Jason Lai
204995126
Dis 1A

CS 118
April 16, 2020

Homework 2

1 Problem 1

How does the web server (e.g., Amazon) identify users when you do the Internet shopping? Please briefly explain how it works with HTTP protocol step by step.

The web server identifies users when you do Internet shopping using cookies. It works with the HTTP protocol as follows: first, the user sends an HTTP request to the web server to get a web page. Upon receiving the request, the server then generates a unique number and creates an entry in the server's backend database indexed by this number. The server then sends an HTTP response back to the user with a line that says `Set-Cookie:` followed by the unique number in the cookie header field of the response. Upon receiving the server's response, the user's browser will put this unique number into its cookie file. Then, while the user is Internet shopping, for each HTTP request the user makes to the web server to get a new web page, the user's browser will find the associated cookie for the user and this website, and will send an HTTP request to the server with the line `Cookie:` followed by the number in the cookie header field of the request. Since the cookie is now sent in the HTTP requests to the server, this cookie serves as an identifier for the user whenever the user makes a request to the server, thereby allowing the server to identify users and track their activity when they are doing things like Internet shopping.

2 Problem 2

Suppose within your Web browser you click on a link to obtain a Web page from a web server *S*. The web page is a HTML file and the HTML file further contains references to 9 small JPEG files on the same server *S*. However, *S*'s IP address is not cached in your local host, so a DNS lookup is required. Suppose that n DNS servers are visited by your browser before you get *S*'s IP address. Let RTT_1, RTT_2, ..., RTT_n denote the RTTs (round-trip time) of visiting each of the n DNS server and RTT_0 denote the RTT between the local host and *S*. If we ignore file transmission time for DNS responses, HTML file, and JPEG files, how much time elapses from when the client clicks on the link until the client receives all objects with:

(a) Non-persistent HTTP with no parallel TCP connections?

(b) Non-persistent HTTP with the browser configured for 5 parallel connections?

(c) Persistent HTTP with no parallel TCP connections?

(d) Persistent HTTP with the browser configured for arbitrarily many parallel connections?

a) The time that elapses when using non-persistent HTTP with no parallel TCP connections is $20RTT_0 + (RTT_1 + ... + RTT_n)$.

b) The time that elapses when using non-persistent HTTP with 5 parallel TCP connections is $6RTT_0 + (RTT_1 + ... + RTT_n)$.

c) The time that elapses when using persistent HTTP with no parallel TCP connections is $11RTT_0 + (RTT_1 + ... + RTT_n)$.

d) The time that elapses when using persistent HTTP with arbitrarily many parallel connections is $3RTT_0 + (RTT_1 + ... + RTT_n)$.

Work:

a)
Since the local host has to query every DNS server to get the IP address, we have $RTT_1 + RTT_2 + ... + RTT_n$ time to get the IP address from the domain name. After, we assume the local host caches the IP address.

$2RTT_0$ time to establish a TCP connection and get the base HTML file.

Since it's non-persistent with no parallelism, we open and close a TCP connection for every

referenced object: $9 \times 2RTT_0 = 18RTT_0$.

$2RTT_0 + 18RTT_0 + RTT_1 + RTT_2 + ... + RTT_n = 20RTT_0 + (RTT_1 + ... + RTT_n)$

b)
Since the local host has to query every DNS server to get the IP address, we have $RTT_1 + RTT_2 + ... + RTT_n$ time to get the IP address from the domain name. After, we assume the local host caches the IP address.

$2RTT_0$ time to establish a TCP connection and get the base HTML file.

Non-persistent HTTP with 5 parallel connections means we can establish 5 concurrent TCP connections at a time, so we $2RTT_0$ for the first 5 referenced JPEG files, and another $2RTT_0$ for the remaining 4: $2RTT_0 + 2RTT_0 = 4\ RTT_0$.

$2RTT_0 + 4RTT_0 + RTT_1 + RTT_2 + ... + RTT_n = 6RTT_0 + (RTT_1 + ... + RTT_n)$

c)
Since the local host has to query every DNS server to get the IP address, we have $RTT_1 + RTT_2 + ... + RTT_n$ time to get the IP address from the domain name. After, we assume the local host caches the IP address.

