LAB WORKBOOK

Computer Networks

Department of Computer Science & Engineering

University of Engineering & Technology, Lahore

This workbook is written for assisting the undergraduate students for the Lab Exercises of course on “”Computer Networks”.

Note: The contents of this document have been compiled from various academic resources. We have tried to give references of the content included in this lab manual. If some references are missing we are open to add them.

For Further information, please contact

Email: [faisal\_hayat@hotmail.com](mailto:faisal_hayat@hotmail.com)

LAB 1

Understanding Ping, Traceroute and Whois

**Source : Wikipedia.org**

Ping

**Ping** is a [computer network](https://en.wikipedia.org/wiki/Computer_network) administration utility used to test the reachability of a [host](https://en.wikipedia.org/wiki/Host_(network)) on an [Internet Protocol](https://en.wikipedia.org/wiki/Internet_Protocol) (IP) network and to measure the [round-trip time](https://en.wikipedia.org/wiki/Round-trip_time) for messages sent from the originating host to a destination computer. The name comes from [active sonar](https://en.wikipedia.org/wiki/Sonar#Active_sonar) terminology which sends a pulse of sound and listens for the echo to detect objects underwater. With computer operating systems Ping or PING stands for Packet INternet Groper but is ordinarily written as "ping" instead of the proper acronym for which it stands.

Ping operates by sending [Internet Control Message Protocol](https://en.wikipedia.org/wiki/Internet_Control_Message_Protocol) (ICMP) *echo request* [packets](https://en.wikipedia.org/wiki/Packet_(information_technology)) to the target host and waiting for an ICMP response. In the process it measures the time from transmission to reception (*round-trip time*) and records any [packet loss](https://en.wikipedia.org/wiki/Packet_loss). The results of the test are printed in the form of a statistical summary of the response packets received, including the minimum, maximum, and the [mean](https://en.wikipedia.org/wiki/Mean_(average)) round-trip times, and sometimes the [standard deviation](https://en.wikipedia.org/wiki/Standard_deviation) of the mean.

Depending on the implementation, the ping command can be run with various command line switches to enable special operational modes. Example options include: specifying the packet size used as the probe, automatic repeated operation for sending a specified count of probes, and time stamping.

Ping may be abused as a simple form of [denial-of-service attack](https://en.wikipedia.org/wiki/Denial-of-service_attack) in the form of a [ping flood](https://en.wikipedia.org/wiki/Ping_flood), in which the attacker overwhelms the victim with ICMP echo request packets.

# ping -n 5 www.example.com

PING www.example.com (192.0.43.10) 56(84) bytes of data.

64 bytes from 43-10.any.icann.org (192.0.43.10): icmp\_seq=1 ttl=250 time=80.5 ms

64 bytes from 43-10.any.icann.org (192.0.43.10): icmp\_seq=2 ttl=250 time=80.4 ms

64 bytes from 43-10.any.icann.org (192.0.43.10): icmp\_seq=3 ttl=250 time=80.3 ms

64 bytes from 43-10.any.icann.org (192.0.43.10): icmp\_seq=4 ttl=250 time=80.3 ms

64 bytes from 43-10.any.icann.org (192.0.43.10): icmp\_seq=5 ttl=250 time=80.4 ms

--- www.example.com ping statistics ---

5 packets transmitted, 5 received, 0% packet loss, time 4006ms

rtt min/avg/max/mdev = 80.393/80.444/80.521/0.187 ms

The utility summarizes its results after completing the ping probes. The shortest [round trip time](https://en.wikipedia.org/wiki/Round-trip_delay_time) was 80.393 ms, the average was 80.444 ms, and the maximum value was 80.521 ms. The measurement had a [standard deviation](https://en.wikipedia.org/wiki/Standard_deviation) of 0.187 ms.

Ping Attacks

Ping of death

Largest packet size a computer can handle is normally 65,535 byte. So, sending a 65,536-byte ping [packet](https://en.wikipedia.org/wiki/Packet_(information_technology)) would violate the [Internet Protocol](https://en.wikipedia.org/wiki/Internet_Protocol) as written in [RFC 791](https://tools.ietf.org/html/rfc791), but a packet of such a size can be sent if it is fragmented; when the target computer reassembles the packet, a [buffer overflow](https://en.wikipedia.org/wiki/Buffer_overflow) can occur, which often causes a [system crash](https://en.wikipedia.org/wiki/Crash_(computing)). This is called ping of death.

Ping flood

A **ping flood** is a simple [denial-of-service attack](https://en.wikipedia.org/wiki/Denial-of-service_attack) where the attacker overwhelms the victim with [ICMP Echo Request (ping)](https://en.wikipedia.org/wiki/Ping) [packets](https://en.wikipedia.org/wiki/Packet_(information_technology)). This is most effective by using the flood option of ping which sends ICMP packets as fast as possible without waiting for replies. Most implementations of ping require the user to be [privileged](https://en.wikipedia.org/wiki/Superuser) in order to specify the flood option. It is most successful if the attacker has more [bandwidth](https://en.wikipedia.org/wiki/Bandwidth_(computing)) than the victim (for instance an attacker with a [DSL](https://en.wikipedia.org/wiki/DSL) line and the victim on a dial-up [modem](https://en.wikipedia.org/wiki/Modem)). The attacker hopes that the victim will respond with [ICMP Echo Reply](https://en.wikipedia.org/wiki/ICMP_Echo_Reply)packets, thus consuming both outgoing bandwidth as well as incoming bandwidth. If the target system is slow enough, it is possible to consume enough of its CPU cycles for a user to notice a significant slowdown.

Smurf attack

The **Smurf attack** is a way of generating significant [computer network](https://en.wikipedia.org/wiki/Computer_network) traffic on a victim network. This is a type of [denial-of-service attack](https://en.wikipedia.org/wiki/Denial-of-service_attack) that floods a system via [spoofed](https://en.wikipedia.org/wiki/IP_address_spoofing) broadcast [ping](https://en.wikipedia.org/wiki/Ping_(networking_utility))messages.

This attack relies on a perpetrator sending a large amount of [ICMP](https://en.wikipedia.org/wiki/Internet_Control_Message_Protocol) echo request (ping) traffic to [IP](https://en.wikipedia.org/wiki/Internet_Protocol) broadcast addresses, all of which have a spoofed source [IP](https://en.wikipedia.org/wiki/Internet_Protocol) address of the intended victim. If the routing device delivering traffic to those broadcast addresses delivers the IP broadcast to all hosts (for example via a [layer 2](https://en.wikipedia.org/wiki/Layer_2) broadcast), most hosts on that IP network will take the ICMP echo request and reply to it with an echo reply, multiplying the traffic by the number of hosts responding. On a multi-access broadcast network, hundreds of machines might reply to each packet.

Traceroute

**Traceroute** is a [computer network](https://en.wikipedia.org/wiki/Computer_network) diagnostic tool for displaying the route (path) and measuring transit delays of [packets](https://en.wikipedia.org/wiki/Packet_(information_technology)) across an [Internet Protocol](https://en.wikipedia.org/wiki/Internet_Protocol) (IP) network.

Traceroute sends a sequence of [Internet Control Message Protocol](https://en.wikipedia.org/wiki/Internet_Control_Message_Protocol) (ICMP) [echo request](https://en.wikipedia.org/wiki/ICMP_Echo_Request#Echo_request) packets addressed to a destination host. Determining the intermediate routers traversed involves adjusting the [time-to-live](https://en.wikipedia.org/wiki/Time_to_live) (TTL), aka **hop limit**, Internet Protocol parameter. Frequently starting with a value like 128 (Windows) or 64 (Linux), routers decrement this and discard a packet when the TTL value has reached zero, returning the ICMP error message[ICMP Time Exceeded](https://en.wikipedia.org/wiki/ICMP_Time_Exceeded).

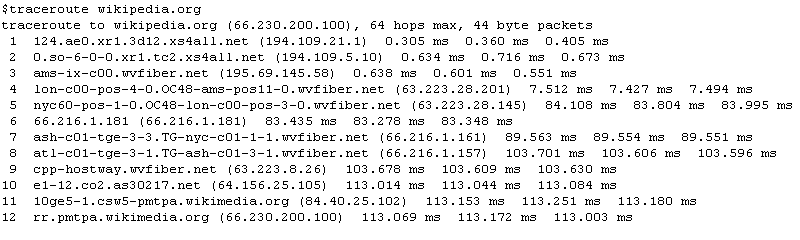
Traceroute works by increasing the TTL value of each successive set of packets sent. The first set of packets sent have a **hop limit** value of 1, expecting that they are not forwarded by the first router. The next set have a **hop limit** value of 2, so that the second router will send the error reply. This continues until the destination host receives the packets and returns an Reply message.

Traceroute uses the returned ICMP messages to produce a list of routers that the packets have traversed. The timestamp values returned for each router along the path are the delay (aka latency) values, typically measured in milliseconds for each packet.

The originating host expects a reply within a specified number of seconds. If a packet is not acknowledged within the expected timeout, an asterisk is displayed. The hosts listed may not be hosts used by other packets. The Internet Protocol does not require that packets between two hosts take the same route. Also note that if the host at hop number N does not reply, the hop will be skipped in the output.

The traceroute utility usually has an option to specify use of ICMP echo request (type 8) instead, as used by the [Windows](https://en.wikipedia.org/wiki/Windows) **tracert** utility. If a network has a firewall and operates both MS Windows and Unix-like systems, both protocols must be enabled inbound through the firewall.

There are also traceroute implementations that use TCP packets, such as [tcptraceroute](https://en.wikipedia.org/w/index.php?title=Tcptraceroute&action=edit&redlink=1) or [layer four traceroute](https://en.wikipedia.org/wiki/Layer_four_traceroute).



WHOIS

**WHOIS** (pronounced as the phrase *who is*) is a query and response [protocol](https://en.wikipedia.org/wiki/Communications_protocol) that is widely used for querying [databases](https://en.wikipedia.org/wiki/Database) that store the registered users or assignees of an [Internet](https://en.wikipedia.org/wiki/Internet) resource, such as a [domain name](https://en.wikipedia.org/wiki/Domain_name), an [IP address](https://en.wikipedia.org/wiki/IP_address) block, or an [autonomous system](https://en.wikipedia.org/wiki/Autonomous_system_(Internet)), but is also used for a wider range of other information. The protocol stores and delivers database content in a human-readable format.  The WHOIS protocol is documented in [RFC 3912](https://tools.ietf.org/html/rfc3912)

The WHOIS system originated as a method for system administrators to obtain contact information for IP address assignments or [domain name](https://en.wikipedia.org/wiki/Domain_name) administrators. The use of the data in the WHOIS system has evolved into a variety of uses, including

* Supporting the security and stability of the Internet by providing contact points for network operators and administrators, including ISPs, and certified computer incident response teams;
* Determining the registration status of domain names.
* <https://whois.icann.org/en>
* Try whois Wikipedia.org
* <https://www.ultratools.com/tools/ipWhoisLookup>

whois -h com.whois-servers.net example.com

[Querying com.whois-servers.net]

[com.whois-servers.net]

Whois Server Version 2.0

Domain names in the .com and .net domains can now be registered

with many different competing registrars. Go to http://www.internic.net

for detailed information.

Domain Name: EXAMPLE.COM

Registrar: RESERVED-INTERNET ASSIGNED NUMBERS AUTHORITY

Whois Server: whois.iana.org

Referral URL: http://res-dom.iana.org

Name Server: A.IANA-SERVERS.NET

Name Server: B.IANA-SERVERS.NET

Status: clientDeleteProhibited

Status: clientTransferProhibited

Status: clientUpdateProhibited

Updated Date: 26-mar-2004

Creation Date: 14-aug-1995

Expiration Date: 13-aug-2011

>>> Last update of whois database: Tue, 17 Aug 2010 02:23:52 UTC <<<

Pathping

**PathPing** is a [network utility](https://en.wikipedia.org/w/index.php?title=Network_utility&action=edit&redlink=1) supplied in [Windows NT](https://en.wikipedia.org/wiki/Windows_NT) and beyond that combines the functionality of [ping](https://en.wikipedia.org/wiki/Ping) with that of [tracert](https://en.wikipedia.org/wiki/Traceroute).

It provides details of the path between two [hosts](https://en.wikipedia.org/wiki/Computer) *and* Ping-like statistics for each node in the path based on samples taken over a time period, depending on how many nodes are between the start and end host.

The advantages of *PathPing* over **ping** and **traceroute** are that each node is pinged as the result of a single command, and that the behavior of nodes is studied over an extended time period, rather than the default *ping* sample of four messages or default *traceroute* single route trace.

