University of California, Berkeley
CS 294-1 Multimedia
Prof. Larry Rowe
Fall, 1998

Project Report

WebTP: A User-centric Network Infrastructure Optimized for Web Browsing

Submitted by:
Hoi-Sheung Wilson So

10th December, 98.

1. Introduction

Web browsers are usually very responsive, but sometimes, after clicking on the link, you wait and wait, the logo animation keep juggling, but nothing happens. You finally give up, and hit the stop button. And suddenly, you see partially loaded HTML page with some broken pictures in them. Obviously, the browser was getting stuck loading some huge picture over a slow link. Why should the browser start loading pictures before finishing the HTML file? Have you ever hoped that the browser would ask you your preferences in choosing what to download first?

Today, web sites are very graphics intensive. For example, a typical web site, such as the CNN home page on December 6, 1998 has a total size of 140KB (HTML, GIFs, and JPEGs only). It takes at least 20 seconds to download this first page using a 56K modem, assuming no protocol overhead. Unfortunately, the Internet is often congested, and drops, delays, or re-orders packets. Danskin et al. [DDS95] reported the loss rate of the link from Dartmouth College to Stanford University was never below 15% from 10am to 10pm when sending at 34.4KBps.

Given a limited amount of imperfect bandwidth, how can we make web browsing as responsive to user needs as possible? Ideally, we want to find out the user's expected utility of each object. Depending on the utilities of different objects, the browser can allocate available bandwidth among them to achieve maximum user satisfaction.

2. Problem Definition

2.1 Utility

The utility of an object is the usefulness of an object to the user. It can be measured in terms of how much the user is willing to pay for this object, or in terms of user satisfaction. It depends on many factors such as the current state of the network, the object being measured, user preferences, web browser capabilities, etc. Large objects are less attractive to a user who is connected to the Internet through a slow link. Pictures used to decorate web pages are less useful than others that convey information. The utility of an object differs for different viewers. Even for the same viewer, it still changes rapidly with time because user actions change the usefulness of an object. For example, when a user follows a link on the current page to a new one, any object on the current page that has not finished downloading can now be ignored.

Finding an accurate model of the utility of an object is very important. But even if we had such a function, we still lack an infrastructure that allows applications to allocate network resources in a way that will maximize the expected utility of a page. And due to the lack of such an infrastructure, it is very difficult to carry out experiments that help us understand humans' model of utility of web pages!

2.2 Goals

To tackle this somewhat recursive problem, we hope to start by designing an infrastructure that is optimized for web-style communications by taking into account user-requirements. This infrastructure will allow user-agents (e.g., browsers) to control how network resources are allocated and consumed in order to best satisfy user needs. The short-term goal for this semester was to build a prototype of the infrastructure to gives us insights into the actual requirements of such infrastructure.

3. Problems with HTTP/TCP

Unfortunately, the protocols linking together the World Wide Web today, namely HTTP and TCP have several fundamental limitations that render them unsuitable for building our proposed infrastructure.

3.1 HTTP/TCP Incompatibilities with Utility-based Model

First, neither HTTP [F+96] nor TCP provides a way to specify bandwidth allocation between multiple connections. Second, TCP [Jac88] hides all the flow control mechanisms from upper layers. Without network state information such as round-trip time and available bandwidth, upper-layer applications or protocols cannot easily implement their own bandwidth allocation algorithm. Third, sender-driven systems such as TCP is less responsive to user needs because the user is on the receiver side. Any changes in user needs must first propagate to the sender-side over the network before it can have any effects on the network protocol operations.

3.2 Other HTTP/TCP Problems

1. Each TCP connection has to go through a 3-way handshake to setup or tear down a connection. For small web transfers, the round-trip latency becomes a significant overhead.
2. TCP resets window size to 1 after an idle period. For small transfers, the window size never

reaches its full potential before the connection is stalled again.

3. Browsers often open several parallel HTTP connections to the same server. These connections have the same end-points, yet operate competitively to probe the currently available bandwidth.
4. TCP breaks the HTTP data stream at arbitrary boundaries and exacerbates the effects of packet loss and re-ordering.

