# Transport service requirements: common apps

| application | data loss | throughput | time sensitive? |
|---|---|---|---|
| file transfer/download | no loss | elastic | no |
| e-mail | no loss | elastic | no |
| Web documents | no loss | elastic | no |
| real-time audio/video | loss-tolerant | audio: 5Kbps-1Mbps<br>video:10Kbps-5Mbps | yes, 10’s msec |
| streaming audio/video | loss-tolerant | same as above | yes, few secs |
| interactive games | loss-tolerant | Kbps+ | yes, 10’s msec |
| text messaging | no loss | elastic | yes and no |

# Internet transport protocols services

## *TCP service:*

- *reliable transport* between sending and receiving process
- *flow control:* sender won't overwhelm receiver
- *congestion control:* throttle sender when network overloaded
- *does not provide:* timing, minimum throughput guarantee, security
- *connection-oriented:* setup required between client and server processes

## *UDP service:*

- *unreliable data transfer* between sending and receiving process
- *does not provide:* reliability, flow control, congestion control, timing, throughput guarantee, security, or connection setup.

Q: why bother? *Why* is there a UDP?

# Internet transport protocols services

| application | application layer protocol | transport protocol |
|---|---|---|
| file transfer/download | FTP [RFC 959] | TCP |
| e-mail | SMTP [RFC 5321] | TCP |
| Web documents | HTTP 1.1 [RFC 7320] | TCP |
| Internet telephony | SIP [RFC 3261], RTP [RFC 3550], or proprietary | TCP or UDP |
| streaming audio/video | HTTP [RFC 7320], DASH | TCP |
| interactive games | WOW, FPS (proprietary) | UDP or TCP |

# Application layer: overview

- Principles of network applications
- Web and HTTP
- E-mail, SMTP, IMAP
- The Domain Name System DNS

- P2P applications
- video streaming and content distribution networks
- socket programming with UDP and TCP

# Web and HTTP

*First, a quick review…*

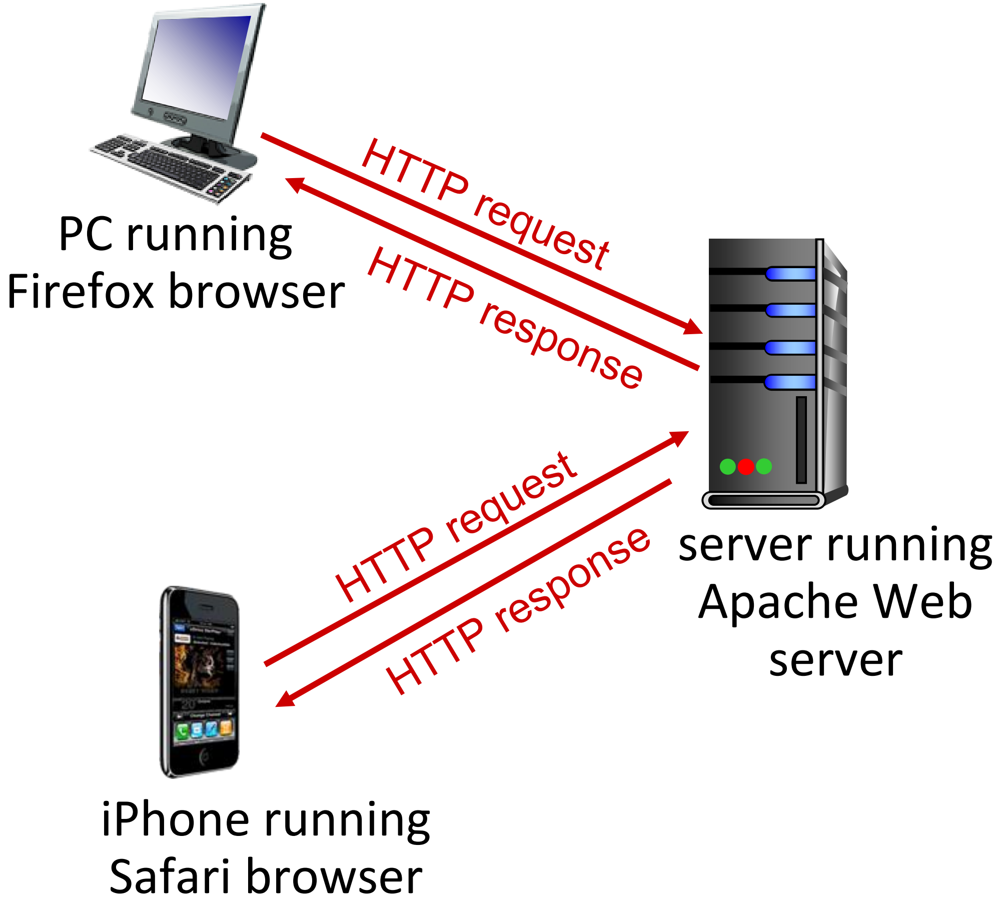
- web page consists of ***objects,*** each of which can be stored on different Web servers
- object can be HTML file, JPEG image, Java applet, audio file,…
- web page consists of *base HTML-file* which includes *several referenced objects, each* addressable by a *URL,* e.g.,

$$\underbrace{\texttt{www.someschool.edu}}_{\text{host name}}\underbrace{\texttt{/someDept/pic.gif}}_{\text{path name}}$$

# HTTP overview

**HTTP: hypertext transfer protocol**

- Web's application layer protocol
- client/server model:
  - *client:* browser that requests, receives, (using HTTP protocol) and "displays" Web objects
  - *server:* Web server sends (using HTTP protocol) objects in response to requests


PC running Firefox browser
HTTP request
HTTP response
server running Apache Web server
HTTP request
HTTP response
iPhone running Safari browser

# HTTP overview (continued)

*HTTP uses TCP:*

- client initiates TCP connection (creates socket) to server, port 80
- server accepts TCP connection from client
- HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)
- TCP connection closed

