

Application Layer (II)

Command Line Utility: nc

- Netcat (nc), an utility for reading from and writing to network connections using TCP or UDP.
- It can be driven by other programs or scripts.
- Good tool for network debugging and investigation.
 - E.g., Scan all ports up to 100

```
gloin:~> nc -z -v dori.cs.colby.edu 1-100
```

- REF

- <https://en.wikipedia.org/wiki/Netcat>
- <https://www.digitalocean.com/community/tutorials/how-to-use-netcat-to-establish-and-test-tcp-and-udp-connections-on-a-vps>

- Another application of using nc is to transfer files.

Client

```
mbp-190250:~ yingli$ echo "Hello gloin" > test.txt
mbp-190250:~ yingli$ cat test.txt
Hello gloin
mbp-190250:~ yingli$ nc gloin.cs.colby.edu 12345 < test.txt
mbp-190250:~ yingli$
```

Server

```
gloin:~> nc -l 12345 > output.txt
gloin:~> cat output.txt
Hello gloin
gloin:~>
```

- In the above example, I was trying to send a file, test.txt, from my laptop to gloin, and saved the file into output.txt.
- The gloin is the server in this case, waiting for the file from client, my laptop, listening on port 12345.
- My laptop is the client in this case, which connects to the server on port 12345 and sends files to it.

- E.g., SSH to dori and gloin for the example.

```
dori:~> nc gloin.cs.colby.edu 12345
Hi gloin, this is dori
Hi dori, I'm gloin, nice to meet you
```

Client

2. connecting the client to a server on gloin on port 12345

3. typing a message to the server "Hi gloin, this is dori"

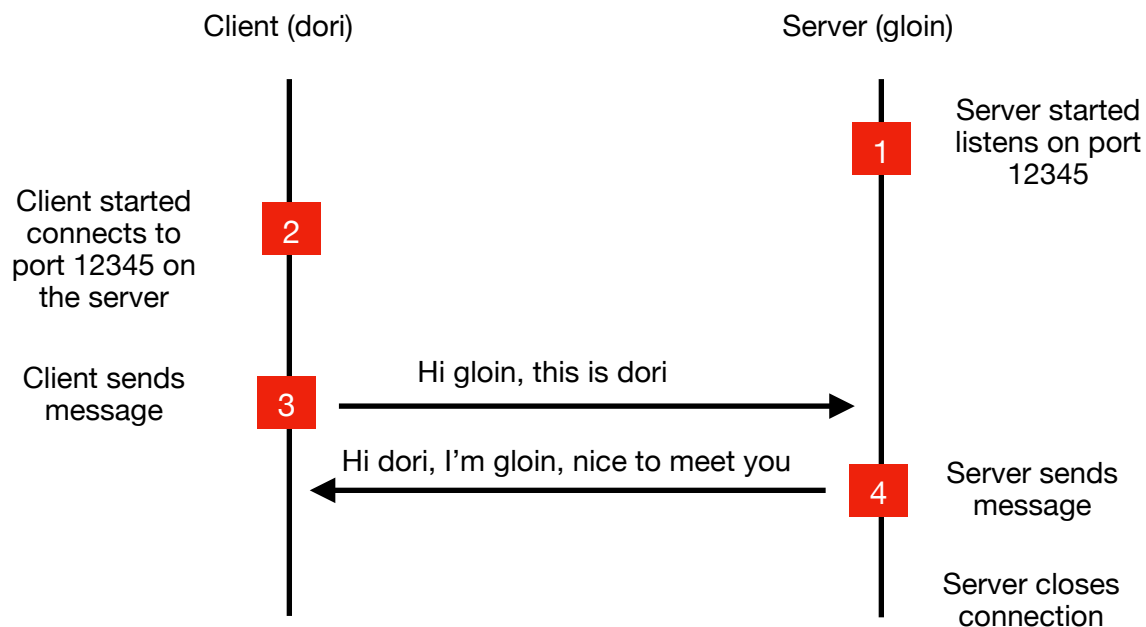
```
gloin:~> nc -l 12345
Hi gloin, this is dori
Hi dori, I'm gloin, nice to meet you
```

Server

1. starting server on port 12345

4. typing response to the client and breaking the connection by type CTRL + C

- Sequence diagram for the above example



Socket API

- We can use `nc` on the command line to build a connection between two nodes. We can also use socket API to set up a connection when programming.
- Socket API is designed to support any protocol - not just TCP/UDP/IP.
- It was originally defined in C, but adopted by essentially all programming languages.

