

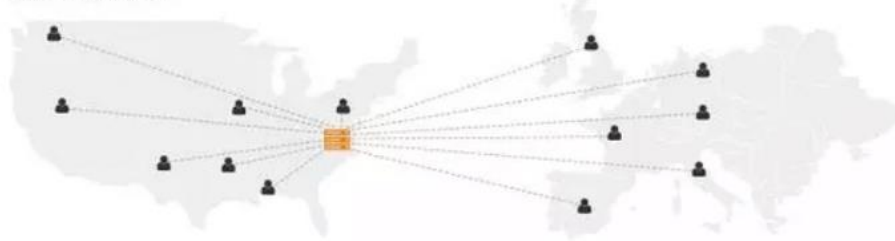
CS 305 Lab Tutorial

Lab 6 CDN & WebSocket

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

Part 1. CDN

Without a CDN



With a CDN

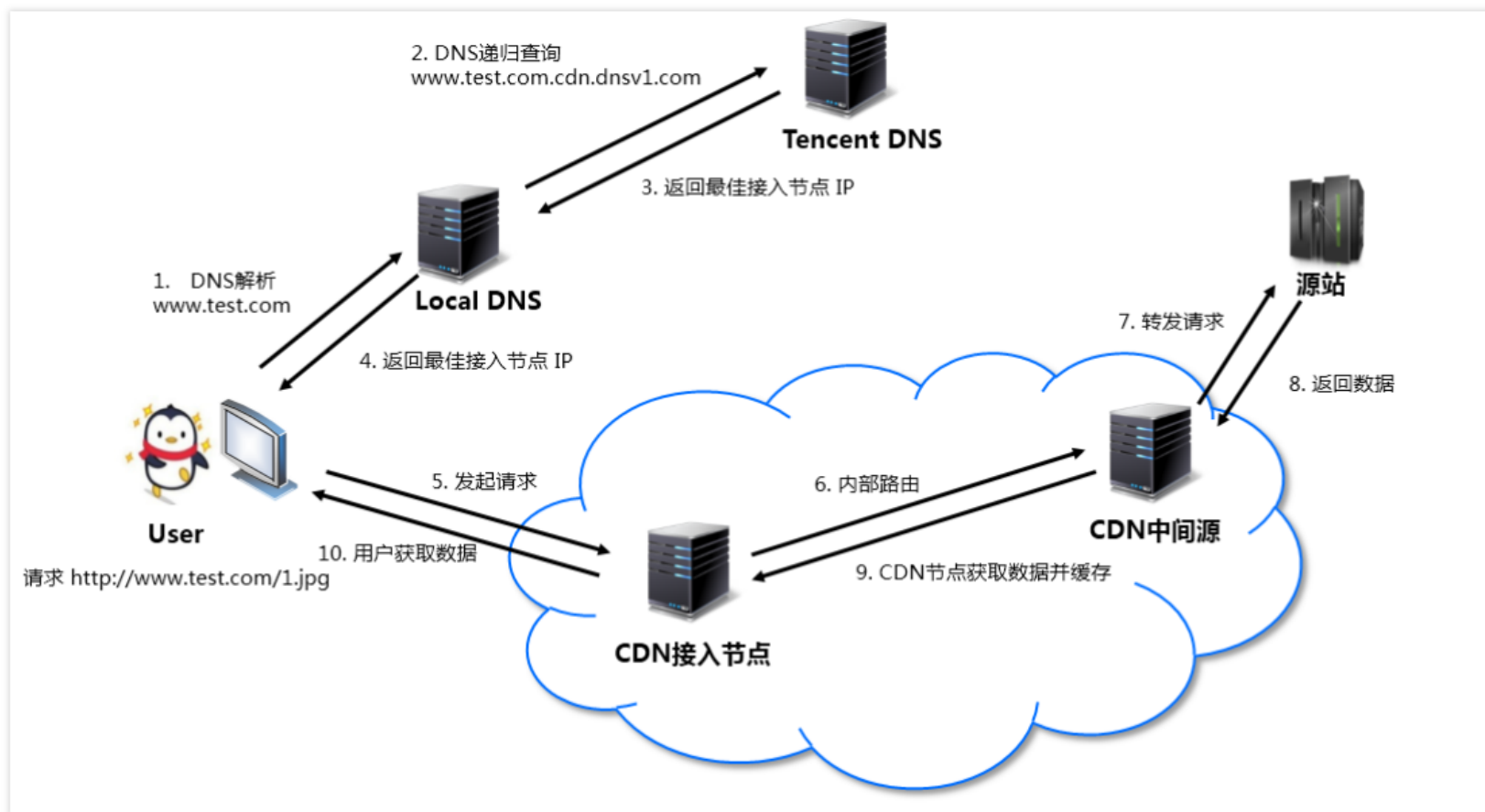


CDN (Content Delivery Network) is to cache content on a node closer to the user to improve the user experience.

What are the scenarios of CDN?

- **Big flow** website, such as: online video, games, pictures, audio, social, e-commerce, download stations, etc.
- CDN is suitable for a certain level of **static resource** access, including html, js, css, apk, mp3, flv, jpg, gif, mp4, flv and all other static resources.

