CS305 Lab 4
Advanced HTTP & Web Programming

Dept. Computer Science and Engineering
Southern University of Science and Technology

SUSTech
Southern University of Science and Technology

thanks to HHQ.Zhang

Part A.

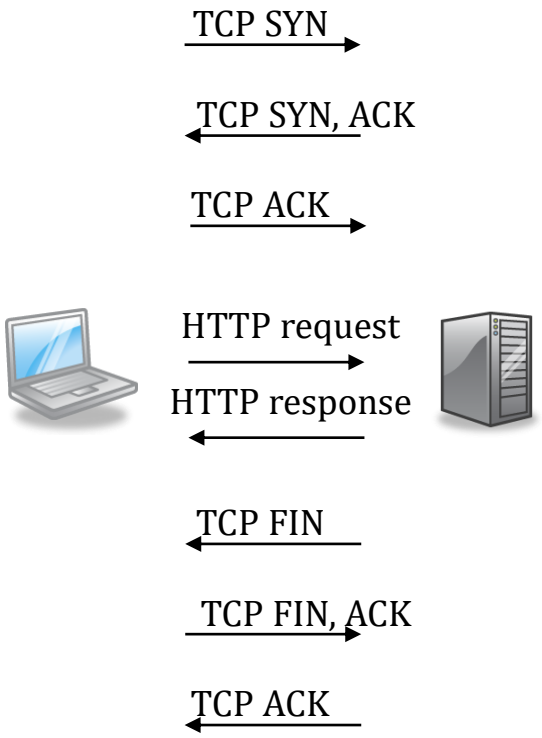
Advanced HTTP

Part A.1
Connection and transfer encoding

- Connection management
  - Persistent connection, parallel connection
  - Connection: close
- Content-Length vs. Chunked transfer encoding
  - Reducing latency of response

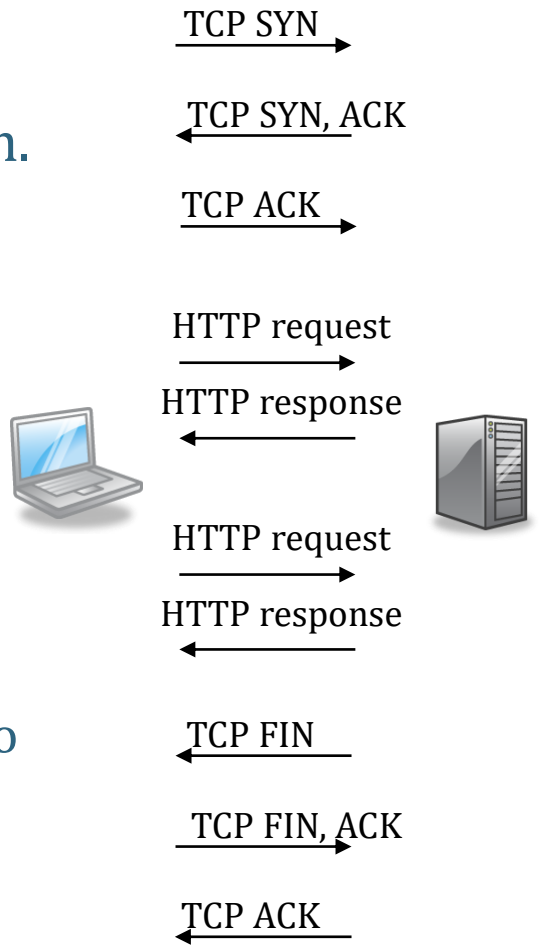
Problem in HTTP/1.0 connection

- HTTP/1.0 uses a new connection for each HTTP transaction.
- Making TCP connection slow.
  - takes three packets to establish
  - takes three packets to close
- A web page typically contains many embedded images. HTTP/1.0 would make many TCP connections to load a web page.
  - Slow page loading



Persistent and Parallel Connection

- Persistent connection: multiple requests and responses are sent through one TCP connection.
  - Default in HTTP/1.1
  - Browsers keep TCP connection after page load. Why?
- Parallel connection: A web browser opens several TCP connections to a web site and downloads components of a web page concurrently.
  - using tcp.srcport and tcp.dstport in Wireshark to trace a http session

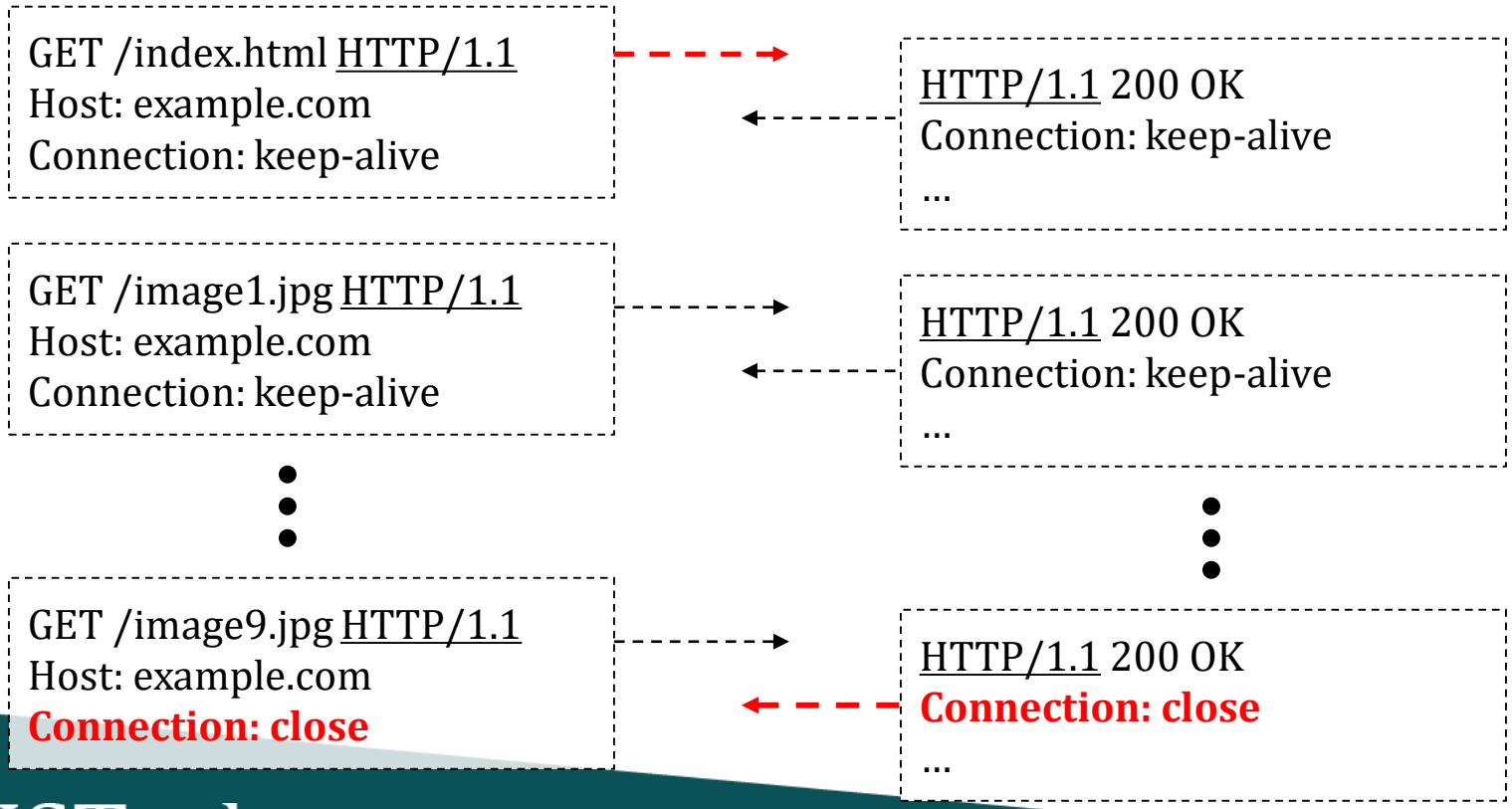


Connection:

- The Connection: header indicates whether to keep or close the current connection.
- Connection: keep-alive. Default, may be omitted.
- Connection: close
  - In a request, the client asks the server to close the connection after sending the response
  - In a response, the server indicates that it will close the connection after sending this response

Persistent connection

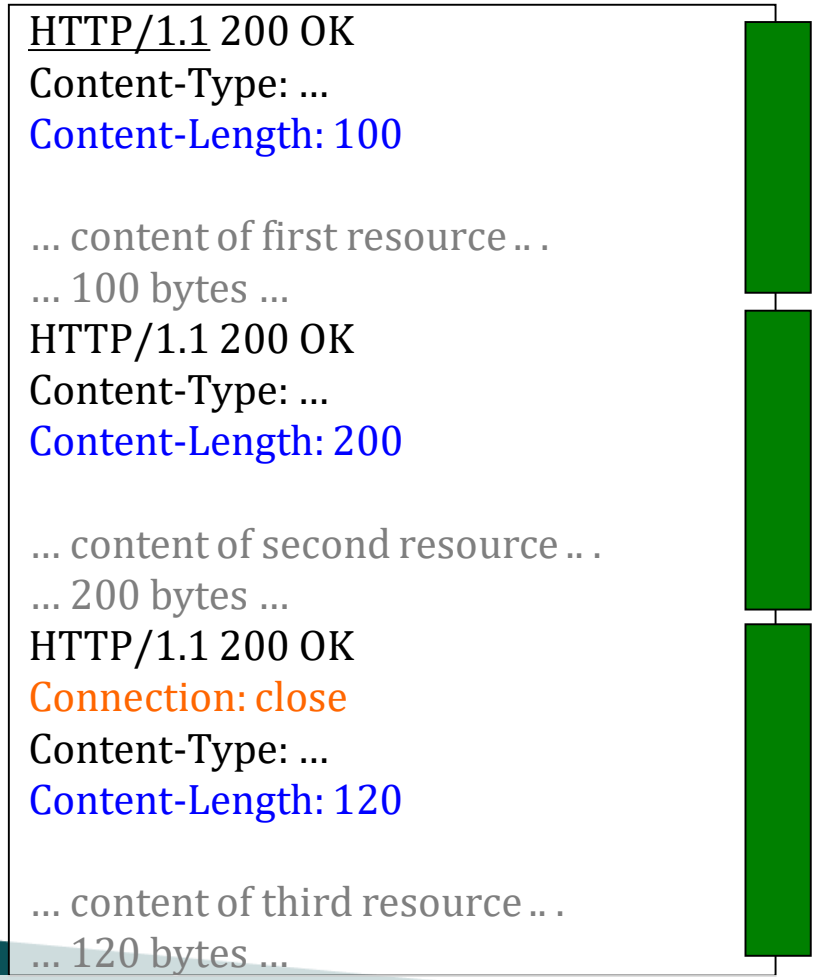
The client creates a connection before sending the first request. Subsequent requests and responses are transferred in this TCP connection.