Tracing route to wikipedia.com [207.142.131.235]

over a maximum of 30 hops:

0 simonslaptop [192.168.0.11]

1 192.168.0.1

2 thus1-hg2.ilford.broadband.bt.net [217.32.64.73]

3 217.32.64.34

4 217.32.64.110

5 anchor-border-1-4-0-2-191.router.demon.net [212.240.162.126]

6 anchor-core-2-g0-0-1.router.demon.net [194.70.98.29]

7 ny1-border-1-a1-0-s2.router.demon.net [194.70.97.66]

8 ge-8-0-153.ipcolo1.NewYork1.Level3.net [209.246.123.177]

9 ae-0-51.bbr1.NewYork1.Level3.net [64.159.17.1]

10 so-2-0-0.mp1.Tampa1.Level3.net [209.247.11.201]

11 ge-6-0.hsa2.Tampa1.Level3.net [64.159.1.10]

12 unknown.Level3.net [63.208.24.2]

13

Computing statistics for 325 seconds...

Source to Here This Node/Link

Hop RTT Lost/Sent = Pct Lost/Sent = Pct Address

0 simonslaptop [192.168.0.11]

0/ 100 = 0% |

1 0ms 0/ 100 = 0% 0/ 100 = 0% 192.168.0.1

0/ 100 = 0% |

Trace complete.

LAB Exercise

1. Traceroute to the following destinations/countries choosing a specific site of your own choice hosted in each of the mentioned countries and write down the main route from your site/country to the destination. You are required to mention the main nodes along with country/city name. You have to use the whois database to find out the required information. (You may use

<http://www.ip-adress.com/whois>, or

<http://www.apnic.net/apnic-info/whois_search2>, or

[www.whois.net](http://www.whois.net))

<http://www.monitis.com/traceroute/>

**Asia**

Singapore

China

India

**Middle East**

Saudi Arabia

Dubai

**Europe**

Germany

Italy

**United States**

**Mexico**

**Brazil**

**Canada**

**Australia**

**Newzealand**

**South Africa**

1. Write down at least three different observations about behavior of traceroute including an observation mentioning an “abnormal” output or something you are not expecting.
2. You can use traceroute to approximately find out earth diameter, can you explain how?

**LAB 2**

**Understanding DNS Working (Using nslookup and dig)**

**Ref: Wikipedia**

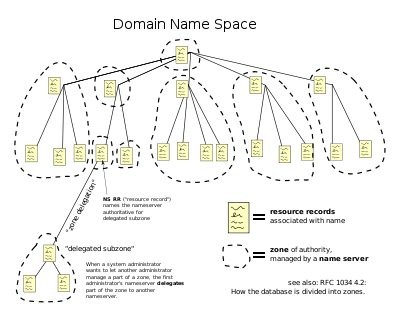
The **Domain Name System** (**DNS**) is a hierarchical distributed naming system for computers, services, or any resource connected to the [Internet](http://en.wikipedia.org/wiki/Internet) or a [private network](http://en.wikipedia.org/wiki/Private_network). It associates various information with [domain names](http://en.wikipedia.org/wiki/Domain_name) assigned to each of the participating entities. A **Domain Name Service** resolves queries for these names into [IP addresses](http://en.wikipedia.org/wiki/IP_address) for the purpose of locating computer services and devices worldwide. By providing a worldwide, distributed [keyword](http://en.wikipedia.org/wiki/Index_term)-based redirection service, the Domain Name System is an essential component of the functionality of the [Internet](http://en.wikipedia.org/wiki/Internet).

**DNS Records**

A DNS name server is a server that stores the DNS records for a domain name, such as address (A) records, name server (NS) records, and mail exchanger (MX) records (see also [list of DNS record types](http://en.wikipedia.org/wiki/List_of_DNS_record_types)); a DNS name server responds with answers to queries against its database.

### Domain name space

The domain name space consists of a [tree](http://en.wikipedia.org/wiki/Tree_data_structure) of domain names. Each node or leaf in the tree has zero or more *resource records*, which hold information associated with the domain name. The tree sub-divides into *zones* beginning at the [root zone](http://en.wikipedia.org/wiki/DNS_root_zone). A [DNS zone](http://en.wikipedia.org/wiki/DNS_zone) may consist of only one domain, or may consist of many domains and sub-domains, depending on the administrative authority delegated to the manager.

[](http://en.wikipedia.org/wiki/File:Domain_name_space.svg)

[http://bits.wikimedia.org/static-1.20wmf11/skins/common/images/magnify-clip.png](http://en.wikipedia.org/wiki/File:Domain_name_space.svg)

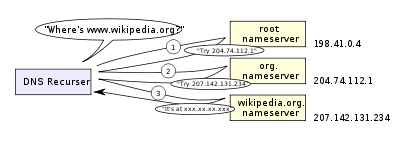
The hierarchical Domain Name System, organized into zones, each served by a name server

#### Authoritative name server

An *authoritative* name server is a name server that gives [answers](http://en.wikipedia.org/wiki/Domain_Name_System#Authoritative_answer) that have been configured by an original source, for example, the domain administrator or by dynamic DNS methods, in contrast to answers that were obtained via a regular DNS query to another name server. An authoritative-only name server only returns answers to queries about domain names that have been specifically configured by the administrator.

### Address resolution mechanism

Domain name resolvers determine the appropriate domain name servers responsible for the domain name in question by a sequence of queries starting with the right-most (top-level) domain label.

[](http://en.wikipedia.org/wiki/File:An_example_of_theoretical_DNS_recursion.svg)

[http://bits.wikimedia.org/static-1.20wmf11/skins/common/images/magnify-clip.png](http://en.wikipedia.org/wiki/File:An_example_of_theoretical_DNS_recursion.svg)

A DNS recursor consults three nameservers to resolve the address www.wikipedia.org.

The process entails:

1. A network host is configured with an initial cache (so called *hints*) of the known addresses of the [root nameservers](http://en.wikipedia.org/wiki/Root_nameserver). Such a *hint file* is updated periodically by an administrator from a reliable source.
2. A query to one of the root servers to find the server authoritative for the top-level domain.
3. A query to the obtained TLD server for the address of a DNS server authoritative for the second-level domain.
4. Repetition of the previous step to process each domain name label in sequence, until the final step which returns the IP address of the host sought.

The diagram illustrates this process for the host www.wikipedia.org.

### Record caching

The DNS Resolution Process reduces the load on individual servers by *caching* DNS request records for a period of time after a response. This entails the local recording and subsequent consultation of the copy instead of initiating a new request upstream. The time for which a resolver caches a DNS response is determined by a value called the [time to live](http://en.wikipedia.org/wiki/Time_to_live) (TTL) associated with every record.

### Reverse lookup

A reverse lookup is a query of the DNS for domain names when the IP address is known. Multiple domain names may be associated with an IP address.

**DNS Zone**

A **DNS zone** is a portion of a domain name space using the [Domain Name System](http://en.wikipedia.org/wiki/Domain_Name_System) (DNS) for which administrative responsibility has been delegated.

## Internet infrastructure DNS zones

The [arpa](http://en.wikipedia.org/wiki/.arpa) top-level domain serves as a delegation zone for various technical infrastructure aspects of DNS and the Internet, and does not follow the registration and delegation system of the country and generic domains. The name *arpa* is a remnant of the [ARPANET](http://en.wikipedia.org/wiki/ARPANET), one of the predecessor stages of today's Internet. Intended as a transition aid to the modern DNS system, deleting the arpa domain was later found to be impractical. It is now officially the acronym for Address and Routing Parameter Area. It contains sub-zones used for reverse resolution of IP addresses to host names ([IPv4](http://en.wikipedia.org/wiki/IPv4): in-addr.arpa, [IPv6](http://en.wikipedia.org/wiki/IPv6): ip6.arpa), telephone number mapping ([ENUM](http://en.wikipedia.org/wiki/ENUM), e164.arpa), and uniform resource identifier resolution (uri.arpa, urn.arpa). Although the administrative structure of this domain and its sub-domains is different, the technical delegation into zones of responsibility is similar and the DNS tools and servers used are identical to any other zone. Sub-zones are delegated by components of the respective resources. For example, 8.8.2.5.5.2.2.0.0.8.1.e164.arpa., which might represent an E.164 telephone number in the [ENUM](http://en.wikipedia.org/wiki/ENUM) system, might be sub-delegated at suitable boundaries of the name. Examples of IP addresses in the reverse DNS zone are: 166.188.77.208.in-addr.arpa, resolving to the domain namewww.example.com. In the case of [IP addresses](http://en.wikipedia.org/wiki/IP_address), the reverse zones are always delegated to the [Internet service provider](http://en.wikipedia.org/wiki/Internet_service_provider) (ISP) to which the IP address block is assigned. When an ISP allocates a range to a customer, it usually also delegates the management of that space to the customer by insertion of name server resource records (pointing to the customers DNS facilities) into their zone. Notably, however, many ISPs serving individual end-users, such as homes or small businesses with only one IP address do not do so.

## Example of zone authority in DNS queries

As an example of the DNS resolving process, consider the role of a recursive [DNS resolver](http://en.wikipedia.org/wiki/Domain_Name_System#DNS_resolvers) attempting to look up the address "en.wikipedia.org.". It begins with a list of addresses for the most authoritative name servers it knows about – the [root zone name servers](http://en.wikipedia.org/wiki/Root_name_server) (indicated by the full stop or period), which contains name server information for all [top-level domains](http://en.wikipedia.org/wiki/Top-level_domain) of the Internet.

When querying one of the root name servers, it is possible that the [root zone](http://en.wikipedia.org/wiki/DNS_root_zone) will not directly contain a record for "en.wikipedia.org.", in which case it will provide a referral to the authoritative name servers for the "org." [top level domain](http://en.wikipedia.org/wiki/Top_level_domain) (TLD). The resolver is issued a referral to the authoritative name servers for the "org." zone, which it will contact for more specific information. Again when querying one of the "org." name servers, the resolver may be issued with another referral to the "wikipedia.org." zone, whereupon it will again query for "en.wikipedia.org.". Since (as of July 2010) "en.wikipedia.org." is a [CNAME](http://en.wikipedia.org/wiki/CNAME) to "text.wikimedia.org." (which is in turn a CNAME to "text.esams.wikimedia.org."), and the "wikipedia.org." name servers also happen to contain authoritative data for the "wikimedia.org." zone, the resolution of this particular query occurs entirely within the queried name server, and the resolver will receive the address record it requires with no further referrals.

If the last name server queried did not contain authoritative data for the target of the CNAME, it would have issued the resolver with yet another referral, this time to the zone *text.wikimedia.org.*. However, since the resolver had previously determined the authoritative name servers for the zone *org.*, it does not need to begin the resolution process from scratch but instead start at zone *org.*, thus avoiding another query to the root name servers.

There is no requirement that resolving should involve any referrals at all. Looking up *en.wikipedia.org.* on the root name servers always results in referrals, but if an [alternative DNS root](http://en.wikipedia.org/wiki/Alternative_DNS_root) is used which is set up to contain a record for *en.wikipedia.org.*, then the record is returned on the first query.

**Using nslookup**

In it is most basic operation, *nslookup* tool allows the host running the tool to query any specified DNS server for a DNS record. The queried DNS server can be a root DNS server, a top-level-domain DNS server, an authoritative DNS server, or an intermediate DNS server (see the textbook for definitions of these terms). To accomplish this task, *nslookup* sends a DNS query to the specified DNS server, receives a DNS reply from that same DNS server, and displays the result.

Consider the first command:

**nslookup www.mit.edu**

Now consider the second command:

**nslookup –type=NS mit.edu**

finally consider the third command:

**nslookup www.uet.edu.pk bitsy.mit.edu**

In this example, we indicate that we want to the query sent to the DNS server bitsy.mit.edu rather than to the default DNS server of UET. Thus, the query and reply transaction takes place directly between our querying host and bitsy.mit.edu. In this example, the DNS server bitsy.mit.edu provides the IP address of the host

www.uet.edu.pk, which is a web server at the UET Lahore.

**Lab Exercise 1:**

Do the following (and write down the results):

1. Run *nslookup* to obtain the IP address of a Web server of UET.

2. Run *nslookup* to determine the authoritative DNS servers for UET.

3. Run *nslookup* to determine the mail servers for UET.

**Using Dig**

**dig** (domain information groper) is a [network administration](http://en.wikipedia.org/wiki/Network_administration) [command-line](http://en.wikipedia.org/wiki/Command-line) tool for querying [Domain Name System](http://en.wikipedia.org/wiki/Domain_Name_System) (DNS) [name servers](http://en.wikipedia.org/wiki/Name_server).

Dig is useful for network troubleshooting and for educational purposes. Dig can operate in interactive command line mode or in batch mode by reading requests from an operating system file. When a specific name server is not specified in the command invocation, it will use the operating systems default resolver, usually configured via the [resolv.conf](http://en.wikipedia.org/wiki/Resolv.conf) file. Without any arguments it queries the [DNS root zone](http://en.wikipedia.org/wiki/DNS_root_zone).

