Introduction

Creating a client and server connection socket program, communication is called bilateral after the ends of TCP/IP socket are connected. The *server* program is a socket that can be configured as a typical web server. Meanwhile, the *client* program acts as the applications that are communicating with the server.

Process

Server Side

When a client wanted to access the server (any web server), client program creates a connection to the server using the program below. On the basis of standard library documentation, the server program receives incoming messages and echos them back to the sender. This is the start of creating a thing we call the TCP/IP socket.

```
import socket
import sys
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_address = ('localhost', 10000)
sock.connect(server_address)
```

With the right socket connection though an IP address and proper port you would be successful in creating your connection. Now we are binded the socket to the port. Calling listen() puts the socket to server mode and will accept for the incoming connection.

```
sock.bind(server_address)
sock.listen(1)
while True:
    connection, client_address = sock.accept()
```

The program above will also be our *Connection Setup Phase(CSP)*. This is the first phase in the protocol where the client informs the server that it wants to conduct active network measurements in order to compute the RTT and throughput of its path to the server.

The accept() function will open a connection between the server and client through thee IP address of the client. In this part we call this as the *Measurement Phase(MP)*. In this phase, the client starts sending probe messages to the server in order to make the appropriate measurements required for computing the mean RTT or the mean throughput of the path connecting it to the server.

```
   try:
        print '1 OK'
        print >>sys.stderr, 'connection from', client_address
        # Receive the data in small chunks and retransmit it
        while True:
            data = connection.recv(1)
            print 'm  1  1'
            print >>sys.stderr, 'received "%s"' % data
            if data:
                        #add delay
                print >>sys.stderr, 'sending data back to the client'
                connection.sendall(data)
            else:
                print >>sys.stderr, 'no more data from', client_address
                break
```

Data is being read from the connection with recv() function and transmitted with sendall() function.

We go now on the third phase which is the *Connection Termination Phase (CTP)*. In this phase, the client and the server attempt to gracefully terminate the connection. After sending the messages in TCP and when communication is finished, the connection between server and client must be close using close() function. See the next example below which uses try: finally block to ensure the close() is always called and print error in the event occur.

```
      try:
            # Clean up the connection
            print '1 OK: Closing Connection'
            print 't  .'
            connection.close()
        except ValueError:
            print '404 ERROR: Invalid Connection Termination Message'
```

Client Side

We are done how server setup its side for the client to communicate within, client program sets up its socket differently from the way server does. Instead of binding to a port and listening, it uses connect() function to attach the socket to the remote address. Again this part serves as the CSP to client.

```
import socket
import sys
```

Now to create TCP/IP socket.

```
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_address = ('localhost', 10000)
```

This part connects the socket to the port where the server is listening.

```
sock.connect(server_address)
```

Connection is established at this point. The data can be send using the sendall() function and receiving the message using recv() function. This part is the MP of the client where we measure the round trip time(rtt) and throughput (TPUT).

```
try:
    # Send data
    message = 'String of message'
    t1 = time.time()
    print >>sys.stderr, 'sending "%s"' % message
    sock.sendall(message)

    # Look for the response
    amount_received = 0
    amount_expected = len(message)

    while amount_received < amount_expected:
        data = sock.recv(100)
        amount_received += len(data)
        print >>sys.stderr, 'received "%s"' % data
```

This part measures the RTT and TPUT, getting time before and after sending and using the t2- t1 formula, we get the round trip time(RTT) measurement.

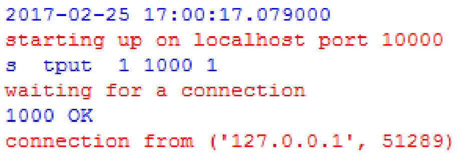
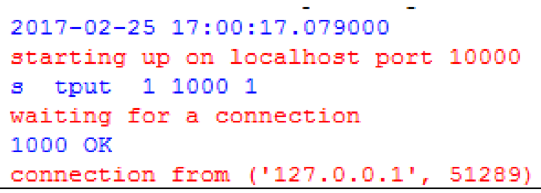
```
    t2 = time.time()
    print 'RTT is', t2 - t1, 'seconds'
```

TPUT is the troughput of its path to the server. We measure that by the formula (TCP Receive Window)/RTT unit of bytes/seconds.

```
    print 'TPUT is', 1000/(t2-t1), 'bytes/seconds'
```