The server closes the connection after the last response.

Multiple messages in a connection

- A browser receives multiple responses in a TCP connection.
- To break the byte stream into messages, it must know the end of each message.
- One solution is that the server declares Content-Length for each response.



Three responses transferred in a TCP stream

Problems of Content-Length

- Content-Length header is sent before the message body.
- If a resource is generated dynamically by a server-side script (e.g. ASPX, PHP), the web server can determine Content-Length only after the script finishes execution.
  - The server first has to buffer the whole response before it can start sending the response.
- Efficiency problems:
  - Larger memory overhead
  - Slower response time

Chunked Transfer-Encoding

分块传输编码

- Chunked transfer-encoding enables a web server to start transmitting the beginning parts of a response while it is still generating the rest.
  - Does not send Content-Length. Sends Transfer-Encoding: chunked instead.
  - A long response body is divided into several pieces called chunks.
  - Before sending each chunk, the server sends its length in hexadecimal.
  - After sending the last chunk, the server sends a 0.

Response with Content-Length

```
HTTP/1.1 200 OK
Date: Wed, 19 Mar 2008 01:46:57 GMT
Content-Type: text/plain
Content-Length: 42

abcdefghijklmnopqrstuvwxyz1234567890abcdef
```

Response with chunked body

```
HTTP/1.1 200 OK
Date: Wed, 19 Mar 2008 01:46:57 GMT
Content-Type: text/plain
Transfer-Encoding: chunked

1a                                  长度+内容
abcdefghijklmnopqrstuvwxyz
10
1234567890abcdef
0
```

Length of a chunk in Hex

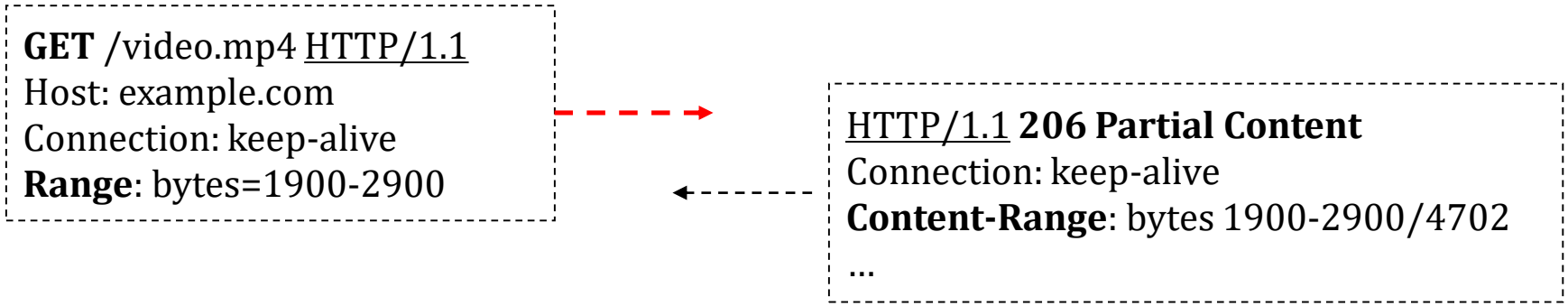
Length of a chunk in Hex

Length of 0 means no more chunks

Partial Content

- How to retrieve a slice of resource?
  - Request: 请求部分内容：用range
    - Range: <unit>=<range-start>-<range-end>
      e.g. Range: bytes=200-1000, 2000-6576, 19000-
  - Response:
    - HTTP/1.1 206 Partial Content
    - Content-Range: <unit> <range-start>-<range-end>/<size>
      e.g. Content-Range: bytes 21010-47021/47022
- References:
  - https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Range
  - https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Range
  - https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/206

Example:


GET /video.mp4 HTTP/1.1
Host: example.com
Connection: keep-alive
Range: bytes=1900-2900
HTTP/1.1 **206 Partial Content**
Connection: keep-alive
Content-Range: bytes 1900-2900/4702
…

Part A.2
State, session and security

- Session management
  - HTTP is stateless 无状态协议
  - Cookies
  - As in common web framework (e.g. asp.net, php)
  - Session hijacking
- Encryption and SSL
  - Secured login vs. full-session HTTPS
  - Partially secure web page

HTTP is stateless

- Statelessness means that every HTTP request happens in complete isolation.
- When the client makes a HTTP request, it includes all information necessary for the server to fulfill that request.
- The server never relies on information from previous requests. If that information was important, the client would have sent it again in this request.
  - A web server does not retain info between processing of requests from a user session
  - The client and server do not need to maintain a common state

(from O'Reilly RESTful web service)

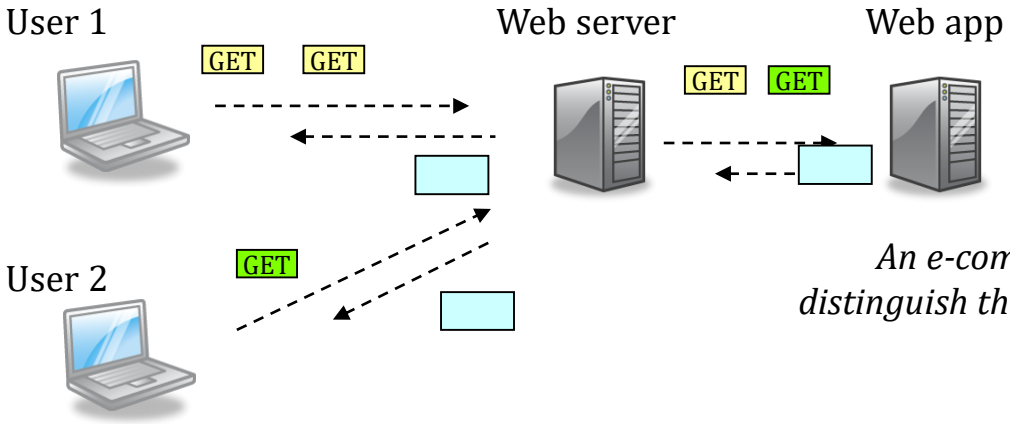
Example

- A browser keeps sending same (or similar) headers to a web server in a series of requests, e.g.
  - Host
  - User-agent
  - Content negotiation
- Each request contain the information from the full URL (absolute URL).

```
GET /wiki/Internet HTTP/1.1
Host: en.wikipedia.org
User-Agent: Mozilla/5.0 ... Firefox/3.5.3
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
```

Session Management in Web App

- Although HTTP is stateless, web app needs to maintain states in processing requests from a user session.
  - e.g. Has a user logged in?
    Which requests come from the user?
- The client needs to attach session identifier in each request



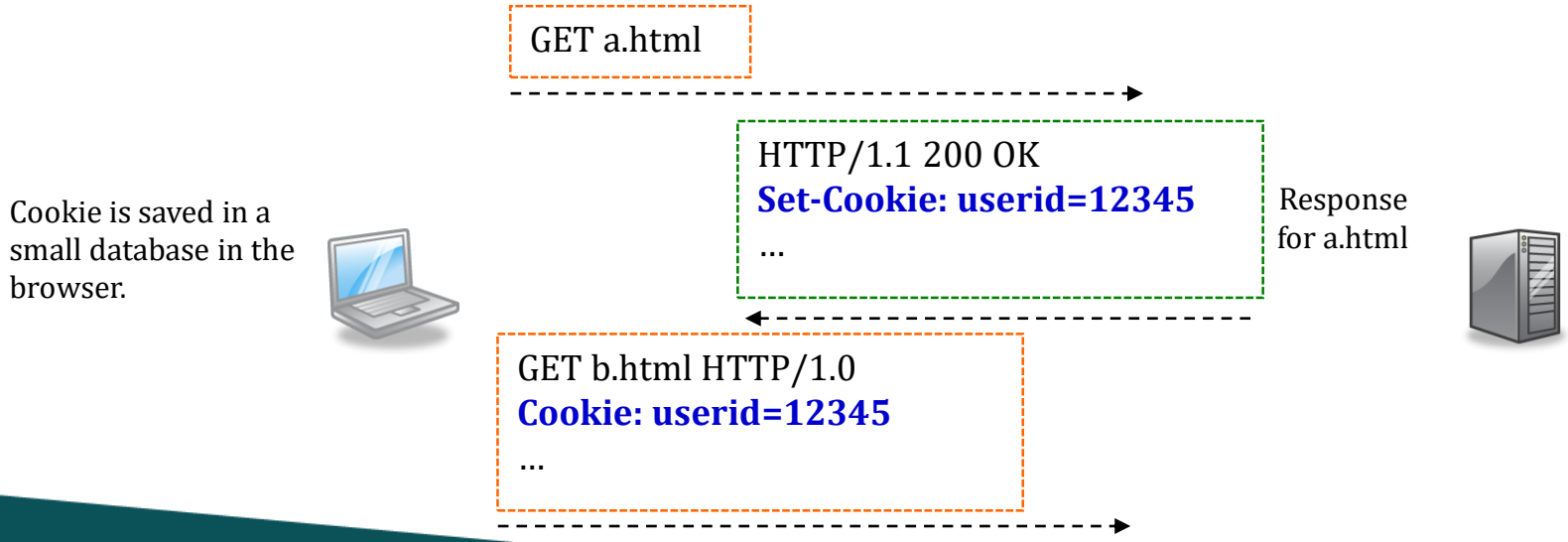
An e-commerce web app must distinguish the user sessions of requests.