Part 1. CDN



Part 1. CDN

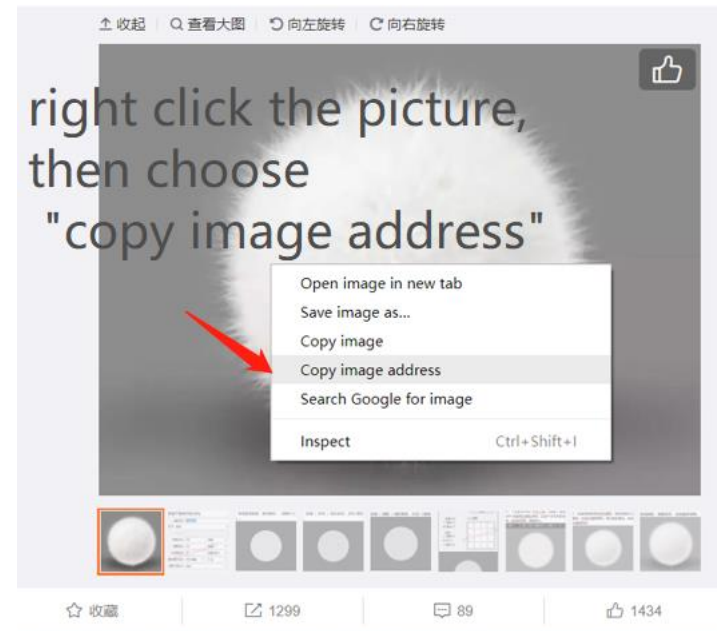
Q: How long is the cache time of file on the CDN server ?

A:

- The caching time of files refers to the cache time cycle of files in browsers.
- The CDN cache server strictly adheres to the standard HTTP protocol, and the cache time is controlled by the Cache-Control and Expires response headers in the HTTP response header
- Html file cache time viewing: Look at the Cache-Control in the HTTP header, such as "Cache-Control max-age = 2592000 (seconds)" to indicate that the file will be cached for 30 days. At this point, unless you use manual refresh, the newly opened browser page will not go back to the source to retrieve the file during the file cache cycle.

Demo 1

1. Find a web sit which may use CDN, find a static resource on it
2. To get the URL of this static resource(such as a picture)
3. Using command “curl” to get the content of this static resource
4. Check if this is on the CDN node or not based on the command output



The result of curl

```
C:\Users\Administrator>curl https://d4.sina.com.cn/202110/15/1581703.jpg --head
HTTP/1.1 200 OK
Server: nginx
Date: Mon, 18 Oct 2021 01:25:27 GMT
Content-Type: image/jpeg
Content-Length: 86080
Connection: keep-alive
X-RequestId: 0cbc680a-2110-1809-0045-0894eff93828
X-Requester: GRPS000000ANONYMOUSE
Last-Modified: Fri, 15 Oct 2021 07:16:45 GMT
X-Filesize: 86080
ETag: "96292990b34d783193938a31187d892c"
x-amz-meta-crc32: 12D0DBA9
x-amz-meta-uploadlocation: /ad4
Cache-Control: max-age=604800
Access-Control-Allow-Headers: Origin, Content-Type, Accept, Range, Content-Length
Access-Control-Allow-Methods: GET, PUT, POST, DELETE, OPTIONS, HEAD
Access-Control-Max-Age: 31536000
Access-Control-Allow-Origin: *
Expires: Mon, 25 Oct 2021 01:00:45 GMT
Edge-Copy-Time: 1634518845207
Age: 1483
Via: https/1.1 dfwx.guangdong.union.163 (ApacheTrafficServer/6.2.1 [cRs f ])
X-Cache: HIT.163
X-Via-CDN: f=edge, s=dfwx.guangdong.union.163.nb.sinaedge.com, c=10.245.100.15;f=Edge, s=dfwx.guangdong.union.163, c=103.116.123.163
X-Via-Edge: 16345203279890f64f50aa37b74676d6ca013
```

A static resource which is cached on a CDN node

Common CDN service providers(1)

- Tencent
 - X-Cache-Lookup:Hit From MemCache
represents the memory that hit the CDN node
 - X-Cache-Lookup:Hit From Disktank
indicates the disk that hit the CDN node
 - X-Cache-Lookup:Hit From Upstream
indicates that the CDN was not hit

```
C:\Users\wq>curl https://inews.gting.com/om_ls/02pUbRFwi4lbTsdNojSk_7zG3nu  
rWCmsjEX64TAZFIBHoAA_870492/0 -i -v
```

```
< X-Cache-Lookup: Cache Hit  
X-Cache-Lookup: Cache Hit  
< Access-Control-Expose-Headers: X-Client-Ip  
Access-Control-Expose-Headers: X-Client-Ip  
< Access-Control-Expose-Headers: X-Server-Ip  
Access-Control-Expose-Headers: X-Server-Ip  
< Access-Control-Expose-Headers: X-Upstream-Ip  
Access-Control-Expose-Headers: X-Upstream-Ip
```

Common CDN service providers(2)

- Alibaba
 - X-Cache:HIT TCP_MEM_HIT
represents hit cache
 - X-Cache:MISS TCP_MISS
indicates a cache miss

```
< X-Cache: HIT TCP_HIT dirn:13:280073047
X-Cache: HIT TCP_HIT dirn:13:280073047
< X-Swift-SaveTime: Sat, 21 Oct 2023 00:50:06 GMT
X-Swift-SaveTime: Sat, 21 Oct 2023 00:50:06 GMT
< X-Swift-CacheTime: 27802806
X-Swift-CacheTime: 27802806
< s-rt: 4
s-rt: 4
```


Common CDN service providers(3)

