

**Birla Institute of Technology & Science, Pilani**  
**Computer Networks (CS F303 / IS F303)**  
**Second Semester 2015-2016**

**Laboratory Session- 2**

**Aim:** Explaining HTTP Traffic using Wireshark.

**Objective:**

- a. To learn capturing live packets using a network protocol analysis tool i.e Wireshark.
- b. Analysing HTTP traffic using Wireshark.

**Required Resources:**

- A PC with Wireshark
  - Proper LAN or WAN to capture the packets
- 

**Network Traffic Analyzer: Wireshark**

**Description:**

Wireshark is an open-source and foremost network protocol analyzer. It lets you see what's happening on your network at a microscopic level. It is used for network analysis, troubleshooting and to assist communications protocol development and education. Wireshark does not manipulate packets on the network, but can only analyze those already present, with minimal overhead.

**Wireshark has a rich feature set which includes the following:**

- i. Deep inspection of hundreds of protocols, with more being added all the time
- ii. Live capture and offline analysis
- iii. Standard three-pane packet browser
- iv. Multi-platform: Runs on Windows, Linux, OS X, Solaris and many others
- v. Captured network data can be browsed via a GUI
- vi. The most powerful display filters.
- vii. Capture files compressed with gzip can be decompressed on the fly
- viii. Live data can be read from Ethernet, IEEE 802.11, PPP/HDLC, ATM, Bluetooth, USB, Token Ring, Frame Relay, FDDI, and others (depending on your platform)
- ix. Colouring rules can be applied to the packet list for quick, intuitive analysis

**Important Links and References (Further reading):**

<http://is.gd/RazB76>)

<http://www.wireshark.org/about.html>

## Installing Wireshark:

For Windows OS (Windows )	For Linux OS (Ubuntu)
<ul style="list-style-type: none"><li>Download the latest stable version of Wireshark. (Available at <a href="http://is.gd/Ys7UeV">http://is.gd/Ys7UeV</a>)</li><li>Choose all components for installation, including WinPcap.</li><li>Proceed until completion. Wireshark may now be launched by running the application launcher.</li></ul>	<ul style="list-style-type: none"><li>Use Ubuntu Software Center to install latest version of Wireshark.</li><li>Wireshark has to be run with root privileges, so that it has the required permissions to monitor the network interfaces. To do so, type in the following command in the terminal “sudo wireshark” (without quotes).</li></ul>

## Procedure (How to use Wireshark):

1. Start Wireshark by starting the executable from the installed directory.
2. Select proper interface for capturing packets (See Figure-1).
3. You will see a dynamic list of packets being captured by Wireshark. In order to stop a running capture, press CTRL+E or from the menu, select Capture → Stop (See Figure-2).
4. Various packets may be filtered. For instance, if you would only like to see HTTP packets enter HTTP in the Filter input-box and press Apply (See Figure-3 & 4).