Perspectives

- We can use parameters in Socket API to specify a connection is a reliable, stream-oriented service (TCP) [SOCK_STREAM](#), or an unreliable, datagram service (UDP) [SOCK_DGRAM](#).
- Socket API also leaves parameters to specify a connection is built on IPv4, [AF_INET](#) or IPv6, [AF_INET6](#).

Python Socket Programming (TCP)

- In this class, we use Python to build a TCP connection over IPv4 as an example. But, you can also create an UDP over IPv4 or IPv6 by changing the parameters. You are also encouraged to explore Socket API in other programming languages, which will be a good exercise.
- To build a connection, we need to program a client and server.
 - The server should be launched first and wait for the connection request from the client.

- The client connects to the server using the port the server is listening and server's IP address. In this example, the client uses 'localhost,' indicating that the server is running on the same computer, which will be translated into a local IP address automatically.
- Client side:
- To use socket API, the socket module should be imported first.

```
from socket import *
```
 - To build a connection, a socket instance should be created first, which specifies the IP protocol and TCP/UDP service supporting a connection.

```
socket(AF_INET, SOCK_STREAM)
```
 - Then, we can invoke the connect function of the socket instance to connect to the other point specified by the server name and port number.

```
connect((serverName, serverPort))
```
 - The send function transmits the data to the other point over the socket.

```
send(sentence.encode())
```
 - The recv function receives the data sent by the other point. The buffer size is specified by the parameter of this function in bytes.

```
recv(1024)
```
 - When it's done, we need to mark the socket closed.

```
close()
```
- Server side:
- After creating a socket instance, the program should use bind function to assign an IP and a port to the socket instance.

```
bind(('localhost', serverPort))
```
 - Then, the server should be listening to the incoming requests.

```
listen(0)
```
 - When a client connect request is arrived, the server accept this connection.

```
accept()
```
 - It then can receive data from the other end and send response if necessary.

```
recv(1024); send(capitalizedSentence.encode())
```
 - When it's done, we need to mark the socket closed.

```
close()
```

```

TCPClient > No Selection
1  from socket import *
2
3  serverName = 'localhost'
4  serverPort = 12000
5  modifiedSentence = None
6
7  clientSocket = socket(AF_INET, SOCK_STREAM)
8  clientSocket.connect((serverName, serverPort))
9  sentence = input('Input lowercase sentence:')
10 clientSocket.send(sentence.encode())
11 modifiedSentence = clientSocket.recv(1024)
12 print('From Server: ', modifiedSentence.decode())
13 clientSocket.close()

```

```

mbp-190250:SimpleVersion yingli$ python TCPClient.py
Input lowercase sentence:hello world
('From Server: ', u'HELLO WORLD')
mbp-190250:SimpleVersion yingli$

```

```

TCPClient > No Selection
1 from socket import *
2
3 serverPort = 12000
4 serverSocket = socket(AF_INET, SOCK_STREAM)
5 serverSocket.bind(('localhost', serverPort))
6 serverSocket.listen(0)
7 capitalizedSentence = None
8
9 print("The server is ready to receive")
10 while True:
11     connectionSocket, addr = serverSocket.accept()
12     sentence = connectionSocket.recv(1024).decode()
13     capitalizedSentence = sentence.upper()
14     connectionSocket.send(capitalizedSentence.encode())
15     connectionSocket.close()

```

```

SimpleVersion — python TCPServer.py
mbp-190250:SimpleVersion yingli$ python TCPServer.py
The server is ready to receive