Session Management

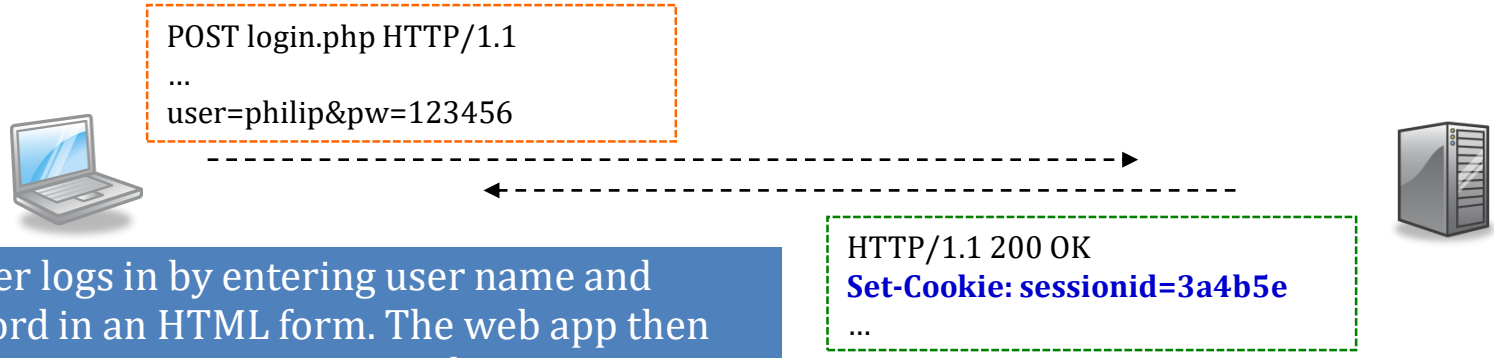
- A web app has to track a user's progress from one request to another.
- Each request has to include some data to identify a user session.
- Common approaches:
  - Cookies
  - Hidden form field (<input type="hidden">)
  - Query string

HTTP Cookies

- Cookies are small pieces of data a web server asks a client to keep and send back in future requests.
  - Servers add header Set-Cookie: name=value in response
  - Clients add header Cookie: name=value in future requests



Example: Typical use of Session id



The user logs in by entering user name and password in an HTML form. The web app then creates a session in memory and returns a session ID.

The browser attaches the cookie in all future requests to the web app. The web app can use the session id to look up application state (e.g. current user, 'session variables')

Cookie attributes

- A web server can restrict the scope of a cookie with attributes:
  - expires : date/time after which this cookie can be deleted
    - If not set, the cookie is deleted when user quits the browser
  - path, domain : the client should only include this cookie for requests in this domain and URL under this path
    - If not set, the default is the domain and path of the response
  - secure: a secure cookie may only be sent through SSL

```
HTTP/1.1 200 OK
Set-Cookie: userid=12345; expires=Fri, 31-Dec-2010
      23:59:59 GMT; path=/; domain=.example.com
...
```

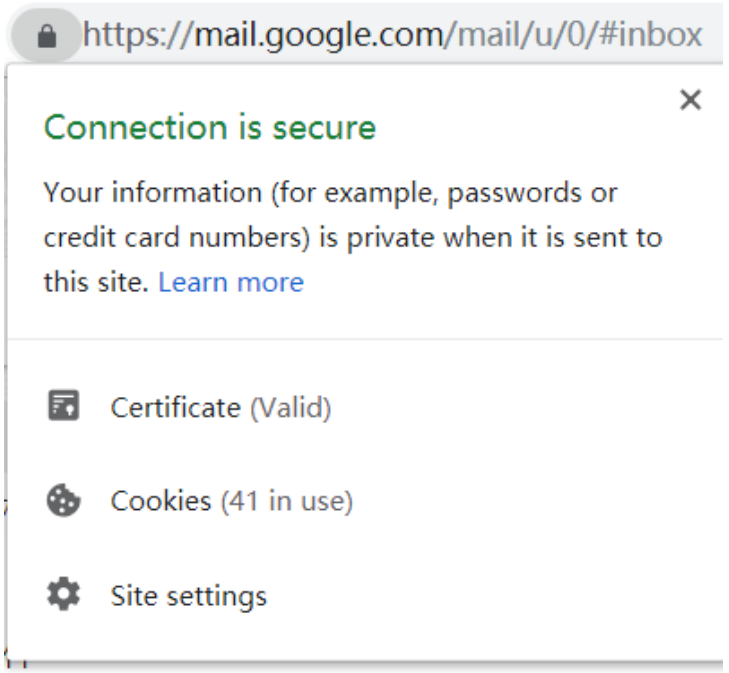
Encryption for HTTP

- **TLS (Transport Layer Security, aka. SSL)** are cryptographic protocols that encrypt data transmitted over a TCP connection.
  - Common versions: TLS1.2, TLS1.3
  - Can run different protocols over TLS, e.g. HTTP, SMTP, IMAP
- Two purposes:
  - Prevent eavesdropping and tampering
    - e.g. Only the client and server of an HTTP transaction can read the request/response
  - Verify the authenticity of the server
    - The server has a valid digital certificate issued by a certificate authority known by the browser

HTTPS

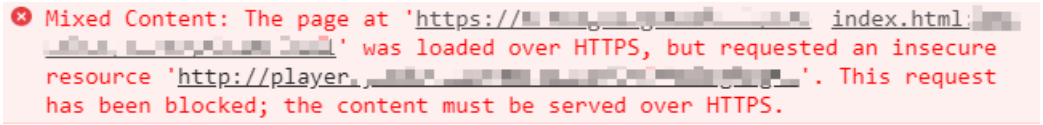
https://mail.google.com/mail/#inbox

- Need to install/trust a digital certificate in the web server
- https – runs HTTP over a secured TCP connection
  - Use port 443
  - Usually TLSv1.2TLSv1.3
- A secure HTTP transaction
  - Attackers cannot read the request and response
  - Proxy (including cache servers) cannot read the messages either



Partially secure web page

- A secure web page (https) that refers to unsecure resources (http)
  - e.g. the HTML page is using https, but the images inside are using http only
  - Unsecure resources may be modified and then added to the supposedly secure HTML page
    - Very serious if these are JavaScript files
  - HTTP requests to unsecure resources may contain cookies and eavesdropped by attackers
    - Problem solved by 'secure' attribute of cookies

