FALMOUTH
UNIVERSITY

Lecture 10: Packet Management

COMP260: Distributed Systems
BSc (Hons) Computing for Games

- Today's session:
  - User Management
  - Data Packet issues
  - Managing Network Data
  - Workshop

- User Management
  - Did everything go ok in the last workshop?
  - Any issues to report?

- Data Packet issues

- Data Packet issues
  - You may have noticed that not all of your messages always make it from the server to the client
    - Typically, we can get round this by throttling server->client sending
      - 1 socket.send() at a time
    - However, pausing the client and running the server will demonstrate this problem

- Data Packet issues

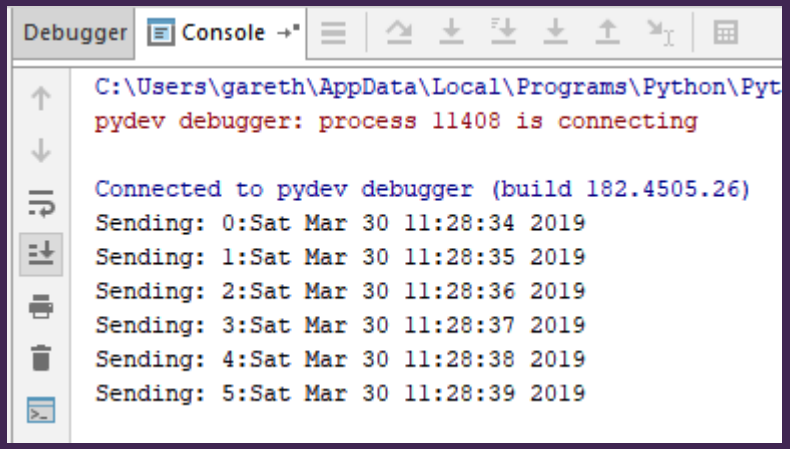
```python
import socket
import time

if __name__ == '__main__':
    mySocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    mySocket.bind(("127.0.0.1", 8222))
    mySocket.listen(5)

    client = mySocket.accept()

    seqID = 0

    while True:
        testString = str(seqID) +":" + time.ctime()

        print('Sending: ' + testString)

        client[0].send(testString.encode())

        seqID+=1
        time.sleep(1)
```

Server.py

```python
import socket

if __name__ == '__main__':
    mySocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    mySocket.connect(("127.0.0.1", 8222))

    while True:
        data = mySocket.recv(4096)
        print(data.decode("utf-8"))
```

Client.py

- Data Packet issues
  - Running the code

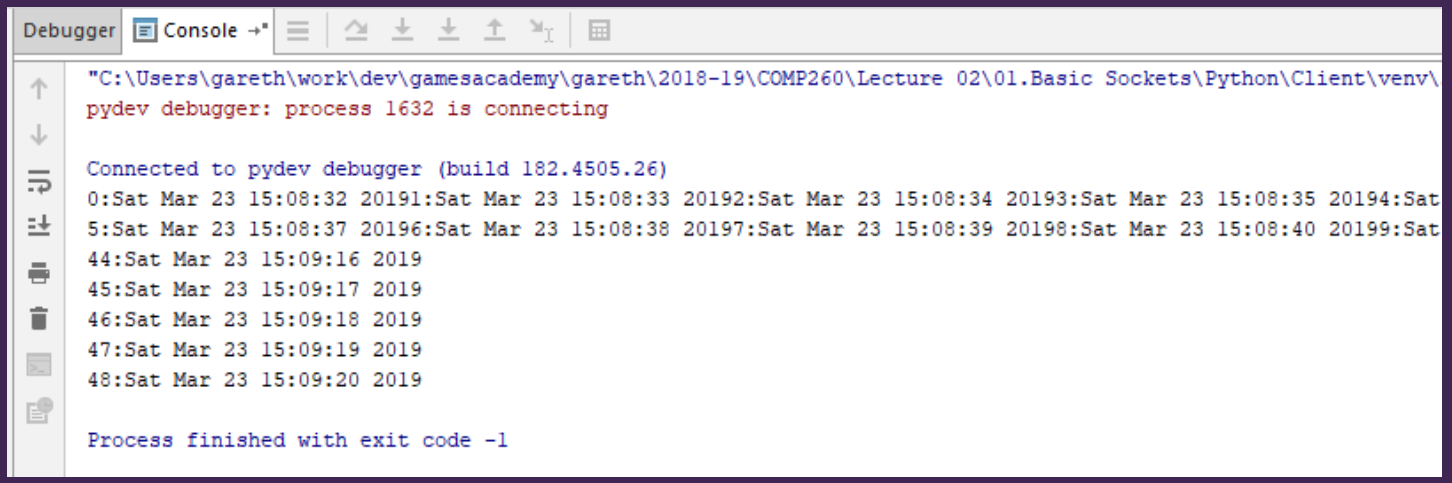



```
import socket

if __name__ == '__main__':
    mySocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    mySocket.connect(("127.0.0.1", 8222))

    while True:
        data = mySocket.recv(4096)
        print(data.decode("utf-8"))
```

