DATANET

Weekly assignment 1

Martin Jørgensen (tzk173)
martinnjorgensen@gmail.com

May 3, 2013

1 Theoretical part

1.1 Store and forward

1.1.1 Proccesing and delay

A key reason for delays in packet switching networks is outputbuffer queing delay. It is the result of each node collecting the entire packet before sending in on to the next node.

1.1.2 Transmission speed

1. **Part 1:**

$$\begin{aligned} d_{nodal} &= 2ms + 1ms + 5ms = 8ms \\ d_{prop0} &= 20m/2.4 \cdot 10^8 m/s = 8.333 \cdot 10^{-5} ms \\ d_{prop1} &= 5m/2.4 \cdot 10^8 m/s = 2.083 \cdot 10^{-5} ms \\ d_{prop2} &= 750m/2.4 \cdot 10^8 m/s = 0.003ms \\ d_{lastLink} &= 24ms \\ d_{total} = d_{nodal} + d_{prop0} + d_{prop1} + d_{prop2} + d_{lastLink} &= 32.003ms \end{aligned}$$

   The last entry d_{total} is the time it takes for a packet to go from the students laptop, to diku.dk, if multiplied by 2, this will give us the RTT.

$$RTT = d_{total} \cdot 2 = 64.006ms$$

2. **Part 2:**
   First we will calculate the time it takes to transmit the $640KB * 8b/B = 5120Kb$ over the different connections:

$$\begin{aligned} t_{WiFi} &= 5120Kb/54000Kb/s = 0.094s \\ t_{wire0} &= 5120Kb/100000Kb/s = 0.051s \\ t_{wire1} &= 5120Kb/2000Kb/s = 2.560s \\ t_{wire2} &= 5120Kb/1000000Kb/s = 0.005s \\ t_{total} &= 0.094 + 0.051 + 2.56 + 0.005 = 2.7s \end{aligned}$$

   To calculate the total upload time we need to add half of the RTT to the above t_{total}:

$$t_{upload} 2.7s + 0.032s = 2.732s$$

1.2 Buffers and latency

1. **Part 1:**
   Since the link from the wifi to the modem is the fastest link, the maximum delay a package would experience would be if the modem-buffer was completely full and needed to be emptied in order for the modem to take new input. The modems buffer is $512KB$ and data is leaving at a rate of $2Mb/s$. Thus the maximum delay would be:

$$\frac{512KB * 8b/B}{2000Kb/s} = 2.048s$$

2. **Part 2:**
   A reason why BDP is a good rule of thumb for buffer-size is that it describes the amount of data that can be in a link. (where the data is send, but not yet recieved) If any buffer on the network is capable of storing the amount of data that the link itself can transmit, the buffers will never overflow.

   If a student sends a request that is larger than the amount of free space in the buffer of the recieving node, the request will be ignored by the node.

1.3 HTTP

1.3.1 HTTP semantics

1. **Part1:**
   The main difference between `GET` and `POST` is the way they pass data. `GET` sends data through the URL and `POST` sends data through the body of the request.

2. **Part2:**
   The `HOST` header is needed to help a server sort our internally- ambiguous URL's. For instance of a server hosts several websites(with their own domains) on the same IP-address, the `HOST` field will help the server decide what page to serve.

3. **Part3:**
   Using the following request from `telnet google.com 80`:

```
GET / HTTP/1.1
Host:  google.com
```

   The result is a HTTP response with a `301 Moved permanently`, which means the object was moved somewhere else. The status code in the response (`301`) is important because it helps the client retrieve the correct page. Upon recieving a `301` a browser will by itself retrieve the correct page. Other responses can occur as well, each is an indication on how to handle the response.

1.3.2 HTTP headers and fingerprinting

1. **Part 1:**
   The purpose of the `Set-cookie` and `Cookie` fields is to allow servers to track/identify user despite the webserver being stateless. It allows the server to instruct a client to always send data (for instance a userID) along when it requests objects from the server.

   Cookies can be used for tracking as long as the client do not wipe their cookie records. In that case, to keep tracking the client the server will need a login system so it can reassociate the client with the cookie.

2. **Part 2:**
   Two fields that can spill informatino is the `Server` and `User-agent` fields. They tell information about the server that answers the request, and the client the send it respectively. This information can be used in an eventual attack on either part. This spill could be prevented by stripping the fields or reduce the amount of information in the fields.

3. **Part 3:**

   The `ETag` field can be used to check the version of a document to see if it should be fetched again, or the cached version will do. It works like a versioning number/fingerprint and is shorter to transmit than the `Last-modified-time`.

1.3.3 The case of Deep Packet Inspection

1. **Part 1:**
   A fairly obvius way to filter requests for grooveshark would be to check the `Host` field, all packets with a grooveshark URL in their `Host` field could then be sorted or redirected.

2. **Part 2:**
   This is a violation since the routers wll have to go to the application layer in order to read the `Host` field, which is not needed in order to forward a package, that would only require accessing the transport layer, for reading the TCP header.

   A way to prevent this layer-violation would be to encrypt the content of the TCP packet so only the destination would be able to see the payload. This would prevent the ISP from inspecting anything other than the TCP-header.

2 Practical Section

2.1 Question 1

For this particular protocol, the buffer size is not a bad choice, as the protocol does not encourage longer payloads than 2014 bytes. The choice of using a message termination character/string is possible but the `/echo` messages could contain the symbol which would give false results. Instead one could take a page from the HTTP protocol and send a header before the payload with the length of the content (both for commands and response) which would solve the problems of both solutions.

2.2 Question 2

For thisd particular client/server a dead socket can safely be closed. The client should then try to reconnect again (and if it have not recieved a response it should send the command again.)

2.3 Question 3

I assembled the following test suite for my program:

1. Send several /ping commands after each other and check the replies.
2. Send commands that contain the legit commands. (for instance `/pinga`) and check if the client rejected the commands.
3. Sending `/calc` commands with numbers greater than the limit for pythons `float` datatype.
4. Sending calc commands with letters as operand or invalid operations.
5. Sending commands longer than the buffersize.

This have been done with a many to one test (1 server for 3 concurrent clients.) I myself think that both the client and the server is quite robust at this state.

An issue with the server is the way it jsut spawns a new thread for each new connection, this can be abused by creating many connections to the server which in turn would spawn a lot of resource consuming threads on the server. This could be solved using a threadpool instead. Then the server would never be able to create more than a specific number of concurrent threads, which would limit the resource consumption. This solution would increase response times when more clients are connected than threads can be allocated.

2.4 Question 4

The protocol itself is stateless since it does not save any state, like HTTP, the cleint sends a request wich the server then fullfills and responds to. In my implementation however, the connection is kept alive and can be reused for several requests, which makes it a statefull connection.