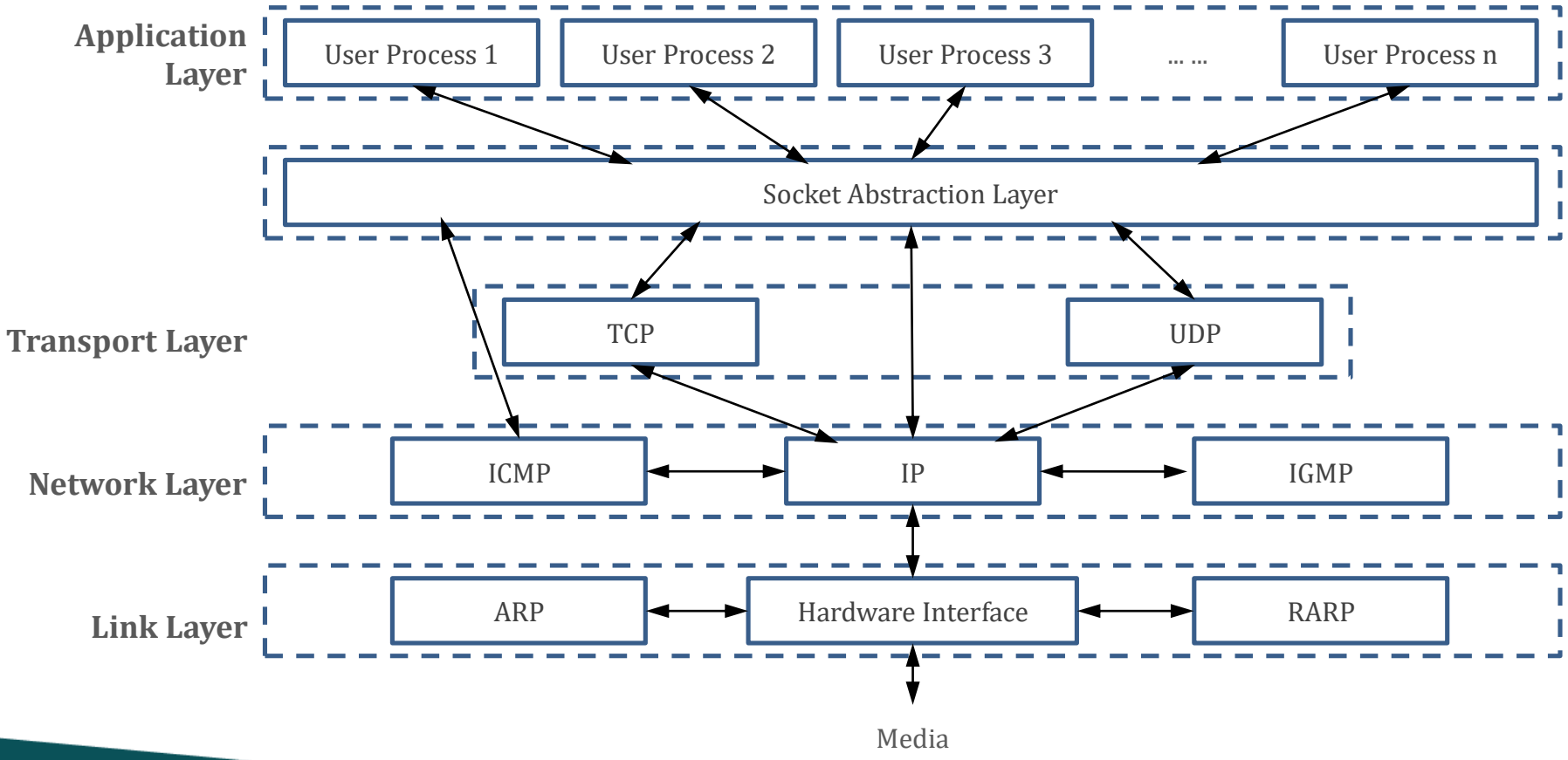


SUSTech
Southern University of Science and Technology

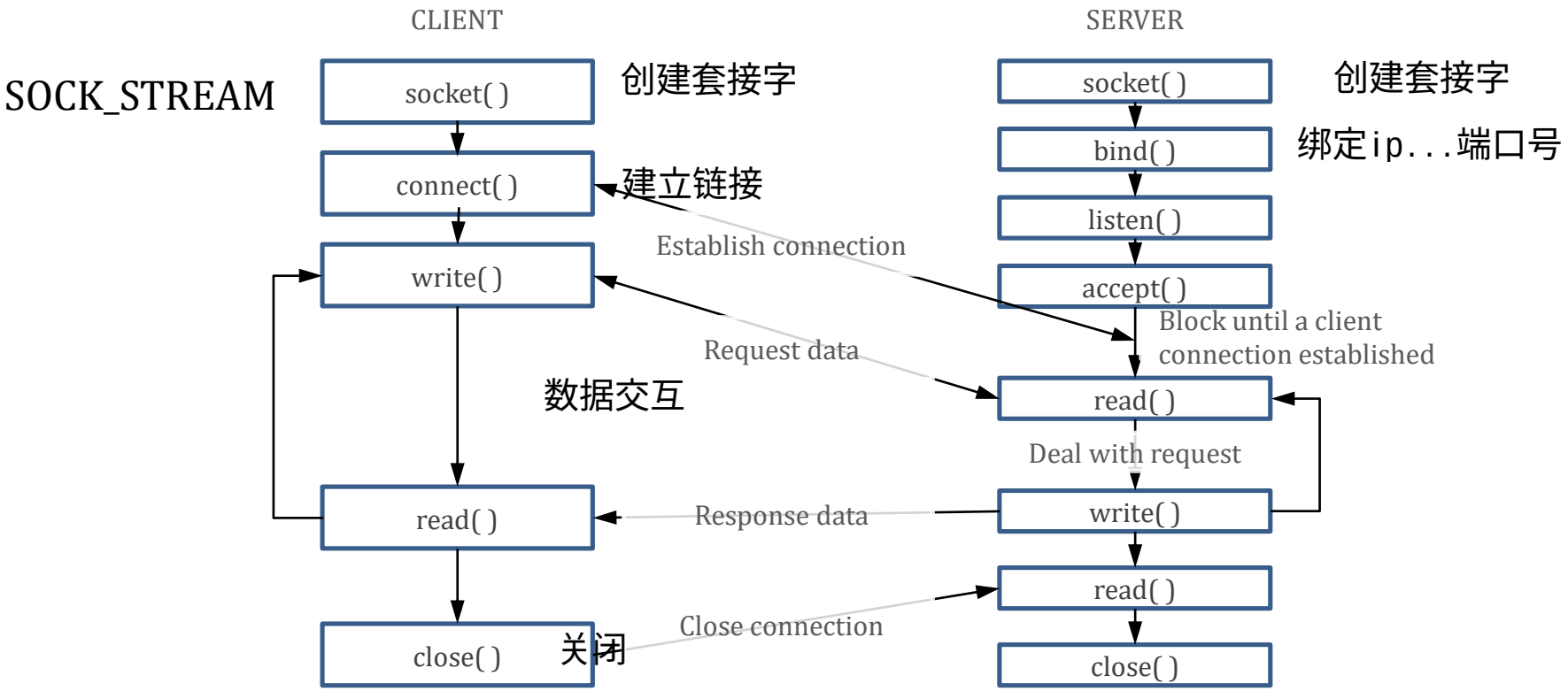
Part B.

Socket Programming

Socket (1)


Application Layer
User Process 1
User Process 2
User Process 3
... ...
User Process n
Socket Abstraction Layer
Transport Layer
TCP
UDP
Network Layer
ICMP
IP
IGMP
Link Layer
ARP
Hardware Interface
RARP
Media

Socket (2)


CLIENT
SERVER
SOCK_STREAM
socket( )
创建套接字
socket( )
创建套接字
bind( )
绑定ip...端口号
connect( )
建立链接
listen( )
Establish connection
write( )
accept( )
Block until a client
connection established
Request data
数据交互
read( )
Deal with request
read( )
Response data
write( )
read( )
Close connection
关闭
close( )
close( )

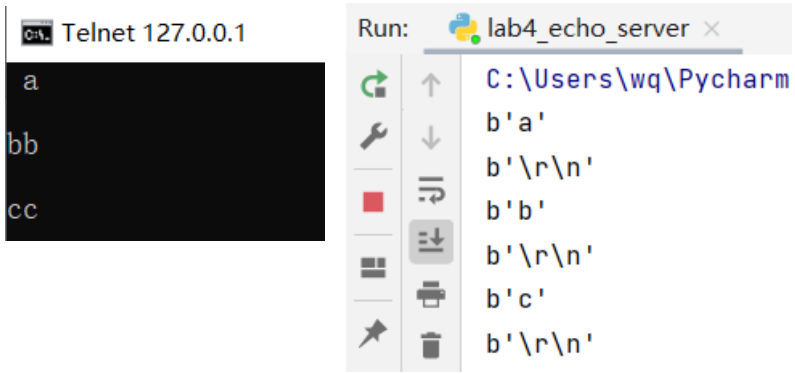
Socket Example 1: Echo Server (1)

```
import socket

def echo():
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.bind(('127.0.0.1', 5555))
    sock.listen(10)
    sock.settimeout(0.5)

    while True:
    try:
            conn, address = sock.accept()
            while True:
                data = conn.recv(2048)
                if data and data != b'exit':
                    conn.send(data)
                    print(data)
                else:
                    conn.close()
                    break
        except socket.timeout:
            continue

if __name__ == "__main__":
    try:
        echo()
    except KeyboardInterrupt:
        pass
```

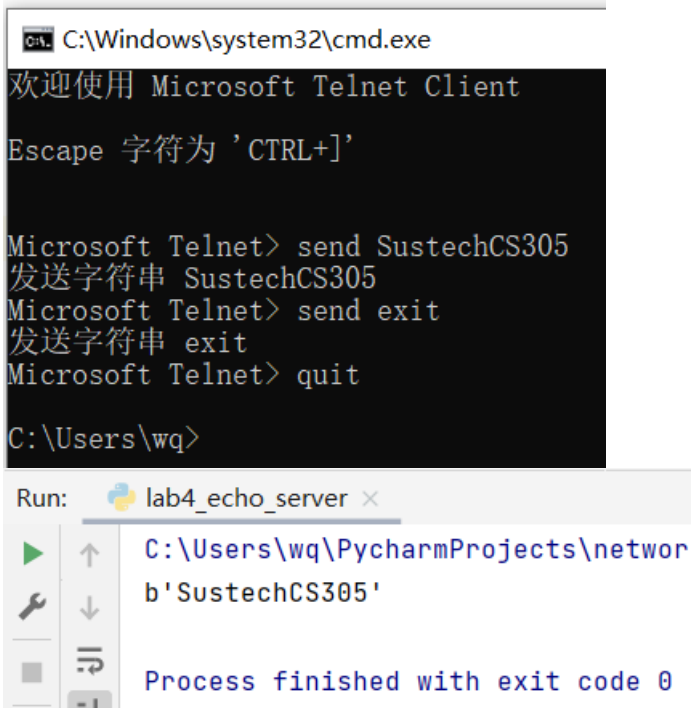
Socket Example 1: Echo Server (2)

Running under Windows

```
C:\Users\wq>telnet 127.0.0.1 5555
```



If we want to send a string instead of characters, shortcut keys “ctrl+]” can be used.

