HAR JSON Parsing

1. To calculate total number of objects per domain we used two methods:
    a. Using HAR file
        i. Total number of objects downloaded were 169.
        ii. In this we only considered objects whose http response status is “200 OK”. Also there were some objects whose http response was 200 OK but null body. They were not taken into account.
    b. Using pcap file
        i. Total number of objects downloaded were 166
        ii. Some objects were not present in pcap file. This may be because pcap capture was closed before the entire webpage was downloaded leading to some inconsistencies.

2. To calculate object size and content size:
    a. object_size = headers_size + body_size
    b. content_size =content_size (In all the objects content_size==body_size)

3. Different types of objects were calculated using content type of response object.

4. TCP connection to each domain
    a. With help of pcap and HAR file, we grouped objects from HAR file as per their domain name.
    b. From pcap file, we found connections made to each domain name. Multiple connection were differentiated using port number and TCP stream id.
    c. tshark command = “tshark -r nytimes.pcap -Y "http && http.host contains #{domain_name}" -T fields -e tcp.port -e frame.time_relative -e _ws.col.Info”

Object tree and Download tree

1) Object Tree
    a) To make an object tree, we considered the referer field in request’s headers of object. In most of the cases, it was 6th element in the array.
    b) The root of the object tree is identified by finding the object without any referer header. It is further cross checked with the first object to be downloaded(which should be the root).
    c) The width of object tree was found to be considerably larger than the depth of tree. In nytimes.com depth turns out to be only 3. This is mainly because most of the objects to be downloaded are referred by index.html page(root page).

2) Download Tree
    a) All the objects were grouped as per domain name.

b) For each domain name, using tshark (filter = “http.host contains #{domain_name}), we calculated number of tcp connections for that domain. Each connection was identified by port number.
c) For each connection, we find objects downloaded on that connection using url.
d) connection_id of tcp stream was calculated using tshark field tcp.stream

Timing Analysis

1) Page load time was calculated from HAR file by finding the difference between earliest start time and latest end time.
2) DNS Time
    a) For most of the objects dns time mentioned in HAR file was 0 ms. This can be because the dns was not flushed before analysis, so instead of launching dns query, browser would have looked it up in its cache. Also dns time have to be at least in order of 20-50 ms.
    b) To calculate precisely, we looked up dns time from pcap file.
    c) Initially, for each domain we calculated dns query launched. Command used is : “tshark -r #{pcap-file} -Y \"dns && ip.src==#{srcip-ip}\" -T fields -e dns.id -e _ws.col.Info | grep #{domain-name}”. This gives id of dns query launched for particular domain.
    d) Next we found dns response corresponding to that id using command “tshark -r #{pcap} -Y \"dns && ip.dst==#{srcip}\" -T fields -e dns.id -e dns.time -e dns.cname”. This gives us dns time for particular domain.
3) Timing information
    a) It was calculated from har file using the connect, wait and receive variable. For almost all the objects, send time was zero. It is consistent with the fact that http_request has very less size and should not take more time.
    b) To calculate active time of the connection, we calculated what was the last request corresponding to tcp stream of that connection from pcap file.
    c) For each connection, active percentage=(sending+waiting+receiving)/active_time and idle percentage=1-active percentage. One trend observed was that if more objects were downloaded on that connection, then active percentage of that connection was high.
    d) For connection where only single objects were downloaded, idle percentage was high.
    e) average_goodput=total_data/total_receive_time. To evaluate maximum goodput, we calculated maximum object downloaded on that connection and goodput for that object download.
    f) It is observed that if object size is more, than goodput of connection is generally high. Also on directly downloading large sized objects, goodput was better than the average goodput of the connection. It was mainly because, in large object size connection was better utilized.
    g) average_goodput_network=total_data_network/total_receive_time_network

h) Average goodput(Network) : 145.30 KB/s
i) Maximum goodput(Network) : 2667 KB/s
j) On comparing maximum of the maximum goodput of the connection and average_goodput of the network, it was observed that download capacity was not effectively utilized. In case of small objects, time wasted for handshake procedure is comparable with receive time of that object. So goodput decrases due to such cases.

4) Browser scheduling policies
   a) On comparing maximum number of parallel tcp connection to same domain, maximum value was found to be 12 (nytimes.com).
   b) On comparing maximum number of parallel tcp connection across network, maximum value was found to be 38(static01.nyt.com). To calculate this, we found parallel connection for each connection to domain and took maximum of those value. Connection is considered parallel, if any of start_time or end_time falls between connection_duration.
   c) CAP
      i) There seem to be cap on connection made to single domain. It is mainly because browser doesn't want to flood the server with too many connections (many server doesn't even allow more than particular number of connections). Moreover, after particular number of parallel connections, the cap is your download link capacity, which will become bottle neck.
      ii) If more number of parallel connections, all the objects will be downloaded slowly. Thus browser experience reduces.
      iii) Regarding maximum number of objects per connections, most browser doesn't support pipelining. So browser, even after following HTTP/1.1, doesn't do pipelining. Reason is because pipelining increases workload of server. Server has to maintain queue and state of the request. So of server is not as good, then there there are many connections problems(and to correct those browser has to handle too many cases). So they avoid pipelining.

Optimizations

1) Optimization-1(collapsing all GET request for each connection)
   a) Using the data aggregated in above analysis, we collapsed all GET request for each tcp connection. This is equivalent in saying that all GET request is launched simultaneously from browser(we are assuming that there is no cap on simultaneous download per connection).
   b) We maintain a queue for each connection. In this queue, we only enqueue those objects whose dependency is met as per object tree. Those object whose dependency are not met will be enqueued only after its referrer is downloaded via some connection.
   c) In a queue for each connection, we have all GET request launched. We consult HAR file for response, and extract corresponding object for timing analysis. Thus

in this case we are improving in idle time between the GET request of each connection.

d) The time taken to download the entire webpage - 12 s (compared to 50s)

2) Optimization-2(collapsing tcp connection for each domain)
   a) As second optimization, we considered all the objects to be downloaded on same tcp connection for each domain.
   b) So in a queue, we have all the GET request to particular domain. And all those GET request are launched on same TCP connection. So in this case, we are improving on connection time for the domain.
   c) This case may or may not see further improvement from case 1. It depends on idle time across connections in case 1. But it is for sure not going to be worse.
   d) The time taken to download the entire webpage - 6 s (compared to 50s)