Report

Home Assignment, CCOM38

Dennis Bennhage, bennhage@student.chalmers.se

Hampus Lidin, lidin@student.chalmers.se

May 21st, 2015

Task 1 :

a) When running the `ipconfig` command, you get all the information about the different interfaces at the host computer. Running the command in the Chalmers network, we get the following information about the WLAN interface:

```
> ifconfig

<output omitted>

Wireless LAN adapter Wireless Network Connection:

   Connection-specific DNS Suffix  . : eduroam.chalmers.se
   Link-local IPv6 Address . . . . . : fe80::6046:472a:63f5:468e%11
   IPv4 Address. . . . . . . . . . . : 129.16.187.2
   Subnet Mask . . . . . . . . . . . : 255.255.240.0
   Default Gateway . . . . . . . . . : 129.16.176.1

<output omitted>
```

The `IPv4 Address` field shows information about the IPv4 configuration for that interface. The unicast IP-address for the local network is then `129.16.187.2/20`, with the broadcast address `129.16.176.1/20`.

b) The hostname can be found using the `nslookup` command on the IP-address obtained in the previous task:

```
> nslookup 129.16.187.2

Server:   res1.chalmers.se
Address:  129.16.1.53

Name:   dhcp-187002.eduroam.chalmers.se
Address:  129.16.187.2
```

This tells us that the hostname for the IP-address `129.16.1.53` is `dhcp-187002.eduroam.chalmers.se`, and was given by the name server `res1.chalmers.se`.

Task 2 :

a) When using `tracert` for the website `www.uio.no`, we get the following results:

```
Tracing route to www.uio.no [129.240.8.200]
over a maximum of 30 hops:

  1    <1 ms    <1 ms    <1 ms  3.home [192.168.1.1]
  2     *        *        *     Request timed out.
  3   296 ms   419 ms   354 ms  10.66.130.65
  4   390 ms   408 ms   412 ms  172.18.4.78
  5   405 ms   409 ms   409 ms  172.18.72.82
  6   406 ms   333 ms   280 ms  172.18.8.105
  7    98 ms    96 ms    98 ms  10.66.54.133
  8   272 ms   414 ms   406 ms  sejar0001—rc2.ip—only.net [213.132.96.193]
  9   620 ms   408 ms   419 ms  sejar0001—rc5.ip—only.net [62.109.44.158]
 10   419 ms   409 ms   613 ms  62.109.44.214
 11   120 ms    55 ms    55 ms  oslo—gw1.uninett.no [193.156.90.1]
 12   228 ms    81 ms    52 ms  uio—gw8.uio.no [128.39.65.18]
 13   111 ms   111 ms    97 ms  mrom—gw2.uio.no [129.240.25.18]
 14    55 ms    66 ms    56 ms  www.uio.no [129.240.8.200]

Trace complete.
```

The route stretches from a home network to the University of Oslo. The first hop is the source location that `tracert` was issued. At the second hop, we can see that the intermediate device didn't respond, so the request timed out. The device has likely blocked any incoming `tracert` requests, which is rather common to do, to prevent possible *denial-of-service* (DoS) attacks. The high delays in hops 3 through 6 could be because of higher prioritized traffic being routed first, leaving lower prioritized traffic to wait in queue. At hop 7 the delays deminishes. In the next three hops, we can see another increase in packet delays. The host name `sejar0001-rc2.ip-only.net` belongs to a company in the city of Stockholm. At hops 11 and 12, the route has stretched down to Kastrup in Denmark, and at hops 13 and 14 the final portion of the route has been laid out towards the web server.

b) `traceroute`, as the name suggests, traces the route of an IP-packet over the Internet. It is usually implemented using the *User Datagram Protocol* (UDP), to transport the packet from the host computer to a specified destination. UDP maintains the route for all packets that it sends, which is essential in order for `traceroute` to work.

`traceroute` initially sets the maximum `Time To Live` (`TTL`) value to 1 in the IP-header. Then after every three packets it sends, the max `TTL` value increments by 1. The first three packets will always stop at the IP-configured interface of the host computer first. Therefore after packets immediately have left the host, the `TTL` will always be 1 less than the max `TTL`. When the first router on the route then receives the first packet, it will decrement the `TTL` value by 1, resulting in a value of 0. This means that the packet can not be forwarded any further and an *Internet Control Message Protocol* (ICMP) message is generated. This message goes all the way back to the source. In the mean time, `traceroute` started counting the time from when the packet was sent, and when the ICMP reply for that packet arrives, it stops the timer. The same scenario occurs for the next two packets. When the ICMP reply for the third packet

arrives, `traceroute` increments the max `TTL` to 3. Then when the next heap of three packets packet gets sent to the first router, the router decrements `TTL` to 1 and the packet is forwarded to the next router. That router then does the same thing and generates an ICMP message (since `TTL` now became 0) that goes back to the host. This is repeated until the max `TTL` is large enough to reach the destination. This way we can put together a table of three roundtrip times for each node on the route.