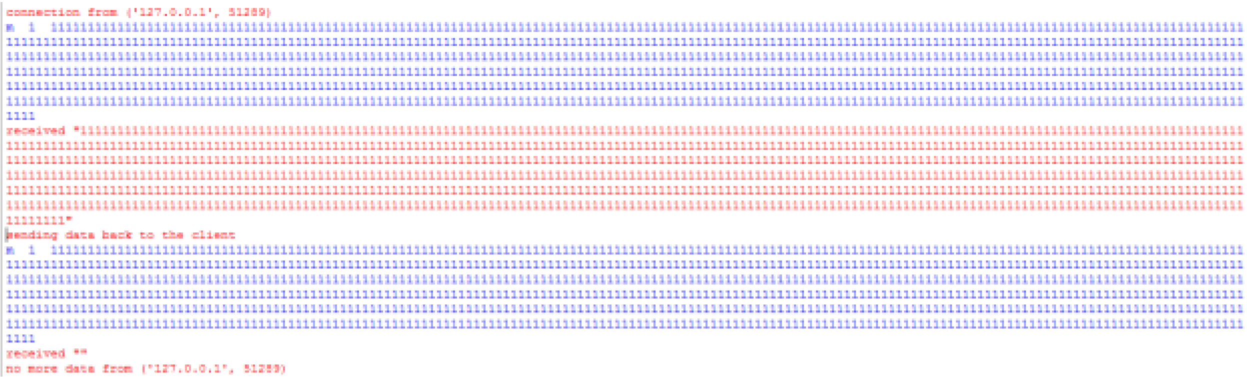
Results

The results shows the probe messages of each protocol (CSP, MP, TCP) and the sending and receiving of each messages between client and server.

	Server	Client
CSP	 2017-02-25 17:00:17.079000 starting up on localhost port 10000 s tput 1 1000 1 waiting for a connection 1000 OK connection from ('127.0.0.1', 51289)	 2017-02-25 17:00:17.079000 starting up on localhost port 10000 s tput 1 1000 1 waiting for a connection 1000 OK connection from ('127.0.0.1', 51289)

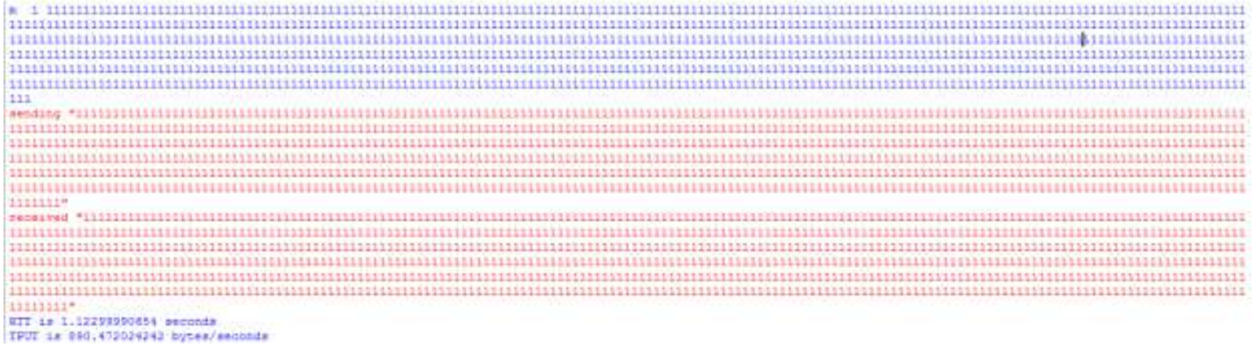
MP

Server:



```
connection from ('127.0.0.1', 51289)
m  1  11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111
received "1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111"
sending data back to the client
m  1  11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111
received ""
no more data from ('127.0.0.1', 51289)
```

Client



```
m: 1 11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
111
sending "111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111"
received "11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111"
RTT is 1.12299990654 seconds
TPUT is 890.472024242 bytes/seconds
```

RTT and TPUT measurement above are printed and computed with the formula explained above.

	Server	Client

TCP	`1000 OK: Closing Connection`
	`t .`
		`t .`
		`closing socket`

Conclusion

I have learned that socket interface provides the basic networking services. Now I understand how applications are connecting to the web server because of this theory. Basically is a client and server bilateral type of communication. Part of the trouble with understanding socket interface is that socket can mean a number of different things, depending on context. Distinguishing between a “client” socket - an endpoint of a conversation, and a “server” socket, which is more like a switchboard operator that accepts a connection and receive incoming and outgoing traffic. The client application which more like a browser thing uses “client” sockets exclusively. This web server is talking to both “server” sockets and “client” sockets. This peak of example help us understand all these things without knowing it can be useful too.