Dig supports [Internationalized Domain Name](http://en.wikipedia.org/wiki/Internationalized_Domain_Name) (IDN) queries.

Dig is part of the [BIND](http://en.wikipedia.org/wiki/BIND) domain name server software suite. Dig replaces older tools such as [nslookup](http://en.wikipedia.org/wiki/Nslookup) and the [host](http://en.wikipedia.org/wiki/Host_(Unix)) program.

**Lab Exercise 2**

Download dig tool for your operating system

1. Run this command: **dig google.com**
   1. What IP addresses are associated with google.com?
   2. Run the same command again. Note that the numbers in the second column of the Answer Section change. Wait a few seconds and run it a third time. What do the numbers in the second column represent?
2. Run this command: **dig google.com ns**
   1. What is different about the information provided by this version of the command?
   2. How many name servers are there for google.com?
3. Run this command google.com mx   
   What information does the output of this version of the command provide?
   1. Run a command to find the number of IP addresses that are associated with yahoo.com.
   2. Run a command to find the number of name servers that there are for yahoo.com.
4. Run this command: **dig www.google.com**
   1. What IP addresses are associated with www.google.com?
   2. How many name servers are there for www.google.com?
   3. What relationship (if any) do you see between google.com and www.google.com?
5. Run this command: **dig mail.google.com**
   1. How many IP addresses are associated with mail.google.com?
   2. What relationship (if any) do you see between google.com and mail.google.com?
6. Run the command:  **dig microsoft.com**

Note the query time

1. Repeat the command and note of the query time
2. Run the command: **dig @ns1.msft.net microsoft.com** and note the query time
3. Repeat the previous command and note the query time
4. Explain the reasons for any observed differences in the query times for each of the commands in parts a. - d.
5. Run each of the following commands and describe what the output represents in each case (also specify what each domain represents).
6. **dig edu ns**
7. **dig ns**
8. Run this command: **dig www.mit.edu +trace**
   1. Summarize what information is provided by the output of the command that uses the trace option. (Some of the info from 8.a. will be useful in answering this question.)
   2. Give the specific path of name servers that provide the information requested in this query.
   3. Repeat the command several times. Look for differences in the outputs when the command is repeated and explain them.

**LAB 3**

**Understanding packet sniffers/protocol analyzers**

**(Using Wireshark)**

**Lab 3a**

Understanding HTTP working with wireshark

**Note : Lab material is from the author of textbook and is available at**

[**http://gaia.cs.umass.edu/wireshark-labs/**](http://gaia.cs.umass.edu/wireshark-labs/)

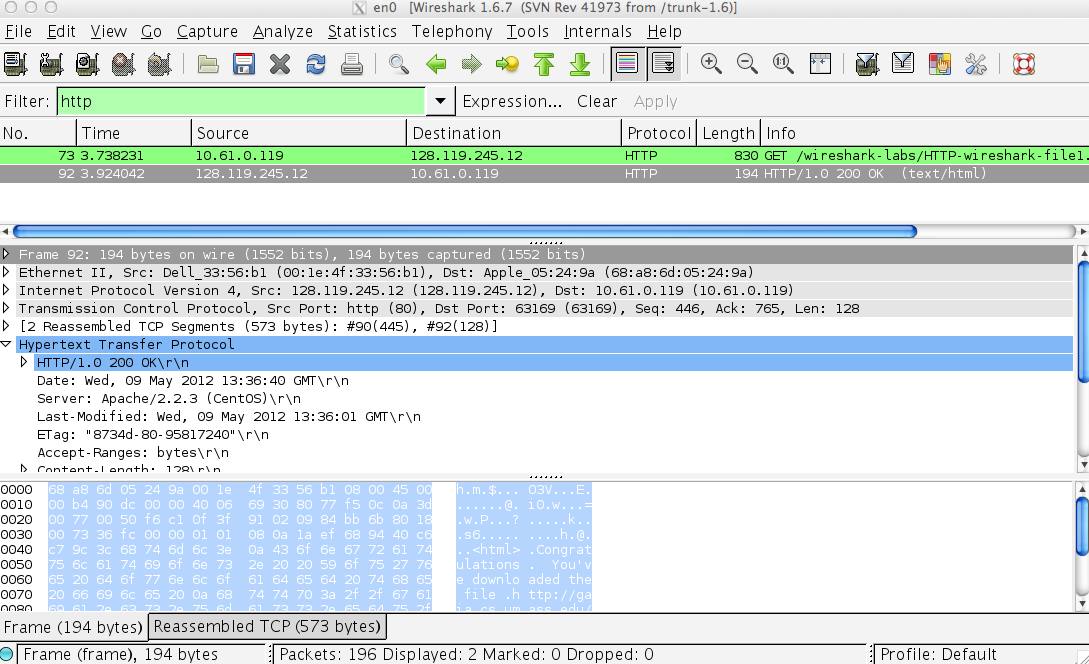
In this lab, we’ll explore several aspects of the HTTP protocol: the basic GET/response interaction, HTTP message formats, retrieving large HTML files, retrieving HTML files with embedded objects, and HTTP authentication and security. Before beginning these labs, you might want to review Section 2.2 of the text.[[1]](#footnote-1)

1. The Basic HTTP GET/response interaction

Let’s begin our exploration of HTTP by downloading a very simple HTML file - one that is very short, and contains no embedded objects. Do the following:

1. Start up your web browser.
2. Start up the Wireshark packet sniffer, as described in the Introductory lab (but don’t yet begin packet capture). Enter “http” (just the letters, not the quotation marks) in the display-filter-specification window, so that only captured HTTP messages will be displayed later in the packet-listing window. (We’re only interested in the HTTP protocol here, and don’t want to see the clutter of all captured packets).
3. Wait a bit more than one minute (we’ll see why shortly), and then begin Wireshark packet capture.
4. Enter the following to your browser  
   <http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file1.html>  
   Your browser should display the very simple, one-line HTML file.
5. Stop Wireshark packet capture.

Your Wireshark window should look similar to the window shown in Figure 1. If you are unable to run Wireshark on a live network connection, you can download a packet trace that was created when the steps above were followed.[[2]](#footnote-2)



**Figure 1:** Wireshark Display after http://gaia.cs.umass.edu/wireshark-labs/ HTTP-wireshark-file1.html has been retrieved by your browser

The example in Figure 1 shows in the packet-listing window that two HTTP messages were captured: the GET message (from your browser to the gaia.cs.umass.edu web server) and the response message from the server to your browser. The packet-contents window shows details of the selected message (in this case the HTTP OK message, which is highlighted in the packet-listing window). Recall that since the HTTP message was carried inside a TCP segment, which was carried inside an IP datagram, which was carried within an Ethernet frame, Wireshark displays the Frame, Ethernet, IP, and TCP packet information as well. We want to minimize the amount of non-HTTP data displayed (we’re interested in HTTP here, and will be investigating these other protocols is later labs), so make sure the boxes at the far left of the Frame, Ethernet, IP and TCP information have a plus sign or a right-pointing triangle (which means there is hidden, undisplayed information), and the HTTP line has a minus sign or a down-pointing triangle (which means that all information about the HTTP message is displayed).

(*Note:* You should ignore any HTTP GET and response for favicon.ico. If you see a reference to this file, it is your browser automatically asking the server if it (the server) has a small icon file that should be displayed next to the displayed URL in your browser. We’ll ignore references to this pesky file in this lab.).

By looking at the information in the HTTP GET and response messages, answer the following questions. When answering the following questions, you should print out the GET and response messages (see the introductory Wireshark lab for an explanation of how to do this) and indicate where in the message you’ve found the information that answers the following questions. When you hand in your assignment, annotate the output so that it’s clear where in the output you’re getting the information for your answer (e.g., for our classes, we ask that students markup paper copies with a pen, or annotate electronic copies with text in a colored font).

1. Is your browser running HTTP version 1.0 or 1.1? What version of HTTP is the server running?
2. What languages (if any) does your browser indicate that it can accept to the server?
3. What is the IP address of your computer? Of the gaia.cs.umass.edu server?
4. What is the status code returned from the server to your browser?
5. When was the HTML file that you are retrieving last modified at the server?
6. How many bytes of content are being returned to your browser?
7. By inspecting the raw data in the packet content window, do you see any headers within the data that are not displayed in the packet-listing window? If so, name one.

In your answer to question 5 above, you might have been surprised to find that the document you just retrieved was last modified within a minute before you downloaded the document. That’s because (for this particular file), the gaia.cs.umass.edu server is setting the file’s last-modified time to be the current time, and is doing so once per minute. Thus, if you wait a minute between accesses, the file will appear to have been recently modified, and hence your browser will download a “new” copy of the document.

2. The HTTP CONDITIONAL GET/response interaction

Recall from Section 2.2.6 of the text, that most web browsers perform object caching and thus perform a conditional GET when retrieving an HTTP object. Before performing the steps below, make sure your browser’s cache is empty. (To do this under Firefox, select *Tools->Clear Recent History* and check the Cache box, or for Internet Explorer, select *Tools->Internet Options->Delete File;* these actions will remove cached files from your browser’s cache.) Now do the following:

* Start up your web browser, and make sure your browser’s cache is cleared, as discussed above.
* Start up the Wireshark packet sniffer
* Enter the following URL into your browser  
  <http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file2.html>  
  Your browser should display a very simple five-line HTML file.
* Quickly enter the same URL into your browser again (or simply select the refresh button on your browser)
* Stop Wireshark packet capture, and enter “http” in the display-filter-specification window, so that only captured HTTP messages will be displayed later in the packet-listing window.
* (*Note:* If you are unable to run Wireshark on a live network connection, you can use the http-ethereal-trace-2 packet trace to answer the questions below; see footnote 1. This trace file was gathered while performing the steps above on one of the author’s computers.)

Answer the following questions:

1. Inspect the contents of the first HTTP GET request from your browser to the server. Do you see an “IF-MODIFIED-SINCE” line in the HTTP GET?
2. Inspect the contents of the server response. Did the server explicitly return the contents of the file? How can you tell?
3. Now inspect the contents of the second HTTP GET request from your browser to the server. Do you see an “IF-MODIFIED-SINCE:” line in the HTTP GET? If so, what information follows the “IF-MODIFIED-SINCE:” header?
4. What is the HTTP status code and phrase returned from the server in response to this second HTTP GET? Did the server explicitly return the contents of the file? Explain.

3. Retrieving Long Documents

In our examples thus far, the documents retrieved have been simple and short HTML files. Let’s next see what happens when we download a long HTML file. Do the following:

* Start up your web browser, and make sure your browser’s cache is cleared, as discussed above.
* Start up the Wireshark packet sniffer
* Enter the following URL into your browser  
  <http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file3.html>  
  Your browser should display the rather lengthy US Bill of Rights.
* Stop Wireshark packet capture, and enter “http” in the display-filter-specification window, so that only captured HTTP messages will be displayed.
* (*Note:* If you are unable to run Wireshark on a live network connection, you can use the http-ethereal-trace-3 packet trace to answer the questions below; see footnote 1. This trace file was gathered while performing the steps above on one of the author’s computers.)

In the packet-listing window, you should see your HTTP GET message, followed by a multiple-packet TCP response to your HTTP GET request. This multiple-packet response deserves a bit of explanation. Recall from Section 2.2 (see Figure 2.9 in the text) that the HTTP response message consists of a status line, followed by header lines, followed by a blank line, followed by the entity body. In the case of our HTTP GET, the entity body in the response is the *entire* requested HTML file. In our case here, the HTML file is rather long, and at 4500 bytes is too large to fit in one TCP packet. The single HTTP response message is thus broken into several pieces by TCP, with each piece being contained within a separate TCP segment (see Figure 1.24 in the text). In recent versions of Wireshark, Wireshark indicates each TCP segment as a separate packet, and the fact that the single HTTP response was fragmented across multiple TCP packets is indicated by the “TCP segment of a reassembled PDU” in the Info column of the Wireshark display. Earlier versions of Wireshark used the “Continuation” phrase to indicated that the entire content of an HTTP message was broken across multiple TCP segments.. We stress here that there is no “Continuation” message in HTTP!

Answer the following questions:

1. How many HTTP GET request messages did your browser send? Which packet number in the trace contains the GET message for the Bill or Rights?
2. Which packet number in the trace contains the status code and phrase associated with the response to the HTTP GET request?
3. What is the status code and phrase in the response?
4. How many data-containing TCP segments were needed to carry the single HTTP response and the text of the Bill of Rights?

4. HTML Documents with Embedded Objects

Now that we’ve seen how Wireshark displays the captured packet traffic for large HTML files, we can look at what happens when your browser downloads a file with embedded objects, i.e., a file that includes other objects (in the example below, image files) that are stored on another server(s).