Task 3 :

a) The TCP/IP model has 4 layers, going from top to bottom; *Application Layer*, *Transport Layer*, *Internet Layer* and *Link Layer*. When applications in the application layer want to send data across the Internet, it has to make use of several protocols in order for the data to successfully arrive at the destination. Each layer has its own protocols and responsibilities, that are transparent to the other layers. For example, the Application Layer can use *HyperText Transfer Protocol* (HTTP) to send data, but how the data is actually sent and how the other protocols work in the other layers, is ignored by HTTP. When HTTP wants to send a message to a remote host, it passes its data to the layer below, the Transport Layer. TCP is often used to make a data transfer for HTTP. TCP breaks up the data into the *Protocol Data Unit* (PDU) called *segment*, where every segment gets its own header. This process is called segmentation.

TCP makes sure that the data is sent across a network in a reliable fashion. How the segments will travel across the network (or any network), is up to the next layer, the Internet Layer. Here IP is used to address every IP-enabled end-point in the network with a unique IP-address. IP encapsulates the segment into the Internet Layer PDU called *packet*, also by adding a header with the essential information about the data and transfer. How the packet is transferred between to directly connected devices in the network is up to the Link Layer. The Link Layer's responsibility lies in making sure that the data is sent across a link between two devices. The packet gets a header and a trailer in an encapsulation called *frame*. The frame can be sent on any medium, as long as there is protocols for it to handle the process. When a frame has been sent across the medium, a new header and trailer is added for the next link transfer.

When the first frame arrives at the destination, decapsulation begins. The frame header and trailer are removed and the packet is revealed to the Internet Layer. After checking that the right packet has arrived, it removes its header as well and sends it to the Transport Layer. The Transport Layer also checks the validity of the segment, then decapsulate it and passes it on to the right application. The application then checks the message and does whatever it wants with the data.

b) A *Peer-to-Peer-Protocol* (P2PP) is a protocol that establishes communication between two entities, or hosts, in the layer that it operates in. P2PPs are only concerned about the responsibilities of their own layers, and do not bother about the implementation of other layers and protocols. For example, TCP establishes a connection over a network, by performing the *three-way handshake* between two hosts. The two hosts operate in a similar way as each other, but none of them care about how the data is being transferred across the network. They only care about the reliability and verification of the transfer, while delegating the routing, media access control and other things to lower layers.

Task 4 :

a) Blablabla.

```
www.chalmers.se:

  HEAD / HTTP/1.0

  HTTP/1.1 401 Unauthorized
  Content-Length: 16
  Content-Type: text/html; charset=utf-8
  Server: Microsoft-IIS/7.5
  SPRequestGuid: 1b34a246-0f4b-4010-9e90-6fd42daab1e4
  X-SharePointHealthScore: 0
  WWW-Authenticate: NTLM
  X-Powered-By: ASP.NET
  MicrosoftSharePointTeamServices: 14.0.0.7102
  X-MS-InvokeApp: 1; RequireReadOnly
  Date: Mon, 18 May 2015 08:20:33 GMT
  Connection: close
```

The type of web server `www.chalmers.se` is using is `Microsoft-IIS/7.5`. We get the response `401 Unauthorized`, which means that we do not have access to URL resources without providing user authentication (log in somewhere with a username and a password). The authentication protocol in this case, which is specified in the `WWW-Authenticate` field of the HTTP header, is `NTLM`.

Content-Length is the size of the content that is being sent, in octets(bytes). Content-Type indicates what type of media is being sent. SPRequestGuid contains diagnostic information about server problems. X-SharePointHealthScore is a number between 0 and 10 where 0 indicates low server load and 10 indicates high server load. X-Powered-By specifies which technology is used to support the web application. MicrosoftSharePointTeamServices indicates which version of Microsoft SharePoint is installed. X-MS-InvokeApp specifies if the application wants to use DirectInvoke, which lets a user open for example a PDF without first saving it to their computer. RequiredReadOnly means that it opens in Read-Only mode.

```
www.tue.nl:

HEAD / HTTP/1.0

HTTP/1.1 301 Moved Permanently
```

```
Date: Mon, 18 May 2015 08:24:17 GMT
Server: Apache/2.2.22 (Ubuntu)
X-Powered-By: PHP/5.3.10-1ubuntu3.18
Location: http://www.tue.nl/
Vary: Accept-Encoding
Content-Type: text/html


Connection closed by foreign host.
```

