Assignment 1

Songtuan Lin u6162630

April 7, 2019

Question 1

For simplicity, I will assume the *HTTP Request* message the server received request a Web Page.

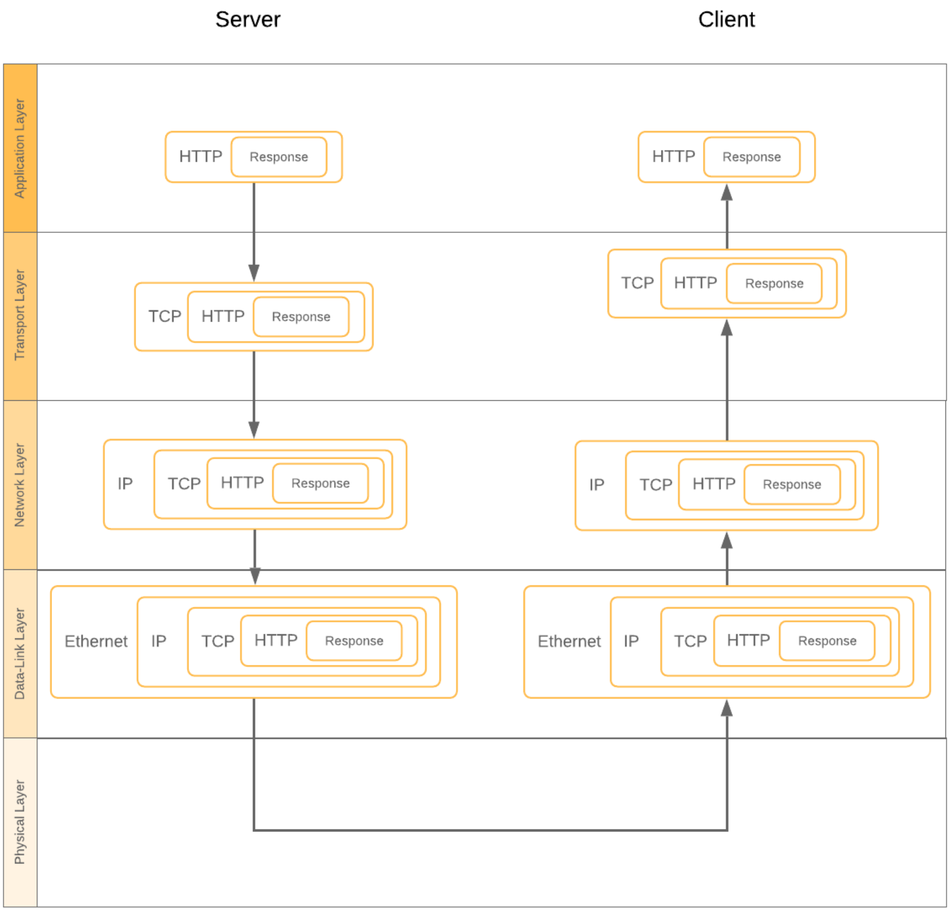


Figure 1: Transmission through Layers

By comparing Figure 1 with the original Figure, it can be seen that the HTTP Response message will go through the same Network Layer as the HTTP Request message. The only difference shown in these two figures is the *Request* field and *Response* field. Particularity, in this example, the client *request* a Web Page and the server *response* the corresponding html file. Indeed, this simple observation implies a general rule for the communication through computer network:

> The message delivery within computer network must go through a series of fixed stages and Each stage will accomplish some specific functions to support the delivery.

Technically, each stage correspond to one Layer and form the modern *Layered-Architecture*, as illustrated in

Figure 1. Furthermore, in order to maintain the generality of the message flow within Computer Network, which means the message can be decoded(understood) by different computer in this network, additional rules that specify how each layer accomplish their function have been made. These layer-specific rules are called *protocol*, e.g. HTTP in Application Layer, TCP in Transport Layer, etc. Generally, each protocol need to add some extra information to the original message in order to accomplish their function. This additional information is called *head*. e.g. HTTP head, TCP head, etc. In the following part, I will briefly explain the function and some common protocols corresponding to each layer:

- **Application Layer**: Application Layer produce the message that will be delivered and transfer it to next Transport Layer. Some popular Application Layer protocols include: HTTP, FTP, etc.
- **Transport Layer**: Transport Layer accomplish the preparation tasks before the delivery, e.g. specify the source and target application of the message, maintain the connection between client and server. The common Transport Layer protocols include: TCP and UDP.
- **Network Layer**: Network Layer determine the route of message delivery, which is one of the most important function. The common Network Layer protocols include OSPF, BGP, etc.
- **Data-Link Layer**: Data-Link Layer further determine the route based on the route calculated by Network Layer. The common protocols in Data-Link Layer include Ethernet, WIFI protocol family, etc.
- **Physical Layer**: Physical Layer is responsible for the actual transmission.

The above explanation may be too abstract. Indeed, we can think Computer Network as a real world package delivery system, hence, we can have a concrete understand about Layer-Architecture by going through an example: Assume a package delivery company have a package that want to be delivered from Australia to China. The entire delivery process, from Computer Network perspective, can be expressed as:

1. The Application Layer get a package from customer that need to be delivered.
2. The Transport Layer confirm this package delivery order.
3. The Network Layer determine the route of delivery. Particularly, it determine which country will be in the delivery path, e.g. Australia → Indian → China
4. The Data-Link Layer determine the route that how package will be delivered within country.
5. The physical Layer carry the package and do the actual transmission.

The above discussion give a theory explanation about Layer-Architecture. Indeed, by using software tools like Wireshark and cURL, we can do some practical experiments to see how Layered-Architecture work. These practical experiments will be demonstrated in the rest question within this assignment.

Question 2

In this question, I will use cURL to inspect HTTP protocol. This experiment can illustrate the format of HTTP head. In order to complete this question, firstly, I need to configure my web server following the setting up instruction described in this question. Particularly, the web server I used is a droplet from Digital-Ocean with IP address: 188.166.167.192. After I have deployed my web server, I can send a HTTP request through cURL by using following command:

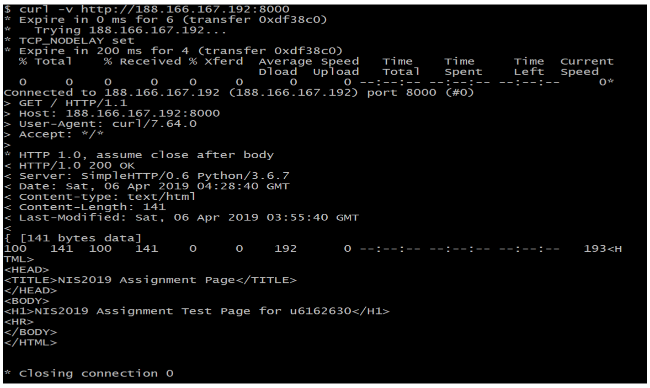
```
curl -v http://188.166.167.192:8000
```

The output of this cURL command is shown in Figure 2, In which, there are several important content:

1. HTTP Request message, which consist the lines start with '>'.
2. HTTP Response message with the HTML file required by client, which consist of the lines start with '<'.

By inspecting these two HTTP messages, we can identify that the format of these messages follow the definition in [Fielding et al., 1999], which is the standard of HTTP protocol. Even though cURL provide a useful tool to generate HTTP message, the majority of HTTP message in real world is produced by Web Browser. Hence, in Question 3, I will demonstrate how to use HTTP Request to retrieve the same Web Page through Web Browser.

Instead of using cURL to send HTTP Request, we can use simple Web Browser to retrieve the same Web Page, which will be shown in Question 3.



```
$ curl -v http://188.166.167.192:8000
* Expire in 0 ms for 6 (transfer 0xdf38c0)
*   Trying 188.166.167.192...
* TCP_NODELAY set
* Expire in 200 ms for 4 (transfer 0xdf38c0)
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0*
Connected to 188.166.167.192 (188.166.167.192) port 8000 (#0)
> GET / HTTP/1.1
> Host: 188.166.167.192:8000
> User-Agent: curl/7.64.0
> Accept: */*
>
* HTTP 1.0, assume close after body
< HTTP/1.0 200 OK
< Server: SimpleHTTP/0.6 Python/3.6.7
< Date: Sat, 06 Apr 2019 04:28:40 GMT
< Content-type: text/html
< Content-Length: 141
< Last-Modified: Sat, 06 Apr 2019 03:55:40 GMT
<
{ [141 bytes data]
100   141  100   141    0     0    192      0 --:--:-- --:--:-- --:--:--   193<H
TML>
<HEAD>
<TITLE>NIS2019 Assignment Page</TITLE>
</HEAD>
<BODY>
<H1>NIS2019 Assignment Test Page for u6162630</H1>
<HR>
</BODY>
</HTML>


* Closing connection 0
```