Do the following:

* Start up your web browser, and make sure your browser’s cache is cleared, as discussed above.
* Start up the Wireshark packet sniffer
* Enter the following URL into your browser  
  <http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file4.html>  
  Your browser should display a short HTML file with two images. These two images are referenced in the base HTML file. That is, the images themselves are not contained in the HTML; instead the URLs for the images are contained in the downloaded HTML file. As discussed in the textbook, your browser will have to retrieve these logos from the indicated web sites. Our publisher’s logo is retrieved from the www.aw-bc.com web site. The image of the cover for our 5th edition (one of our favorite covers) is stored at the manic.cs.umass.edu server.
* Stop Wireshark packet capture, and enter “http” in the display-filter-specification window, so that only captured HTTP messages will be displayed.
* (*Note:* If you are unable to run Wireshark on a live network connection, you can use the http-ethereal-trace-4 packet trace to answer the questions below; see footnote 1. This trace file was gathered while performing the steps above on one of the author’s computers.)

Answer the following questions:

1. How many HTTP GET request messages did your browser send? To which Internet addresses were these GET requests sent?
2. Can you tell whether your browser downloaded the two images serially, or whether they were downloaded from the two web sites in parallel? Explain.

5 HTTP Authentication

Finally, let’s try visiting a web site that is password-protected and examine the sequence of HTTP message exchanged for such a site. The URL

http://gaia.cs.umass.edu/wireshark-labs/protected\_pages/HTTP-wireshark-file5.html is password protected. The username is “wireshark-students” (without the quotes), and the password is “network” (again, without the quotes). So let’s access this “secure” password-protected site. Do the following:

* Make sure your browser’s cache is cleared, as discussed above, and close down your browser. Then, start up your browser
* Start up the Wireshark packet sniffer
* Enter the following URL into your browser  
  <http://gaia.cs.umass.edu/wireshark-labs/protected_pages/HTTP-wireshark-file5.html>  
  Type the requested user name and password into the pop up box.
* Stop Wireshark packet capture, and enter “http” in the display-filter-specification window, so that only captured HTTP messages will be displayed later in the packet-listing window.
* (*Note:* If you are unable to run Wireshark on a live network connection, you can use the http-ethereal-trace-5 packet trace to answer the questions below; see footnote 2. This trace file was gathered while performing the steps above on one of the author’s computers.)

Now let’s examine the Wireshark output. You might want to first read up on HTTP authentication by reviewing the easy-to-read material on “HTTP Access Authentication Framework” at <http://frontier.userland.com/stories/storyReader$2159>

Answer the following questions:

1. What is the server’s response (status code and phrase) in response to the initial HTTP GET message from your browser?
2. When your browser’s sends the HTTP GET message for the second time, what new field is included in the HTTP GET message?

The username (wireshark-students) and password (network) that you entered are encoded in the string of characters (d2lyZXNoYXJrLXN0dWRlbnRzOm5ldHdvcms=) following the “Authorization: Basic” header in the client’s HTTP GET message. While it may appear that your username and password are encrypted, they are simply encoded in a format known as Base64 format. The username and password are *not* encrypted! To see this, go to <http://www.motobit.com/util/base64-decoder-encoder.asp> and enter the base64-encoded string d2lyZXNoYXJrLXN0dWRlbnRz and decode. *Voila!* You have translated from Base64 encoding to ASCII encoding, and thus should see your username! To view the password, enter the remainder of the string Om5ldHdvcms= and press decode. Since anyone can download a tool like Wireshark and sniff packets (not just their own) passing by their network adaptor, and anyone can translate from Base64 to ASCII (you just did it!), it should be clear to you that simple passwords on WWW sites are not secure unless additional measures are taken.

Fear not! As we will see in Chapter 8, there are ways to make WWW access more secure. However, we’ll clearly need something that goes beyond the basic HTTP authentication framework!

**Lab 3b**

Understanding TCP with wireshark

**Note : Lab material is from the author of textbook and is available at**

[**http://gaia.cs.umass.edu/wireshark-labs/**](http://gaia.cs.umass.edu/wireshark-labs/)

In this lab, we’ll investigate the behavior of the celebrated TCP protocol in detail. We’ll do so by analyzing a trace of the TCP segments sent and received in transferring a 150KB file (containing the text of Lewis Carrol’s *Alice’s Adventures in Wonderland*) from your computer to a remote server. We’ll study TCP’s use of sequence and acknowledgement numbers for providing reliable data transfer; we’ll see TCP’s congestion control algorithm – slow start and congestion avoidance – in action; and we’ll look at TCP’s receiver-advertised flow control mechanism. We’ll also briefly consider TCP connection setup and we’ll investigate the performance (throughput and round-trip time) of the TCP connection between your computer and the server.

Before beginning this lab, you’ll probably want to review sections 3.5 and 3.7 in the text[[3]](#footnote-3).

1. Capturing a bulk TCP transfer from your computer to a remote server

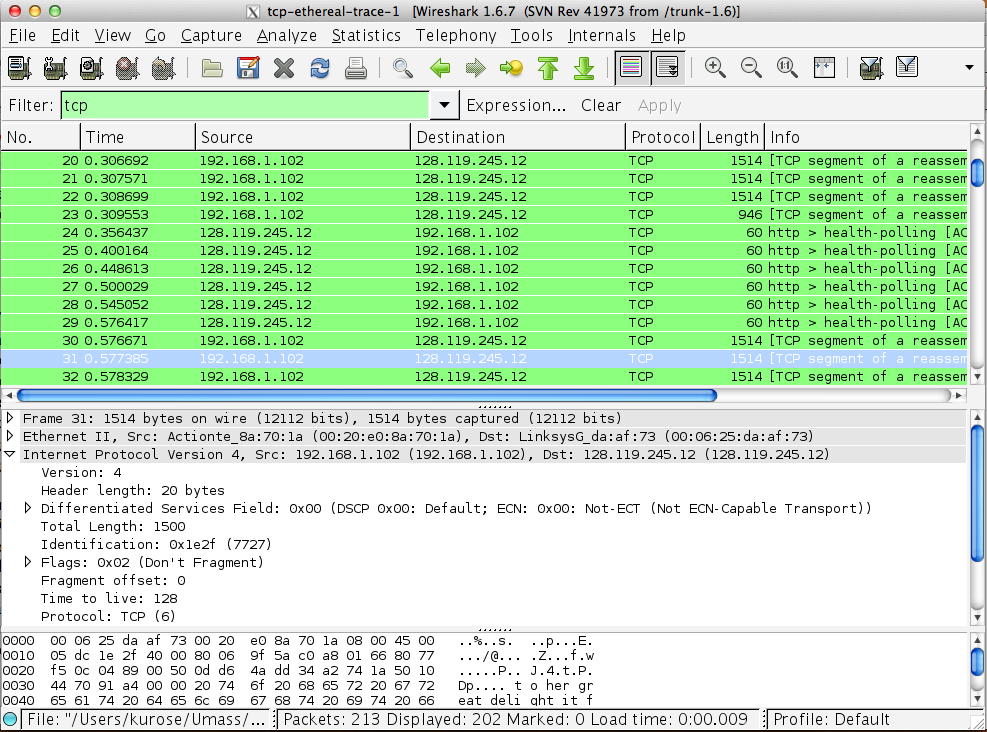
Before beginning our exploration of TCP, we’ll need to use Wireshark to obtain a packet trace of the TCP transfer of a file from your computer to a remote server. You’ll do so by accessing a Web page that will allow you to enter the name of a file stored on your computer (which contains the ASCII text of *Alice in Wonderland*), and then transfer the file to a Web server using the HTTP POST method (see section 2.2.3 in the text). We’re using the POST method rather than the GET method as we’d like to transfer a large amount of data *from* your computer to another computer. Of course, we’ll be running Wireshark during this time to obtain the trace of the TCP segments sent and received from your computer.

Do the following:

* Start up your web browser. Go the <http://gaia.cs.umass.edu/wireshark-labs/alice.txt> and retrieve an ASCII copy of *Alice in Wonderland.* Store this file somewhere on your computer.
* Next go to <http://gaia.cs.umass.edu/wireshark-labs/TCP-wireshark-file1.html>.
* You should see a screen that looks like:



* Use the *Browse* button in this form to enter the name of the file (full path name) on your computer containing *Alice in Wonderland* (or do so manually).Don’t yet press the “*Upload alice.txt file*” button.
* Now start up Wireshark and begin packet capture *(Capture->Start)* and then press *OK* on the Wireshark Packet Capture Options screen (we’ll not need to select any options here).
* Returning to your browser, press the “*Upload alice.txt file*” button to upload the file to the gaia.cs.umass.edu server. Once the file has been uploaded, a short congratulations message will be displayed in your browser window.
* Stop Wireshark packet capture. Your Wireshark window should look similar to the window shown below.



If you are unable to run Wireshark on a live network connection, you can download a packet trace file that was captured while following the steps above on one of the author’s computers[[4]](#footnote-4). You may well find it valuable to download this trace even if you’ve captured your own trace and use it, as well as your own trace, when you explore the questions below.

2. A first look at the captured trace

Before analyzing the behavior of the TCP connection in detail, let’s take a high level view of the trace.

* First, filter the packets displayed in the Wireshark window by entering “tcp” (lowercase, no quotes, and don’t forget to press return after entering!) into the display filter specification window towards the top of the Wireshark window.

What you should see is series of TCP and HTTP messages between your computer and gaia.cs.umass.edu. You should see the initial three-way handshake containing a SYN message. You should see an HTTP POST message. Depending on the version of Wireshark you are using, you might see a series of “HTTP Continuation” messages being sent from your computer to gaia.cs.umass.edu. Recall from our discussion in the earlier HTTP Wireshark lab, that is no such thing as an HTTP Continuation message – this is Wireshark’s way of indicating that there are multiple TCP segments being used to carry a single HTTP message. In more recent versions of Wireshark, you’ll see “[TCP segment of a reassembled PDU]” in the Info column of the Wireshark display to indicate that this TCP segment contained data that belonged to an upper layer protocol message (in our case here, HTTP). You should also see TCP ACK segments being returned from gaia.cs.umass.edu to your computer.

Answer the following questions, by opening the Wireshark captured packet file *tcp-ethereal-trace-1* in <http://gaia.cs.umass.edu/wireshark-labs/wireshark-traces.zip> (that is download the trace and open that trace in Wireshark; see footnote 2). Whenever possible, when answering a question you should hand in a printout of the packet(s) within the trace that you used to answer the question asked. Annotate the printout[[5]](#footnote-5) to explain your answer. To print a packet, use *File->Print*, choose *Selected packet only*, choose *Packet summary line,* and select the minimum amount of packet detail that you need to answer the question.

1. What is the IP address and TCP port number used by the client computer (source) that is transferring the file to gaia.cs.umass.edu? To answer this question, it’s probably easiest to select an HTTP message and explore the details of the TCP packet used to carry this HTTP message, using the “details of the selected packet header window” (refer to Figure 2 in the “Getting Started with Wireshark” Lab if you’re uncertain about the Wireshark windows.
2. What is the IP address of gaia.cs.umass.edu? On what port number is it sending and receiving TCP segments for this connection?

If you have been able to create your own trace, answer the following question:

1. What is the IP address and TCP port number used by your client computer (source) to transfer the file to gaia.cs.umass.edu?

Since this lab is about TCP rather than HTTP, let’s change Wireshark’s “listing of captured packets” window so that it shows information about the TCP segments containing the HTTP messages, rather than about the HTTP messages. To have Wireshark do this, select *Analyze->Enabled Protocols.* Then uncheck the HTTP box and select *OK*. You should now see a Wireshark window that looks like:



This is what we’re looking for - a series of TCP segments sent between your computer and gaia.cs.umass.edu. We will use the packet trace that you have captured (and/or the packet trace *tcp-ethereal-trace-1* in <http://gaia.cs.umass.edu/wireshark-labs/wireshark-traces.zip>; see earlier footnote) to study TCP behavior in the rest of this lab.

3. TCP Basics

Answer the following questions for the TCP segments:

1. What is the sequence number of the TCP SYN segment that is used to initiate the TCP connection between the client computer and gaia.cs.umass.edu? What is it in the segment that identifies the segment as a SYN segment?
2. What is the sequence number of the SYNACK segment sent by gaia.cs.umass.edu to the client computer in reply to the SYN? What is the value of the Acknowledgement field in the SYNACK segment? How did gaia.cs.umass.edu determine that value? What is it in the segment that identifies the segment as a SYNACK segment?
3. What is the sequence number of the TCP segment containing the HTTP POST command? Note that in order to find the POST command, you’ll need to dig into the packet content field at the bottom of the Wireshark window, looking for a segment with a “POST” within its DATA field.
4. Consider the TCP segment containing the HTTP POST as the first segment in the TCP connection. What are the sequence numbers of the first six segments in the TCP connection (including the segment containing the HTTP POST)? At what time was each segment sent? When was the ACK for each segment received? Given the difference between when each TCP segment was sent, and when its acknowledgement was received, what is the RTT value for each of the six segments? What is the EstimatedRTT value (see Section 3.5.3, page 239 in text) after the receipt of each ACK? Assume that the value of the EstimatedRTT is equal to the measured RTT for the first segment, and then is computed using the EstimatedRTT equation on page 239 for all subsequent segments.