For a more detailed description of these problems, see [GCMW98].

4. Design Principles

In order to build a new infrastructure that avoids the problems of HTTP and TCP, we have identified several promising design principles.

4.1 User-centric Optimization

Users are the ultimate sink of any information. They also pay for the network connection. Therefore, they should be brought into the loop of decision. Whenever making design or implementation decisions, we should always aim at improving the network performance perceived by the user. We should leave as much flexibility as possible to the users or their software agents.

4.2 Application Level Framing

Downloading a web page often involves transferring many small independent files, such as the text file (HTML) and some pictures (JPEGs, GIFs). Browsers either use one TCP connection per object (HTTP/1.0) or one (or a few) connection per page (HTTP/1.1). The former incurs high overhead, and the latter is more prone to performance problems in the event of packet drops or reordering. There is room for a new connectionless protocol that uses ALF principles to multiplex the transfers of multiple objects sharing the same path, and limit the effects of a single packet drops or re-ordering to only one object transfer.

4.3 Receiver-driven Transport Protocol

As mentioned earlier, the goal of WebTP is to build an infrastructure that is responsive to user requirements that are changing all the time. To be able to respond quickly to changes in user needs, the network resource management must be located as close to the user as possible. Because the user is the receiver of information, the transport protocol should preferably be driven from the receiving end.

5. Design Overview

The goal of the project is to build a framework which allows applications to allocate network resources to achieve maximum utility for the user. Building such a framework requires designing and implementing at least four components:

1. An ADU naming scheme which has enough expressive power to name the ADUs of various data formats used in web pages, such as JPEG, GIFs, HTML, WAV etc, so that each ADU can be processed independently of each other.
2. An object request protocol that understands this ADU naming scheme, and provides a mechanism to allocate bandwidth based on the utility of the ADU.
3. A transport protocol that allows the object request protocol to have control over the bandwidth allocated to each object transfer.
4. A browser that understands user preferences and can translate preferences into the transport priorities of ADUs. (Strictly speaking, the browser is not part of the infrastructure, but there is no way to verify the usefulness of the infrastructure other than building a browser.)

6. Prototype Design & Implementation

To try to solve this complex problem in one semester is infeasible. Indeed, one can argue that designing such a complex system without first building some prototypes is impossible. So we followed the advice of Butler Lampson [Lamp83], and started building a prototype. The main purpose of building the first prototype is to let us better understand the requirements of each of the four components.

6.1 ADU Naming Scheme

For the first prototype, we limit ourselves to only the most common object type on the web. The most common object types on the web are HTML, JPEG, and GIF files. HTML files are usually relatively small. Pages that take a long time to load are mostly due to the embedded pictures, not due to the HTML file. GIFs are commonly used for computer-generated graphics and they tend to be relatively small in size. JPEG are used for color photos which are usually larger in both screen size and in file size. Hence, for the first prototype, we focus specifically on JPEG. We want the pictures to be progressively refinable so that bandwidth allocation can be at a fine-grained level. One subset of the JPEG standard [JPEG] defines a picture format commonly known as "Progressive JPEGs." Progressive JPEGs differ from "baseline JPEGs" in the way Discrete Cosine Transform (DCT) coefficients are encoded in the file. Coefficients encoding the lower frequency signals of each DCT block appear near the beginning of a file. Higher frequency coefficients that give a crisp image are located near the end of a file. Therefore, if a picture is loaded from the beginning to the end, the rendered picture will become progressively richer in details, and hence the name Progressive JPEG.

A typical progressive JPEG has the following structure:

Header
Quantization Tables
Huffman Tables
Scan 0 (DCT coefficients of lowest frequency signals)
…
Scan N-1 (DCT coefficients of highest frequency signals)

(Fig.1: typical structure of a progressive JPEG image)

One natural way to break up a JPEG file is to specify the first ADU (U1) to include everything from the beginning of the file up to the end of Scan 0. The second ADU (U2) includes just Scan 1, and so on. Each ADU name is a tuple <URL, offset, length>. U1 by itself can render a low quality version of the picture. Any subset {Ui: $1 \leq i \leq N$} can render a picture. This definition of an ADU name might seem odd at first sight because using offset and length to name an ADU masks the semantics of the ADU. Nevertheless, this does not violate the ALF definition [CT89]: "break the data into suitable aggregates, and the lower levels should preserve these boundaries."