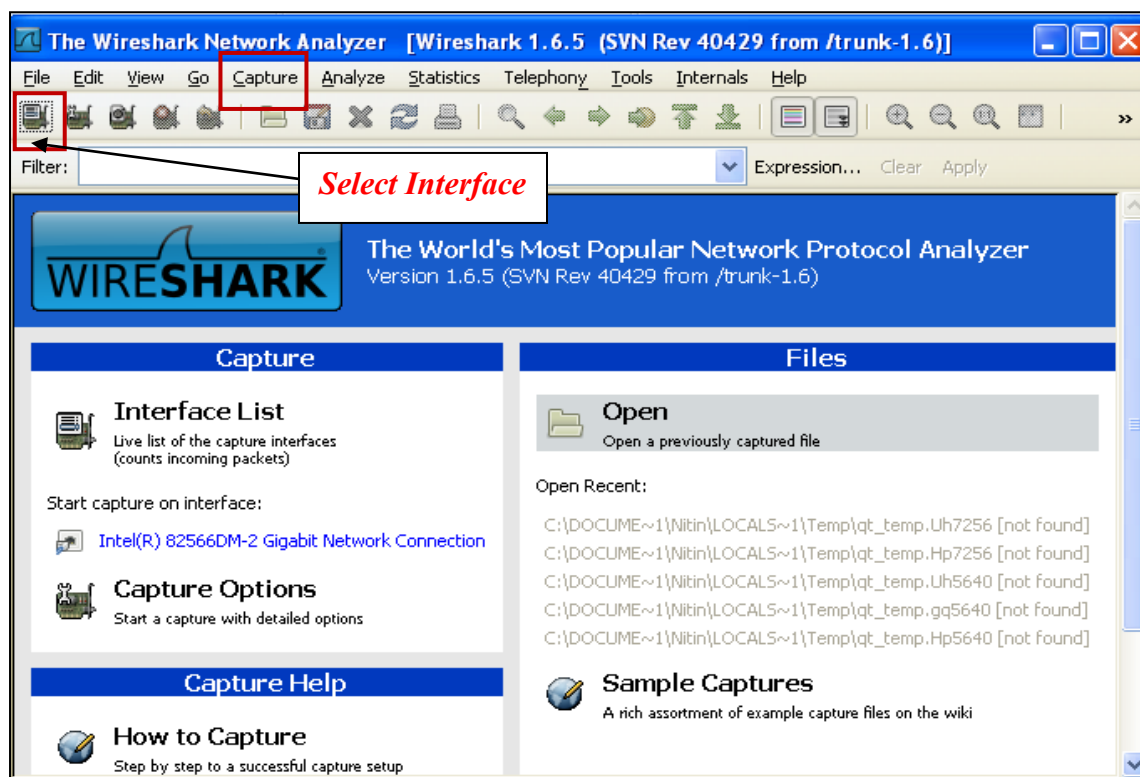


Figure 1

Intel(R) 82566DM-2 Gigabit Network Connection [Wireshark 1.6.5 (SVN Rev 40429 from /trunk-1.6)]

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

Filter: Expression... Clear Apply

No.	Time	Source	Destination	Protocol	Length	Info
969	27.499910	192.168.10.100	107.22.163.94	TCP	54	afs > https [ACK] Seq=414 Ack=2352 win=65285
970	27.499915	192.168.10.100	31.13.72.81	TCP	54	insitu-conf > https [ACK] Seq=1280 Ack=768 w
971	27.500085	173.194.36.111	192.168.10.100	TLSv1	87	Application Data
972	27.584920	103.4.253.46	192.168.10.100	HTTP	889	HTTP/1.1 200 OK
973	27.585229	103.4.253.46	192.168.10.100	TCP	60	http > valisys-lm [FIN, ACK] Seq=836 Ack=265
974	27.585254	192.168.10.100	103.4.253.46	TCP	54	valisys-lm > http [ACK] Seq=2651 Ack=837 win
975	27.585739	192.168.10.100	103.4.253.46	TCP	54	valisys-lm > http [FIN, ACK] Seq=2651 Ack=83
976	27.586075	103.4.253.46	192.168.10.100	TCP	60	http > valisys-lm [ACK] Seq=837 Ack=2652 win
977	27.702038	192.168.10.100	173.194.36.111	TCP	54	imtc-mcs > https [ACK] Seq=1026 Ack=452 win=
978	27.718082	199.59.150.10	192.168.10.100	TLSv1	99	Application Data
979	27.902225	192.168.10.100	199.59.150.10	TCP	54	liberty-lm > https [ACK] Seq=1450 Ack=226 wi
980	27.902465	199.59.150.10	192.168.10.100	TLSv1	458	Application Data, Application Data, Applicat
981	27.944648	103.246.148.128	192.168.10.100	TCP	1414	[TCP segment of a reassembled PDU]
982	28.103396	192.168.10.100	103.246.148.128	TCP	54	taligent-lm > http [ACK] Seq=462 Ack=1361 wi
983	28.103408	192.168.10.100	199.59.150.10	TCP	54	liberty-lm > https [ACK] Seq=1450 Ack=630 wi
984	28.103577	103.246.148.128	192.168.10.100	HTTP	172	HTTP/1.1 200 OK (application/x-javascript)
985	28.195894	192.241.245.89	192.168.10.100	HTTP	354	HTTP/1.1 302 Found
986	28.197917	192.168.10.100	103.243.35.50	TCP	62	utcd > http [SYN] Seq=0 win=65535 Len=0 MSS=
987	28.198733	103.243.35.50	192.168.10.100	TCP	62	http > utcd [SYN, ACK] Seq=0 Ack=1 win=5840
988	28.198751	192.168.10.100	103.243.35.50	TCP	54	utcd > http [ACK] Seq=1 Ack=1 win=65535 Len=
989	28.199140	192.168.10.100	103.243.35.50	HTTP	571	GET /foo.gif HTTP/1.1