`www.tue.nl` is using the web server type `Apache/2.2.22 (Ubuntu)`. 301 Move Permanently means that the resource we are requesting has been redirected to a new URL. The new URL is specified in the Location field. In this case www.tue.nl is redirecting to http://www.tue.nl/. The Vary field specifies which fields of the request header to take into account when trying to find the right object in the cache.

b) Blablabla.

```
www.chalmers.se:

  HEAD / HTTP/1.1

  HTTP/1.1 400 Bad Request
  Content-Length: 334
  Content-Type: text/html; charset=us-ascii
  Server: Microsoft-HTTPAPI/2.0
  Date: Mon, 18 May 2015 09:15:38 GMT
  Connection: close

  www.tue.nl:

  HEAD / HTTP/1.1

  HTTP/1.1 400 Bad Request
  Date: Mon, 18 May 2015 09:16:38 GMT
  Server: Apache/2.2.22 (Ubuntu)
  Vary: Accept-Encoding
  Connection: close
  Content-Type: text/html; charset=iso-8859-1
```

Blablabla.

c) Blablabla.

```
www.chalmers.se:

  HEAD / HTTP/1.1
  Host: www.chalmers.se

  HTTP/1.1 302 Redirect
  Content-Length: 164
  Content-Type: text/html; charset=UTF-8
  Location: http://www.chalmers.se/Pages/default.aspx
  Server: Microsoft-IIS/7.5
  SPRequestGuid: f25d9db4-5345-4c46-b7fc-292c52258273
  X-SharePointHealthScore: 0
  X-Powered-By: ASP.NET
  MicrosoftSharePointTeamServices: 14.0.0.7102
  X-MS-InvokeApp: 1; RequireReadOnly
  Date: Mon, 18 May 2015 09:27:28 GMT

www.tue.nl:
```

```
HEAD / HTTP/1.1
Host: www.tue.nl

HTTP/1.1 200 OK
Date: Mon, 18 May 2015 09:28:56 GMT
Server: Apache/2.2.22 (Ubuntu)
X-Powered-By: PHP/5.3.10-1ubuntu3.18
Set-Cookie: fe_typo_user=e4577ddec4a01568b6eefae39b0dcf4b; path=/
Expires: Mon, 18 May 2015 10:21:29 GMT
Cache-Control: max-age=3144
Vary: Accept-Encoding
Content-Type: text/html; charset=utf-8

Connection closed by foreign host.
```

In an HTTP 1.1 request you have to specify the name of the host of the resource you are requesting. You can also specify a port number, but we did not do that since it defaults to 80 if you do not specify one. If we do not include a host name in the request, the server responds with 400 Bad Request, as we saw in 4b.

Say for example that we are hosting several different websites on a single machine. They all share an IP address. To be able to send the request to the correct website, we need to specify a host name. This is what the host header field is used for; differentiating between multiple hosts on the same IP.

Task 5 :

a) *Cookie-files*, or *cookies*, are web files that are stored in the web browser. When browsing a website, a pop-up window might appear asking the user if it wants to enable cookies in the current web browser. If the user accepts, the website will keep track of what the user do on the website and then tells the browser to store this information as cookies on the computer.

When using the latest version of *Firefox* web browser, you can view what cookies are stored on your computer. By going to the *Privacy* tab in *Options* and clicking *remove individual cookies*, you can see all cookies in a list. As an example, the following text shows the contents of a cookie after visiting Amazon's website:

```
Name:       skin
Content:    noskin
Domain:     .amazon.com
Path:       /
Send For:   Any type of connection
Expires:    At end of session
```

Here we see a cookie named `skin` that was stored while browsing `www.amazon.com`, with the content `noskin`. The cookie also contains information about the domain, the *Uniform Resource Indentificator* (URI) path for the cookie and when the cookie expires.

b) Using `telnet`, we can find out about the cookie header-line:

```
> telnet www.amazon.com 80

HEAD / HTTP/1.1
Host: www.amazon.com


Response:

  <output omitted>

  Set—Cookie: skin=noskin; path=/; domain=.amazon.com

  <output omitted>
```

To find out more about the syntax of the header-line, we have to consult the *Request For Comments* (RFC) document for cookies. RFC 6265 specifies the latest standards for cookies. The header contains three name-value pairs. The first one specifies the name of the cookie and the content it carries. In this case the name of the cookie is `skin`, with the content `noskin`. `path` specifies the URI for the cookie, which has a value of \. This tells the host to store the default-value in the cookie. The last field is the `domain`. Here we see that the server tells us that `amazon.com` is the domain for the cookie (leading "." are omitted).