- Seven Cattle

```
< X-Log: X-Log;IMAGESLIM:152;BOOTS-PROXY:152;BOOTS-PROXY:153;BOOTS-  
X-Log: X-Log;IMAGESLIM:152;BOOTS-PROXY:152;BOOTS-PROXY:153;BOOTS-P  
< X-M-Log: QNM:jjh1504;SRCPROXY:jjh1535;QNM:jjh1503;SRCPROXY:jjh15  
X-M-Log: QNM:jjh1504;SRCPROXY:jjh1535;QNM:jjh1503;SRCPROXY:jjh1540  
< X-M-Reqid: QlcAACLW3WizzmwX  
X-M-Reqid: QlcAACLW3WizzmwX  
< X-Qiniu-Zone: 0  
X-Qiniu-Zone: 0  
< X-Qnm-Cache: Miss  
X-Qnm-Cache: Miss  
< X-Qnm-Cache: Miss,RawProxy  
X-Qnm-Cache: Miss,RawProxy  
< X-Reqid: QlcAACLW3WizzmwX  
X-Reqid: QlcAACLW3WizzmwX  
< X-Slim-Origin: 0  
X-Slim-Origin: 0  
< X-Svr: IO  
X-Svr: IO  
< X-Ser: BC154_dx-lt-yd-jiangsu-huaian-25-cache-2, BC20_yd-zhejiang  
X-Ser: BC154_dx-lt-yd-jiangsu-huaian-25-cache-2, BC20_yd-zhejiang-  
< X-Cache: HIT from BC20_yd-zhejiang-wenzhou-20-cache-1(baishan)  
X-Cache: HIT from BC20_yd-zhejiang-wenzhou-20-cache-1(baishan)
```

Part 2. WebSocket

- The WebSocket Protocol is designed to supersede existing bidirectional communication technologies that use HTTP as a transport layer to benefit from existing infrastructure (proxies, filtering, authentication).
- The WebSocket Protocol attempts to address the goals of existing bidirectional HTTP technologies in the context of the existing HTTP infrastructure.
 - support HTTP proxies and intermediaries
 - does not limit WebSocket to HTTP, and future implementations could use a simpler handshake over a dedicated port without reinventing the entire protocol.

<https://www.rfc-editor.org/rfc/rfc6455.txt>

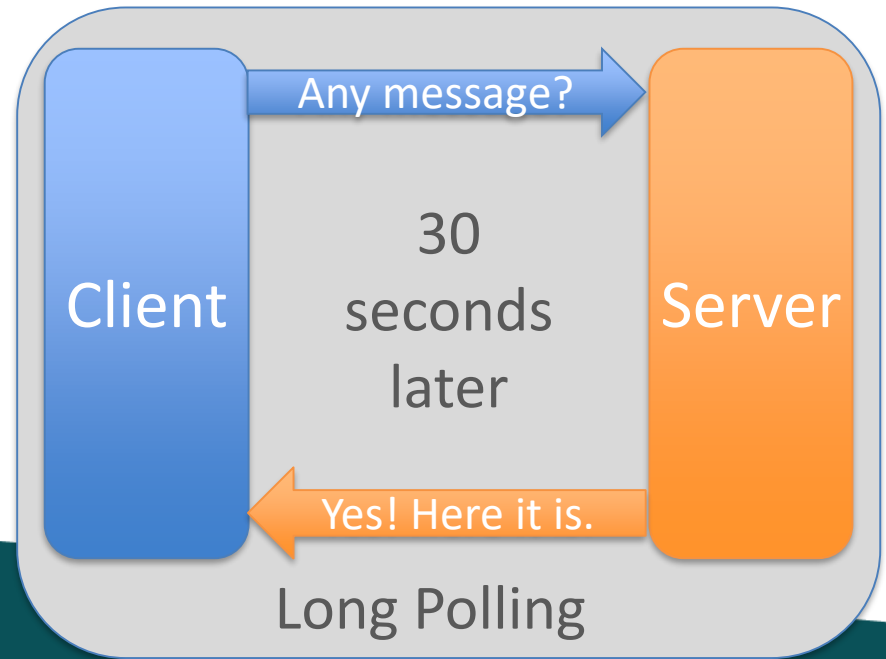
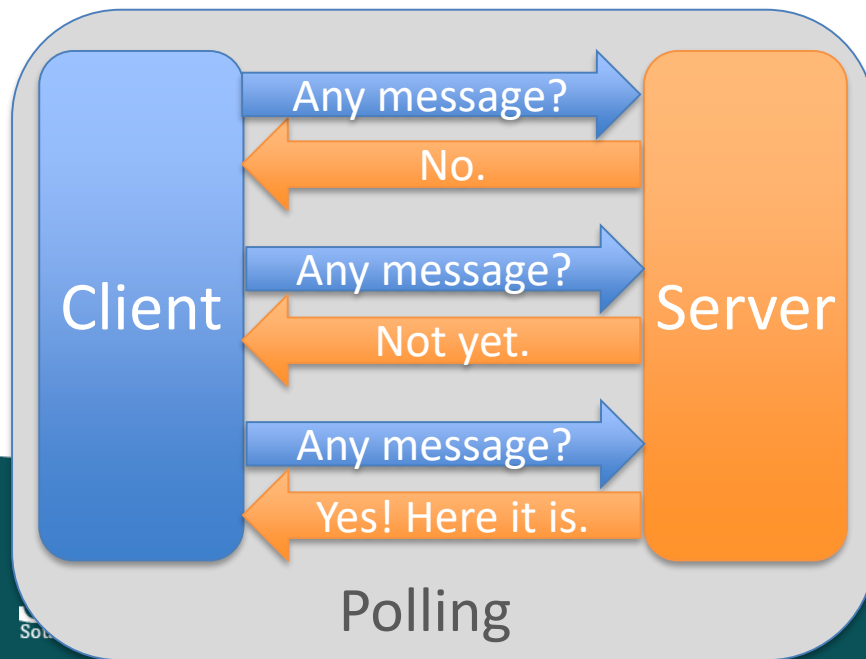
Main features of WebSocket

- Application layer protocol
- Established on TCP protocol
- Use the port 80
- Full duplex
- ws-URL= "ws:" "://" host [":" port] path ["?" query]
 - ws://example.com/chat
- wss-URL= "wss:" "://" host [":" port] path ["?" query]

<https://www.rfc-editor.org/rfc/rfc6455.txt>

HTTP polls for new messages

- Historically, creating web applications that need bidirectional communication between a client and a server (e.g., instant messaging and gaming applications) has required an abuse of HTTP to poll the server for updates while sending upstream notifications as distinct HTTP calls.