Figure 2

Intel(R) 82566DM-2 Gigabit Network Connection [Wireshark 1.6.5 (SVN Rev 40429 from /trunk-1.6)]

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

**Select Filter**

Filter: Expression... Clear Apply

**Wireshark: Display Filter - Profile: Default**

Edit

Display Filter

- ethernet broadcast
- No ARP
- IP only
- IP address 192.168.0.1
- IP address isn't 192.168.0.1, don't use != for this!
- IPX only
- TCP only**
- UDP only
- UDP port isn't 53 (not DNS), don't use != for this!

Properties

Filter name: TCP only

Filter string: tcp Expression...

Help OK Apply Cancel

No.	Time	Source	Destination	Protocol	Length	Info
969	27.499910	192.168.10.100	107.22.163.94	TCP	54	afs > https [ACK] seq=414 Ack
970	27.499915	192.168.10.100	31.13.72.81	TCP	54	insitu-conf > https [ACK] seq
971	27.500085	173.194.36.111	192.168.10.100	TLSv1	87	Application Data
972	27.584920	103.4.253.46	192.168.10.100	HTTP	889	HTTP/1.1 200 OK
973	27.585229	103.4.253.46	192.168.10.100	TCP	60	http > valisys-lm [FIN, ACK]
974	27.585254	192.168.10.100	103.4.253.46	TCP	54	valisys-lm > http [ACK] Seq=2
975	27.585739	192.168.10.100	103.4.253.46	TCP	54	valisys-lm > http [FIN, ACK]
976	27.586075	103.4.253.46	192.168.10.100	TCP	60	http > valisys-lm [ACK] Seq=8
977	27.702038	192.168.10.100	173.194.36.111	TCP	54	imtc-mcs > https [ACK] seq=10
978	27.718082	199.59.150.10	192.168.10.100	TLSv1	99	Application Data
979	27.902225	192.168.10.100	199.59.150.10	TCP	54	liberty-lm > https [ACK] seq=
980	27.902465	199.59.150.10	192.168.10.100	TLSv1	458	Application Data, Application
981	27.944648	103.246.148.128	192.168.10.100	TCP	1414	[TCP segment of a reassembled
982	28.103396	192.168.10.100	103.246.148.128	TCP	54	taligent-lm > http [ACK] Seq=
983	28.103408	192.168.10.100	199.59.150.10	TCP	54	liberty-lm > https [ACK] Seq=
984	28.103577	103.246.148.128	192.168.10.100	HTTP	172	HTTP/1.1 200 OK (application
985	28.195894	192.241.245.89	192.168.10.100	HTTP	354	HTTP/1.1 302 Found
986	28.197917	192.168.10.100	103.243.35.50	TCP	62	utcd > http [SYN] Seq=0 win=6
987	28.198733	103.243.35.50	192.168.10.100	TCP	62	http > utcd [SYN, ACK] Seq=0
988	28.198751	192.168.10.100	103.243.35.50	TCP	54	utcd > http [ACK] Seq=1 Ack=1
989	28.199140	192.168.10.100	103.243.35.50	HTTP	571	GET /foo.gif HTTP/1.1
990	28.199140	192.168.10.100	103.243.35.50	TCP	60	http > utcd [ACK] Seq=1 Ack=5
991	28.199140	192.168.10.100	103.243.35.50	TCP	54	taligent-lm > http [ACK] Seq=
992	28.199140	192.168.10.100	103.243.35.50	HTTP	278	HTTP/1.1 304 Not Modified
993	28.199140	192.168.10.100	103.243.35.50	TCP	62	symplex > http [SYN] Seq=0 wi

Figure 3

# **Hyper Text Transfer Protocol (HTTP)**

## **Brief Introduction**

The Hyper Text Transport Protocol is a text-based request-response client-server protocol. A HTTP client (e.g. a web browser such as Mozilla) performs a HTTP request to a HTTP server (e.g. the Apache HTTP server), which in return will issue a HTTP response. The HTTP protocol header is text-based, where headers are written in text lines.