$2RTT_0$ time to establish a TCP connection and get the base HTML file.

Since we are using persistent HTTP, the TCP connection remains open when we transfer the 9 referenced JPEG files, so we have $9 \times RTT_0$ to request each file and get the response from the server.

$2RTT_0 + 9RTT_0 + RTT_1 + RTT_2 + ... + RTT_n = 11RTT_0 + (RTT_1 + ... + RTT_n)$

d)
Since the local host has to query every DNS server to get the IP address, we have $RTT_1 + RTT_2 + ... + RTT_n$ time to get the IP address from the domain name. After, we assume the local host caches the IP address.

$2RTT_0$ time to establish a TCP connection and get the base HTML file.

Since we are using persistent HTTP, the TCP connection remains open after the first HTTP request. Furthermore, since our browser is configured for arbitrarily many parallel connec-

tions, then all 9 referenced JPEG objects can be transferred concurrently, so we have one RTT_0 for all 9 referenced objects to be transferred. Notice we don't have to open a new TCP connection for each parallel connection, since our initial HTTP request's TCP connection is still open.

$2RTT_0 + RTT_0 + RTT_1 + RTT_2 + ... + RTT_n = 3RTT_0 + (RTT_1 + ... + RTT_n)$

3 Problem 3

How does SMTP marks the end of a message body? How about HTTP? Can HTTP use the same method as SMTP to mark the end of the message body?

SMTP marks the end of a message body with a line consisting of a single period or more formally, `CRLF.CRLF`, where `CR` is carriage return \r and `LF` is line feed \n. HTTP, on the other hand, differs depending on whether the message has an entity body or not. In HTTP requests, usually the entity body field of the message is left empty (unless it's a `POST`), so the message will end after the header lines, and is marked by a line with just a carriage return followed by a line feed. For an HTTP response message, the entity body will typically contain the object that was requested, so a header line called `Content-Length:` is used to indicate the number of bytes in the object being sent in the entity body; by specifying the exact size of the object, this is the way that HTTP "marks" the end of a message body with an object.

No, HTTP cannot use the same method as SMTP to mark the end of the message body, as the contents of the message body for SMTP and HTTP may differ. An SMTP message body is entirely made up of 7-bit ASCII, so it can use the single line with a period to mark the end of the message. However, the entity body in HTTP messages can be of various types, as indicated by the header line `Content-Type:`, and this includes text of a different encoding than ASCII like UTF-8 or even just binary data. Binary data, in particular, makes it infeasible to end HTTP messages the same way SMTP does, as the object's binary data could contain bytes that match a line with a single period, so the message may be mistakenly marked to end before the actual end of the message. Therefore, it's better to instead use a header field specifying the size of the entity body to account for the different types of objects being sent.

4 Problem 4

Suppose your department has a local DNS server for all computers in the department.

(a) Suppose you are an ordinary user (i.e., not a network/system administrator). Can you determine if an external Web site was likely accessed from a computer in your department a couple of seconds ago? Explain.

(b) Now suppose you are a system administrator and can access the caches in the local DNS servers of your department. Can you propose a way to roughly determine the Web servers (outside your department) that are most popular among the users in your department? Explain.

a) Yes, you can use the command: `dig` (Domain Information Groper) to query to local DNS server for the specific hostname; `dig` will then display the query time. If the Web site was accessed from a computer in the department a couple of seconds ago, then its record would be cached in the local DNS server, so `dig` would report a query time of around 0ms. Otherwise, if the Web site's record was not cached in the server, then the query time should be much larger, as without a cached record, the local DNS server will do a DNS lookup.

b) To determine the Web servers that are most popular among the users in the department, you can simply look at and record the caches of all of the local DNS servers over a period of time. The IP addresses of web servers that are popular in the department will show up the most frequently in the caches of the local DNS servers, therefore those Web servers will be more frequently seen across all of the data you record of the caches periodically.