Server.py

- If client is paused in debugger, data will still be sent
- But not processed

- Data Packet issues



  - Socket is effectively a pipe from server to client
    - This is kind of ok a lot of the time
    - In this case we ‘just’ end up with a lack of <CR> to break up the lines, but this looks nasty
    - Could be worse if we need to process the input strings

- Data Packet issues
  - In part, this isn't helped by the large value in the socket.recv

```
import socket

if __name__ == '__main__':
    mySocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    mySocket.connect(("127.0.0.1", 8222))

    while True:
        data = mySocket.recv(4096)
        print(data.decode("utf-8"))
```

  - How do we know what size of data we are dealing with?
  - And how can we work out how many messages we are dealing with?

- Managing Network Data

- Managing Network Data
  - Need to think of a more robust approach to network data
    - This will also tie into packet data security
    - Effectively our current data packet is:
      - <DATA.N>
      - Never a good plan to have N bytes of data when we don't know what N is

- Managing Network Data
  - Approach 1: data parsing
    - The data that comes of out socket.recv is an array of bytes
      - We could:
        - » 1. read an arbitrary about of bytes from socket.recv() and parse the array of bytes
          - Like C/C++ style string parsing
        - » 2. read the stream with socket.recv(1) to pop byte data and parse
      - However,
        - » In this case, this data has no terminators like C-style strings, so we can't take that approach
        - » We will need to create our own packet form to parse

- Managing Network Data
  - Approach 2: fixed length packets
    - Let's define a protocol for data communications where each message is 100 bytes in the length (other sizes are available
    - Effectively a data packet of:
      - <DATA.100>

- Managing Network Data
  - Approach 2: fixed length packets
    - Let's define a protocol for data communications where each message is 100 bytes in the length (other sizes are available

```
data = bytearray(100)

while True:
    testString = str(seqID) + ':' + time.ctime()

    for index in range(len(data)):
        data[index]=0

    src = testString.encode()
    for index in range(len(src)):
        data[index] = src[index]

    client[0].send(data)

    print('Sent: '+str(seqID))
    seqID+=1
    time.sleep(1)
```

Server.py

```
if __name__ == '__main__':
    mySocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    mySocket.connect(("127.0.0.1", 8222))

    while True:
        data = mySocket.recv(100)
        print(data.decode("utf-8"))
```