*HTTP is "stateless"*

- server maintains *no* information about past client requests

# HTTP connections: two types

*Non-persistent HTTP*

1. TCP connection opened
2. at most one object sent over TCP connection
3. TCP connection closed

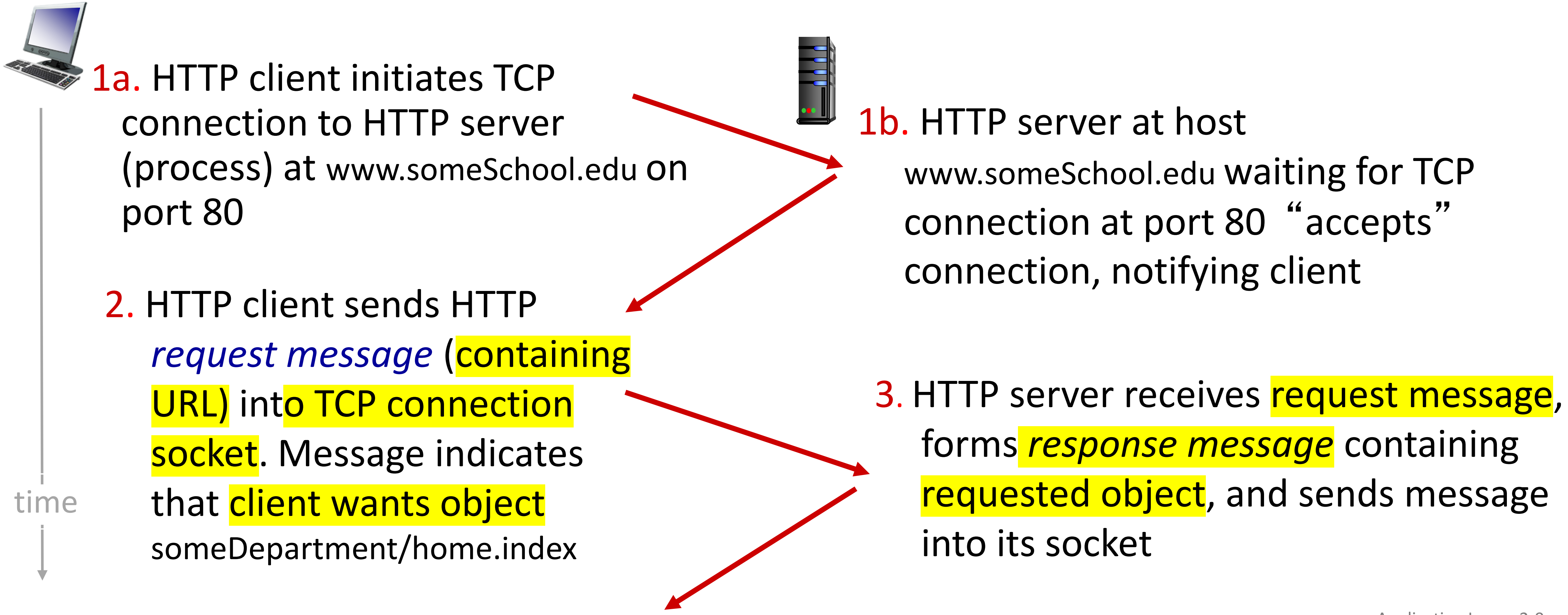
downloading multiple objects required multiple connections

*Persistent HTTP*

- TCP connection opened to a server
- multiple objects can be sent over *single* TCP connection between client, and that server
- TCP connection closed

# Non-persistent HTTP: example

User enters URL: **`www.someSchool.edu/someDepartment/home.index`**
(containing text, references to 10 jpeg images)



1a. HTTP client initiates TCP connection to HTTP server (process) at www.someSchool.edu on port 80

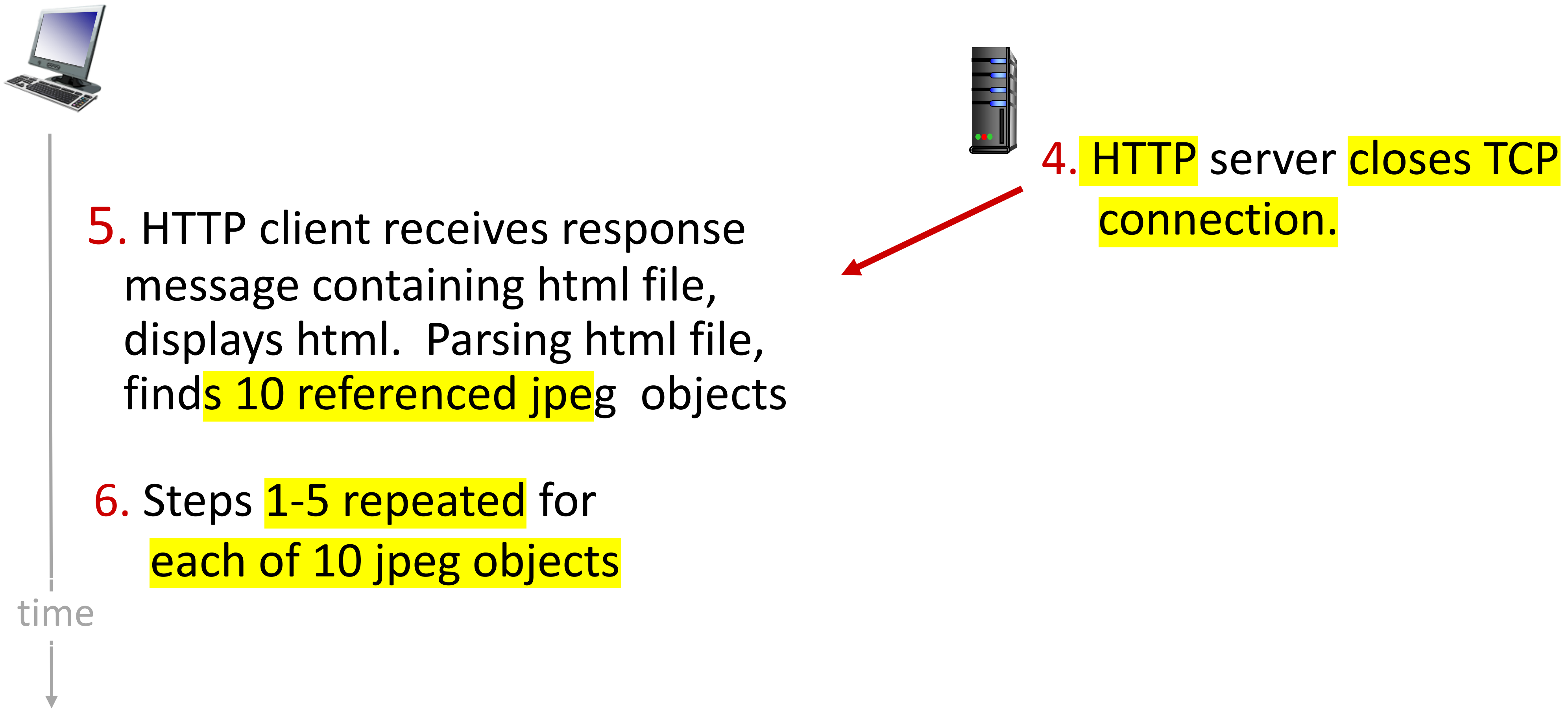
1b. HTTP server at host www.someSchool.edu waiting for TCP connection at port 80 "accepts" connection, notifying client