HTTP/1.1 allows for client-server connections to be pipelined, whereby multiple requests can be sent (often in the same packet), without waiting for a response from the server. The only restriction is the server **MUST** return the responses in the same order as they were received. This enables greater efficiency, especially on revalidation.

An encrypted variant named HTTPS is also available. This is often used where privacy of data is necessary, e.g. when using online banking. The HTTPS protocol is in fact two protocols running on top of each other. The first protocol is a security protocol like SSL, TLS or PCT. The second protocol, which runs on top of this security protocol, is HTTP. The URLs starting with https:// really are only a shorthand notation for the end user. The web browser will read the URI scheme (https://), initiate the security protocol to the server, and once this secure connection is established, issue a HTTP request over it with the URI specified in the request.

## **History**

The Hyper Text Transfer Protocol (HTTP) was initiated at the CERN in Geneve (Switzerland), where it emerged (together with the HTML presentation language) from the need to exchange scientific information on a computer network in a simple manner. The first public HTTP implementation only allowed for plain text information, and almost instantaneously became a replacement of the GOPHER service. One of the first text-based browsers was LYNX which still exists today; a graphical HTTP client appeared very quickly with the name NCSA Mosaic. Mosaic was a popular browser back in 1994. Soon the need for a more rich multimedia experience was born, and the markup language provided support for a growing multitude of media types.

## **Protocol Dependencies**

- **MIME\_multipart:** HTTP uses MIME\_multipart to encode its messages.
- **TCP:** Typically, HTTP uses TCP as its transport protocol. The well known TCP port for HTTP traffic is 80. A HTTP proxy often uses a different port; typical values are 81, 3128, 8000 and 8080. However, HTTP can use other transport protocols as well.

## Capturing HTTP Protocol Packets

Here are the steps for capturing and analysing FTP Protocol:

### Step: 1 Start a Wireshark capture

- Close all unnecessary network traffic, such as the web browser, to limit the amount traffic during the Wireshark capture (not necessarily).
- Start the Wireshark capture.

Step: 2 **Start HTTP a Session:** By opening a website using your internet browser.

Step 3: **Stop the Wireshark capture (after some time.).**

Step 4: **View the Wireshark Main Window.**

- Wireshark captured many packets during the HTTP session
- You can see the various protocols working underneath HTTP in Wirshark (ie TCP, IP etc).  
OR go to **Statistics → Protocol Hierarchy** to find the same.

Step 5: **Select a HTTP Stream in Wireshark Main Window and analyse it. (See the section given below)**

## Analyzing HTTP Protocol

Consider following figure for capturing HTTP protocol packets using filter (circled red in the Figure- 4)