```

How to modify the code so that the connection between the client and server can end till the user input “quit”? Otherwise, the client can keep sending strings to the server and receive capitalized strings (while loop before line 7 in client side).

```

TCPClient > No Selection
1 from socket import *
2
3 serverName = 'localhost'
4 serverPort = 12000
5 modifiedSentence = None
6
7 while modifiedSentence != 'QUIT':
8     clientSocket = socket(AF_INET, SOCK_STREAM)
9     clientSocket.connect((serverName, serverPort))
10    sentence = input('Input lowercase sentence:')
11    clientSocket.send(sentence.encode())
12    modifiedSentence = clientSocket.recv(54321).decode()
13    print('From Server:', modifiedSentence)
14    clientSocket.close()

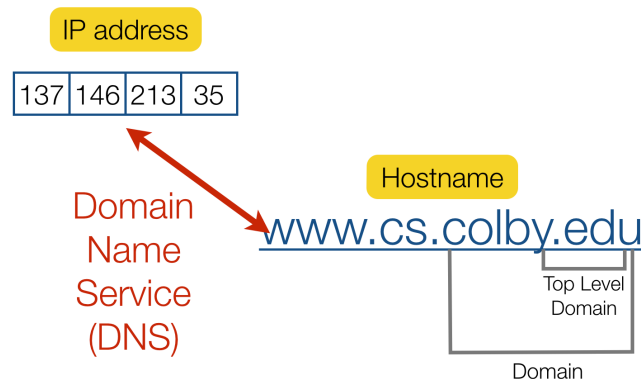
```

```

TCPClient > No Selection
1 from socket import *
2
3 serverPort = 12000
4 serverSocket = socket(AF_INET, SOCK_STREAM)
5 serverSocket.bind('', serverPort)
6 serverSocket.listen(0)
7 capitalizedSentence = None
8
9 print("The server is ready to receive")
10 while capitalizedSentence != "QUIT":
11     connectionSocket, addr = serverSocket.accept()
12     sentence = connectionSocket.recv(54321).decode()
13     capitalizedSentence = sentence.upper()
14     connectionSocket.send(capitalizedSentence.encode())
15     connectionSocket.close()

```

Domain Name Service



- The information contained in “www.cs.colby.edu” is three-fold:
 - Hostname (cs.colby.edu)
 - Domain (colby.edu)

- Top-level domain (.edu)
 - Other top-level domains
 - By characteristics of the organization (.com, .co, .org, .net, and .gov)
 - .com: commercial business
 - .co: new domain extension launched on 07/20/2010; (google.co.uk, google.co.jp)
 - .net: networking services and Internet service provider (Comcast residential email address)
 - .org: non-profit organizations (ieee.org)
 - By geography (.uk, .cn, google.com.hk)
- Domain name service provides the **mapping** between **hostnames and IP** addresses:
 - Is the mapping one-to-one, one-to-many, or many-to-one? [All are possible.]
 - one-to-one: cs.colby.edu
 - one-to-many: whois www.google.com [round robin]
 - many-to-one: Shared Web Hosting
 - A web hosting service where **many websites reside on one web server** connected to the Internet.
 - **System administration is necessary**. Benefit for users who do not want to deal with it, but hindrance to power users who want more control.
 - It is **cheaper** than dedicated server hosting.
 - Name-based virtual hosting: shared IP hosting, the virtual hosts serve multiple hostnames on a single machine with a single IP address.
 - **HTTP/1.1 makes it possible**. A web browser requests a resource from a web server using HTTP/1.1 it includes the requested hostname as part of the request.
 - REF: https://en.wikipedia.org/wiki/Shared_web_hosting_service
 - No matter which type of mapping it is, the **mapping is in both directions**.
- If you are asked to implement a system to provide domain name service, there are two possible solutions, centralized database and fully distributed database. Which solution you will select?
 - If choosing the centralized one, the system has to face the possibility of a single point of failure, large traffic volume, distant centralized database, and maintenance challenges (as it's a huge database).
 - So, the fully distributed solution will be a better solution.
- The system to provide domain name service is actually a distributed redundant, hierarchical database.
 - Root Name Servers
 - 13 logical root name servers are operated by 12 independent organizations
 - logical names in the form letter.root-servers.net, where letter ranges from a to m
 - each operator uses redundant computer equipment to provide reliable service

- all (except B-root) operate in multiple geographical locations using anycast addressing
- Top-level domain (TLD) servers
 - There is a TLD server cluster for each of the top-level domain (com, org, net, edu, and gov, and uk, fr, ca and jp)
 - Verisign Global Registry Services maintains the TLD servers for .com
 - Educause maintains the TLD servers for .edu
- Authoritative DNS server
 - Every organization with publicly accessible hosts (e.g. web servers and mail servers) on the Internet must provide publicly accessible DNS records.
 - Implement own server or pay to have these records stored in an authoritative DNS server
 - Most universities and companies implement and maintain their own primary and secondary (backup) authoritative DNS servers.
- Local DNS server
 - not strictly belong to the hierarchy
 - default name server, close to the host, may be in the same LAN
 - act as a proxy
 - can be used for DNS caching
- Query delegation
 - iterative (non-recursive): local domain name resolution
 - recursive: domain name resolution

Disclaimer: Notes adapted from the textbook and online resources.