Task 6 :

a) The `nslookup` command can be used to find information about the DNS servers of a domain. We will use `utoronto.ca` (University of Toronto) for this task. Here are the results of our `nslookup` for `utoronto.ca`:

```
> nslookup —type=mx utoronto.ca

Server:  res1.chalmers.se
Address:  129.16.1.53

Non—authoritative answer:
utoronto.ca     MX preference = 10, mail exchanger = k.mx.utoronto.ca
utoronto.ca     MX preference = 10, mail exchanger = d.mx.utoronto.ca
utoronto.ca     MX preference = 10, mail exchanger = b.mx.utoronto.ca
utoronto.ca     MX preference = 10, mail exchanger = c.mx.utoronto.ca
utoronto.ca     MX preference = 10, mail exchanger = g.mx.utoronto.ca
utoronto.ca     MX preference = 10, mail exchanger = a.mx.utoronto.ca
```

```
utoronto.ca     MX preference = 10, mail exchanger = l.mx.utoronto.ca
utoronto.ca     MX preference = 10, mail exchanger = e.mx.utoronto.ca
utoronto.ca     MX preference = 10, mail exchanger = f.mx.utoronto.ca
utoronto.ca     MX preference = 10, mail exchanger = j.mx.utoronto.ca

utoronto.ca     nameserver = bay.cs.utoronto.ca
utoronto.ca     nameserver = ns2.utoronto.ca
utoronto.ca     nameserver = ns1.utoronto.ca
utoronto.ca     nameserver = ns7.utoronto.ca
bay.cs.utoronto.ca      internet address = 128.100.1.1
ns1.utoronto.ca internet address = 128.100.100.129
ns2.utoronto.ca internet address = 128.100.72.168
ns7.utoronto.ca internet address = 162.243.71.42
```

To find the mail servers we set the type parameter to `MX` (Mail eXchange). We find that there are 10 mail servers for `utoronto.ca`, with all of them having equal priority (preference = 10). All of them having equal priority means that we do not care which of the mail servers we use first.

We also find the DNS servers and their respective IP addresses.

b) We use `www.google.com` for this task.

```
> nslookup www.google.com

Server:   res1.chalmers.se
Address:    129.16.1.53

Non-authoritative answer:
Name:     www.google.com
Addresses:  2a00:1450:4010:c02::68
            74.125.205.106
            74.125.205.103
            74.125.205.105
            74.125.205.99
            74.125.205.147
            74.125.205.104
```

We can see that www.google.com has 6 different IP addresses. Multiple IP addresses can be used to balance load between web servers. There can be several copies of the same web site, each having its own IP address but all using the same DNS servers. The first user who sends a request is sent to the first IP address, the second person to the second IP address and so on. This reduces the amount of requests to each web server and provides redundancy in case a web server should go down.s

Multiple IP addresses can also be used to have different IP addresses for different services.

Task 7 :

a) TCP uses *timeout* to prevent possible delays in the transfer. If one segment doesn't get acknowledged (ACK'ed), it would be bad if the whole transfer had to stall while waiting for a reply. Instead it uses a timer to count down a segments maximum *Round Trip Time* (RTT). When the timer ticks down to zero, the segment is considered lost

and a duplicate segment is sent. As the transfer proceeds, TCP will calculate new timeouts as it "learns" that the segments arrives either faster or slower. The tighter the timeout window is, the faster TCP can discover that a segment is lost.

b) When calculating the timeout window, TCP makes use of the *recorded* RTTs and the *estimated* RTTs. We could just simply calculate the window by adding a little more time to the recorded RTT, but that would lead to big variations and unstable transfer flow. Instead we only count for a portion of the latest RTT sample and add it to the complement portion of the previous calculation. This is what's called the estimated RTT, and unlike the recorded RTT, it's less prone to fluctuation.

   To actually determine a value for the timeout window, we need to calculate the deviation of the sample RTT from the estimated RTT. Big fluctuations between these should affect the deviation value more, and contrary small fluctuations should affect it less. By adding an appropriate multiple of this deviation to the estimated RTT, we get an even calculation of the timeout value, that adapts fairly quickly to data rate changes.