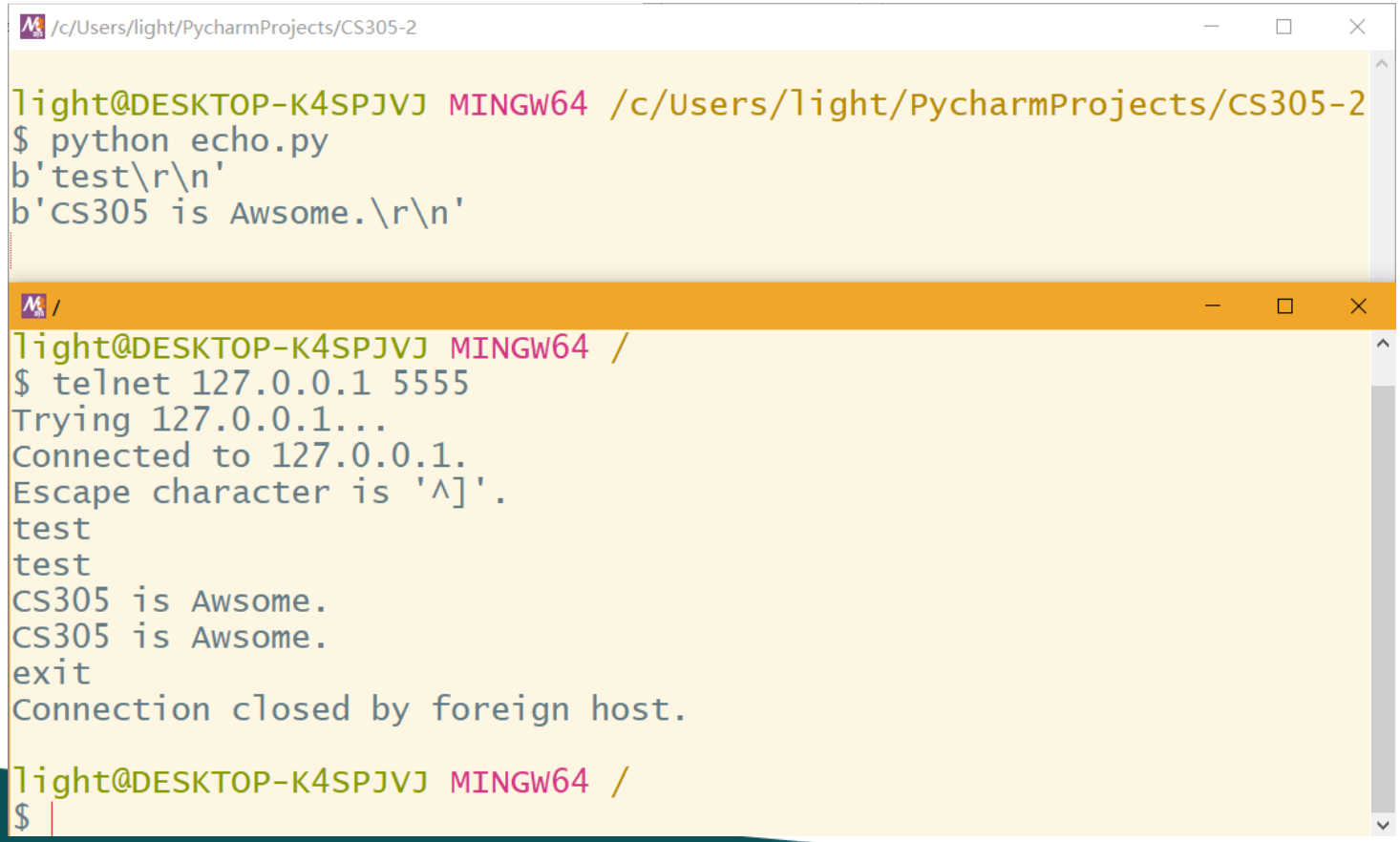


Tips: using command “quit” can exit sending mode.

SUSTech
Southern University of Science and Technology

Socket Example 1: Echo Server (3)

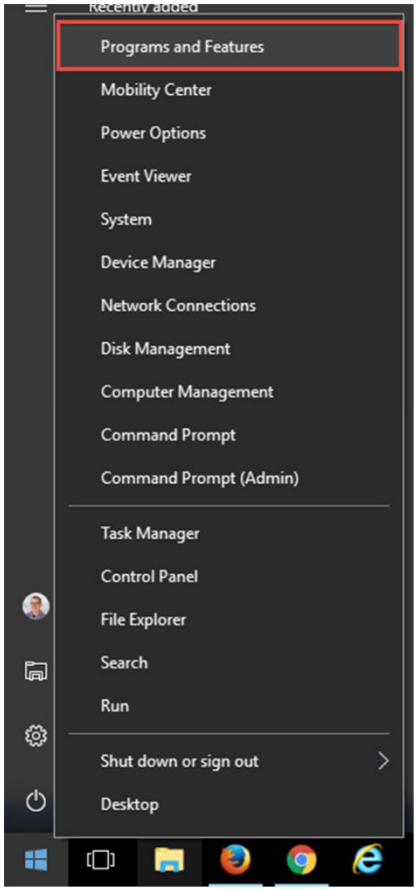
Running under other OS

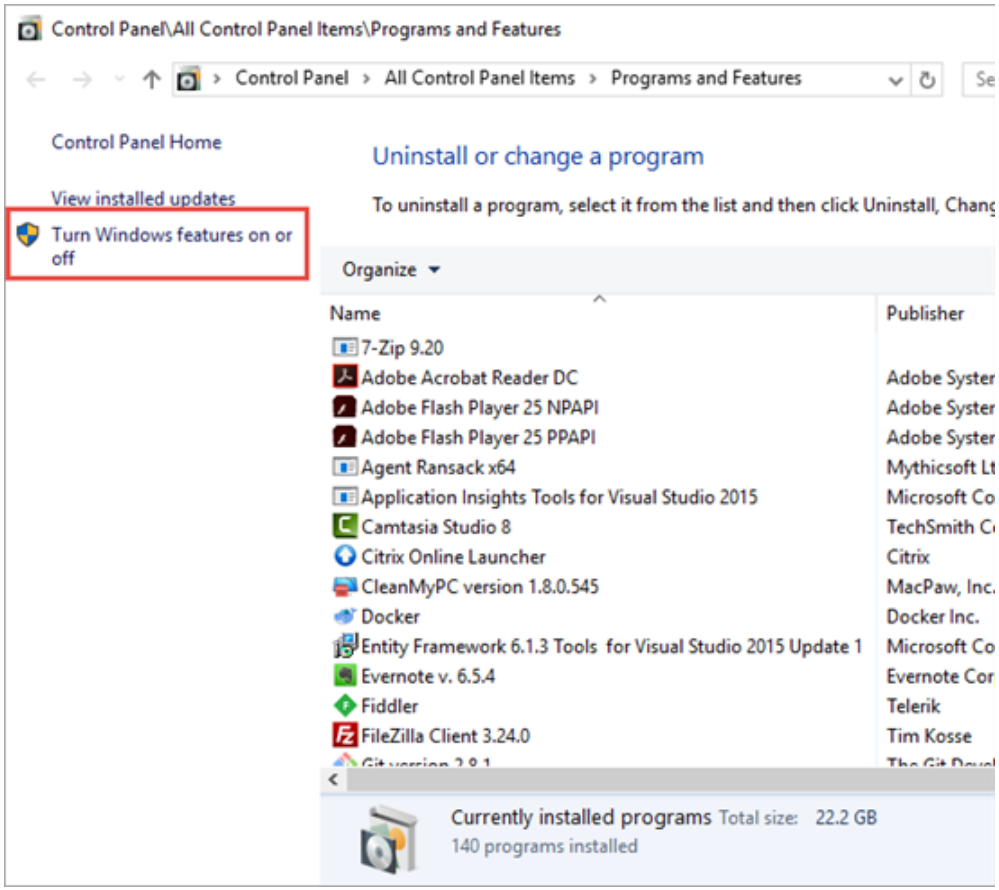


Socket Example 1: Echo Server (4)

- 1. Run the server echo on Linux(or macOS) and Windows separately, is there any difference?
- 2. How to exit the loop? Can you design a method to quit elegantly?
- 3. Is there anyway to improve the server to make it work the same on different OS?

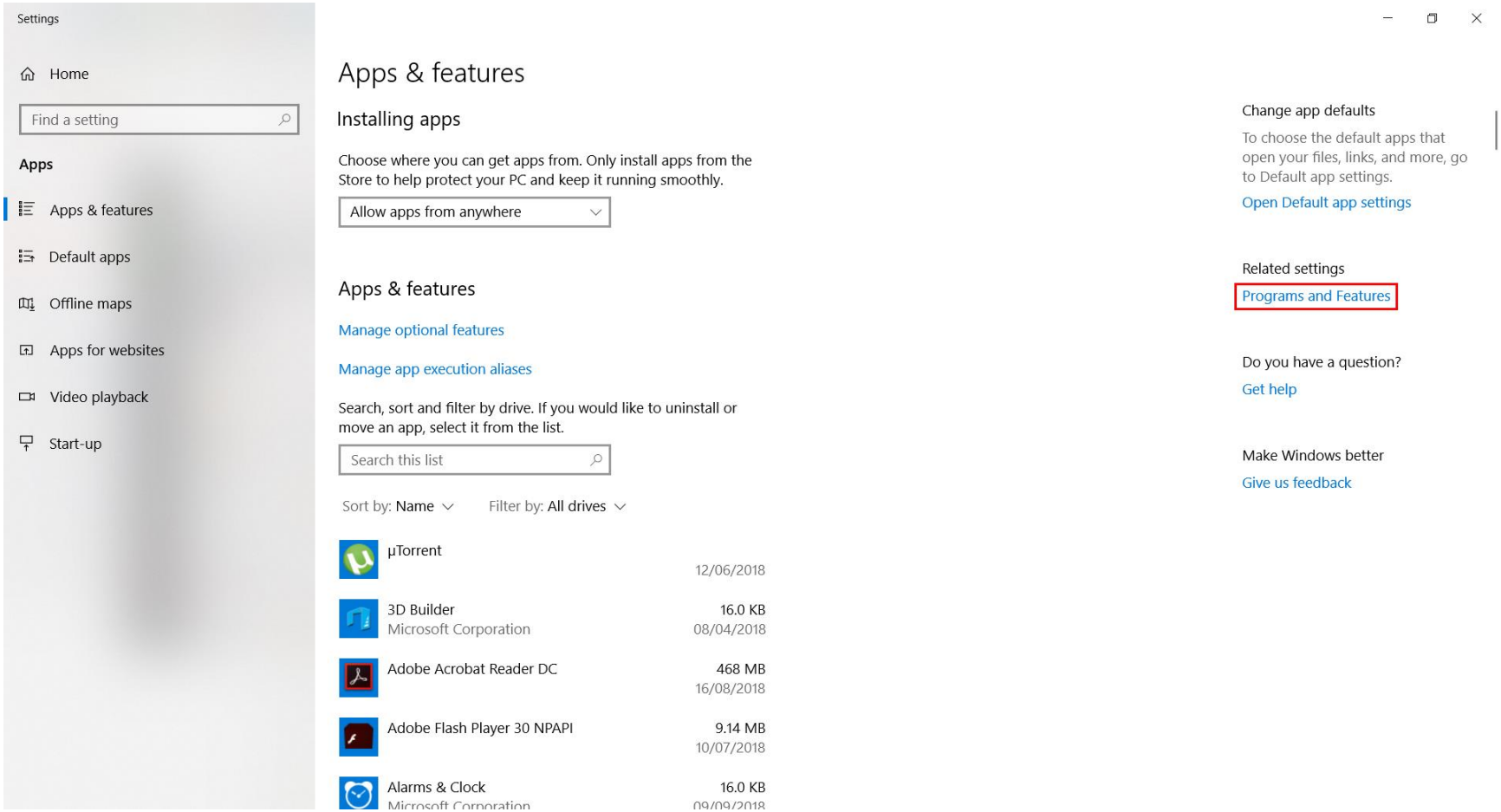
Tip 1: Enable Telnet on Windows (1)