2. HTTP client sends HTTP *request message* (containing URL) into TCP connection socket. Message indicates that client wants object someDepartment/home.index

3. HTTP server receives request message, forms *response message* containing requested object, and sends message into its socket

# Non-persistent HTTP: example (cont.)

User enters URL: **`www.someSchool.edu/someDepartment/home.index`**
(containing text, references to 10 jpeg images)



4. HTTP server closes TCP connection.

5. HTTP client receives response message containing html file, displays html. Parsing html file, finds 10 referenced jpeg objects

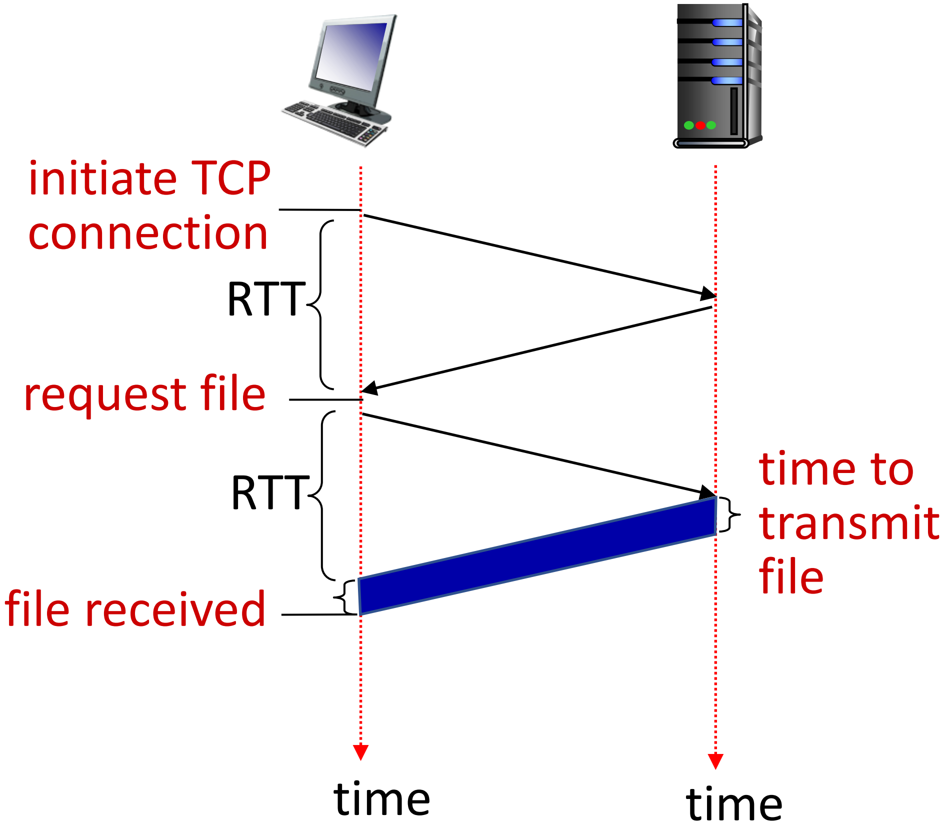
6. Steps 1-5 repeated for each of 10 jpeg objects

# Non-persistent HTTP: response time

RTT (definition): time for a small packet to travel from client to server and back

HTTP response time (per object):

- one RTT to initiate TCP connection
- one RTT for HTTP request and first few bytes of HTTP response to return
- obect/file transmission time



*Non-persistent HTTP response time = 2RTT+ file transmission time*

# Persistent HTTP (HTTP 1.1)

## *Non-persistent HTTP issues:*

- requires 2 RTTs per object
- OS overhead for *each* TCP connection
- browsers often open multiple parallel TCP connections to fetch referenced objects in parallel

## *Persistent HTTP (HTTP1.1):*

- server leaves connection open after sending response
- subsequent HTTP messages between same client/server sent over open connection
- client sends requests as soon as it encounters a referenced object
- as little as one RTT for all the referenced objects (cutting response time in half)