Client.py

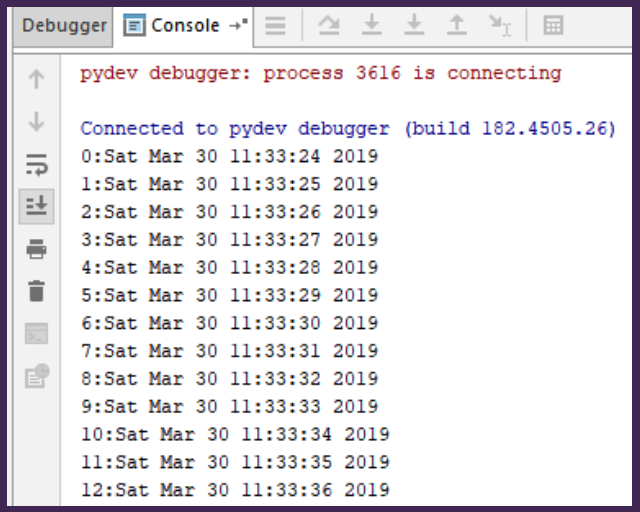
- Managing Network Data
  - Approach 2: fixed length packets
    - If I run the client & server but leave the client on a breakpoint, the server will send messages and they will get queued up

```
Connected to pydev debugger (build 182.4505.26)
Sent: 0
Sent: 1
Sent: 2
Sent: 3
Sent: 4
Sent: 5
Sent: 6
Sent: 7
Sent: 8
Sent: 9
Sent: 10
Sent: 11
```

Server.py

```
7
8       while True:
9           data = mySocket.recv(100)
10          print(data.decode("utf-8"))
11
```

Client.py



- Managing Network Data
  - Approach 2: fixed length packets
    - If I step over the client breakpoint, a single message will come out of the socket.recv() function, rather than a huge wad of data

```
Connected to pydev debugger (build 182.4505.26)
0:Sat Mar 23 15:38:33 2019
```

    - If I keep single-stepping through the client, one message per loop will be received until the receive socket is empty

- Managing Network Data
  - Approach 2: fixed length packets
    - This gives us good control over data, but it does require an arbitrary sized packet
      - This is good if most of the data is roughly the same size
      - But bad if there's a huge range of sizes, say 1byte to 1Kb

- Managing Network Data
  - Approach 3: variable length packets
    - Clearly, if a fixed length packet wont work, the obvious next step is a variable length packet
    - This means we need to send the size of the packet as a data header
    - And the packet data
    - Effectively a data packet of:
      - <LENGTH.2><DATA.N>
      - (assuming up 2^16 bytes of data – 32kb)

- Managing Network Data
  - Approach 3: variable length packets

```
while True:
    testString = str(seqID) + ':' + time.ctime()

    header = len(testString).to_bytes(2, byteorder='little')

    client[0].send(header)
    client[0].send(testString.encode())


    print('Sent: '+str(seqID))
    seqID+=1
    time.sleep(1)
```

Server.py

```
while True:
    payloadSize = int.from_bytes(mySocket.recv(2),'little')
    print(payloadSize)
    data = mySocket.recv(payloadSize)
    print(data.decode("utf-8"))
```