Reference: https://social.technet.microsoft.com/wiki/contents/articles/38433.windows-10-enabling-telnet-client.aspx

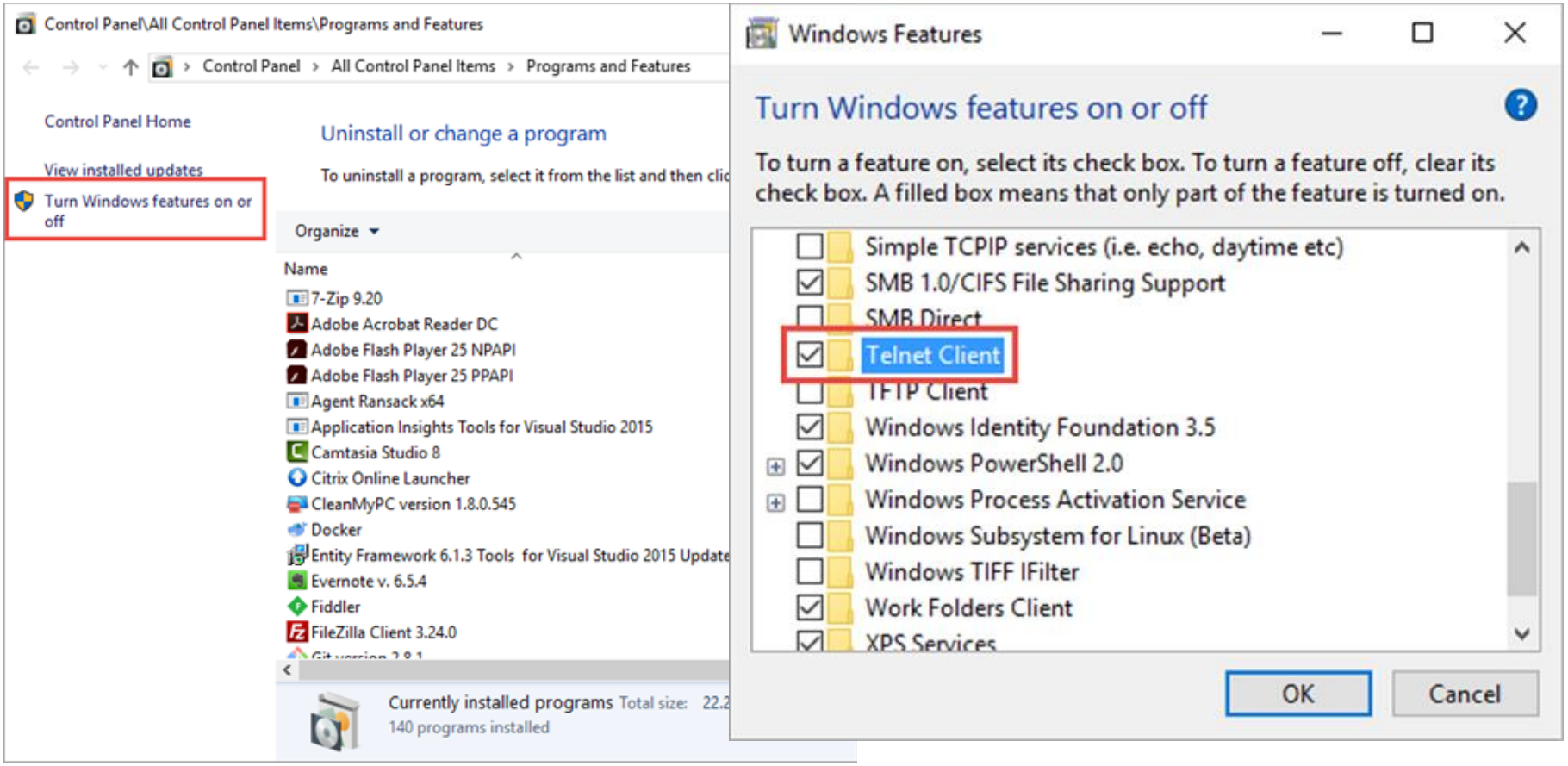
Tip 1: Enable Telnet on Windows (2)



Reference: https://social.technet.microsoft.com/wiki/contents/articles/38433.windows-10-enabling-telnet-client.aspx

SUSTech
Southern University of Science and Technology

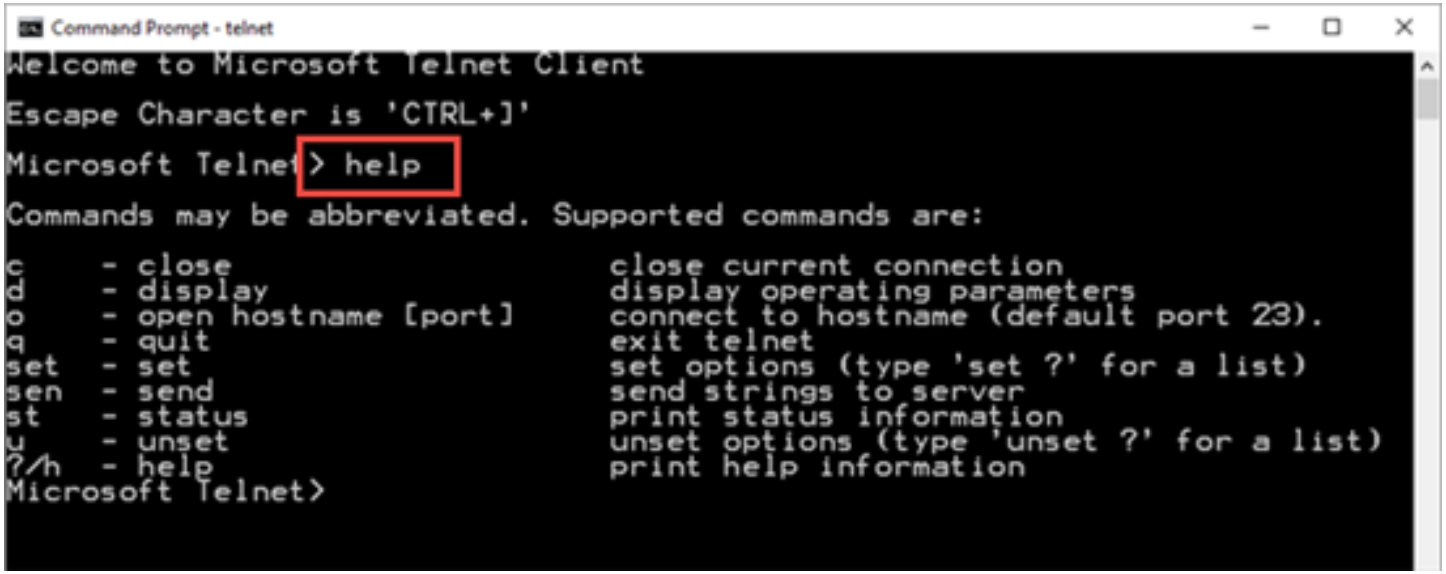
Tip 1: Enable Telnet on Windows (3)



Reference: https://social.technet.microsoft.com/wiki/contents/articles/38433.windows-10-enabling-telnet-client.aspx

Tip 1: Enable Telnet on Windows (4)

- Verify
  - Win+R, run “cmd”
  - Type “telnet”, press enter



Reference:
https://social.technet.microsoft.com/wiki/contents/articles/38433.windows-10-enabling-telnet-client.aspx

Example 2: Mimic a Simple Web Server (1)

```
def web():
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.bind(('127.0.0.1', 8080))
    sock.listen(10)
    while True:
        conn, address = sock.accept()
        data = conn.recv(2048).decode().split('\r\n')
        print(data[0].split(' '))
        res = err404
        if data[0].split(' ')[1] == '/':
            res = hello
        for line in res :
            conn.send(line)
        conn.close()

if __name__ == "__main__":
    try:
        web()
    except KeyboardInterrupt:
        pass
```

```
import socket

hello = [b'HTTP/1.0 200 OK\r\n',
    b'Connection: close'
    b'Content-Type:text/html; charset=utf-8\r\n',
    b'\r\n',
    b'<html><body>Hello World!<body></html>\r\n',
    b'\r\n']

err404 = [b'HTTP/1.0 404 Not Found\r\n',
    b'Connection: close'
    b'Content-Type:text/html; charset=utf-8\r\n',
    b'\r\n',
    b'<html><body>404 Not Found<body></html>\r\n',
    b'\r\n']
```

Example 2: Mimic a Simple Web Server (2)

```
/c/Users/light/PycharmProjects/CS305-2

light@DESKTOP-K4SPJVJ MINGW64 /c/Users/light/PycharmProjects/CS305-2
$ python web_hello.py
['GET', '/', 'HTTP/1.1']
['GET', '/not-exist', 'HTTP/1.1']
```