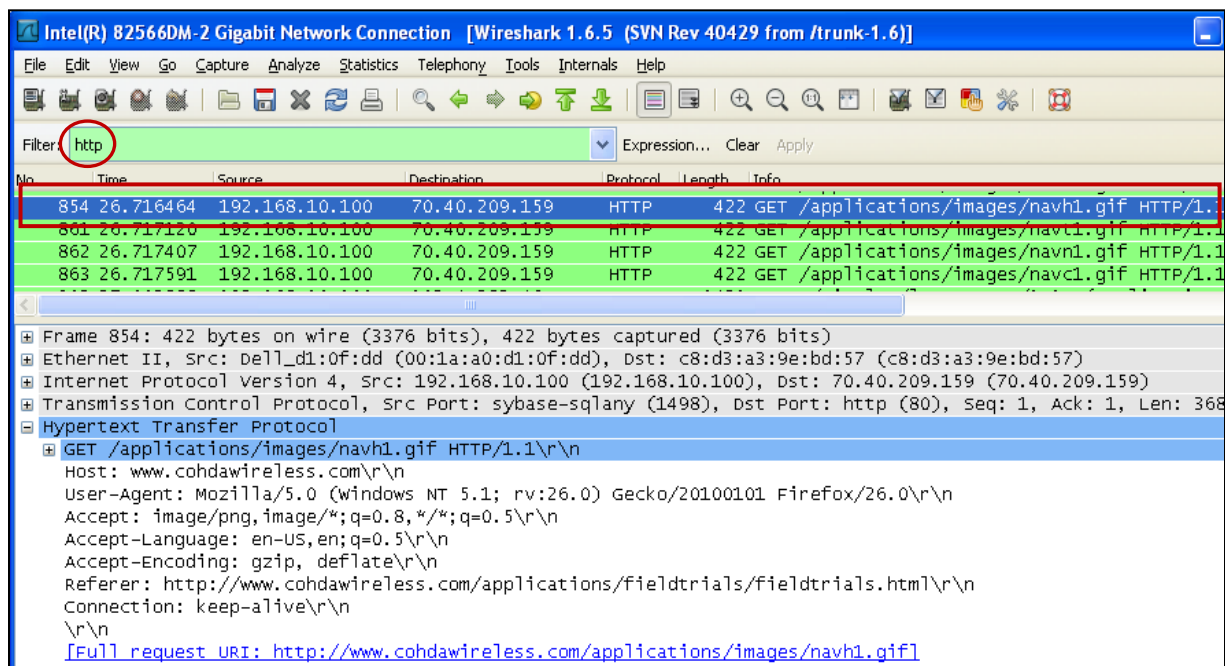


Figure 4

## Example Traffic

### Request by an end-users browser

This user wants to access the web site "www.freebsd.org", so they type in `http://http://wiki.wireshark.org/Hyper_Text_Transfer_Protocol` into their browser and hit enter.

After the usual DNS resolution to find the IP address for:

`http://wiki.wireshark.org/Hyper_Text_Transfer_Protocol.org`, a connection is initiated via TCP to the web server (SYN; SYN, ACK; ACK). The very next thing to be sent to the web server by the browser/client is the following plain text request:



Figure 5

You can see this request in Wireshark as shown in below figure:

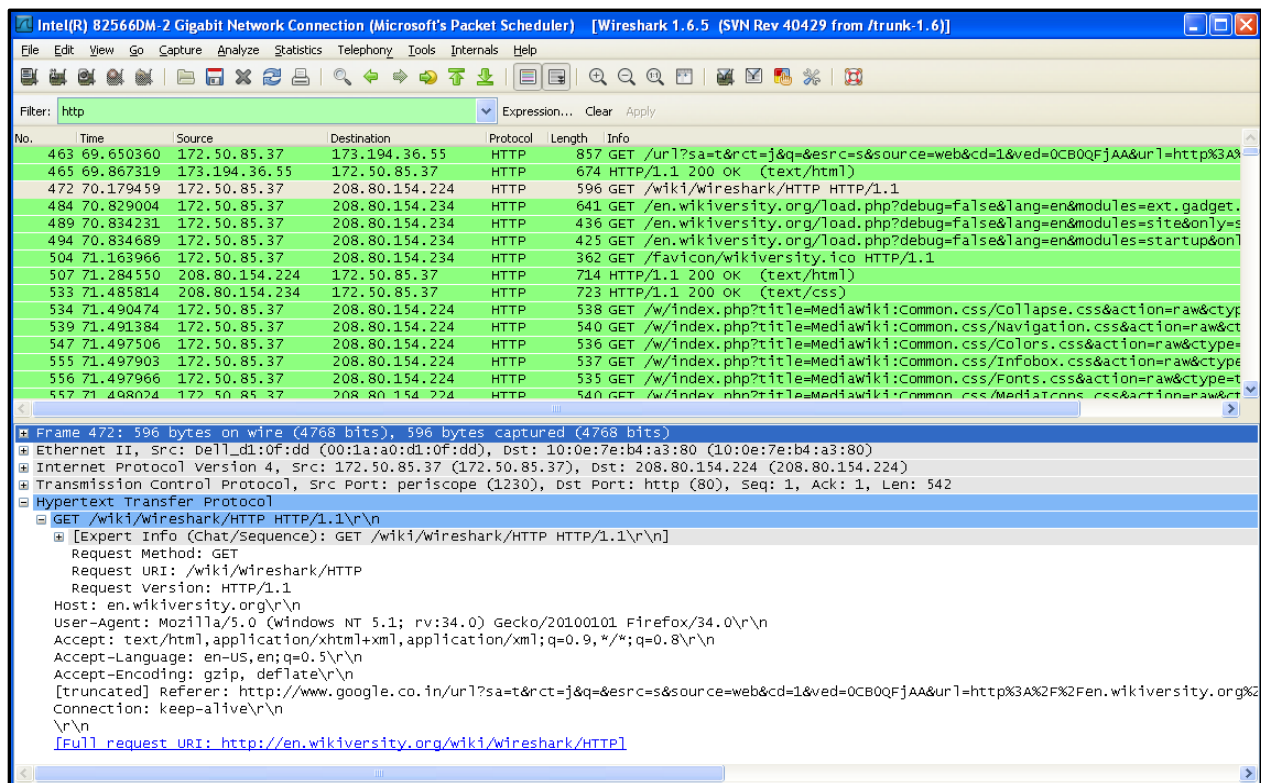
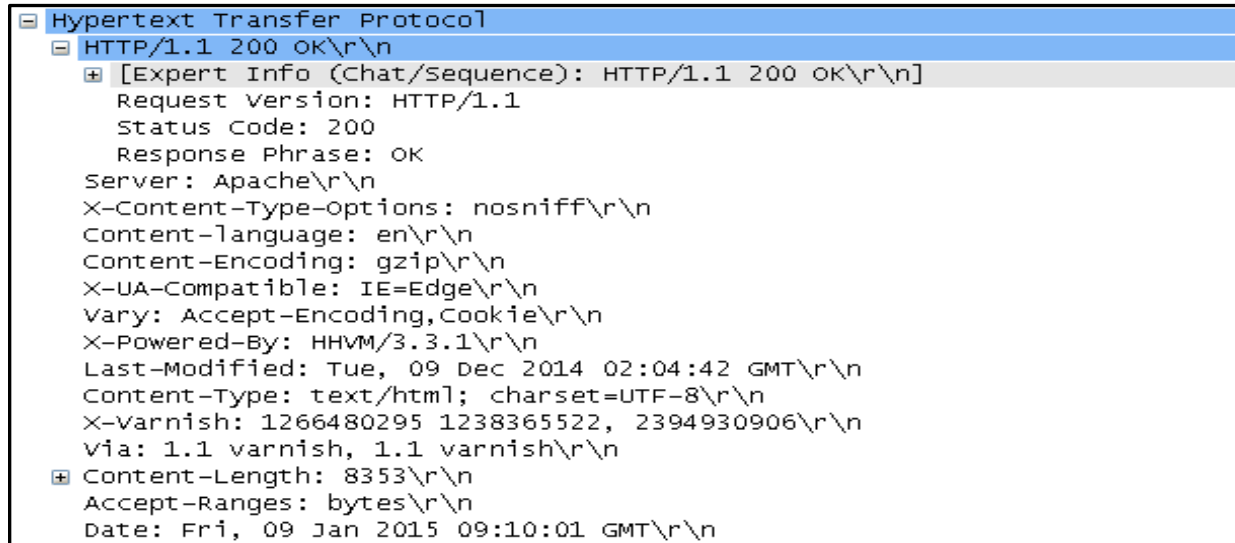


Figure 6

The server knows the browser/client is done with its traffic when it receives a blank line with a carriage return + line feed (\r\n).