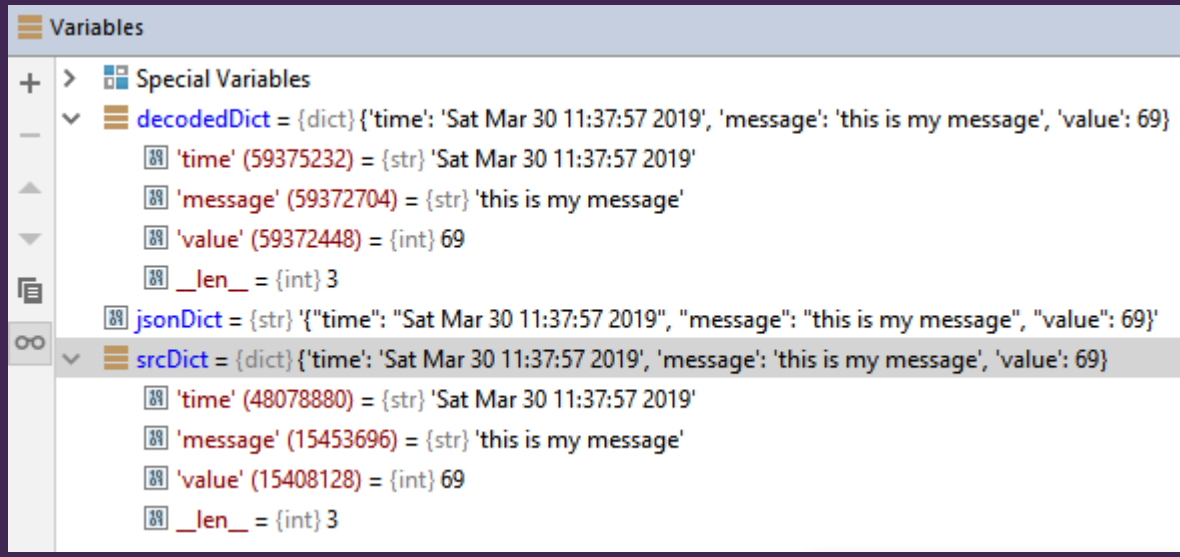
Client.py

- Sending size+data is the core of serialisation / deserialization
- See C# serialisation for more details

- Managing Network Data
  - Approach 4: JSON
    - JSON allows us to send complex data as an encoded text string that can be examined as an organised data type in Python
      - Saw this in COMP130 last year

- Managing Network Data
  - Approach 4: JSON
    - JSON allows us to send complex data as an encoded text string that can be examined as an organised data type in Python
      - Saw this in COMP130 last year

```
srcDict = {"time": time.ctime()
    , "message": 'this is my message'
    , "value": 69}

jsonDict = json.dumps(srcDict)
decodedDict = json.loads(jsonDict)
```



- Managing Network Data
  - Approach 4: JSON
    - Combine this with the variable length packet, so:
      - 1. Make Python data dictionary
      - 2. Convert to JSON
      - 3. Send packet size len(JSON data)
      - 4.Send JSON data as bytes (JSON.encode)
    - On the client
      - 1. Read payload size
      - 2. Read payload
      - 3. JSON.loads(payload)

- Managing Network Data
  - Approach 4: JSON

    - Effectively this is still a data packet of:
      - <LENGTH.2><DATA.N>

      - With more data encoding in the <DATA.N> part

- Managing Network Data
  - Approach 4: JSON

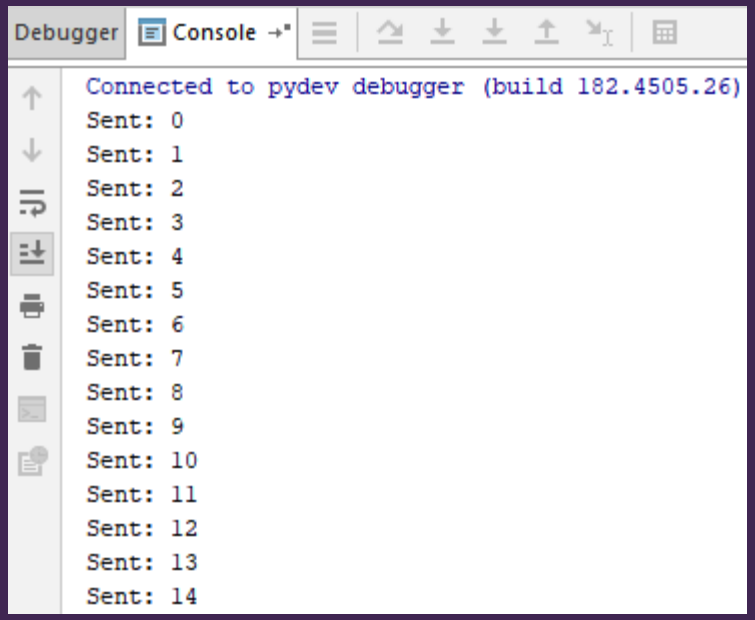
```
while True:
    mydict = {"time": time.ctime()
        , "message": ""
        , "value": seqID}

    mydict['message'] = str(mydict['value']) + ':' +mydict['time']

    jsonPacket = json.dumps(mydict)

    header = len(jsonPacket).to_bytes(2, byteorder='little')

    client[0].send(header)
    client[0].send(jsonPacket.encode())


    print('Sent: '+ str(mydict['value']) )
    seqID+=1
    time.sleep(1)
```