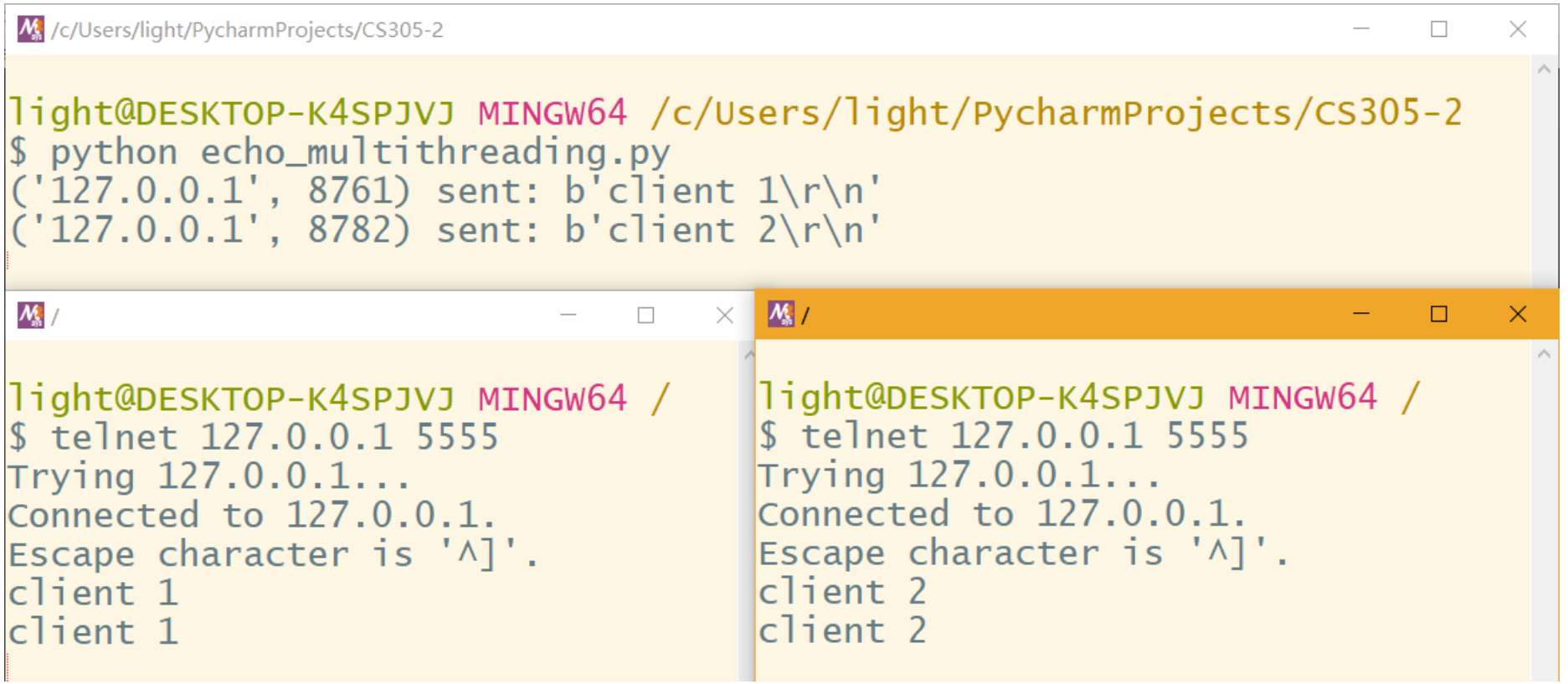
```
/

light@DESKTOP-K4SPJVJ MINGW64 /
$ curl 127.0.0.1:8080
<html><body>Hello World!<body></html>


light@DESKTOP-K4SPJVJ MINGW64 /
$ curl 127.0.0.1:8080/not-exist
<html><body>404 Not Found<body></html>
```

Example 3: Echo Server Multithreading (1)

```
import socket, threading

class Echo(threading.Thread):
    def __init__(self, conn, address):
        threading.Thread.__init__(self)
        self.conn = conn
        self.address = address

    def run(self):
        while True:
            data = self.conn.recv(2048)
            if data and data != b'exit':
                self.conn.send(data)
                print('{} sent: {}'.format(self.address, data))
            else:
                self.conn.close()
                return
```

```
def echo():
    sock = socket.socket(socket.AF_INET,
                         socket.SOCK_STREAM)
    sock.bind(('127.0.0.1', 5555))
    sock.listen(10)
    while True:
        conn, address = sock.accept()
        Echo(conn, address).start()


if __name__ == "__main__":
    try:
        echo()
    except KeyboardInterrupt:
        pass
```

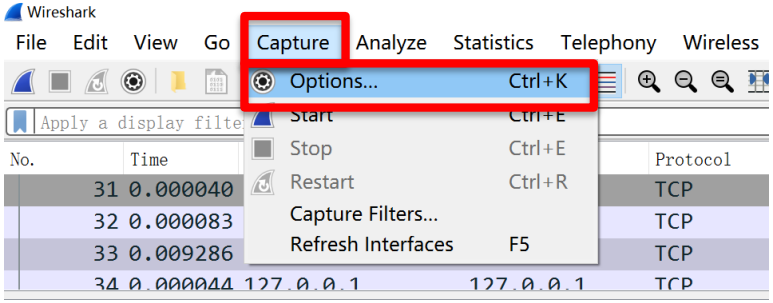
Example 3: Echo Server Multithreading (2)

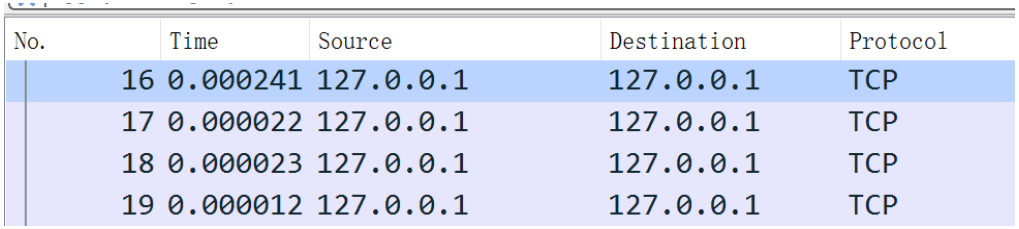


Practice 4.1 (Optional)

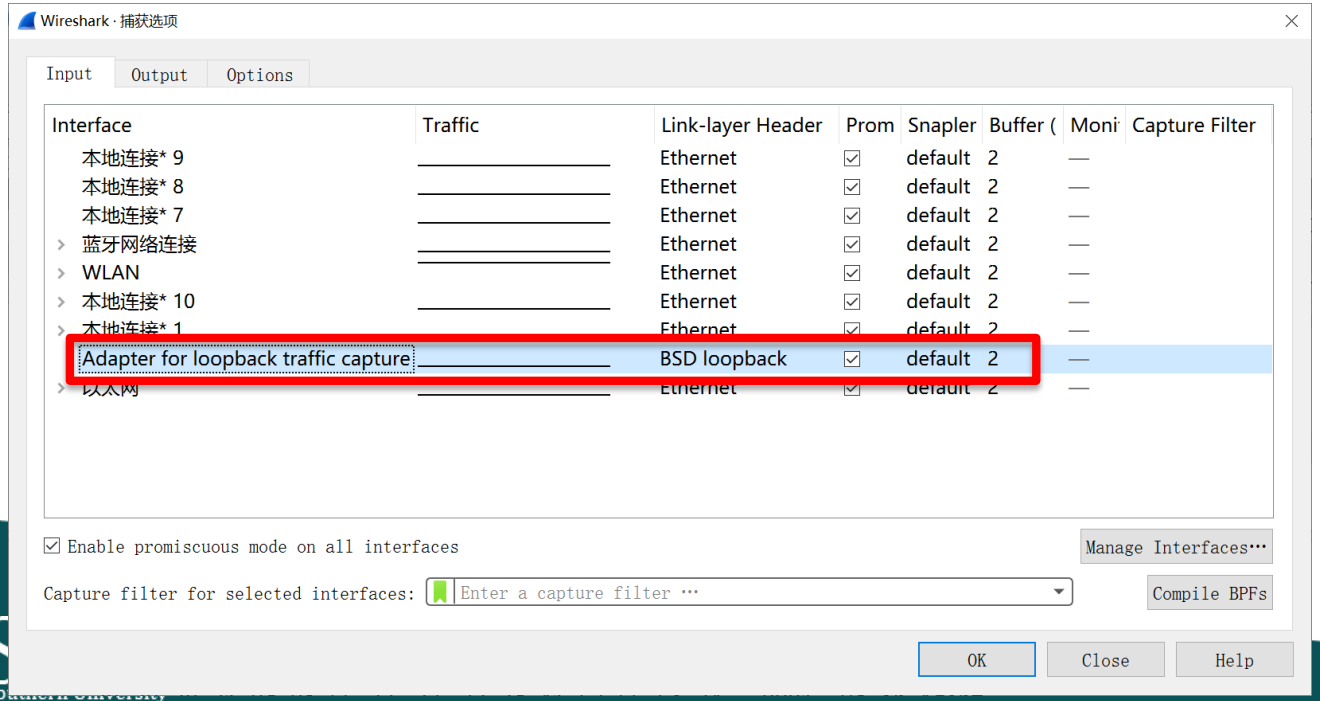
- Run all the demos on your PC.
- Use Wireshark to capture and analyze the packets when running the demos, list the source IPs, source port numbers, destination IPs, destination port numbers and response's status code of each connections.

Tip 2: capture loopback traffic





No.	Time	Source	Destination	Protocol
16	0.000241	127.0.0.1	127.0.0.1	TCP
17	0.000022	127.0.0.1	127.0.0.1	TCP
18	0.000023	127.0.0.1	127.0.0.1	TCP
19	0.000012	127.0.0.1	127.0.0.1	TCP



Part C.

Configure your PC as an HTTP server

Using http.server(1)

- Run command “python –m http.server” on your PC to create the simplest HTTP server.
- Official documentation: https://docs.python.org/3/library/http.server.html
- The server listens to port 8000 by default. The default can be overridden by passing the desired port number as an argument.
- By default, the server binds itself to all interfaces. The option -b/--bind specifies a specific address to which it should bind.
- By default, the server uses the current directory. The option -d/--directory specifies a directory to which it serves the files.