# HTTP request message

- two types of HTTP messages: *request, response*
- HTTP request message:
  - ASCII (human-readable format)

carriage return character

line-feed character

request line (GET, POST, HEAD commands)

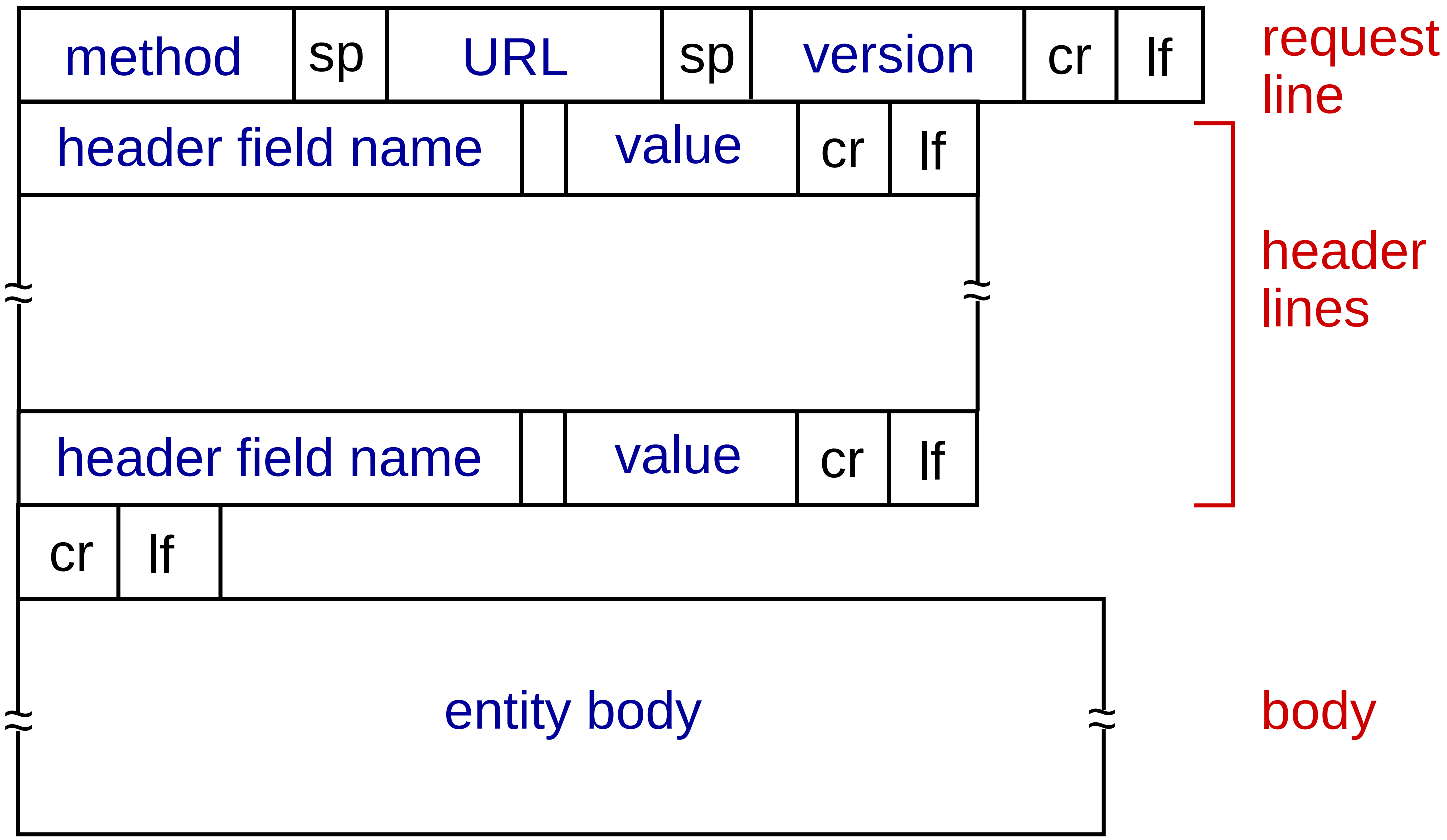
header lines

carriage return, line feed at start of line indicates end of header lines

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n
```

* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

# HTTP request message: general format



# Other HTTP request messages

## GET method: (Read)

- include user data in URL field of HTTP GET request message

## POST method: (Create)

- web page often includes form input
- user input sent from client to server in entity body of HTTP POST request message

```
www.somesite.com/animalsearch?monkeys&banana
```

## HEAD method: (only header)

- requests headers (only) that would be returned *if* specified URL were requested with an HTTP GET method.

## PUT method: (update)

- uploads new file (object) to server
- completely replaces file that exists at specified URL with content in entity body of POST HTTP request message

# HTTP response message

status line (protocol status code status phrase)

header lines

data, e.g., requested HTML file