Server.py

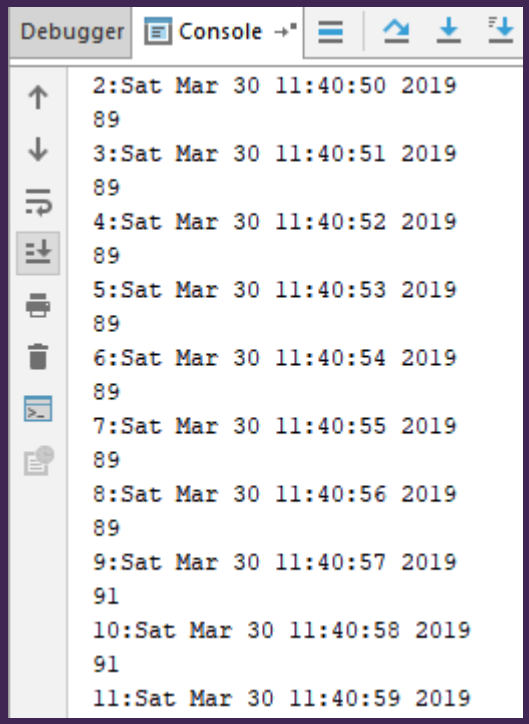
```
while True:
    payloadSize = int.from_bytes(mySocket.recv(2),'little')
    print(payloadSize)
    payloadData = mySocket.recv(payloadSize)

    myData = json.loads(payloadData)

    print(myData['message'])
```

Client.py

- Managing Network Data
  - Approach 4: JSON



Server.py



Client.py

- Managing Network Data
  - Are these my packets?
    - We've seen that (currently) everyone's clients and servers are fairly interchangeable
      - You're using the same protocol
        - » data packets of byte streams
    - Let's put a stamp on our packets so we know they belong to the application

- Managing Network Data
  - Are these my packets?
    - Already, we are assuming a packet format of
      - <LENGTH.2><DATA.N>
    - If we add an ID to the packet, we'll know it's ours
      - <ID.4><LENGTH.2><DATA.N>

- Managing Network Data
  - Are these my packets?

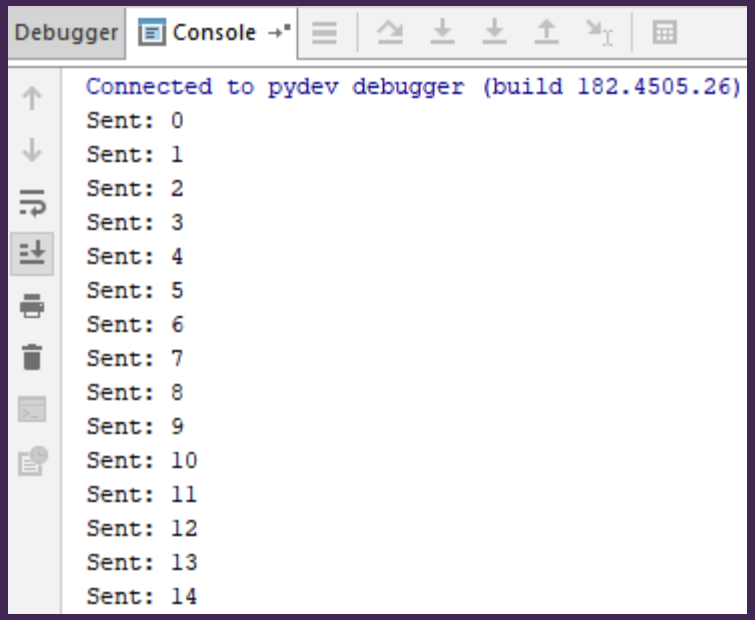
```
packetID = 'GMUD'

while True:
    mydict = {"time": time.ctime()
        , "message": ""
        , "value": seqID}

    mydict['message'] = str(mydict['value']) + ':' +mydict['time']

    jsonPacket = json.dumps(mydict)

    header = len(jsonPacket).to_bytes(2, byteorder='little')

    client[0].send(packetID.encode())
    client[0].send(header)
    client[0].send(jsonPacket.encode())


    print('Sent: '+ str(mydict['value']) )
    seqID+=1
    time.sleep(1)
```