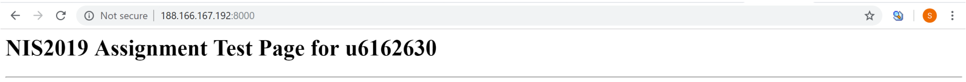
Figure 2: Output of cURL

Question 3

The HTML file that been created in web server can be retrieved through web browser by typing the following command:

http://188.166.167.192:8000

The result that displayed in web browser is shown in Figure 3:



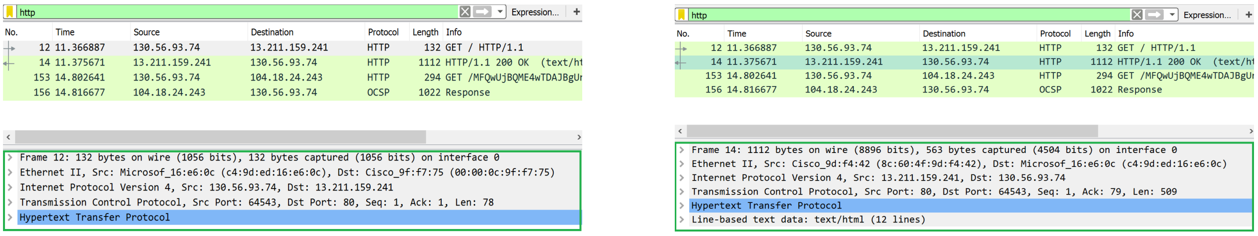
NIS2019 Assignment Test Page for u6162630

Figure 3: Web Page

In this experiment, the web browser will first send a HTTP Request message to the server. After that, this request message will go through the Layers-Architecture described in Question 1 with the corresponding protocol-head been added, *i.e.* the TCP head been added in Transport Layer, the IP head been added in Network Layer and the Ethernet head been added in Data-Link Layer. This process can be well illustrated by the Figure 1-4 in Question 1. When the server receive the request, it will reply with a HTTP Response message contain the Web Page required by client. This response message, as indicated in Figure 1, will go through the same Layers and arrive at the client side. Hence, the web browser can display the Web Page. One more thing that need to be noticed within this process is: When the message arrive at the receiver side, *e.g.* the server in Figure 1-4 and client in Figure 1, it will go through the same Layer-Architecture but in inverse order. In the mean time ,each Layer would remove the corresponding protocol head and deliver the message to upper Layer.

Question 4

The Question 2 and Question 3 has demonstrated some basic property of HTTP protocol, include its' message format and general delivery process. Indeed, we can also inspect protocols' head other than HTTP by using Wireshark, which is a tool can be used to capture the messages flow within Computer Network. I will demonstrate how to use Wireshark to capture HTTP Request and Response message and use this result to further illustrate Layer-Architecture Model. The following Figure 4a and 4b are the capture panel of Wireshark, in which, the first two messages are the HTTP message between my computer and the server with IP address 13.211.159.241:



No.	Time	Source	Destination	Protocol	Length	Info
12	11.366887	130.56.93.74	13.211.159.241	HTTP	132	GET / HTTP/1.1
14	11.375671	13.211.159.241	130.56.93.74	HTTP	1112	HTTP/1.1 200 OK (text/ht
153	14.802641	130.56.93.74	104.18.24.243	HTTP	294	GET /MFQwUjBQME4wTDAJBgUr
156	14.816677	104.18.24.243	130.56.93.74	OCSP	1022	Response

```
> Frame 12: 132 bytes on wire (1056 bits), 132 bytes captured (1056 bits) on interface 0
> Ethernet II, Src: Microsof_16:e6:0c (c4:9d:ed:16:e6:0c), Dst: Cisco_9f:f7:75 (00:00:0c:9f:f7:75)
> Internet Protocol Version 4, Src: 130.56.93.74, Dst: 13.211.159.241
> Transmission Control Protocol, Src Port: 64543, Dst Port: 80, Seq: 1, Ack: 1, Len: 78
> Hypertext Transfer Protocol
```