```
HTTP/1.1 200 OK\r\n
Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Tue, 30 Oct 2007 17:00:02
   GMT\r\n
ETag: "17dc6-a5c-bf716880"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html; charset=ISO-8859-
   1\r\n
\r\n
data data data data data ...
```

* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

# HTTP response status codes

- status code appears in 1st line in server-to-client response message.
- some sample codes:

200 OK

- request succeeded, requested object later in this message

301 Moved Permanently

- requested object moved, new location specified later in this message (in Location: field)

400 Bad Request

- request msg not understood by server

404 Not Found

- requested document not found on this server

505 HTTP Version Not Supported

# Wireshark Basics & Demo

Anand Baswade

anand@iitbhilai.ac.in

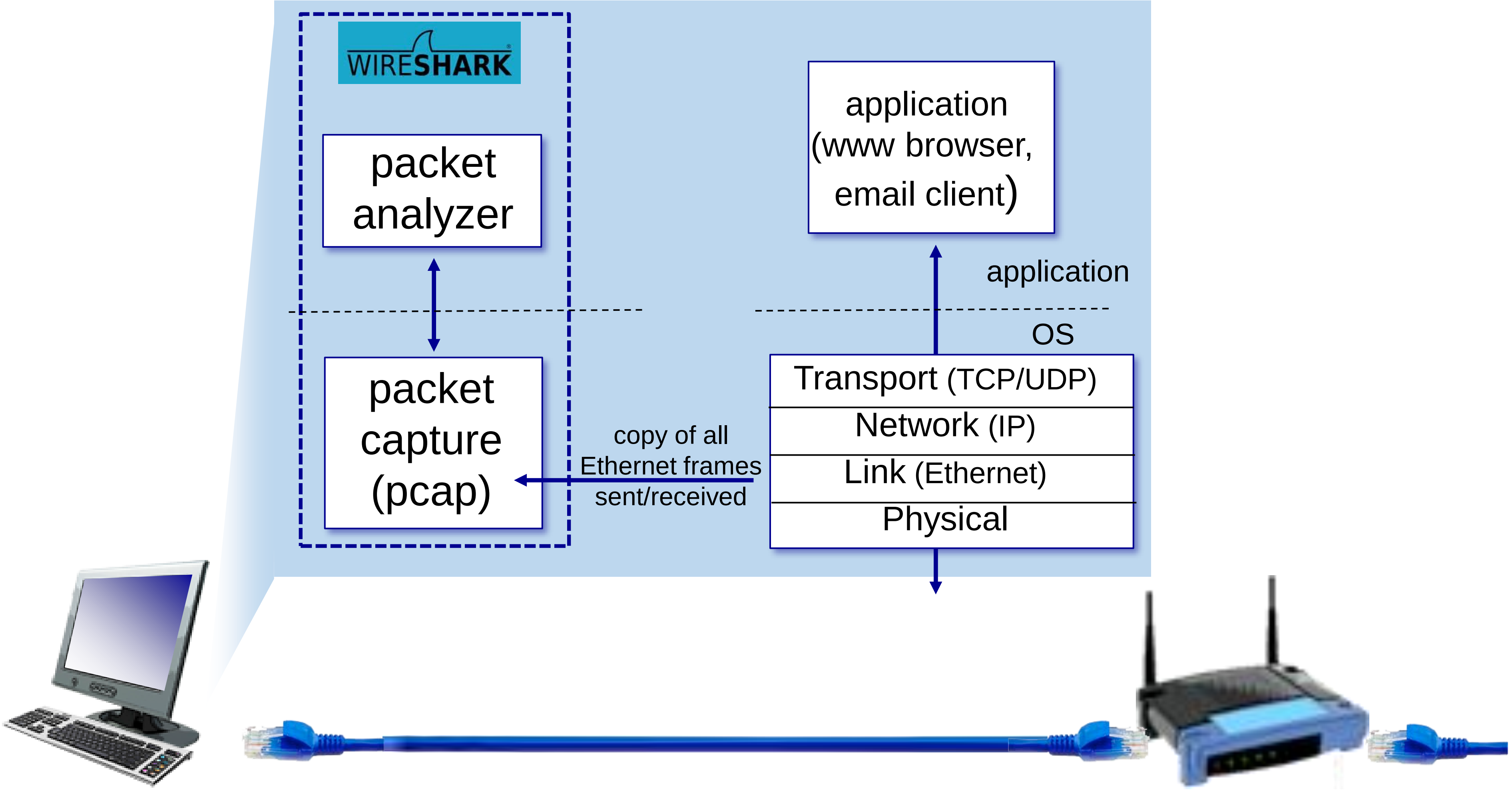
# Wireshark

- **Install Wireshark**
- For installing wireshark in Ubuntu system type in the following command in the terminal:

- **$ sudo apt-get install wireshark**

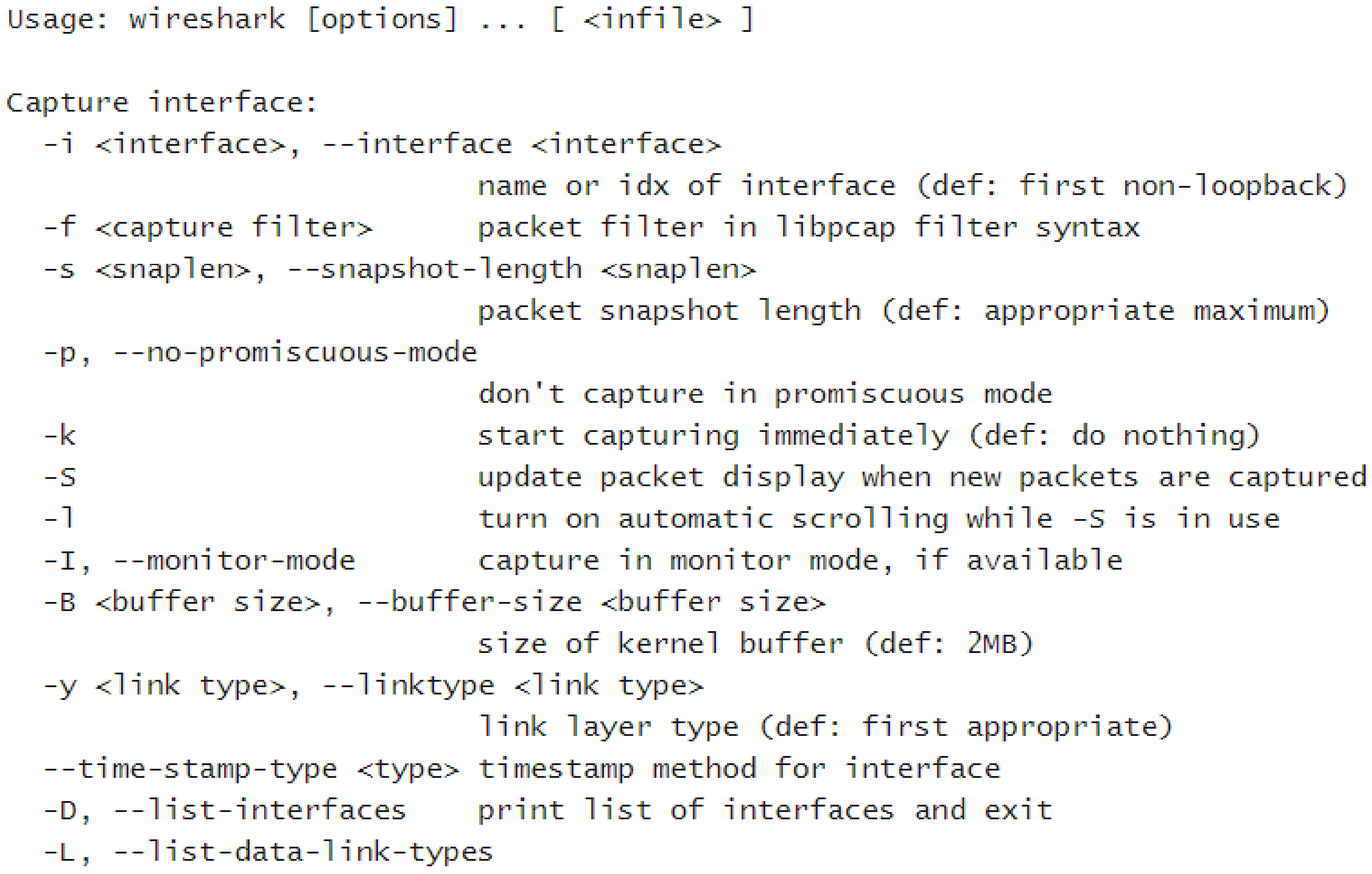
- For further details you can refer: (https://www.wireshark.org/download.html)

# How Wireshark works?



Source: Top-down approach slides: Introduction: 1-20

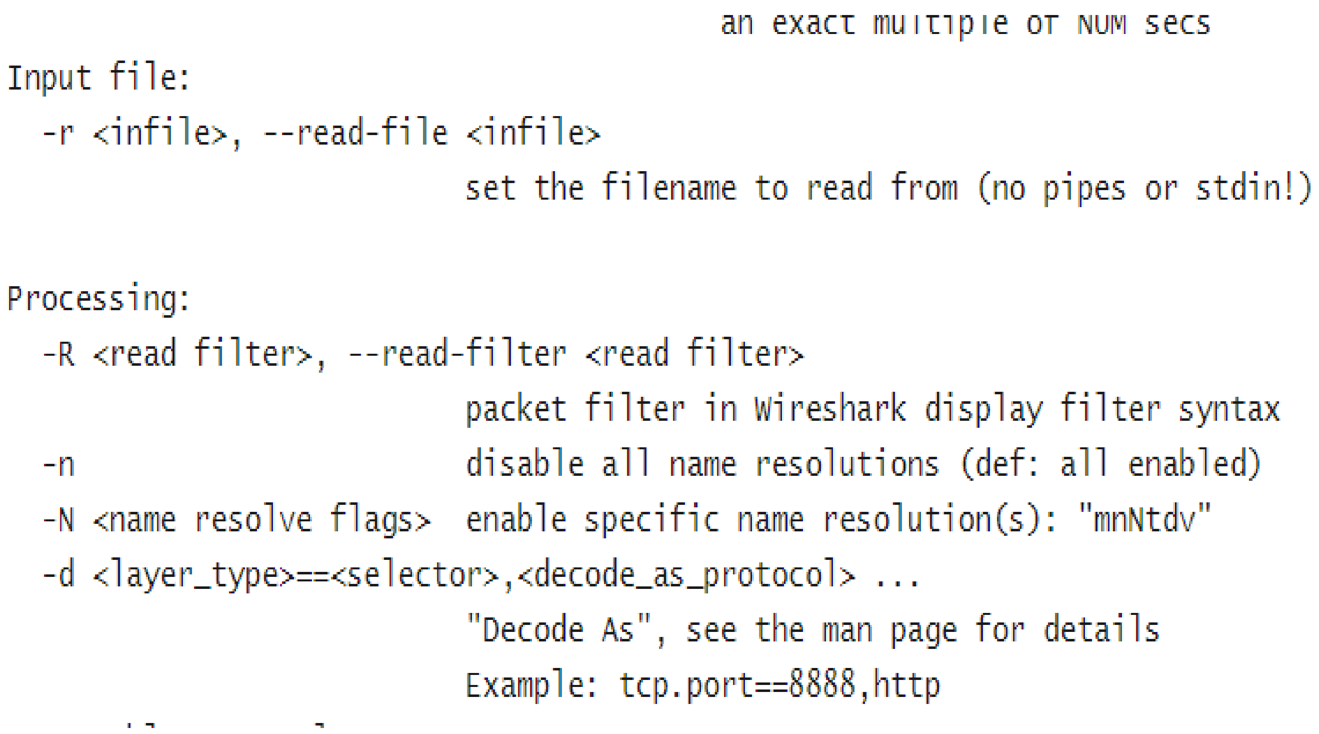
# Wireshark Cont..