WebSocket protocol overview

- Two parts
 - Handshakes: Opening Handshake & Closing Handshake
 - Data transfer
- The WebSocket Protocol is an independent TCP-based protocol. Its only relationship to HTTP is that its handshake is interpreted by HTTP servers as an Upgrade request.
 - Uses port 80 for regular WebSocket connections
 - Uses port 443 for WebSocket connections tunneled over TLS
 - Can not establish a connection with servers of pre-existing protocols like SMTP and HTTP

Opening handshake

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
```

client



example.com



“dGhlIHNhbXBsZSBub25jZQ==”

Step1. concatenate “258EAF45-E914-47DA-95CA-C5AB0DC85B11”

Step2. take the SHA-1 hash

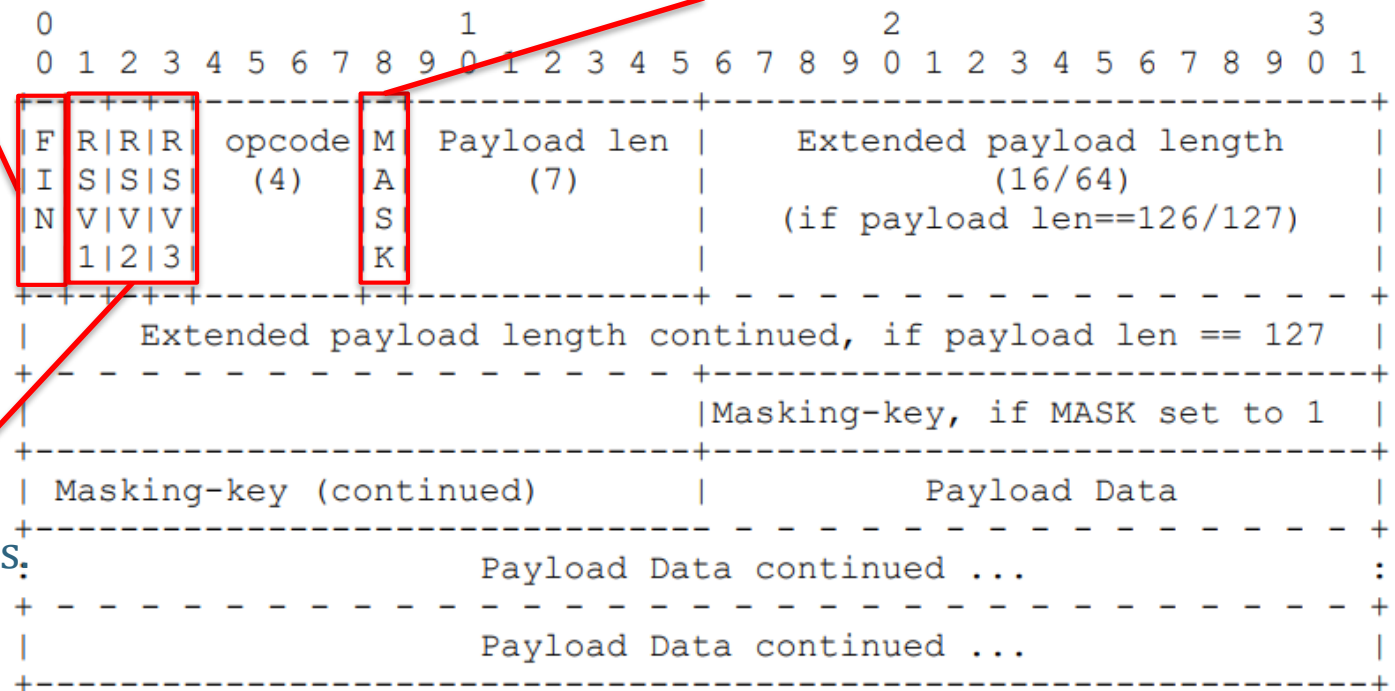
Step3. base64-encoded the hash value

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept:
s3pPLMBiTxaQ9kYGzzhZRbK+x0o=
Sec-WebSocket-Protocol: chat
```

Data framing

Indicates that this is the final fragment in a message.

Defines whether the "Payload data" is masked.



Data framing (continued)

- Opcode (4 bits)
 - %x0(0000) denotes a continuation frame
 - %x1(0001) denotes a text frame
 - %x2(0010) denotes a binary frame
 - %x3-7(0011-0111) are reserved for further non-control frames
 - %x8(1000) denotes a connection close
 - %x9(1001) denotes a ping
 - %xA(1010) denotes a pong
 - %xB-F(1011-1111) are reserved for further control frames

Data framing (continued)

- Payload len
 - 7 bits: if 0~125 bytes
 - 7 + 16 bit: if the 7 bits equals to 126
 - 7 + 64 bits: if the 7 bits equals to 127
 - payload length = the length of the "Extension data" + the length of the "Application data"
- Masking key
 - 0 bits: if the MASK bit is set to 0
 - 32 bits: if the MASK bit is set to 1
 - The masking key is a 32-bit value chosen at random by the client.