*Note:* Wireshark has a nice feature that allows you to plot the RTT for each of the TCP segments sent. Select a TCP segment in the “listing of captured packets” window that is being sent from the client to the gaia.cs.umass.edu server. Then select: *Statistics->TCP Stream Graph->Round Trip Time Graph.*

1. What is the length of each of the first six TCP segments?[[6]](#footnote-6)
2. What is the minimum amount of available buffer space advertised at the received for the entire trace? Does the lack of receiver buffer space ever throttle the sender?
3. Are there any retransmitted segments in the trace file? What did you check for (in the trace) in order to answer this question?
4. How much data does the receiver typically acknowledge in an ACK? Can you identify cases where the receiver is ACKing every other received segment (see Table 3.2 on page 247 in the text).
5. What is the throughput (bytes transferred per unit time) for the TCP connection? Explain how you calculated this value.

4. TCP congestion control in action

Let’s now examine the amount of data sent per unit time from the client to the server. Rather than (tediously!) calculating this from the raw data in the Wireshark window, we’ll use one of Wireshark’s TCP graphing utilities - *Time-Sequence-Graph(Stevens*) - to plot out data.

* Select a TCP segment in the Wireshark’s “listing of captured-packets” window. Then select the menu : *Statistics->TCP Stream Graph-> Time-Sequence-Graph(Stevens*). You should see a plot that looks similar to the following plot, which was created from the captured packets in the packet trace [*tcp-ethereal-trace-1* in <http://gaia.cs.umass.edu/wireshark-labs/wireshark-traces.zip>](http://gaia.cs.umass.edu/ethereal-labs/traces/lab3-1-trace) (see earlier footnote ):



Here, each dot represents a TCP segment sent, plotting the sequence number of the segment versus the time at which it was sent. Note that a set of dots stacked above each other represents a series of packets that were sent back-to-back by the sender.

Answer the following questions for the TCP segments the packet trace [*tcp-ethereal-trace-1* in <http://gaia.cs.umass.edu/wireshark-labs/wireshark-traces.zip>](http://gaia.cs.umass.edu/ethereal-labs/traces/lab3-1-trace)

1. Use the *Time-Sequence-Graph(Stevens*) plotting tool to view the sequence number versus time plot of segments being sent from the client to the gaia.cs.umass.edu server. Can you identify where TCP’s slowstart phase begins and ends, and where congestion avoidance takes over? Comment on ways in which the measured data differs from the idealized behavior of TCP that we’ve studied in the text.
2. Answer each of two questions above for the trace that you have gathered when you transferred a file from your computer to gaia.cs.umass.edu

**Lab 4**

Understanding Discrete Event Simulation

**Note: Most of material is from various Internet resources**

**Introduction:**

**Evaluation Spectra:**

**What is a Simulation?**

* **Goal of Simulation**

Study system performance / operation

Real-system not available, is complex/costly or dangerous (eg: space simulations, flight simulations).

Simulation quickly evaluate design alternatives (eg: different system configurations).

And we can evaluate complex functions for which closed form formulas or numerical techniques not available.

**What ‘s in a simulation program?**

*simulated time:* internal (to simulation program) variable that keeps track of simulated time

*system “state”:* variables maintained by simulation program define system “state”

e.g., may track number (possibly order) of packets in queue, current value of retransmission timer

*events:* points in time when system changes state

each event has associated *event time*

e.g., arrival of packet to queue, departure from queue

precisely at these points in time that simulation must take action (change state and may cause new future events)

model for time between events (probabilistic) caused by external environment

**Discrete Event Simulation Program Structure**

**Simulation of GI/G/1/S Queueing System :**

*GI* (general independent): the distribution of the *Ai*’s

*G* (general): the distribution of the *Si*’s

Symbols: *M* (exponential), *Ek* (*k*-Erlang), *D* (deterministic times)

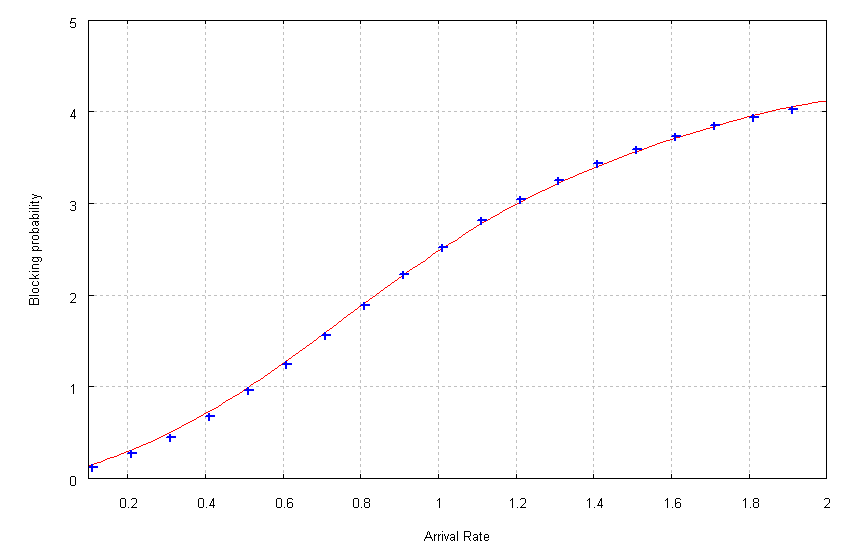
*M/M/1*: a single-server queueing system with exponential interarrival and service times and a FIFO discipline



the utilization factor of the queueing system

**Performance Measures**

**Simulation of a GI/G/1/S queueing system, code help**



**Exercise 1**

Alternative routing problem: simulate the following network of queues, all queues have equal capacity, departures of first queue is fed to the Q2 until it is filled upto threshold = 3 after that remaining customers are routed to the Q3, use the simulation program to find out the mean customers in all three queues for different arrival rates and blocking probability of Q1 and Q3.

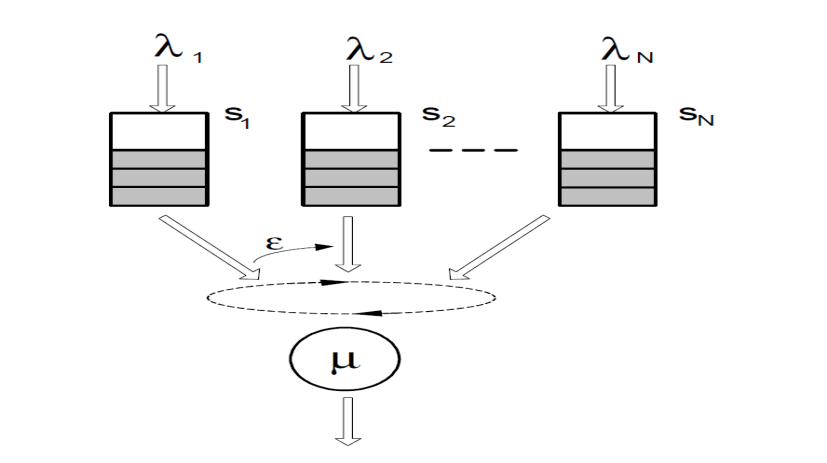
**Exercise 2**

**Round Robin Scheduling**

Queues are polled in a round robin manner.

One customer is served from each queue if not empty.

If queue is empty next queue is polled with switchover.



Follow the pseudo code to write a simulation program for a round robin queueing system , and draw the mean number of customers in a queue and mean blocking for each queue against a uniform varying arrival rate in each queue.

**Lab 5**

Designing an IP Address Scheme

## Source: www.cisco.com

An IP address is an address used in order to uniquely identify a device on an IP network. The address is made up of 32 binary bits, which can be divisible into a network portion and host portion with the help of a subnet mask. The 32 binary bits are broken into four octets (1 octet = 8 bits). Each octet is converted to decimal and separated by a period (dot). For this reason, an IP address is said to be expressed in dotted decimal format (for example, 172.16.81.100). The value in each octet ranges from 0 to 255 decimal, or 00000000 - 11111111 binary.

Here is how binary octets convert to decimal: The right most bit, or least significant bit, of an octet holds a value of 20. The bit just to the left of that holds a value of 21. This continues until the left-most bit, or most significant bit, which holds a value of 27. So if all binary bits are a one, the decimal equivalent would be 255 as shown here:

1 1 1 1 1 1 1 1

128 64 32 16 8 4 2 1 (128+64+32+16+8+4+2+1=255)

Here is a sample octet conversion when not all of the bits are set to 1.

0 1 0 0 0 0 0 1

0 64 0 0 0 0 0 1 (0+64+0+0+0+0+0+1=65)

And this is sample shows an IP address represented in both binary and decimal.

10. 1. 23. 19 (decimal)

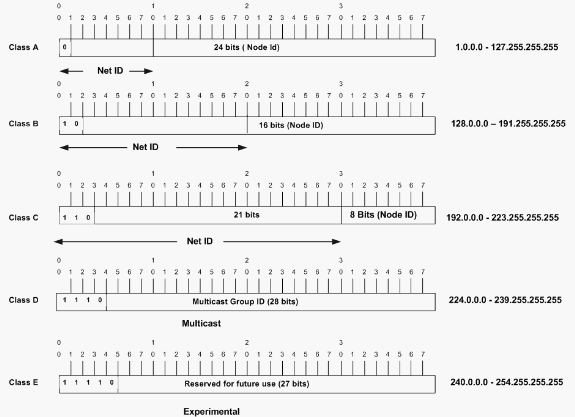
00001010.00000001.00010111.00010011 (binary)

These octets are broken down to provide an addressing scheme that can accommodate large and small networks. There are five different classes of networks, A to E. This document focuses on addressing classes A to C, since classes D and E are reserved and discussion of them is beyond the scope of this document.

**Note:**Also note that the terms "Class A, Class B" and so on are used in this document to help facilitate the understanding of IP addressing and subnetting. These terms are rarely used in the industry anymore because of the introduction of [classless interdomain routing (CIDR)](http://www.cisco.com/en/US/tech/tk365/technologies_tech_note09186a00800a67f5.shtml#cidr).

Given an IP address, its class can be determined from the three high-order bits. [Figure 1](http://www.cisco.com/en/US/tech/tk365/technologies_tech_note09186a00800a67f5.shtml#figone) shows the significance in the three high order bits and the range of addresses that fall into each class. For informational purposes, Class D and Class E addresses are also shown.

**Figure 1**



In a Class A address, the first octet is the network portion, so the Class A example in [Figure 1](http://www.cisco.com/en/US/tech/tk365/technologies_tech_note09186a00800a67f5.shtml" \l "figone) has a major network address of 1.0.0.0 - 127.255.255.255. Octets 2, 3, and 4 (the next 24 bits) are for the network manager to divide into subnets and hosts as he/she sees fit. Class A addresses are used for networks that have more than 65,536 hosts (actually, up to 16777214 hosts!).

In a Class B address, the first two octets are the network portion, so the Class B example in [Figure 1](http://www.cisco.com/en/US/tech/tk365/technologies_tech_note09186a00800a67f5.shtml#figone) has a major network address of 128.0.0.0 - 191.255.255.255. Octets 3 and 4 (16 bits) are for local subnets and hosts. Class B addresses are used for networks that have between 256 and 65534 hosts.

In a Class C address, the first three octets are the network portion. The Class C example in [Figure 1](http://www.cisco.com/en/US/tech/tk365/technologies_tech_note09186a00800a67f5.shtml#figone) has a major network address of 192.0.0.0 - 233.255.255.255. Octet 4 (8 bits) is for local subnets and hosts - perfect for networks with less than 254 hosts.

**Network Masks**

A network mask helps you know which portion of the address identifies the network and which portion of the address identifies the node. Class A, B, and C networks have default masks, also known as natural masks, as shown here:

Class A: 255.0.0.0

Class B: 255.255.0.0

Class C: 255.255.255.0

An IP address on a Class A network that has not been subnetted would have an address/mask pair similar to: 8.20.15.1 255.0.0.0. To see how the mask helps you identify the network and node parts of the address, convert the address and mask to binary numbers.

8.20.15.1 = 00001000.00010100.00001111.00000001

255.0.0.0 = 11111111.00000000.00000000.00000000

Once you have the address and the mask represented in binary, then identifying the network and host ID is easier. Any address bits which have corresponding mask bits set to 1 represent the network ID. Any address bits that have corresponding mask bits set to 0 represent the node ID.

8.20.15.1 = 00001000.00010100.00001111.00000001

255.0.0.0 = 11111111.00000000.00000000.00000000

-----------------------------------

net id | host id

netid = 00001000 = 8

hostid = 00010100.00001111.00000001 = 20.15.1

**Understanding Subnetting**

Subnetting allows you to create multiple logical networks that exist within a single Class A, B, or C network. If you do not subnet, you are only able to use one network from your Class A, B, or C network, which is unrealistic.