5 Problem 5

Consider distributing a file of $F = 15$ Gbits to N peers. The server has an upload rate of $u_s = 30$ Mbps, and each peer has a download rate of $d_1 = 2$ Mbps and an upload rate of u. For N = 10 and 100, and $u = 300$ Kbps and 2 Mbps, prepare a chart giving the minimum distribution time for each of the combinations of N and u for both client-server distribution and P2P distribution (i.e., there are 8 distribution time values in total). In this problem, we assume $1K = 1 \times 10^3$, $1M = 1 \times 10^6$, $1G = 1 \times 10^9$.

Please fulfill your answers into the following two charts and briefly explain how you get them.

Client-Server distribution

u/N	10	100
300Kbps	7500 seconds	50000 seconds
2Mbps	7500 seconds	50000 seconds

To get the client-server distribution chart, for each value of N peers, I simply calculate the maximum of the file size divided by the rate at which the server uploads for N peers and the file size divided by the download rate for one peer i.e $max\{NF/u_s, F/d_{min}\}$. Note that the download rate for all peers is the same, and the upload rate does not matter, since the server is uploading the file to all N peers, so no peer needs to upload.

P2P distribution

u/N	10	100
300Kbps	7500 seconds	25000 seconds
2Mbps	7500 seconds	7500 seconds

To get the P2P distribution chart, I calculate the max of the file size divided by the server upload rate, the file size divided by the download rate for one peer, and the number of peers N times the file size divided by the sum of the upload rate of all N peers and the server upload rate i.e. $max\{F/u_s, F/d_{min}, NF/(u_s + \mathrm{S}u_i)\}$; I do this according to each value of N peers and each value of u for the upload rates of these N peers. Here, since distribution of the file is peer-to-peer, the upload rates of the peers is relevant.

Work:

Client-Server
$F = 15 \times 10^9$ bits

$u_s = 30 \times 10^6$ bps
$d_1 = 2 \times 10^6$ bps
$D_{\text{c-s}} \geq max\{NF/u_s, F/d_{min}\}$

$N = 10$
$D_{\text{c-s}} \geq max\{10(15 \times 10^9/30 \times 10^6), 15 \times 10^9/2 \times 10^6\} = max\{5000, 7500\} = 7500$ seconds

$N = 100$
$D_{\text{c-s}} \geq max\{100(15 \times 10^9/30 \times 10^6), 15 \times 10^9/2 \times 10^6\} = max\{50000, 7500\} = 50000$ seconds

P2P
$F = 15 \times 10^9$ bits
$u_s = 30 \times 10^6$ bps
$d_1 = 2 \times 10^6$ bps
$u_1 = 300 \times 10^3$ bps
$u_2 = 2 \times 10^6$ bps
$D_{\text{P2P}} \geq max\{F/u_s, F/d_{min}, NF/(u_s + \text{S}u_i)\}$

$N = 10$
$D_{\text{P2P,1}} \geq max\{15 \times 10^9/30 \times 10^6, 15 \times 10^9/2 \times 10^6, 10[15 \times 10^9/(30 \times 10^6 + 10(300 \times 10^3)]\} = max\{500, 7500, 4545.45\} = 7500$ seconds
$D_{\text{P2P,2}} \geq max\{15 \times 10^9/30 \times 10^6, 15 \times 10^9/2 \times 10^6, 10[15 \times 10^9/(30 \times 10^6 + 10(2 \times 10^6)]\} = max\{500, 7500, 3000\} = 7500$ seconds

$N = 100$
$D_{\text{P2P,1}} \geq max\{15 \times 10^9/30 \times 10^6, 15 \times 10^9/2 \times 10^6, 100[15 \times 10^9/(30 \times 10^6 + 100(300 \times 10^3)]\} = max\{500, 7500, 25000\} = 25000$ seconds
$D_{\text{P2P,2}} \geq max\{15 \times 10^9/30 \times 10^6, 15 \times 10^9/2 \times 10^6, 100[15 \times 10^9/(30 \times 10^6 + 100(2 \times 10^6)]\} = max\{500, 7500, 6521.73913\} = 7500$ seconds