Data framing examples

- A single-frame unmasked text message
 - 0x81 0x05 0x48 0x65 0x6c 0x6c 0x6f (contains "Hello")
- A single-frame masked text message
 - 0x81 0x85 0x37 0xfa 0x21 0x3d 0x7f 0x9f 0x4d 0x51 0x58 (contains "Hello")
- A fragmented unmasked text message
 - 0x01 0x03 0x48 0x65 0x6c (contains "Hel")
 - 0x80 0x02 0x6c 0x6f (contains "lo")
- Unmasked Ping request and masked Ping response
 - 0x89 0x05 0x48 0x65 0x6c 0x6c 0x6f (contains a body of "Hello", but the contents of the body are arbitrary)
 - 0x8a 0x85 0x37 0xfa 0x21 0x3d 0x7f 0x9f 0x4d 0x51 0x58 (contains a body of "Hello", matching the body of the ping)
- 256 bytes binary message in a single unmasked frame
 - 0x82 0x7E 0x0100 [256 bytes of binary data]

Closing the connection

- An endpoint MUST send a Close control frame(opcode = 1000)
- Connection Close Code
 - 1000: a normal closure.
 - 1001: an endpoint is "going away".
 - 1002: a protocol error occurs.
 - 1003: an endpoint has received a type of data it cannot accept.
 - 1004, 1005, 1006, 1015: Reserved.
 - 1007: an endpoint has received an inconsistent type of data.
 - 1008: an endpoint has received a message violates its policy.
 - 1009: an endpoint has received a message that is too big to process.
 - 1010: the client has expected the server to negotiate one or more extensions but received no response about that.
 - 1011: the server encountered an unexpected condition.

Example 1: Mimic a WebSocket Server

```
import asyncio
import websockets

async def echo(websocket, path):
    async for message in websocket:
        message = "I got your message: {}".format(message)
        await websocket.send(message)

asyncio.get_event_loop().run_until_complete(
    websockets.serve(echo, '127.0.0.1', 8766))
asyncio.get_event_loop().run_forever()
```

Example 2: Mimic a WebSocket Client

```
import asyncio
import websockets

async def echo(uri):
    async with websockets.connect(uri) as websocket:
        while True:
            message = input("Write down your message:")
            await websocket.send(message)
            print("<", message)
            recv_text = await websocket.recv()
            print("> {}".format(recv_text))

asyncio.get_event_loop().run_until_complete(
    echo('ws://127.0.0.1:8766'))
```

Apply a display filter ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
10	0.000387	127.0.0.1	127.0.0.1	HTTP	307	GET / HTTP/1.1
11	0.000035	127.0.0.1	127.0.0.1	TCP	44	8766 → 6724 [ACK] Seq=1 Ack=264 Win=2619648 Len=0
12	0.002069	127.0.0.1	127.0.0.1	HTTP	346	HTTP/1.1 101 Switching Protocols

> Transmission Control Protocol, Src Port: 6724, Dst Port: 8766, Seq: 1, Ack: 1, Len: 263

▼ Hypertext Transfer Protocol

> GET / HTTP/1.1\r\n

Host: 127.0.0.1:8766\r\n

Upgrade: websocket\r\n

Connection: Upgrade\r\n

Sec-WebSocket-Key: 8Hh1RpEngq3e+mXOmDINvQ==\r\n

Sec-WebSocket-Version: 13\r\n

Sec-WebSocket-Extensions: permessage-deflate; client_max_window_bits\r\n

User-Agent: Python/3.9 websockets/10.2\r\n

\r\n

[Full request URI: <http://127.0.0.1:8766/>]

[HTTP request 1/1]

[Response in frame: 12]

[Community ID: 1:UevRM2Dfbi6SaPxt5m6ld6vBavM=]

激活 Windows
转到“设置”以激活 Windows。

Apply a display filter ... <Ctrl-/> → +

No.	Time	Source	Destination	Protocol	Length	Info
10	0.000387	127.0.0.1	127.0.0.1	HTTP	307	GET / HTTP/1.1
11	0.000035	127.0.0.1	127.0.0.1	TCP	44	8766 → 6724 [ACK] Seq=1 Ack=264 Win=2619648 Len=0
12	0.002069	127.0.0.1	127.0.0.1	HTTP	346	HTTP/1.1 101 Switching Protocols

< →

> Transmission Control Protocol, Src Port: 8766, Dst Port: 6724, Seq: 1, Ack: 264, Len: 302

▼ Hypertext Transfer Protocol

> HTTP/1.1 101 Switching Protocols\r\n

Upgrade: websocket\r\n

Connection: Upgrade\r\n

Sec-WebSocket-Accept: TD/u6+TjNMIH6bkOMbPh0f4KFwA=\r\n

Sec-WebSocket-Extensions: permessage-deflate; server_max_window_bits=12; client_max_window_bits=12\r\n

Date: Tue, 05 Apr 2022 04:10:41 GMT\r\n

Server: Python/3.9 websockets/10.2\r\n

\r\n

[HTTP response 1/1]

[Time since request: 0.002104000 seconds]

[\[Request in frame: 10\]](#)

[Request URI: http://127.0.0.1:8766/]

[Community ID: 1:UevRM2Dfbi6SaPxt5m6ld6vBavM=]

激活 Windows
转到“设置”以激活 Windows。

websocket

No.	Time	Source	Destination	Protocol	Length	Info
14	3.265380	127.0.0.1	127.0.0.1	WebSocket	57	WebSocket Text [FIN] [MASKED]
16	0.000827	127.0.0.1	127.0.0.1	WebSocket	73	WebSocket Text [FIN]

<

▼ WebSocket

- 1... = Fin: True
- .100 = Reserved: 0x4
- .1... = Per-Message Compressed: True
- 0001 = Opcode: Text (1)
- 1... = Mask: True
- .000 0111 = Payload length: 7
- Masking-Key: 58905743
- Masked payload
- Payload

▼ Line-based text data (1 lines)

hello

[Community ID: 1:UevRM2Dfbi6SaPxt5m6ld6vBavM=]

0000 68 65 6c 6c 6f hello

websocket

No.	Time	Source	Destination	Protocol	Length	Info
14	3.265380	127.0.0.1	127.0.0.1	WebSocket	57	WebSocket Text [FIN] [MASKED]
16	0.000827	127.0.0.1	127.0.0.1	WebSocket	73	WebSocket Text [FIN]

> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1

> Transmission Control Protocol, Src Port: 8766, Dst Port: 6724, Seq: 303, Ack: 277, Len: 29

▼ WebSocket

1... .. = Fin: True

.100 .. = Reserved: 0x4

.1... .. = Per-Message Compressed: True

.... 0001 = Opcode: Text (1)

0... .. = Mask: False

.001 1011 = Payload length: 27

Payload

▼ Line-based text data (1 lines)

I got your message: hello

[Community ID: 1:UevRM2Dfbi6SaPxt5m6ld6vBavM=]

0000

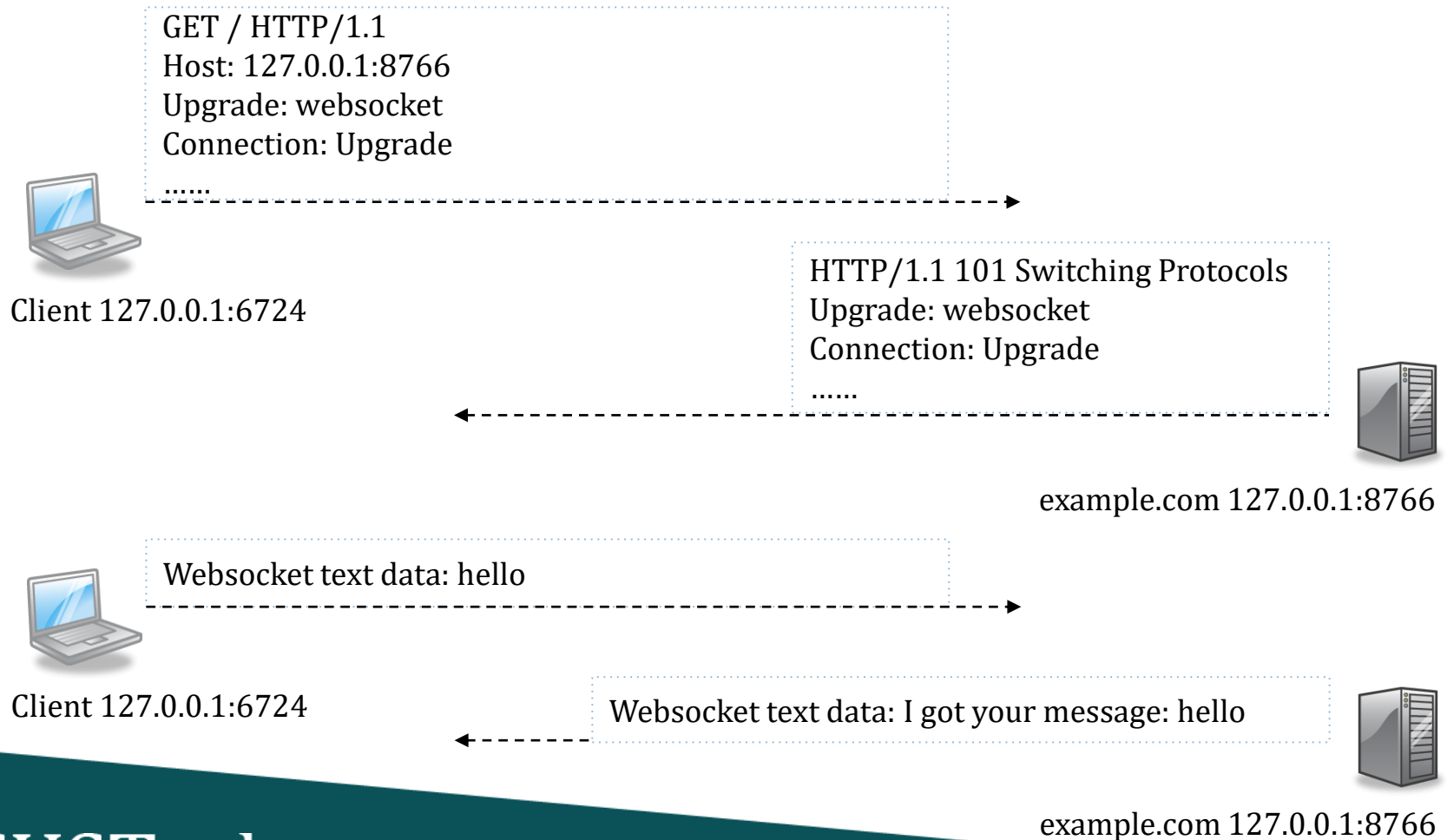
49 20 67 6f 74 20 79 6f 75 72 20 6d 65 73 73 61

I got yo ur messa

SUSTech

Southern University
 of Science and Technology

Process analysis



Example 3: Use curl

```
> curl --include --no-buffer --header "Connection: Upgrade" --header  
"Upgrade: websocket" --header "Host: example.com:80" --header "Origin:  
http://example.com:80" --header "Sec-WebSocket-Key:  
GVsbG8sIHdvcmxkIQ==" --header "Sec-WebSocket-Version: 13"  
http://example.com:80/ -i -v
```

```
C:\Users\wq>curl --include --no-buffer --header "Connection: Upgrade" --header "Upgrade: websocket"  
" --header "Host: example.com:80" --header "Origin: http://example.com:80" --header "Sec-WebSocke  
t-Key: GVsbG8sIHdvcmxkIQ==" --header "Sec-WebSocket-Version: 13" http://example.com:80/ -i -v  
* Trying 2606:2800:220:1:248:1893:25c8:1946...  
* TCP_NODELAY set  
* Connected to example.com (2606:2800:220:1:248:1893:25c8:1946) port 80 (#0)  
> GET / HTTP/1.1  
> Host: example.com:80  
> User-Agent: curl/7.55.1  
> Accept: */*  
> Connection: Upgrade  
> Upgrade: websocket  
> Origin: http://example.com:80  
> Sec-WebSocket-Key: GVsbG8sIHdvcmxkIQ==  
> Sec-WebSocket-Version: 13  
>  
< HTTP/1.1 200 OK  
HTTP/1.1 200 OK
```