## Response from the server

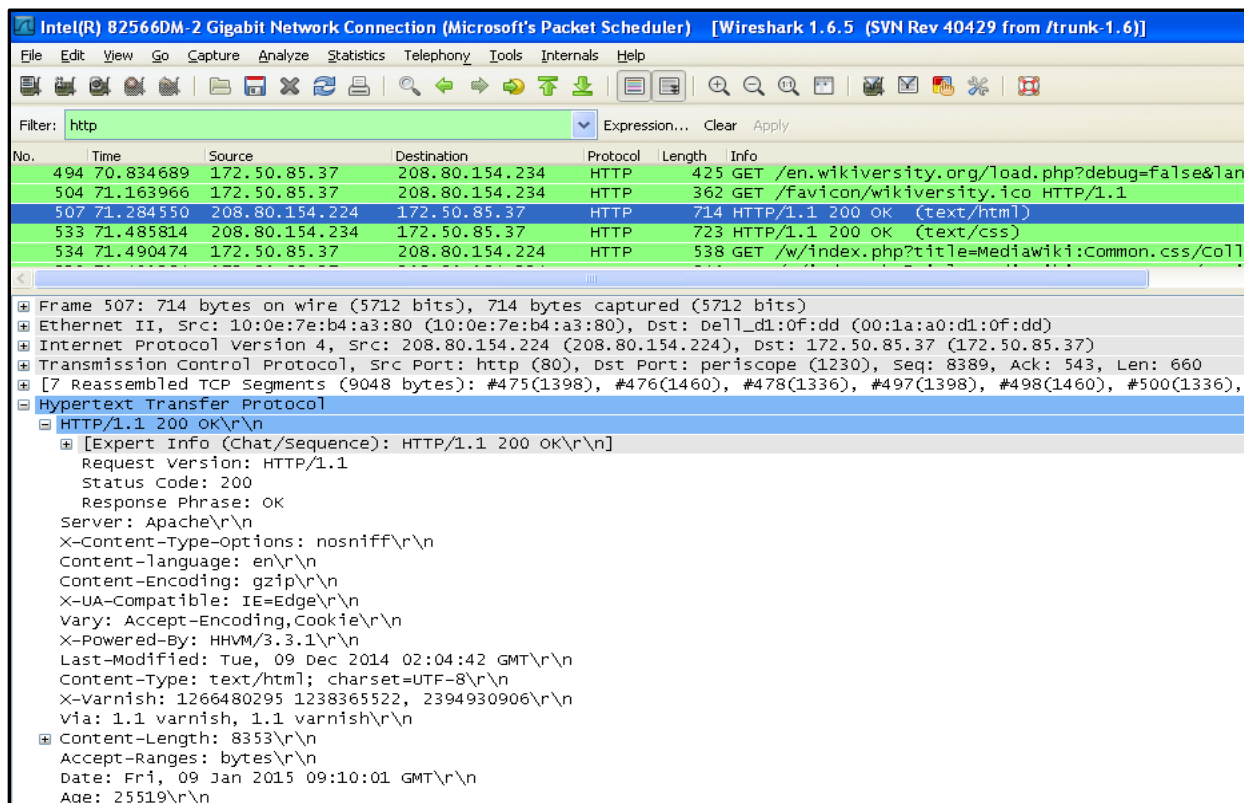
The response is also in plain text:



```
Hypertext Transfer Protocol
  HTTP/1.1 200 OK\r\n
    [Expert Info (Chat/Sequence): HTTP/1.1 200 OK\r\n]
      Request Version: HTTP/1.1
      Status Code: 200
      Response Phrase: OK
      Server: Apache\r\n
      X-Content-Type-Options: nosniff\r\n
      Content-Language: en\r\n
      Content-Encoding: gzip\r\n
      X-UA-Compatible: IE=Edge\r\n
      Vary: Accept-Encoding, Cookie\r\n
      X-Powered-By: HHVM/3.3.1\r\n
      Last-Modified: Tue, 09 Dec 2014 02:04:42 GMT\r\n
      Content-Type: text/html; charset=UTF-8\r\n
      X-varnish: 1266480295 1238365522, 2394930906\r\n
      via: 1.1 varnish, 1.1 varnish\r\n
    Content-Length: 8353\r\n
    Accept-Ranges: bytes\r\n
    Date: Fri, 09 Jan 2015 09:10:01 GMT\r\n
```

Figure 7

You can see this response in Wireshark as shown in below figure: Thus client and server start communication.



**Inference:**

- Learned, how to use Wireshark as network analyser.
- Learned, capturing and working (Client- Server Request/Response) HTTP protocol.
- Learned, analysis of HTTP traffic.

**Note:**

- a. Further Analysis of various network protocols will be done in future laboratory sessions.*
- b. Do it yourself: Configuration of HTTP Server.*