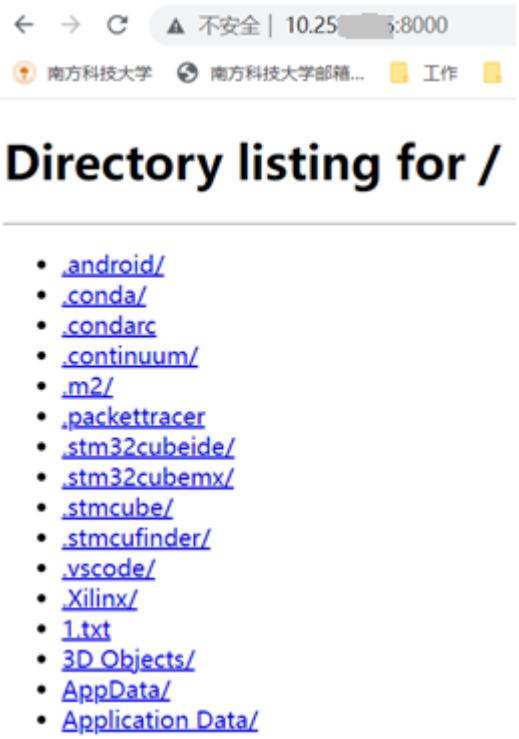
Using http.server(2)

- Server

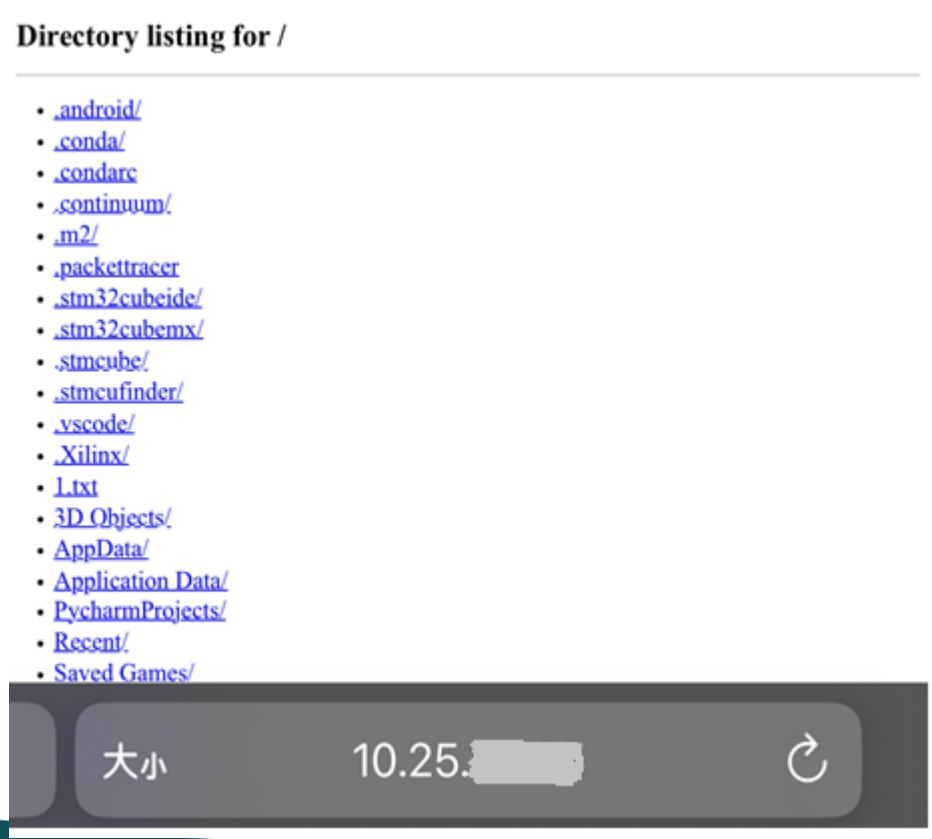
```
C:\Users\wq>python -m http.server
Serving HTTP on :: port 8000 (http://[::]:8000/) ...
::ffff:10.25.70.41 - - [06/Mar/2023 10:03:31] "GET / HTTP/1.1" 200 -
::ffff:10.25.70.41 - - [06/Mar/2023 10:03:31] code 404, message File not found
::ffff:10.25.70.41 - - [06/Mar/2023 10:03:31] "GET /favicon.ico HTTP/1.1" 404 -
::ffff:10.25.70.41 - - [06/Mar/2023 10:03:46] "GET /test.txt HTTP/1.1" 200 -
::ffff:10.16.6.199 - - [06/Mar/2023 10:03:57] "GET / HTTP/1.1" 200 -
::ffff:10.16.6.199 - - [06/Mar/2023 10:04:08] "GET /Documents/ HTTP/1.1" 200 -
::ffff:10.16.6.199 - - [06/Mar/2023 10:04:11] code 404, message No permission to list directory
::ffff:10.16.6.199 - - [06/Mar/2023 10:04:11] "GET /Documents/My%20Pictures/ HTTP/1.1" 404 -
::ffff:10.16.6.199 - - [06/Mar/2023 10:04:22] code 404, message No permission to list directory
::ffff:10.16.6.199 - - [06/Mar/2023 10:04:22] "GET /Cookies/ HTTP/1.1" 404 -
::ffff:10.25.[redacted] - - [06/Mar/2023 10:04:39] "GET / HTTP/1.1" 200 -
::ffff:10.25.[redacted] - - [06/Mar/2023 10:04:44] "GET /Downloads/ HTTP/1.1" 200 -
```

Using http.server(3)

- Client 1(PC)



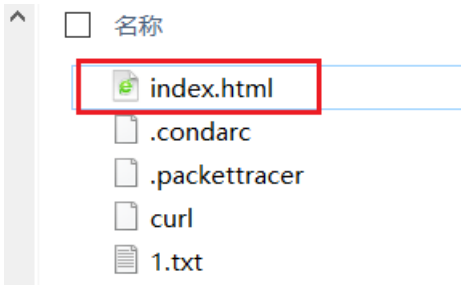
- Client 2 (mobile phone)



SUSTech
Southern University
of Science and Technology

Using http.server(4)

- Add a file named “index.html” under the root of server.



File content:

```
<html>
This is a test demo of CS305
</html>
```

- Access the URL again.

Practice 4.2

- Suppose your IP address is ***.***.***.***
- Run command “python –m http.server” on your PC.
- Use Wireshark to capture and analyze the packets.
  - 2-1. When accessing the web server from your own PC, which URL will work, “***.***.***.***:8000” or “127.0.0.1:8000” ?
  - 2-2. When accessing the web server from your own PC, which interface should you choose?
  - 2-3. Let your classmate to access your web server, which URL will work, “***.***.***.***:8000” or “127.0.0.1:8000” ?
  - 2-4. Let your classmate to access your web server, which interface should you choose if you want to capture the traffic?
- Capture and analyze the packets, list the source IPs, source port numbers, destination IPs, destination port numbers and response’s status code of each connections.

Using flask(1)

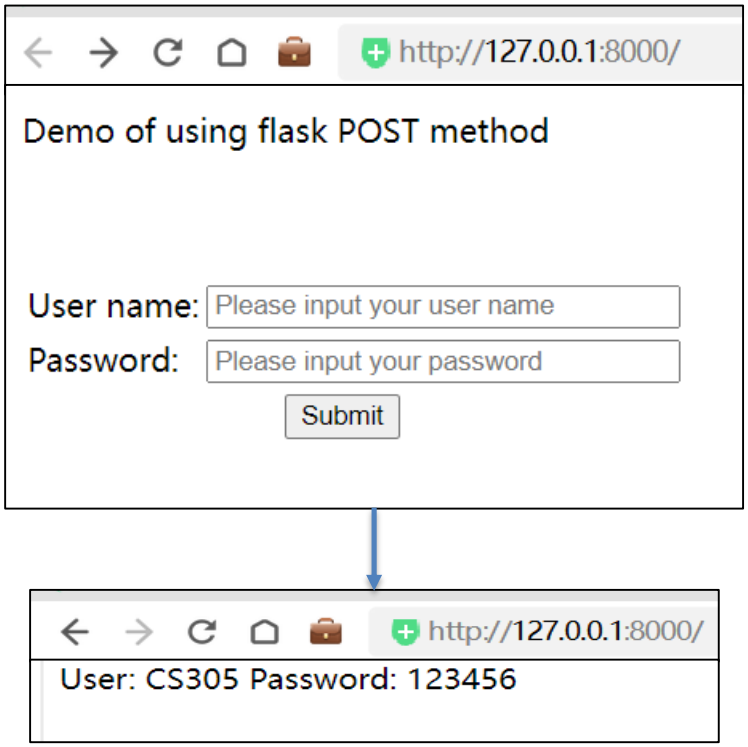
- A lightweight web application framework written in Python.
- Official documentation: https://flask.palletsprojects.com/en/2.2.x/
- Flask depends on the Jinja template engine and the Werkzeug WSGI toolkit.
- Jinja is a fast, expressive, extensible templating engine.
- Werkzeug is a comprehensive WSGI web application library. (WSGI: Web Server Gateway Interface)

Using flask(2)

```
from flask import Flask
from flask import request
from flask import render_template

app = Flask(__name__)

@app.route("/", methods=['GET', 'POST'])
def main():
    if request.method == 'GET':
        return render_template('main.html')
    if request.method == 'POST':
        user = request.form.get('user')
        password = request.form.get('password')
        res = "User: %s\nPassword: %s" % (user, password)
        return res

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=8000, debug=True)
```

Using flask(3)

- Client 1

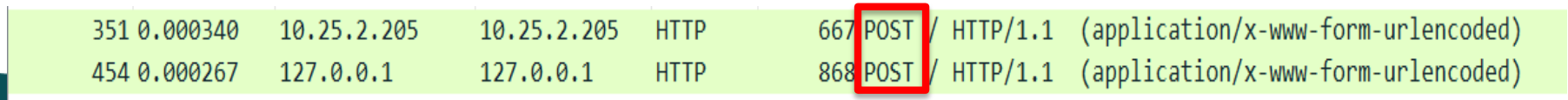


- Client 2

SUSTech
Southern University
of Science and Technology

Tip 3:

- Flask uses port 5000 by default, you can choose other port numbers when running as demo shows.
- When running the demo, you should put main.html in a folder named “templates”. For example, suppose the python project locates under “D:\CS305\http”, and then you should put main.html in “D:\CS305\http\templates”.

Practice 4.3

- Suppose your IP address is ***.***.***.***
- Run the demo using flask on your PC.
- Use Wireshark to capture and analyze the packets.
  - 3-1. When accessing the web server from your own PC, which interface should you choose?
  - 3-2. Let your classmate to access your web server, which interface should you choose if you want to capture the traffic?
- If we need analyze the packages using POST method, what display filter should be set? Are the packages request messages or response messages?