(a) Request Message

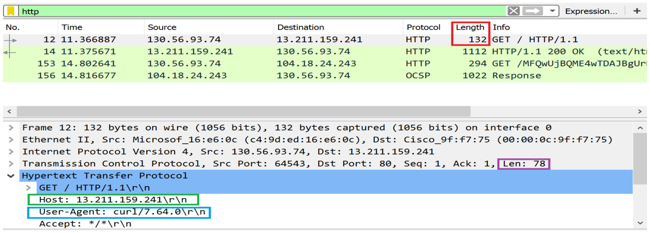
No.	Time	Source	Destination	Protocol	Length	Info
12	11.366887	130.56.93.74	13.211.159.241	HTTP	132	GET / HTTP/1.1
14	11.375671	13.211.159.241	130.56.93.74	HTTP	1112	HTTP/1.1 200 OK (text/ht
153	14.802641	130.56.93.74	104.18.24.243	HTTP	294	GET /MFQwUjBQME4wTDAJBgUr
156	14.816677	104.18.24.243	130.56.93.74	OCSP	1022	Response

```
> Frame 14: 1112 bytes on wire (8896 bits), 563 bytes captured (4504 bits) on interface 0
> Ethernet II, Src: Cisco_9d:f4:42 (8c:60:4f:9d:f4:42), Dst: Microsof_16:e6:0c (c4:9d:ed:16:e6:0c)
> Internet Protocol Version 4, Src: 13.211.159.241, Dst: 130.56.93.74
> Transmission Control Protocol, Src Port: 80, Dst Port: 64543, Seq: 1, Ack: 79, Len: 509
> Hypertext Transfer Protocol
> Line-based text data: text/html (12 lines)
```

(b) Response Message

Figure 4: Capture Result

The content of green box within Figure 4a and 4b illustrate the format of the captured message. Each line within the box corresponding to one protocol head, e.g. The line start with Transmission Control Protocol correspond to TCP head. Particularly, Figure 4a indicate the format of HTTP Request message and Figure 4b indicate the HTTP Response message. Indeed, these result can be used as a good example to illustrate my discussion of Layer-Architecture in Question 1. I will have a formal discussion of this example in Question 5. In this question, I will use the capture result from Wireshark to inspect some specific fields within protocol head. I will first start with Request message, as illustrated in Figure 5:



No.	Time	Source	Destination	Protocol	Length	Info
12	11.366887	130.56.93.74	13.211.159.241	HTTP	132	GET / HTTP/1.1
14	11.375671	13.211.159.241	130.56.93.74	HTTP	1112	HTTP/1.1 200 OK (text/ht
153	14.802641	130.56.93.74	104.18.24.243	HTTP	294	GET /MFQwUjBQME4wTDAJBgUr
156	14.816677	104.18.24.243	130.56.93.74	OCSP	1022	Response

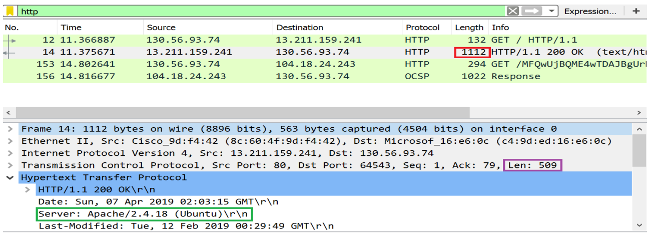
```
> Frame 12: 132 bytes on wire (1056 bits), 132 bytes captured (1056 bits) on interface 0
> Ethernet II, Src: Microsof_16:e6:0c (c4:9d:ed:16:e6:0c), Dst: Cisco_9f:f7:75 (00:00:0c:9f:f7:75)
> Internet Protocol Version 4, Src: 130.56.93.74, Dst: 13.211.159.241
> Transmission Control Protocol, Src Port: 64543, Dst Port: 80, Seq: 1, Ack: 1, Len: 78
v Hypertext Transfer Protocol
   > GET / HTTP/1.1\r\n
     Host: 13.211.159.241\r\n
     User-Agent: curl/7.64.0\r\n
     Accept: */*\r\n
```