Server.py

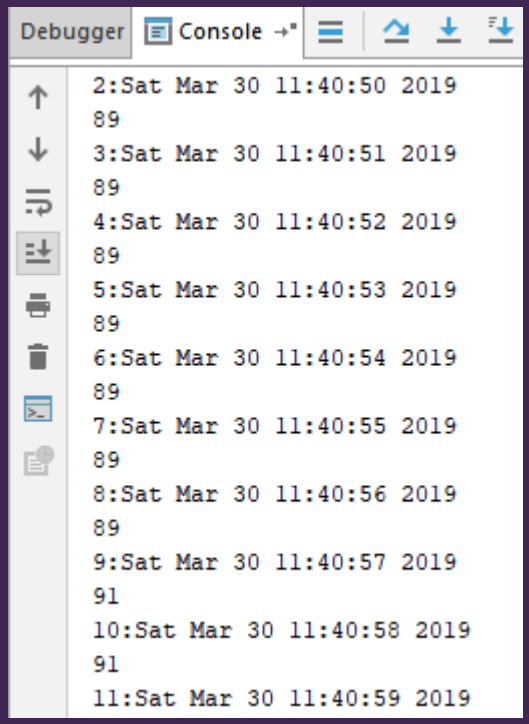
```
while True:
    packetID = mySocket.recv(4)

    if packetID.decode("utf-8") == 'GMUD':
        payloadSize = int.from_bytes(mySocket.recv(2),'little')
        print(payloadSize)
        payloadData = mySocket.recv(payloadSize)

        myData = json.loads(payloadData)

        print(myData['message'])
    else:
        print('Invalid Packet')
```

Client.py

- Managing Network Data
  - Are these my packets?



Server.py



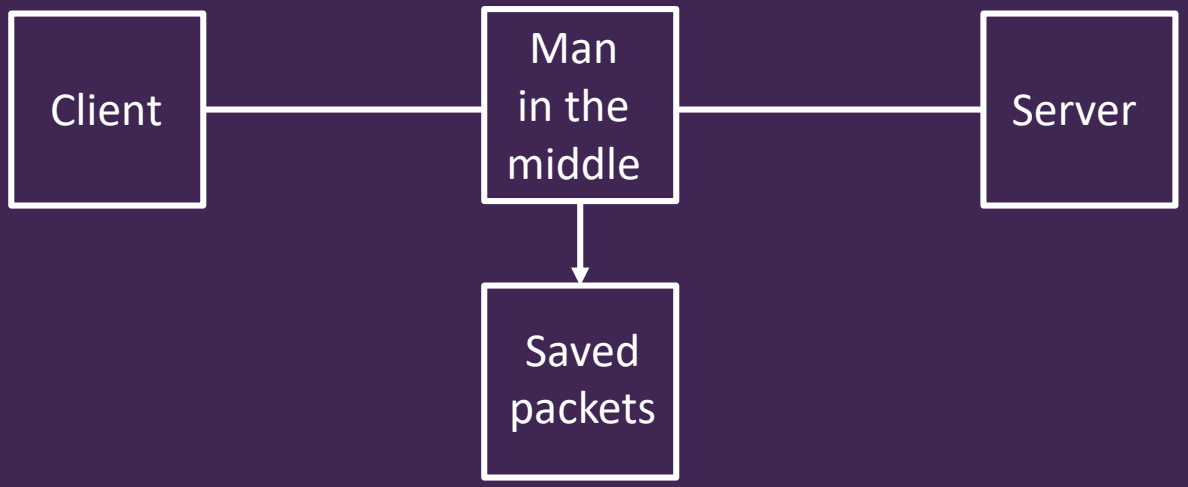
Client.py

- Managing Network Data
  - Passive and active attacks from wk5
    - Passive: reading the contents
      - How do we stop people?
        - » Encryption / Decryption (next week)
    - Active: ‘doing things with packets’
      - Message replay
      - Message modification
      - Message spamming