Each data link on a network must have a unique network ID, with every node on that link being a member of the same network. If you break a major network (Class A, B, or C) into smaller subnetworks, it allows you to create a network of interconnecting subnetworks. Each data link on this network would then have a unique network/subnetwork ID. Any device, or gateway, connecting *n* networks/subnetworks has *n* distinct IP addresses, one for each network / subnetwork that it interconnects.

In order to subnet a network, extend the natural mask using some of the bits from the host ID portion of the address to create a subnetwork ID. For example, given a Class C network of 204.17.5.0 which has a natural mask of 255.255.255.0, you can create subnets in this manner:

204.17.5.0 - 11001100.00010001.00000101.00000000

255.255.255.224 - 11111111.11111111.11111111.11100000

--------------------------|sub|----

By extending the mask to be 255.255.255.224, you have taken three bits (indicated by "sub") from the original host portion of the address and used them to make subnets. With these three bits, it is possible to create eight subnets. With the remaining five host ID bits, each subnet can have up to 32 host addresses, 30 of which can actually be assigned to a device *since host ids of all zeros or all ones are not allowed* (it is very important to remember this). So, with this in mind, these subnets have been created.

204.17.5.0 255.255.255.224 host address range 1 to 30

204.17.5.32 255.255.255.224 host address range 33 to 62

204.17.5.64 255.255.255.224 host address range 65 to 94

204.17.5.96 255.255.255.224 host address range 97 to 126

204.17.5.128 255.255.255.224 host address range 129 to 158

204.17.5.160 255.255.255.224 host address range 161 to 190

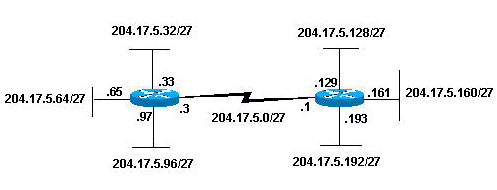
204.17.5.192 255.255.255.224 host address range 193 to 222

204.17.5.224 255.255.255.224 host address range 225 to 254

**Note:**There are two ways to denote these masks. First, since you are using three bits more than the "natural" Class C mask, you can denote these addresses as having a 3-bit subnet mask. Or, secondly, the mask of 255.255.255.224 can also be denoted as /27 as there are 27 bits that are set in the mask. This second method is used with [CIDR](http://www.cisco.com/en/US/tech/tk365/technologies_tech_note09186a00800a67f5.shtml#cidr). With this method, one of these networks can be described with the notation prefix/length. For example, 204.17.5.32/27 denotes the network 204.17.5.32 255.255.255.224. When appropriate the prefix/length notation is used to denote the mask throughout the rest of this document.

The network subnetting scheme in this section allows for eight subnets, and the network might appear as:

**Figure 2**



Notice that each of the routers in [Figure 2](http://www.cisco.com/en/US/tech/tk365/technologies_tech_note09186a00800a67f5.shtml" \l "figtwo) is attached to four subnetworks, one subnetwork is common to both routers. Also, each router has an IP address for each subnetwork to which it is attached. Each subnetwork could potentially support up to 30 host addresses.

This brings up an interesting point. The more host bits you use for a subnet mask, the more subnets you have available. However, the more subnets available, the less host addresses available per subnet. For example, a Class C network of 204.17.5.0 and a mask of 255.255.255.224 (/27) allows you to have eight subnets, each with 32 host addresses (30 of which could be assigned to devices). If you use a mask of 255.255.255.240 (/28), the break down is:

204.17.5.0 - 11001100.00010001.00000101.00000000

255.255.255.240 - 11111111.11111111.11111111.11110000

--------------------------|sub |---

Since you now have four bits to make subnets with, you only have four bits left for host addresses. So in this case you can have up to 16 subnets, each of which can have up to 16 host addresses (14 of which can be assigned to devices).

Take a look at how a Class B network might be subnetted. If you have network 172.16.0.0 ,then you know that its natural mask is 255.255.0.0 or 172.16.0.0/16. Extending the mask to anything beyond 255.255.0.0 means you are subnetting. You can quickly see that you have the ability to create a lot more subnets than with the Class C network. If you use a mask of 255.255.248.0 (/21), how many subnets and hosts per subnet does this allow for?

172.16.0.0 - 10101100.00010000.00000000.00000000

255.255.248.0 - 11111111.11111111.11111000.00000000

-----------------| sub |-----------

You are using five bits from the original host bits for subnets. This allows you to have 32 subnets (25). After using the five bits for subnetting, you are left with 11 bits for host addresses. This allows each subnet so have 2048 host addresses (211), 2046 of which could be assigned to devices.

**Note:**In the past, there were limitations to the use of a subnet 0 (all subnet bits are set to zero) and all ones subnet (all subnet bits set to one). Some devices would not allow the use of these subnets. Cisco Systems devices allow the use of these subnets when the**ip subnet zero** command is configured.

**Examples**

**Sample Exercise 1**

Now that you have an understanding of subnetting, put this knowledge to use. In this example, you are given two address / mask combinations, written with the prefix/length notation, which have been assigned to two devices. Your task is to determine if these devices are on the same subnet or different subnets. You can do this by using the address and mask of each device to determine to which subnet each address belongs.

DeviceA: 172.16.17.30/20

DeviceB: 172.16.28.15/20

**Determining the Subnet for DeviceA:**

172.16.17.30 - 10101100.00010000.00010001.00011110

255.255.240.0 - 11111111.11111111.11110000.00000000

-----------------| sub|------------

subnet = 10101100.00010000.00010000.00000000 = 172.16.16.0

Looking at the address bits that have a corresponding mask bit set to one, and setting all the other address bits to zero (this is equivalent to performing a logical "AND" between the mask and address), shows you to which subnet this address belongs. In this case, DeviceA belongs to subnet 172.16.16.0.

**Determining the Subnet for DeviceB:**

172.16.28.15 - 10101100.00010000.00011100.00001111

255.255.240.0 - 11111111.11111111.11110000.00000000

-----------------| sub|------------

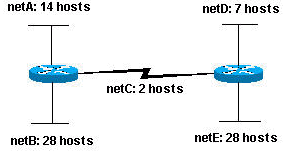
subnet = 10101100.00010000.00010000.00000000 = 172.16.16.0

From these determinations, DeviceA and DeviceB have addresses that are part of the same subnet.

**Sample Exercise 2**

Given the Class C network of 204.15.5.0/24, subnet the network in order to create the network in [Figure 3](http://www.cisco.com/en/US/tech/tk365/technologies_tech_note09186a00800a67f5.shtml#figthree) with the host requirements shown.

**Figure 3**



Looking at the network shown in [Figure 3](http://www.cisco.com/en/US/tech/tk365/technologies_tech_note09186a00800a67f5.shtml" \l "figthree), you can see that you are required to create five subnets. The largest subnet must support 28 host addresses. Is this possible with a Class C network? and if so, then how?

You can start by looking at the subnet requirement. In order to create the five needed subnets you would need to use three bits from the Class C host bits. Two bits would only allow you four subnets (22).

Since you need three subnet bits, that leaves you with five bits for the host portion of the address. How many hosts does this support? 25 = 32 (30 usable). This meets the requirement.

Therefore you have determined that it is possible to create this network with a Class C network. An example of how you might assign the subnetworks is:

netA: 204.15.5.0/27 host address range 1 to 30

netB: 204.15.5.32/27 host address range 33 to 62

netC: 204.15.5.64/27 host address range 65 to 94

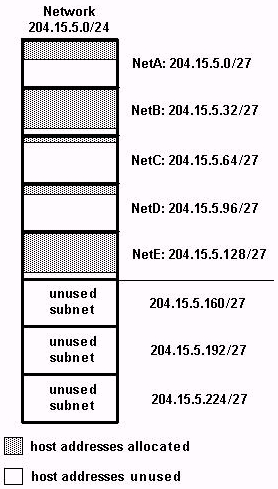
netD: 204.15.5.96/27 host address range 97 to 126

netE: 204.15.5.128/27 host address range 129 to 158

**VLSM Example**

In all of the previous examples of subnetting, notice that the same subnet mask was applied for all the subnets. This means that each subnet has the same number of available host addresses. You can need this in some cases, but, in most cases, having the same subnet mask for all subnets ends up wasting address space. For example, in the [Sample Exercise 2](http://www.cisco.com/en/US/tech/tk365/technologies_tech_note09186a00800a67f5.shtml#ex2) section, a class C network was split into eight equal-size subnets; however, each subnet did not utilize all available host addresses, which results in wasted address space. [Figure 4](http://www.cisco.com/en/US/tech/tk365/technologies_tech_note09186a00800a67f5.shtml#figfour) illustrates this wasted address space.

**Figure 4**



[Figure 4](http://www.cisco.com/en/US/tech/tk365/technologies_tech_note09186a00800a67f5.shtml" \l "figfour) illustrates that of the subnets that are being used, NetA, NetC, and NetD have a lot of unused host address space. It is possible that this was a deliberate design accounting for future growth, but in many cases this is just wasted address space due to the fact that the same subnet mask is being used for all the subnets.

Variable Length Subnet Masks (VLSM) allows you to use different masks for each subnet, thereby using address space efficiently.

**VLSM Example**

Given the same network and requirements as in [Sample Exercise 2](http://www.cisco.com/en/US/tech/tk365/technologies_tech_note09186a00800a67f5.shtml#ex2) develop a subnetting scheme with the use of VLSM, given:

netA: must support 14 hosts

netB: must support 28 hosts

netC: must support 2 hosts

netD: must support 7 hosts

netE: must support 28 host

Determine what mask allows the required number of hosts.

netA: requires a /28 (255.255.255.240) mask to support 14 hosts

netB: requires a /27 (255.255.255.224) mask to support 28 hosts

netC: requires a /30 (255.255.255.252) mask to support 2 hosts

netD\*: requires a /28 (255.255.255.240) mask to support 7 hosts

netE: requires a /27 (255.255.255.224) mask to support 28 hosts

\* a /29 (255.255.255.248) would only allow 6 usable host addresses

therefore netD requires a /28 mask.

The easiest way to assign the subnets is to assign the largest first. For example, you can assign in this manner:

netB: 204.15.5.0/27 host address range 1 to 30

netE: 204.15.5.32/27 host address range 33 to 62

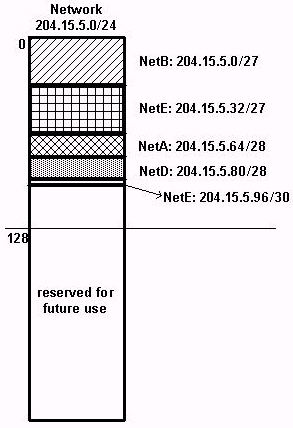
netA: 204.15.5.64/28 host address range 65 to 78

netD: 204.15.5.80/28 host address range 81 to 94

netC: 204.15.5.96/30 host address range 97 to 98

This can be graphically represented as shown in Figure 5:

**Figure 5**



[Figure 5](http://www.cisco.com/en/US/tech/tk365/technologies_tech_note09186a00800a67f5.shtml" \l "figfive) illustrates how using VLSM helped save more than half of the address space.

**CIDR**

Classless Interdomain Routing (CIDR) was introduced to improve both address space utilization and routing scalability in the Internet. It was needed because of the rapid growth of the Internet and growth of the IP routing tables held in the Internet routers.

CIDR moves way from the traditional IP classes (Class A, Class B, Class C, and so on). In CIDR , an IP network is represented by a prefix, which is an IP address and some indication of the length of the mask. Length means the number of left-most contiguous mask bits that are set to one. So network 172.16.0.0 255.255.0.0 can be represented as 172.16.0.0/16. CIDR also depicts a more hierarchical Internet architecture, where each domain takes its IP addresses from a higher level. This allows for the summarization of the domains to be done at the higher level. For example, if an ISP owns network 172.16.0.0/16, then the ISP can offer 172.16.1.0/24, 172.16.2.0/24, and so on to customers. Yet, when advertising to other providers, the ISP only needs to advertise 172.16.0.0/16.

For more information on CIDR, see [RFC 1518](http://www.ietf.org/rfc/rfc1518.txt) leavingcisco.com and [RFC 1519](http://www.ietf.org/rfc/rfc1519.txt) leavingcisco.com.

**Sample Config**

Routers A and B are connected via serial interface.