```
Usage: wireshark [options] ... [ <infile> ]

Capture interface:
  -i <interface>, --interface <interface>
                           name or idx of interface (def: first non-loopback)
  -f <capture filter>      packet filter in libpcap filter syntax
  -s <snaplen>, --snapshot-length <snaplen>
                           packet snapshot length (def: appropriate maximum)
  -p, --no-promiscuous-mode
                           don't capture in promiscuous mode
  -k                       start capturing immediately (def: do nothing)
  -S                       update packet display when new packets are captured
  -l                       turn on automatic scrolling while -S is in use
  -I, --monitor-mode       capture in monitor mode, if available
  -B <buffer size>, --buffer-size <buffer size>
                           size of kernel buffer (def: 2MB)
  -y <link type>, --linktype <link type>
                           link layer type (def: first appropriate)
  --time-stamp-type <type> timestamp method for interface
  -D, --list-interfaces    print list of interfaces and exit
  -L, --list-data-link-types
```

# Cont..



```
                                          an exact multiple of NUM secs
Input file:
  -r <infile>, --read-file <infile>
                           set the filename to read from (no pipes or stdin!)

Processing:
  -R <read filter>, --read-filter <read filter>
                           packet filter in Wireshark display filter syntax
  -n                       disable all name resolutions (def: all enabled)
  -N <name resolve flags>  enable specific name resolution(s): "mnNtdv"
  -d <layer_type>==<selector>,<decode_as_protocol> ...
                           "Decode As", see the man page for details
                           Example: tcp.port==8888,http
```

# Wireshark Cont..

## To start capture

```
$ sudo wireshark -i wlan0    => wireless interface
$ sudo wireshark -i eth0       => wired interface
```

## To save file

```
$ sudo mkdir wireshark
$ cd wireshark
$ sudo wireshark -i wlan0 -w test
```

## Specify duration for capture

