# Improvements to UDP Programs: Avoiding Arbitrary Servers

There are a few improvements that can easily be made to our UDP program. Let's have a look.



### Problem: Replies From Arbitrary Servers #

Note that at the moment, our UDP client accepts replies from *any* machine and assumes that it's the one that it sent the initial message to, evident in the following line,

```
data, address = s.recvfrom(MAX_SIZE_BYTES)
```

Note how the client does not check **who** it is receiving the message from. It just receives a message.

#### Fix with connect() #

There are two quick ways to go about fixing this. The first of which is to use the <code>connect()</code> method to forbid other addresses from sending packets to the client.

```
import socket

MAX_SIZE_BYTES = 65535 # Mazimum size of a UDP datagram

def client(port):
    best = '127 0 0 1'
```

```
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.connect((host, port))
message = input('Input lowercase sentence:' )
data = message.encode('ascii')
s.send(data)
print('The OS assigned the address {} to me'.format(s.getsockname()))
data = s.recv(MAX_SIZE_BYTES)
text = data.decode('ascii')
print('The server replied with {!r}'.format(text))
```

With the <code>sendto()</code> method, we had to specify the IP address and port of the server every time the client wanted to send a message. However, with the <code>connect()</code> method we used, we just use <code>send()</code> and <code>recv()</code> without passing any arguments about which address to send to because the program <code>knows</code> that.

This also means that no server other than the one the client *connected* to can send it messages. The operating system discards any of those messages by default.

#### Disadvantages #

The main disadvantage of this method is that the **client can only be connected to one server at a time**. In most real life scenarios, singular applications connect to *multiple servers*!

#### Fix with Address Matching #

A better, though more tedious approach, to handle multiple servers would be to check the return address of each reply against a list of addresses that replies are expected from.

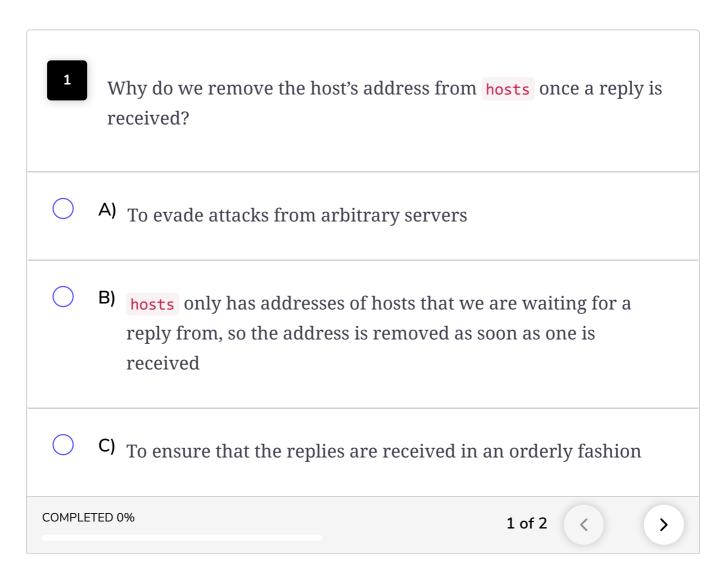
Let's implement it!

```
data, address = s.recvfrom(MAX_SIZE_BYTES)
text = data.decode('ascii')
if(address in hosts):

    print('The server {} replied with {!r}'.format(address, text))
    hosts.remove(address)
else:
    print('message {!r} from unexpected host {}!'.format(text, address))
```

As you can see, we created a list called **hosts** which contains tuples like (*IPaddresses*, *port numbers*) of any host that the client connects to. Upon receiving every message, it checks whether the message is from a host it expects to receive a reply from. As soon as a reply is received, it removes the host from the list.

## Quick Quiz! #



In the next lesson, you're going to try out an exercise for yourself: write a chat app in UDP!