**Router A**

hostname routera

!

ip routing

!

int e 0

ip address 172.16.50.1 255.255.255.0

!(subnet 50)

int e 1

ip address 172.16.55.1 255.255.255.0

!(subnet 55)

int t 0

ip address 172.16.60.1 255.255.255.0

!(subnet 60)

int s 0

ip address 172.16.65.1 255.255.255.0

!(subnet 65)

!S 0 connects to router B

router rip

network 172.16.0.0

**Router B**

hostname routerb

!

ip routing

!

int e 0

ip address 192.1.10.200 255.255.255.240

!(subnet 192)

int e 1

ip address 192.1.10.66 255.255.255.240

!(subnet 64)

int s 0

ip address 172.16.65.2 (same subnet as router A's s 0)

!Int s 0 connects to router A

router rip

network 192.1.10.0

network 172.16.0.0

**Host/Subnet Quantities Table**

Class B Effective Effective

# bits Mask Subnets Hosts

------- --------------- --------- ---------

1 255.255.128.0 2 32766

2 255.255.192.0 4 16382

3 255.255.224.0 8 8190

4 255.255.240.0 16 4094

5 255.255.248.0 32 2046

6 255.255.252.0 64 1022

7 255.255.254.0 128 510

8 255.255.255.0 256 254

9 255.255.255.128 512 126

10 255.255.255.192 1024 62

11 255.255.255.224 2048 30

12 255.255.255.240 4096 14

13 255.255.255.248 8192 6

14 255.255.255.252 16384 2

Class C Effective Effective

# bits Mask Subnets Hosts

------- --------------- --------- ---------

1 255.255.255.128 2 126

2 255.255.255.192 4 62

3 255.255.255.224 8 30

4 255.255.255.240 16 14

5 255.255.255.248 32 6

6 255.255.255.252 64 2

\*Subnet all zeroes and all ones included. These

might not be supported on some legacy systems.

\*Host all zeroes and all ones excluded.

**Exercise 1**

Company ABC has 100 workstations in their head office and the IT manager would like to assign a subnetwork address for each department. He wants to create 5 subnets using the 192.168.1.0/24 block and assign the zfirst subnetwork to their servers and the rest for their 100 workstations. How will we compute the subnetwork requirement?

***Given:***  
Number of workstations = **100**  
Network address = **192.168.1.0/24**  
Subnets required = **5**

Design the network in Packet tracer and configure all routers, show packet are moving between all subnets using pings from various attached hosts/workstations.

**Lab 6**

Router Configurations

(Static routes, ICMP, CDP, Some Important Commands)

## Source: www.cisco.com

Note: Please take help from provided **manual by Cisco from page no. 994**

Setup a network similar to the exercise in last lab. The following steps are intended to be executed on each router unless specifically instructed otherwise.

* + 1. Run **show ip route** and analyze the output on both routers. It should look like

192.168.1.0/27 is subnetted, 6 subnets

C 192.168.1.32 is directly connected, FastEthernet0/0

C 192.168.1.64 is directly connected, FastEthernet0/1

C 192.168.1.96 is directly connected, FastEthernet1/0

R 192.168.1.128 [120/1] via 192.168.1.222, 00:00:12, Serial0/0

R 192.168.1.160 [120/1] via 192.168.1.222, 00:00:12, Serial0/0

C 192.168.1.192 is directly connected, Serial0/0

* + 1. Run **no router rip** on both routers
    2. Run show ip route and analyze the output

192.168.1.0/27 is subnetted, 4 subnets

C 192.168.1.32 is directly connected, FastEthernet0/0

C 192.168.1.64 is directly connected, FastEthernet0/1

C 192.168.1.96 is directly connected, FastEthernet1/0

C 192.168.1.192 is directly connected, Serial0/0

* + 1. Check now that ping is working or not across the routers
    2. How can this situation be changed so that the hosts can **ping** each other?

Add static routes to each router or run a routing protocol, we have exercised the second option and now we want to add the static routes.

In global configuration mode, add a static route on Router1 to network 192.168.1.128 and 192.168.1.160 and on Router2 to network 192.168.1.32, 192.168.1.64 and 192.168.1.96

Like on Router1 enter the following route for 192.168.1.128

**//** **dest network network mask interface //**

**#ip route 192.168.1.128 255.255.255.224 192.168.1.222**

route for 192.168.1.160

**#ip route 192.168.1.160 255.255.255.224 192.168.1.222**

Follow the same logic to enter all possible routes on both routers.

* + 1. Run **show ip route**
    2. Check by Ping between different host across routers and between networks attached with one router
    3. Run **tracert destination** to check the route between two nodes.
    4. **Creating a network map using CDP** (Take help Cisco Manual page no. 883)

CDP discovers and shows information about directly connected Cisco devices, including routers and switches. CDP only shows information about directly connected Cisco devices.

1. Run **show cdp neighbors** on both routers and analyze the output

It should look like

Capability Codes: R - Router, T - Trans Bridge, B - Source Route Bridge

S - Switch, H - Host, I - IGMP, r - Repeater, P - Phone

Device ID Local Intrfce Holdtme Capability Platform Port ID

Switch Fas 0/1 173 S 2950 Fas 0/1

Switch Fas 0/0 173 S 2950 Fas 0/1

Switch Fas 1/0 173 S 2950 Fas 0/1

RouterB Ser 0/0 173 R C2600 Ser 0/0

1. Run now **show cdp neighbor detail** and analyze the output. Analyze the output.

**Some Important things to know**

**No command** is used to reverse or disable commands e.g

* + - ip domain-lookup
    - no ip domain-lookup
    - router rip
    - no router rip
    - ip address 192.168.1.56 255.255.255.0
    - no ip address

**Getting help**

* IOS has a built-in help facility; use “?” to get a list of possible configuration statements
* “?” after the prompt lists all possible commands:
* router#?
* “<partial command> ?” lists all possible subcommands, e.g.:
* router#show ?
* router#show ip ?
* router#con?
* configure connect
* This is different:
* hostel-rtr#conf ?
* memory Configure from NVRAM
* network Configure from a TFTP network host
* overwrite-network Overwrite NV memory from TFTP... network host
* terminal Configure from the terminal
* <cr>

**This also works in configuration mode:**

* router(config)#ip a?
* accounting-list accounting-threshold accounting-transits address-pool alias as-path
* router(config)#int e0/0
* router(config-if)#ip a?
* access-group accounting address

**Can “explore” a command to figure out the syntax:**

* router(config-if)#ip addr ?
* A.B.C.D IP address
* router(config-if)#ip addr n.n.n.n ?
* A.B.C.D IP subnet mask
* router(config-if)#ip addr n.n.n.n m.m.m.m ?
* secondary Make this IP address a secondary address
* <cr>
* router(config-if)#ip addr n.n.n.n m.m.m.m
* router(config-if)#

**TAB character will complete a partial word**

* hostel-rtr(config)#int**<TAB>**
* hostel-rtr(config)#interface et**<TAB>**
* hostel-rtr(config)#interface ethernet 0
* hostel-rtr(config-if)#ip add**<TAB>**
* hostel-rtr(config-if)#ip address ... n.n.n.n m.m.m.m

**Partial commands can also be used:**

* router#conf t
* router(config)#int e0/0
* router(config-if)#ip addr n.n.n.n

**Command history**

* IOS maintains short list of previously typed commands
* up-arrow or ‘^p’ recalls previous command
* down-arrow or ‘^n’ recalls next command

**Lab 7**

Router Configurations

(Access Control Lists)

## Source: www.cisco.com

Note: Please take help from provided **manual by Cisco from page no. 1133**

**Access Control Lists**

* Access Control Lists used to implement security in routers
  + powerful tool for network control
  + filter packets flow in or out of router interfaces
  + restrict network use by certain users or devices
  + deny or permit traffic
* Rules Followed When Traffic Is Compared To An Access Control List
  + Is done in sequential order; line 1, line 2, line 3 e.t.c
  + Is compared with the access list until a match is made; then NO further comparisons are made
  + There is an implicit “deny” at the end of each access list; if a packet does not match in the access list, it will be discarded
* Using Access Control Lists
* Standard IP Access Lists (1 - 99)
  + simpler address specifications
  + generally permits or denies entire protocol suite
* Extended IP Access Lists (100 - 199)
  + more complex address specification
  + generally permits or denies specific protocols
* Syntax of using access lists
  + Standard IP Access List Configuration Syntax
    - access-list access-list-number {permit | deny} source {source-mask}
    - ip access-group access-list-number {in | out}
  + Extended IP Access List Configuration Syntax
    - access-list access-list-number {permit | deny} protocol source {source-mask} destination {destination-mask}
    - ip access-group access-list-number {in | out}
* Where to place access control lists
  + Place **Standard IP** access list close to **destination**
  + Place **Extended IP** access lists close to the **source** of the traffic you want to manage
* Using Wild Cards
  + Are used with access lists to specify a host, network or part of a network
  + To specify an address range, choose the next largest block size e.g.
* to specify 34 hosts, you need a 64 block size
* to specify 18 hosts, you need a 32 block size
* to specify 2 hosts, you need a 4 block size
* Wild Card Masks
  + Are used with the host/network address to tell the router a range of addresses to filter
    - Examples:
    - to specify a host: 81.199.108.1 0.0.0.0
    - to specify a small subnet: 81.199.108.8 – 81.199.108.15 (would be a /29)
    - Block size is 8, and wildcard is always one number less than the block size
    - Cisco access list then becomes: 81.199.108.8 0.0.0.7
  + to specify all hosts on a Class C network: 81.199.108.0 0.0.0.255
* **What are wild card masks**
  + Short cut method to a quick calculation of a network subnet to wildcard:
    - 255 – {netmask bits on subnet mask}
    - to create wild card mask for 81.199.108.160 255.255.255.240
    - 81.199.108.160 0.0.0.15 {255 – 240}
    - to create wild card mask for 81.199.108.0 255.255.252.0

81.199.108.0 0.0.3.255

* **Examples**
  + Router(config)#Access-list access-list-number {permit|deny}{test conditions}
  + Router(config)#{protocol} access-group access-list-number
    - e.g check for IP subnets 81.199.108.80 to 81.199.108.95

81.199.108.80,

Address and Wildcard Mask: **81.199.108.80 0.0.0.15**

* Wildcard bits indicate how to check corresponding address bit
  + - 0=check or match
    - 1=ignore
* Matching Any IP Address
  + - 255.255.255.255
    - or abbreviate the expression using the keyword any
* Matching a specific host
  + - 81.199.108.8 0.0.0.0
    - or abbreviate the wildcard using the IP address preceded by the keyword host

Lab Exercise

Considering Network designed in previous labs as shown

Implement the following access control lists on the two available routers:

* + - 1. Create an access list to deny hosts of network 192.168.1.32 to restrict the access to the other two networks 192.168.1.64 and 192.168.1.96

**(config)#access-list 1 deny 192.168.1. 32 0 0.0.0.31**

access-list 1 permit any any

Interface fa0/1

ip access-group 1 out

Interface fa1/0

ip access-group 1 out

Task 1

Configure an access-list disabling anyone TELNET to R1 and all devices behind it (R2) if the traffic is originated from Internet (here: SP). All other traffic should be permitted.

R1 Configuration:

R1(config)#access-list 100 deny tcp any any eq telnet

R1(config)#access-list 100 permit ip any any

R1(config)#int s0/1

R1(config-if)#ip access-group 100 in

R1(config-if)#end

**Verification:**

For verification,

From R2 to R1 send ping and check response

Then from R2 to R1 try to have a telnet session and check response

Task 2

On R1 remove the previous ACL and configure a new one allowing only HTTP access to 192.168.1.32/27. All other traffic should be discarded.

First attach a server in the network 192.168.1.32 and give it an IP address 192.168.1.35, and turn on HTTP on it, which is by default running in packet tracer when you choose a server.

(config)#no access-list 100

R1(config)#int s0/1

R1(config-if)#no ip access-group 100 in

R1(config-if)#exit

R1(config)#

R1(config)#access-list 101 permit tcp any host 192.168.1.35 eq www

R1(config)#int s0/1

R1(config-if)#ip access-group 101 in

R1(config-if)#

All traffic is denied by explicit deny in the end of the list and only

Task 3

Attach a web-server to the switch of network 192.168.1.64

Assign it a valid ip address from the pool and then deny hosts of network 192.168.1.32 to have an access to this web-server

R1(config)#int s0/1

R1(config-if)#no ip access-group 101 in

R1(config-if)#exit

R1(config)#no access-list 101

R1(config)#access-list 101 deny tcp 192.168.1. 32 0 0.0.0.31 192.168.1. 64 0 0.0.0.31 eq 80

R1(config)#access-list 101 permit ip 192.168.1.32 0 0.0.0.31 0.0.0.0 255.255.255.255

R1(config)#Interface fa0/1

R1(config-if)#ip access-group 101 out

**Lab 8**

Network Simulations (Using NS2)

## Source: USC/ISI

**What is NS**

* Started as REAL in 1989
* Discrete event, packet level simulator
* Written in C++ with Otcl frontend
* Wired, wireless and emulation modes
* Link layer and up for most wired

**Platforms**

* + Most UNIX and UNIX-like systems
  + Linux
  + FreeBSD
  + SunOS/Solaris
  + HP/SGI (with some tweaking)
  + Windows 95/98/NT/ME/2000
  + Tips on build available
  + However validation tests don’t work

**Architecture of NS**

* Object-oriented (C++ and Otcl)
  + Algorithms over large data sets, per packet handling in C++
  + Configuration, “one-time” stuff in Otcl
  + Fast to run, quick to re-configure
* Fine grained object composition
* C++ for “data”
* Per packet action
* OTcl for control
* Configuration, “one-time” task
* Compromise between composibility and speed
* Learning and debugging

**Basic TCL**

set a 43

set b 27

proc test { a b } {

set c [expr $a + $b]

set d [expr [expr $a - $b] \* $c]

for {set k 0} {$k < 10} {incr k} {

if {$k < 5} {

puts “k < 5, pow = [expr pow($d, $k)]”

} else {

puts “k >= 5, mod = [expr $d % $k]”

}

}

**Basic OTCL**

Class Mom

Mom instproc greet {} {

$self instvar age\_

puts “$age\_ years old mom: How are you doing?”