```
$ sudo wireshark -i wlan0 -w test -a duration:10                    (time in seconds)
```

## To read

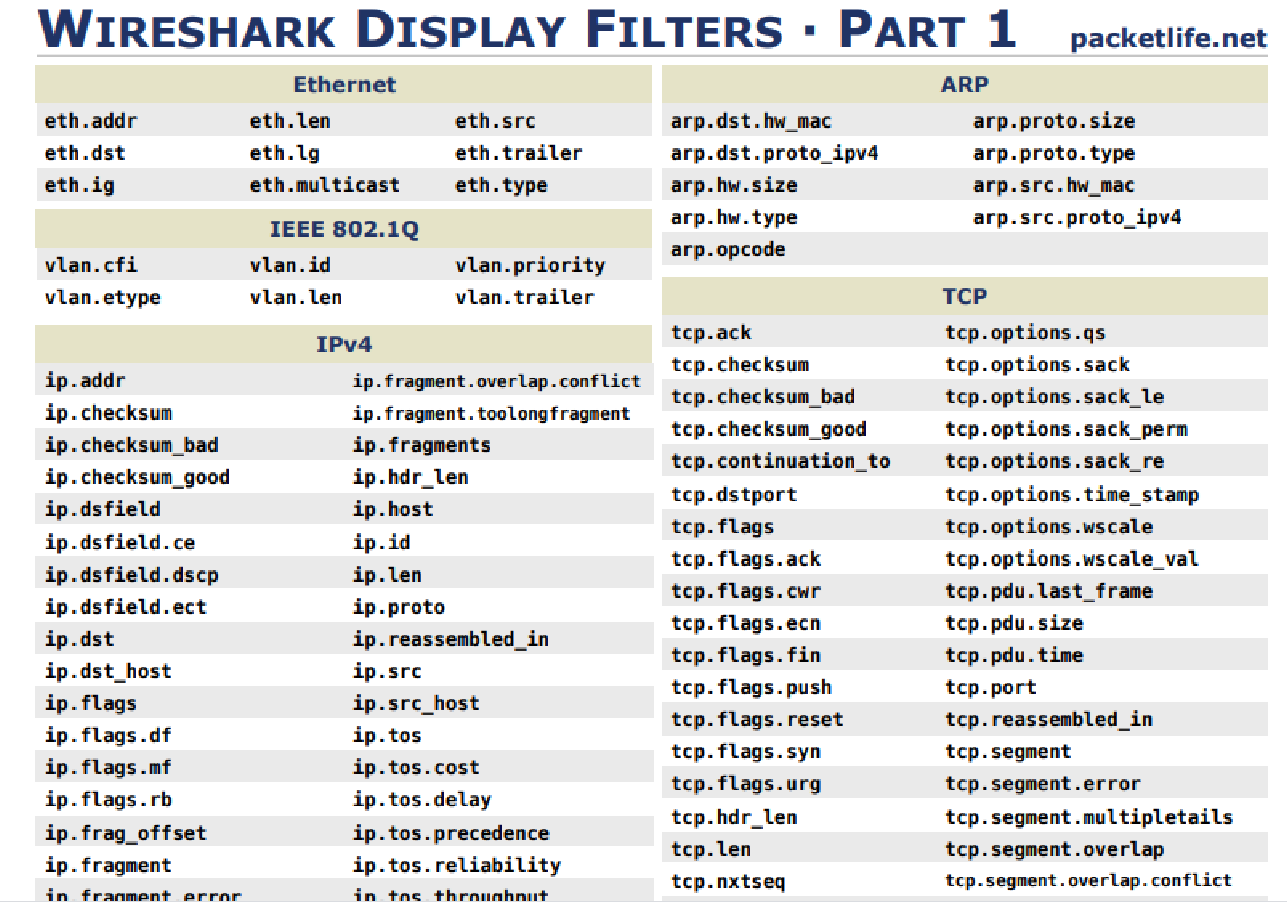
```
$ sudo wireshark -r test
```

# Cont..

- **For Filters**
  - $ sudo wireshark -r test -R "ip.src == 192.168.0.8"
  - $ sudo wireshark -r test -R "ip.src == 192.168.0.8 && ip.dst == 192.168.0.4"
  - $ sudo wireshark -r test -R "ip.src == 192.168.0.8 || ip.src == 192.168.0.4"

- Please find below all the list of filters
  - **Cheatsheet for filters:** http://packetlife.net/media/library/13/Wireshark_Display_Filters.pdf
  - **Cmd line help:** https://www.wireshark.org/docs/wsug_html_chunked/ChCustCommandLine.html

# Wireshark Display Filters



WIRESHARK DISPLAY FILTERS · PART 1 packetlife.net

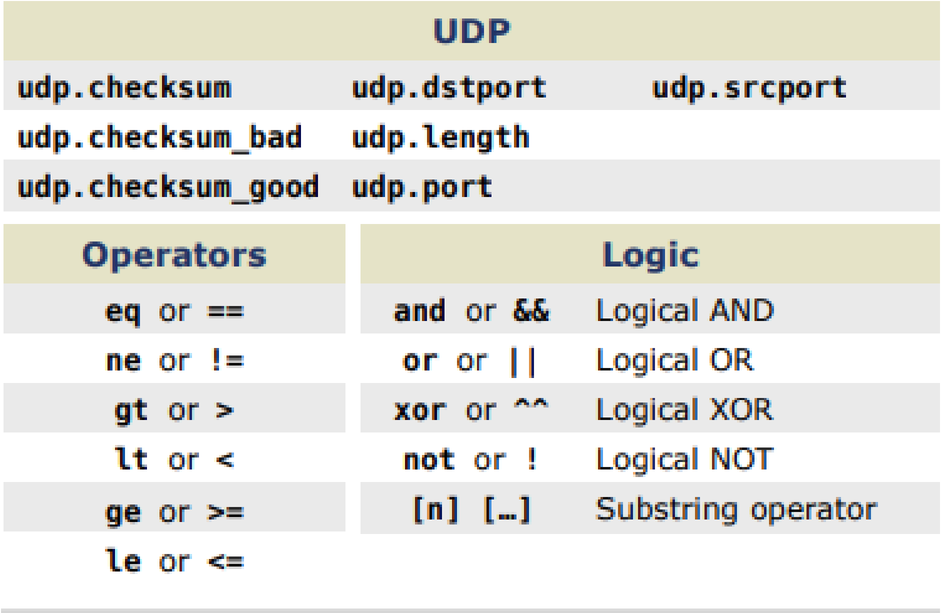
| Ethernet | | |
|---|---|---|
| eth.addr | eth.len | eth.src |
| eth.dst | eth.lg | eth.trailer |
| eth.ig | eth.multicast | eth.type |

| IEEE 802.1Q | | |
|---|---|---|
| vlan.cfi | vlan.id | vlan.priority |
| vlan.etype | vlan.len | vlan.trailer |

| IPv4 | |
|---|---|
| ip.addr | ip.fragment.overlap.conflict |
| ip.checksum | ip.fragment.toolongfragment |
| ip.checksum_bad | ip.fragments |
| ip.checksum_good | ip.hdr_len |
| ip.dsfield | ip.host |
| ip.dsfield.ce | ip.id |
| ip.dsfield.dscp | ip.len |
| ip.dsfield.ect | ip.proto |
| ip.dst | ip.reassembled_in |
| ip.dst_host | ip.src |
| ip.flags | ip.src_host |
| ip.flags.df | ip.tos |
| ip.flags.mf | ip.tos.cost |
| ip.flags.rb | ip.tos.delay |
| ip.frag_offset | ip.tos.precedence |
| ip.fragment | ip.tos.reliability |
| ip.fragment.error | ip.tos.throughput |

| ARP | |
|---|---|
| arp.dst.hw_mac | arp.proto.size |
| arp.dst.proto_ipv4 | arp.proto.type |
| arp.hw.size | arp.src.hw_mac |
| arp.hw.type | arp.src.proto_ipv4 |
| arp.opcode | |

| TCP | |
|---|---|
| tcp.ack | tcp.options.qs |
| tcp.checksum | tcp.options.sack |
| tcp.checksum_bad | tcp.options.sack_le |
| tcp.checksum_good | tcp.options.sack_perm |
| tcp.continuation_to | tcp.options.sack_re |
| tcp.dstport | tcp.options.time_stamp |
| tcp.flags | tcp.options.wscale |
| tcp.flags.ack | tcp.options.wscale_val |
| tcp.flags.cwr | tcp.pdu.last_frame |
| tcp.flags.ecn | tcp.pdu.size |
| tcp.flags.fin | tcp.pdu.time |
| tcp.flags.push | tcp.port |
| tcp.flags.reset | tcp.reassembled_in |
| tcp.flags.syn | tcp.segment |
| tcp.flags.urg | tcp.segment.error |
| tcp.hdr_len | tcp.segment.multipletails |
| tcp.len | tcp.segment.overlap |
| tcp.nxtseq | tcp.segment.overlap.conflict |



| UDP | | |
|---|---|---|
| udp.checksum | udp.dstport | udp.srcport |
| udp.checksum_bad | udp.length | |
| udp.checksum_good | udp.port | |

| Operators |
|---|
| eq or == |
| ne or != |
| gt or > |
| lt or < |
| ge or >= |
| le or <= |

| Logic | |
|---|---|
| and or && | Logical AND |
| or or \|\| | Logical OR |
| xor or ^^ | Logical XOR |
| not or ! | Logical NOT |
| [n] […] | Substring operator |

https://packetlife.net/media/library/13/Wireshark_Display_Filters.pdf

# Wireshark cont..

- Promiscuous mode Vs non-promiscuous mode
- Collecting Statistics/summary in wireshark
- Wireshark I/O graph

# References for Wireshark

- Wireshark User’s Guide: Version 3.7.3
- **Here are some basics on how to get started with Wireshark**
  - https://www.wireshark.org/docs/wsug_html_chunked/
  - http://www.howtogeek.com/104278/how-to-use-wireshark-to-capture-filter-and-inspect-packets/
- Youtube videos for beginners:
  - https://www.youtube.com/watch?v=TkCSr30UojM
  - https://www.youtube.com/watch?v=jvuiI1Leg6w&t=9s