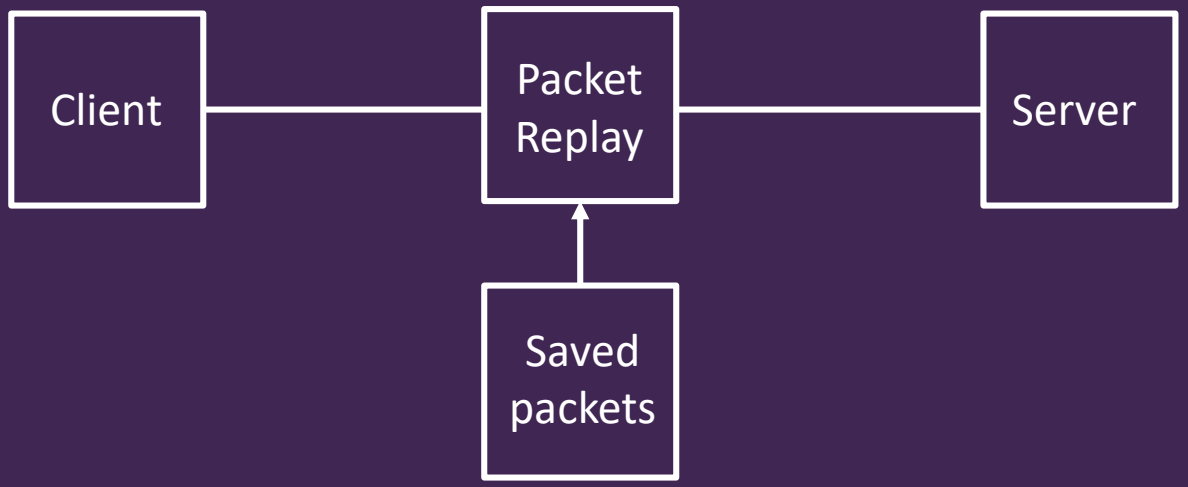
- Managing Network Data
  - Passive and active attacks from wk5
    - Passive: reading the contents
      - How do we stop people?
        - » Encryption / Decryption (next week)
    - Active: ‘doing things with packets’
      - Message replay
        - » Stop messages from being replayed by encoding sequences into packets
      - Message modification
        - » Stop modification by encrypting packet contents
      - Message spamming

- Managing Network Data
  - Using packet sequence numbers to stop replay hacks
    - A replay hack consists of a stream of client -> server messages being captured and then replayed

- Managing Network Data
  - Using packet sequence numbers to stop replay hacks
    - A replay hack consists of a stream of client -> server messages being captured and then replayed



- Managing Network Data
  - Using packet sequence numbers to stop replay hacks
    - A replay hack consists of a stream of client -> server messages being captured and then replayed



- Managing Network Data
  - Using packet sequence numbers to stop replay hacks
    - Simple sequencing
      - Client and server set sequence ID to 0 on connect/accept
      - Client sends sequence as part of packet & increments
      - Server tests on packet receipt
      - Remember, TCP packet will always arrive and in the correct order
    - Session sequencing
      - As above,
      - But server will assign session ID to client on accept
      - Client adds that to packet send data
      - Server tests on packet receipt

- Managing Network Data
  - Using packet sequence numbers to stop replay hacks
    - Simple sequencing
      - If sequence number is out of sync -> packet replay / packet hacking
    - Session sequencing
      - If session ID is wrong -> packet replay / packet hacking