}

Class Kid -superclass Mom

Kid instproc greet {} {

$self instvar age\_

puts “$age\_ years old kid: What’s up, dude?”

**set** mom [**new** Mom]

$mom **set** age\_ 45

**set** kid [**new** Kid]

$kid **set** age\_ 15

$mom greet

$kid greet

**Hello World in NS**

%ns

% set ns [new Simulator]

% $ns at 1 “puts \“Hello World!\””

% $ns at 1.5 “exit”

% $ns run

**Event Driven Simulation**

* Scheduler – main controller of events
* Scheduler clock - simulator virtual time
* [$ns\_ now] returns the current simulator time
* Event queue - holds events in the order of their firing times
* Events have a firing time and a handler function
* Two types of events possible – packets and “at-events”

**Discrete Event Scheduler**

Four types of scheduler:

* + - List: simple linked list, order-preserving, O(N)
    - Heap: O(logN)
    - Calendar: hash-based, fastest, default, O(1)
    - Real-time: subclass of list, sync with real-time, O(N)

**Procedure of NS**

* Create the event scheduler
* [Turn on tracing]
* Create network
* Setup routing
* Insert errors
* Create transport connection
* Create traffic

**Create an Event Scheduler**

* Create event scheduler
  + - * set ns [new Simulator]
* Schedule events
  + - * $ns at <time> <event>
      * <event>: any legitimate ns/tcl commands
* Start scheduler
  + - * $ns run

**Tracing**

* Trace packets on all links
* $ns trace-all [open test.out w]
  + - <event> <time> <from> <to> <pkt> <size> -- <fid> <src> <dst> <seq> <attr>
    - + 1 0 2 cbr 210 ------- 0 0.0 3.1 0 0
* Trace packets on all links in nam format
  + - $ns namtrace-all [open test.nam w]
    - $ns namtrace-all-wireless [open wtest.nam w]
* Turn on tracing on specific links
* $ns trace-queue $n0 $n1
* $ns namtrace-queue $n0 $n1
* Trace-all commands must appear immediately after creating scheduler
* Event tracing
  + $ns eventtrace-all [$file]
  + Add eventtrace *after* trace-all as trace-all file is used as default
  + Example script: ~ns/tcl/ex/tcp-et.tcl

**Creating NS Topology**

* Nodes
  + set n0 [$ns node]
  + set n1 [$ns node]
* Links and queuing
  + $ns duplex-link $n0 $n1 <bandwidth> <delay> <queue\_type>
  + <queue\_type>: DropTail, RED, CBQ, FQ, SFQ, DRR
  + $ns duplex-link $n0 $n1 5Mb 2ms DropTail

**Inserting Errors**

* Creating Error Module
  + set loss\_module [new ErrorModel]
  + $loss\_module set rate\_ 0.01
  + $loss\_module unit pkt
  + $loss\_module ranvar [new RandomVariable/Uniform]
  + $loss\_module drop-target [new Agent/Null]
* Inserting Error Module
  + $ns lossmodel $loss\_module $n0 $n1

**Network Dynamics**

* Link failures
  + Hooks in routing module to reflect routing changes
* Four models
  + $ns rtmodel Trace <config\_file> $n0 $n1
  + $ns rtmodel Exponential {<params>} $n0 $n1
  + $ns rtmodel Deterministic {<params>} $n0 $n1
  + $ns rtmodel-at <time> up|down $n0 $n1
* Parameter list
  + [<start>] <up\_interval> <down\_interval> [<finish>]

**Setup Routing**

* Unicast
  + $ns rtproto <type>
  + <type>: Static, Session, DV, LS, Manual or hierarchical
* Multicast
  + $ns multicast (right after [new Simulator])
  + $ns mrtproto <type>
* <type>: CtrMcast, DM, ST, BST

**Creating TCP Connections**

* One-way TCP sending agent [Tahoe, Reno, NewReno, Sack, Vegas and Fack]
  + set tcp [new Agent/TCP]
  + set tcpsink [new Agent/TCPSink]
  + $ns attach-agent $n0 $tcp
  + $ns attach-agent $n1 $tcpsink
  + $ns connect $tcp $tcpsink

**Creating Traffic on the top of TCP**

* FTP
  + set ftp [new Application/FTP]
  + $ftp attach-agent $tcp
* Telnet
  + set telnet [new Application/Telnet]
  + $telnet attach-agent $tcp

**Fig: Packet flow**

**Generic Script Structure Summary**

set ns [new Simulator]

# [Turn on tracing]

# Create topology

# Setup packet loss, link dynamics

# Create routing agents

# Create:

# - multicast groups

# - protocol agents

# - application and/or setup traffic sources

# Post-processing procs

# Start simulation

**Example TCP Network (A Simple Scenario)**

**Step 1:**

* Scheduler & tracing
  + Create scheduler
    - Set ns [new Simulator]
  + Turn on tracing
    - set f [open out.tr w]
    - $ns trace-all $f
    - Set nf [open out.nam w]
    - $ns namtrace-all $nf

**Step 2**

* Create topology
  + create nodes
    - set n0 [$ns node]
    - set n1 [$ns node]
    - set n3 [$ns node]
    - set n4 [$ns node]

**Step 3**

* create links
  + $ns duplex-link $n0 $n1 5Mb 2ms DropTail
  + $ns duplex-link $n1 $n2 1.5Mb 10ms DropTail
  + $ns duplex-link $n2 $n3 5Mb 2ms DropTail
  + $ns queue-limit $n1 $n2 25
  + $ns queue-limit $n2 $n1 25

**Step 4**

* Create TCP agents
  + - * set tcp [new Agent/TCP]
      * set sink [new Agent/TCPSink]
      * $ns attach-agent $n0 $tcp
      * $ns attach-agent $n3 $sink
      * $ns connect $tcp $sink

**Step 5**

* Attach traffic
  + - * set ftp [new Application/FTP]
      * $ftp attach-agent $tcp
      * #start application traffic
      * $ns at 1.1 “$ftp start”

**Step 6**

* End of simulation wrapper (as usual)
  + $ns at 2.0 “finish”
  + Proc finish {}
    - {
      * global ns f nf
      * close $f
      * close $nf
      * puts “Running nam…”
      * exec nam out.nam &
      * exit 0
      * }
  + $ns run

**Visualization Tools**

Nam-1 (Network AniMator Version 1)

* + - * Packet-level animation
      * Well-supported by ns

Xgraph

* Convert trace output into xgraph format

**NAM interface**

* Color
* Node manipulation
* Link manipulation
* Topology layout
* Protocol state
* Misc

NAM interface (Color)

* Color mapping
  + $ns color 40 red
  + $ns color 41 blue
  + $ns color 42 chocolate
  + Color ↔ flow id association
  + $tcp0 set fid\_ 40 ;# red packets
  + $tcp1 set fid\_ 41 ;# blue packets
  + Color (Nodes)
    - node color red
  + Shape (can’t be changed after sim starts)
    - $node shape box ;# circle, box, hexagon
  + Marks (concentric “shapes”)
    - $ns at 1.0 “$n0 add-mark m0 blue box”
    - $ns at 2.0 “$n0 delete-mark m0”
  + Label (single string)
    - $ns at 1.1 “$n0 label \”web cache 0\””
  + Color (links)
    - $ns duplex-link-op $n0 $n1 color "green"
  + Label
    - $ns duplex-link-op $n0 $n1 label "abced"
  + Dynamics (automatically handled)
    - $ns rtmodel Deterministic {2.0 0.9 0.1} $n0 $n1
  + Asymmetric links not allowed

**Topology Layout NAM**

* “Manual” layout: specify everything
  + $ns duplex-link-op $n(0) $n(1) orient right
  + $ns duplex-link-op $n(1) $n(2) orient right-up
  + $ns duplex-link-op $n(2) $n(3) orient down
  + $ns duplex-link-op $n(3) $n(4) orient 60deg
* If anything missing 🡪 automatic layout

**Resources**

* Ns distribution download
  + http://www.isi.edu/nsnam/ns/ns-build.html
* Installation problems and bug-fix
  + http://www.isi.edu/nsnam/ns/ns-problems.html
* Ns-users mailing list
  + [Ns-users@isi.edu](mailto:Ns-users@isi.edu)
  + See <http://www.isi.edu/nsnam/ns/ns-lists.html>
  + Archives from above URL

**Lab 9**

Term Project (Socket Programming)

## Following link may be used for a good brief introduction on the topic of socket programming

## http://www.buyya.com/java/Chapter13.pdf

The project is a semester-long activity that is performed in Lab groups. The goal is to introduce students how to implement / build small network applications which are required to be tested , demonstrated and evaluated. A brief overview / survey is the part of every project.

**Project Grading**[**\***](http://www.cs.wmich.edu/~alfuqaha/Spring07/cs6030/projects.html#_ftn1)**(20 Marks Max.):**

1. Proposal
2. Checkpoint
3. Final Documented Software & Report
4. Demo and Presentation

**Example Ideas for Projects**

<http://www.sourcecodesworld.com/project-bank/>

<http://www.projectideas.co.in/resources/Java-project-list.pdf>

1. References to figures and sections are for the 6th edition of our text, *Computer Networks, A Top-down Approach, 6th ed., J.F. Kurose and K.W. Ross, Addison-Wesley/Pearson, 2012.* [↑](#footnote-ref-1)
2. Download the zip file [http://gaia.cs.umass.edu/wireshark-labs/wireshark-traces.zip](http://gaia.cs.umass.edu/ethereal-labs/ethereal-traces.zip) and extract the file http-ethereal-trace-1. The traces in this zip file were collected by Wireshark running on one of the author’s computers, while performing the steps indicated in the Wireshark lab. Once you have downloaded the trace, you can load it into Wireshark and view the trace using the *File* pull down menu, choosing *Open*, and then selecting the http-ethereal-trace-1 trace file. The resulting display should look similar to Figure 1. (The Wireshark user interface displays just a bit differently on different operating systems, and in different versions of Wireshark). [↑](#footnote-ref-2)
3. References to figures and sections are for the 6th edition of our text, *Computer Networks, A Top-down Approach, 6th ed., J.F. Kurose and K.W. Ross, Addison-Wesley/Pearson, 2012.* [↑](#footnote-ref-3)
4. Download the zip file <http://gaia.cs.umass.edu/wireshark-labs/wireshark-traces.zip> and extract the file tcp-ethereal-trace-1. The traces in this zip file were collected by Wireshark running on one of the author’s computers, while performing the steps indicated in the Wireshark lab. Once you have downloaded the trace, you can load it into Wireshark and view the trace using the *File* pull down menu, choosing *Open*, and then selecting the tcp-ethereal-trace-1 trace file. [↑](#footnote-ref-4)
5. What do we mean by “annotate”? If you hand in a paper copy, please highlight where in the printout you’ve found the answer and add some text (preferably with a colored pen) noting what you found in what you ‘ve highlight. If you hand in an electronic copy, it would be great if you could also highlight and annotate. [↑](#footnote-ref-5)
6. The TCP segments in the tcp-ethereal-trace-1 trace file are all less that 1460 bytes. This is because the computer on which the trace was gathered has an Ethernet card that limits the length of the maximum IP packet to 1500 bytes (40 bytes of TCP/IP header data and 1460 bytes of TCP payload). This 1500 byte value is the standard maximum length allowed by Ethernet. If your trace indicates a TCP length greater than 1500 bytes, and your computer is using an Ethernet connection, then Wireshark is reporting the wrong TCP segment length; it will likely also show only one large TCP segment rather than multiple smaller segments. Your computer is indeed probably sending multiple smaller segments, as indicated by the ACKs it receives. This inconsistency in reported segment lengths is due to the interaction between the Ethernet driver and the Wireshark software. We recommend that if you have this inconsistency, that you perform this lab using the provided trace file. [↑](#footnote-ref-6)