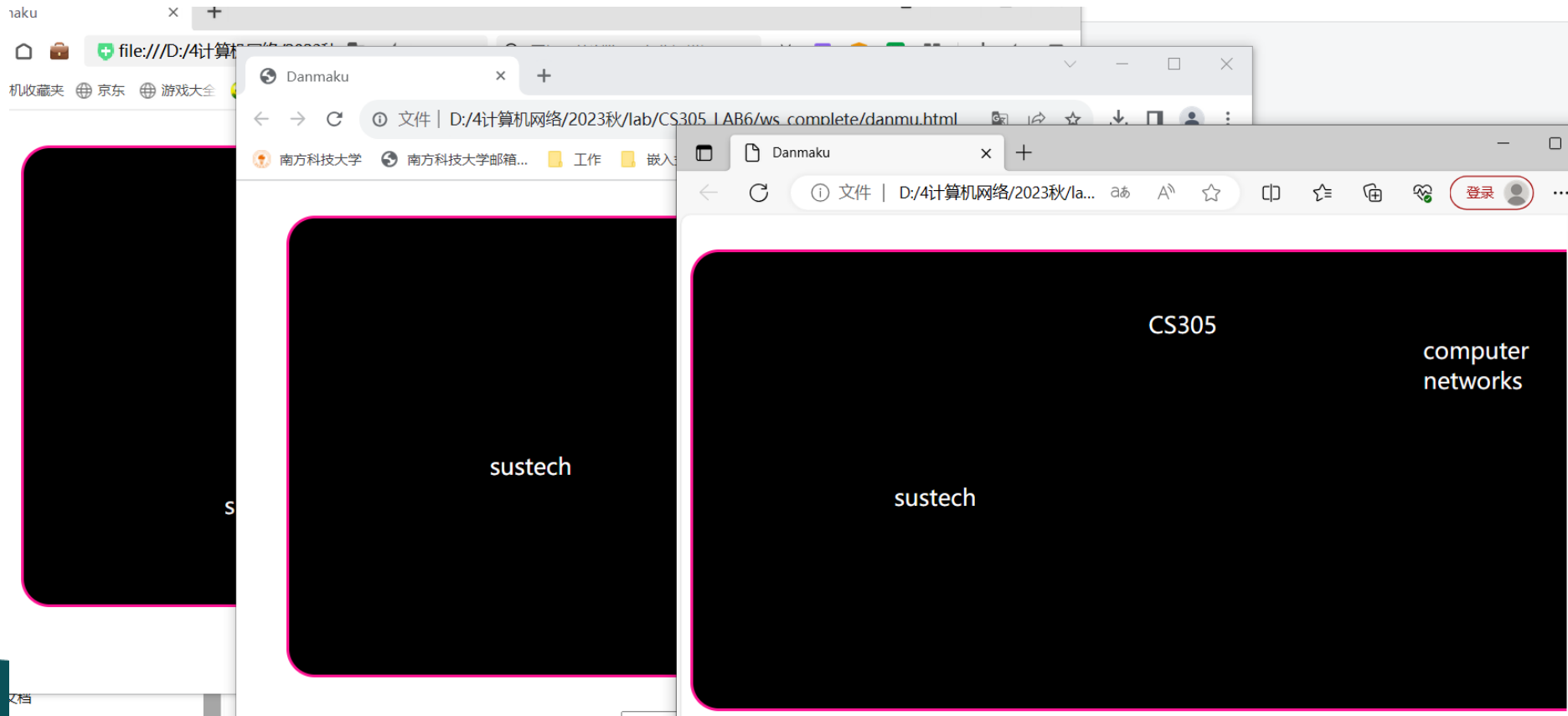
Practice 6.1 Finding a CDN user

- Use curl to get a resource from web which using CDN to upgrade the accessing speed and balance the traffic load (www.sina.com.cn is recommended)
 - From what header field(s) can you tell that this web is using CDN?
 - Use nslookup/dig in your computer to find the IP address of this web site.
 - Use nslookup/dig in your computer to find the IP address of the CDN server you are accessing.
 - Ask your friend who is in another province to practice the same thing(using nslookup/dig to find the IP address of the web site which using CDN, find the IP address of this web site.)