- Managing Network Data
  - Using packet sequence numbers to stop replay hacks
    - Simple sequencing
      - If sequence number is out of sync -> packet replay / packet hacking
    - Session sequencing
      - If session ID is wrong -> packet replay / packet hacking

- Managing Network Data
  - Approach 6: sequence numbers
    - Part 1: send session data

```python
import socket
import time
import json
import random

packetLabel = 'GMUD'

if __name__ == '__main__':
    mySocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    mySocket.bind(("127.0.0.1", 8222))
    mySocket.listen(5)

    client = mySocket.accept()

    sessionData = {"session_id": random.randrange(2^16)
        , "sequence_id": 0}

    jsonPacket = json.dumps(sessionData)

    header = len(jsonPacket).to_bytes(2, byteorder='little')

    client[0].send(packetLabel.encode())
    client[0].send(header)
    client[0].send(jsonPacket.encode())
```

Server.py

```python
import socket
import json

packetLabel = 'GMUD'

if __name__ == '__main__':
    mySocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    mySocket.connect(("127.0.0.1", 8222))

    packetID = mySocket.recv(4)
    payloadSize = int.from_bytes(mySocket.recv(2), 'little')
    payloadData = mySocket.recv(payloadSize)
    sessionData = json.loads(payloadData)
```

Client.py

- Server accepts client and sends session data as json packet
- Client receives json data and decodes

- Managing Network Data
  - Approach 6: sequence numbers
    - Part 2: server sends ‘normal’ data, client receives

```python
while True:
    #send data to client
    clientData = {"time": time.ctime()
        , "message": ""
        , "value": seqID}

    clientData['message'] = str(clientData['value']) + ':' + clientData['time']

    jsonPacket = json.dumps(clientData)

    header = len(jsonPacket).to_bytes(2, byteorder='little')

    client[0].send(packetLabel.encode())
    client[0].send(header)
    client[0].send(jsonPacket.encode())

    ##increment session id as message has been sent
    sessionData['sequence_id'] +=1

    print('Sent: ' + str(clientData['value']))
    seqID+=1
```

Server.py

```python
while True:

    ##receive data from server
    packetID = mySocket.recv(4)

    if packetID.decode("utf-8") == packetLabel:
        payloadSize = int.from_bytes(mySocket.recv(2),'little')
        print(payloadSize)
        payloadData = mySocket.recv(payloadSize)

        myData = json.loads(payloadData)

        print(myData['message'])
    else:
        print('Invalid Packet')

    sessionData['sequence_id'] += 1
```

Client.py

- Session data is updated on both sides

- Managing Network Data
  - Approach 6: sequence numbers
    - Part 3: client sends data to server which is validated

```python
#get a response from the client
packetID = client[0].recv(4)

if packetID.decode("utf-8") == packetLabel:
    payloadSize = int.from_bytes(client[0].recv(2), 'little')
    payloadData = client[0].recv(payloadSize)
    clientSessionData = json.loads(payloadData)

    if (clientSessionData['session_id'] != sessionData['session_id']) \
        or (clientSessionData['sequence_id'] != sessionData['sequence_id']) :
        print('Session data wrong - replay hack?')
    else:
        print('Client valid')

else:
    print('Invalid Packet')
```

Server.py

```python
#send data to server
clientData = {'session_id' : sessionData['session_id']
              ,'sequence_id': sessionData['sequence_id']
              ,'data': 'this is test data'
              }

jsonPacket = json.dumps(clientData)

header = len(jsonPacket).to_bytes(2, byteorder='little')

mySocket.send(packetLabel.encode())
mySocket.send(header)
mySocket.send(jsonPacket.encode())
```

Client.py

- Client sends a message to server with session data
- Server decodes and validates

- Managing Network Data
  - Approach 6: sequence numbers
    - Rather than ‘just’ being about data, it’s also about communications between client and server

- Workshop
  - This week: hosting your server apps on a remote linux server

- Questions