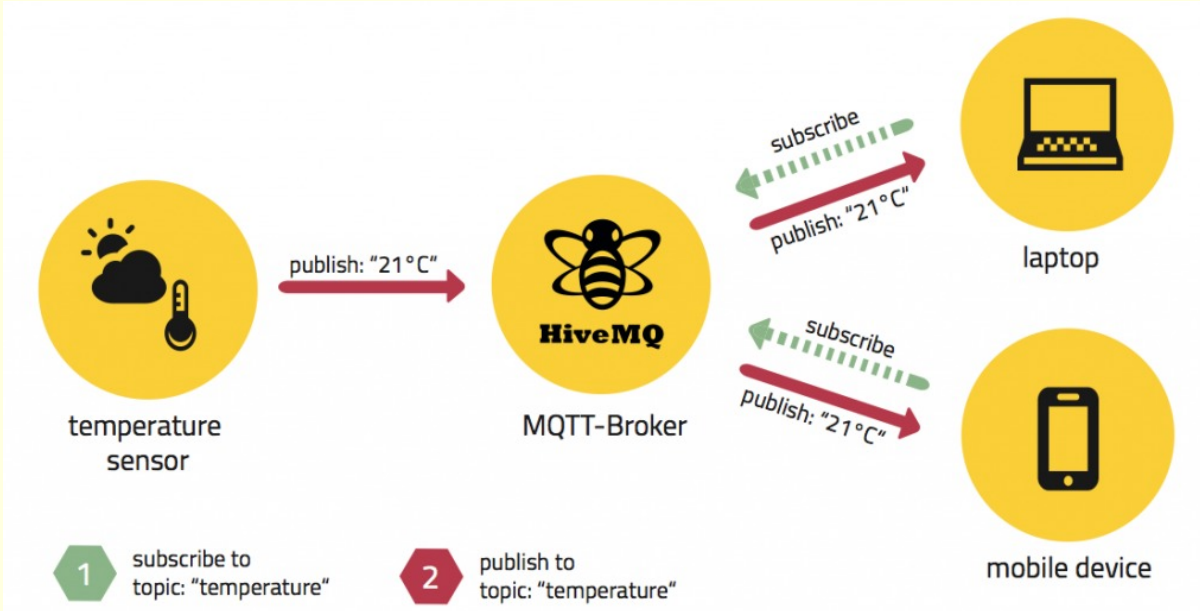
```
import BaseHTTPServer, SimpleHTTPServer

PORT = 8000

Handler = SimpleHTTPServer.SimpleHTTPRequestHandler
httpd = BaseHTTPServer.HTTPServer(("", PORT), Handler)
print "serving at port", PORT
httpd.serve_forever()
```


Device admin via http

- Configuration/admin functions via embedded httpd server such as
  - **`micro-http`**
  - **`mini-http`**
  - **`nginx-light`**
  - **`nginx-full`**
- PHP
- Django provides database integration
  - (Python includes driver-libraries for Postgres and MySQL)
  - **`python-django`**

MQTT

- Lightweight **publisher-subscriber** messaging for IOT environment



MQTT vs Client/Server

- Alternative to the traditional client-server model
  - **Space decoupling**: Publisher and subscriber do not need to know each other (by IP address and port for example)
  - **Time decoupling**: Publisher and subscriber do not need to run at the same time.
  - **Synchronization decoupling**: Operations on both components are not halted during publish or receiving

MQTT

- `mosquitto` and `mosquitto-clients` packages provides
  - MQTT broker
  - MQTT subscriber/publisher client
- `python-mosquitto` provides corresponding Python bindings

MQTT

- Broker `/usr/sbin/mosquitto` has configuration in `/etc/mosquitto/mosquitto.conf`
- Clients `/usr/bin/mosquitto_pub` and `/usr/bin/mosquitto_sub`

```
$ mosquitto_sub -h broker.host.com -v -t temp

$ mosquitto_pub -h broker.host.com -t temp \
                        -m 22.5
```


MQTT publisher – Python bindings

```
from mosquitto import *
import datetime

client = Mosquitto("my_id_pub")
client.connect("localhost")
message = str(datetime.datetime.now())
topic = "temp"
client.publish(topic, message)
```

MQTT subscriber – Python bindings

```
from mosquitto import *
def on_message(mqttc, userdata, mesg):
    print "message: % %" %
    (str(mesg.topic), str(mesg.payload))

client = Mosquitto("my_id_sub")
client.connect("localhost")
client.on_message = on_message
client.subscribe("#")
while True:
    client.loop()
```


Remote Job Submission

- Secure shell **`ssh`** allows for remote login via terminal shell
- Secure copy **`scp`** for file copy over **`ssh`** transport tunnel
- Remote shell **`rsh`** for remote command execution over **`ssh`**
- Secure shell forwarding **`ssh -L`** for socket abstract within **`ssh`** tunnel

ssh authentication via cryptographic key pair

- Why? Submit computationally intensive jobs. Scripts may be automated without authentication password
- Generate key pair via **ssh-keygen**

```
$ ssh-keygen -f abc
$ ls -l abc*
-rw------- 1 dkiong dkiong 1675 Mar 20 14:12 abc
-rw-r--r-- 1 dkiong dkiong  409 Mar 20 14:12 abc.pub
```

- Copy public key over to server account via **ssh-copy-id**

```
$ ssh-copy-id -i abc.pub dkiong@172.27.246.86
```

- ssh login using private key

```
$ ssh -i abc dkiong@host.com
```

- (used by nscc.sg)

Summary

- Network communication to be simplified, standardised and multi-platform
- Abstracts sensor (location)
- Secured communication (confidentiality) via https/encryption