Practice 6.2 Implement a simple だんま(danmaku) system

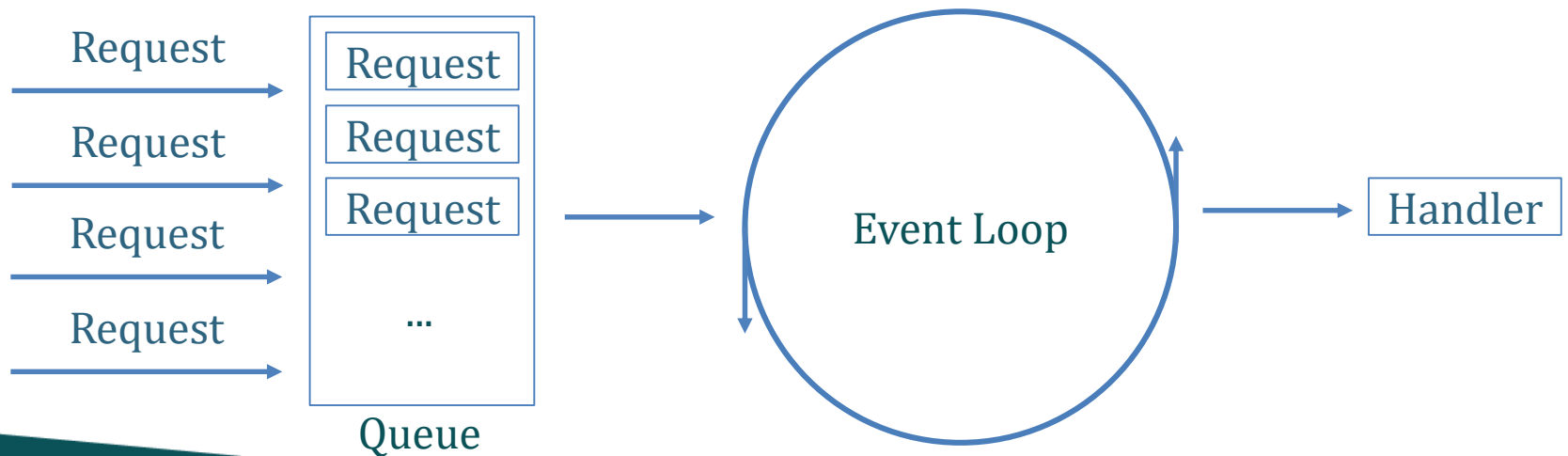
- Use python to implement a simple だんま(danmaku) system.
 - Use a python file as a server and a html file as a client.
 - The system should listen and accept WS messages.
 - Receive danmakus from clients, and send them to all clients and display them on their screens.
 - You should have at least 3 clients, i.e., on at least 3 browser tabs, to show whether danmakus are correctly sent and appear simultaneously on each client.
 - Use Wireshark to capture and analyze the packets when running the system, observe what would the server do after receiving a message from one client.

Practice 6.2 Implement a simple だんま(danmaku) system



Tip: asyncio

- Asyncio: Asynchronous I/O
- Handle requests with a single-threaded concurrently.
- Official documentation:
<https://docs.python.org/3/library/asyncio.html>



Example 4: simple asyncio demo

```
import asyncio
import time
```

```
async def main():
    print(f"Started at {time.strftime('%X')}")

    print('Hello')
    print(f" end at {time.strftime('%X')}")
    await asyncio.sleep(1)
    print('world')
    print(f" end at {time.strftime('%X')}")

    print(f"Finished at {time.strftime('%X')}")

asyncio.run(main())
```

```
Started at 21:59:16
```

```
Hello
```

```
end at 21:59:16
```

```
world
```

```
end at 21:59:17
```

```
Finished at 21:59:17
```

```
Process finished with exit code 0
```


Example 5

```
import asyncio
import time

async def saysth(delay, sth):
    await asyncio.sleep(delay)
    print(f"{sth}: delay {delay}s, end at {time.strftime('%X')}")

async def main():
    print(f"Started at {time.strftime('%X')}")
    await saysth(1, 'hello')
    await saysth(2, 'world')
    await saysth(3, 'SUSTECH')
    await saysth(1, 'CS305')
    await saysth(1, '2023-10-22')
    print(f"Finished at {time.strftime('%X')}")

asyncio.run(main())
```

```
Started at 22:00:46
hello: delay 1s, end at 22:00:47
world: delay 2s, end at 22:00:49
SUSTECH: delay 3s, end at 22:00:52
CS305: delay 1s, end at 22:00:53
2023-10-22: delay 1s, end at 22:00:54
Finished at 22:00:54
```

```
Process finished with exit code 0
```

Example 6

```
import asyncio
import time

async def saysth(delay, sth):
    await asyncio.sleep(delay)
    print(f"{sth}: delay {delay}s, end at {time.strftime('%X')}")

async def main():
    task1 = asyncio.create_task(saysth(1, 'hello'))
    task2 = asyncio.create_task(saysth(2, 'world'))
    task3 = asyncio.create_task(saysth(3, 'SUSTECH'))
    task4 = asyncio.create_task(saysth(1, 'CS305'))
    task5 = asyncio.create_task(saysth(1, '2023-10-22'))
    print(f"Started at {time.strftime('%X')}")
    await task1
    await task2
    await task3
    await task4
    await task5
    print(f"Finished at {time.strftime('%X')}")

asyncio.run(main())
```

```
Started at 22:02:41
hello: delay 1s, end at 22:02:42
CS305: delay 1s, end at 22:02:42
2023-10-22: delay 1s, end at 22:02:42
world: delay 2s, end at 22:02:43
SUSTECH: delay 3s, end at 22:02:44
Finished at 22:02:44
```

```
Process finished with exit code 0
```

Example 7: asyncio Web Hello

```
import asyncio
```

```
async def dispatch(reader, writer):  
    while True:  
        data = await reader.readline()  
        message = data.decode().split(' ')  
        print(data)  
        if data == b'\r\n':  
            break  
        writer.writelines([  
            b'HTTP/1.0 200 OK\r\n',  
            b'Content-Type:text/html; charset=utf-8\r\n',  
            b'Connection: close\r\n',  
            b'\r\n',  
            b'<html><body>Hello World!</body></html>\r\n',  
            b'\r\n'  
        ])  
        await writer.drain()  
        writer.close()
```

```
if __name__ == '__main__':  
    loop = asyncio.get_event_loop()  
    coro = asyncio.start_server(dispatch, '127.0.0.1', 8080, loop=loop)  
    server = loop.run_until_complete(coro)  
  
    # Serve requests until Ctrl+C is pressed  
    print('Serving on {}'.format(server.sockets[0].getsockname()))  
    try:  
        loop.run_forever()  
    except KeyboardInterrupt:  
        pass  
  
    # Close the server  
    server.close()  
    loop.run_until_complete(server.wait_closed())  
    loop.close()
```