Figure 5: Detail of Request Message

In which, we can get the following information:

- **The User-Agent Field in HTTP Head**: cURL 7.64.0, which indicate that this request message was sent by application cURL with version 7.64.0 and is shown within the **blue box**.
- **Host Field in HTTP Head**: 13.211.159.241, which is shown within **green box**. This IP address is server's IP address.
- **Length of Request Message**: 132 bytes, which is shown within **red box**. Notice that this length count in all protocol head.
- **Length of TCP Head**: 78 bytes, which is shown within **purple box**.

After that, we can inspect the detail of Response message as shown in Figure 6.



No.	Time	Source	Destination	Protocol	Length	Info
12	11.366887	130.56.93.74	13.211.159.241	HTTP	132	GET / HTTP/1.1
14	11.375671	13.211.159.241	130.56.93.74	HTTP	1112	HTTP/1.1 200 OK (text/ht
153	14.802641	130.56.93.74	104.18.24.243	HTTP	294	GET /MFQwUjBQME4wTDAJBgUr
156	14.816677	104.18.24.243	130.56.93.74	OCSP	1022	Response

```
> Frame 14: 1112 bytes on wire (8896 bits), 563 bytes captured (4504 bits) on interface 0
> Ethernet II, Src: Cisco_9d:f4:42 (8c:60:4f:9d:f4:42), Dst: Microsof_16:e6:0c (c4:9d:ed:16:e6:0c)
> Internet Protocol Version 4, Src: 13.211.159.241, Dst: 130.56.93.74
> Transmission Control Protocol, Src Port: 80, Dst Port: 64543, Seq: 1, Ack: 79, Len: 509
v Hypertext Transfer Protocol
   > HTTP/1.1 200 OK\r\n
     Date: Sun, 07 Apr 2019 02:03:15 GMT\r\n
     Server: Apache/2.4.18 (Ubuntu)\r\n
     Last-Modified: Tue, 12 Feb 2019 00:29:49 GMT\r\n
```

Figure 6: Detail of Response Message

The following information can be obtained:

- **Server Field in HTTP Head**: Apache/2.4.18, which indicate the response message is given by Apache application with version 2.4.18 and is shown within **green box**.
- **Length of Response Message**: 1112 bytes, shown within **red box**.
- **Length of TCP Head**: 509 bytes, shown within **purple box**.

Question 5

In Question 1, I have mentioned that each layer in Layer-Architecture will add a specific head to original message. Technical, this process is called encapsulation. In more specific:

> **Encapsulation** is the process that add a protocol-specific head to original message and produce a new one.

The Wireshark capture result from Question 4 provide us a concrete example that illustrate how encapsulation work. It can be seen from the capture result that in Transport Layer, it add a TCP head to the HTTP packet from Application Layer and produce a *Transport Layer segment*, which means, Transport Layer *encapsulate* HTTP packet into Transport Layer segment with a TCP head. This encapsulation can be demonstrated as:

$$\overbrace{\boxed{\text{TCP head}} + \underbrace{\boxed{\text{HTTP head}}}_{\text{HTTP packet}}}^{\text{Transport Layer segment}}$$

Furthermore, the Transport Layer segment will be sent to Network Layer and be encapsulated into *Network Layer packet* with IP head. The encapsulation repeat until the message reach the Physical Layer. In the meantime, final message has the format:

Ethernet head	IP head	TCP head	HTTP head

The advantages of using encapsulation is that it separate the function of different Layer, which make the Layer-Architecture easier to maintain and develop.

By the end of this assignment, I have draw a big picture of Computer Network by giving a general description of Layer-Architecture and use some concrete example to illustrate some important concepts like encapsulation, which is the core part of Layer-Architecture. Additionally, the format of some specific protocol head also be inspected by using tools like Wireshark and cURL.

References

[Fielding et al., 1999] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T. (1999). Hypertext transfer protocol – http/1.1. RFC 2616, RFC Editor.