6.2 Object Request Protocol

Once we have a naming scheme which allows us to refer to individual ADUs of a JPEG image, we can design an object request protocol for requesting ADUs. Because we adopted the principle of ALF, lower layer protocol layers are fully aware of the existence of ADUs. The object request protocol can therefore be reduced to an API without adding additional packet headers or trailers. The object request API has a client and a server interface:

6.2.1 Client API:

1. `Request(ADU_Name, Priority);`
Requests the ADU specified by the `ADU_Name` <URL, offset, len> and associate a `Priority` with this request.

2. `UpdatePriority(ADU_Name, New_Priority);`
Changes the priority of an earlier ADU request to the `New_Priority`. Updating the priority of an earlier request to negative infinity cancels the request.

3. `CALLBACK ReplyArrival( ADU_Name, ADU_data);`
Client receives this call back when an ADU arrives. `ADU_Name` specifies the request issued earlier, `ADU_data` contains the server reply.

6.2.2 Server API:

1. `ListenForRequest(port);`
Server application calls this function to register as a listener on this `Port`.

2. `CALLBACK RequestArrival(Assocaition, ADU_Name);`
Server application receives this callback when a request for ADU arrives. Association specifies a pair of endpoints, `<client_host:port, server_port>`

3. `Reply(Association, ADU_Name, ADU_Data);`
Server application calls `Reply()` to serve an ADU request issued by the client

6.3 Transport Protocol

With just static flow control, one can simulate the effects of a bottleneck link at any speed. Therefore, we decided to implement only static flow control in this prototype. The prototype WebTP stack is built on top of UDP, and implemented in Java.

Whenever a client request an ADU, the request is put into a priority queue sorted in descending order of the priorities of the requests. At time intervals regulated by the static flow control mechanisms, the request at the head of the priority queue is sent to the server. In other words, the bandwidth allocation scheme is to allocate 100% bandwidth to the highest priority object, then move on to the next highest priority object, and so on. For example, assuming that we have 3 different ADU requests in our priority queue. The sizes of the requested ADUs are S1 Bytes, S2 Bytes, and S3 Bytes. Further assume that the link speed is X Bytes/second, the delay between the first request and the second therefore should be S1/X [second]. Similarly, the delay between the second and the third should be S2/X [second]. Remember that WebTP is receiver driven; spacing out the requests on the client side in effect limits the server's sending rate, and achieves flow-control. Retransmission uses a window-based algorithm with a static window size.

6.4 Image Browser

The image browser is implemented in Java using the Java Foundation Classes 1.0. It takes two different types of files as input: layout and image files. The layout files are written in a simple language that is much easier to parse than HTML, and contains additional ADU fragmentation information (Fig. 2). The image files supported are JPEGs and GIFs.

Each line of the layout file describes an embedded object (image) and has the following format:

URL:	URL of the object
X, Y:	Coordinates of the object on the screen
W, H:	Dimension of the object on the screen
P:	Preset priority of the object as deemed appropriate by the page designer
URL link:	URL of a page that the browser jumps to when the user clicks on the object
ADU Fragment Lengths:	List of numbers, each of which is the ADU length in bytes. E.g., {512, 234, 1024} indicates that this object consists of 3 ADUs with length of 512, 234 and 1024 bytes respectively.

(Fig.2: structure of each line of the layout file.)

6.4.1 Page Download Process

1. When the user types in an URL of the layout file, the browsers sends a request <URL, 0, *> to the server asking for every byte of the layout file. The whole layout file can be thought of as a single ADU. A typical layout file for testing purposes is smaller than 1KB.
2. After the layout file has been downloaded and parsed, the image browser will assign priorities to different ADU requests depending on the preferences that the users choose.
3. ADUs are requested in descending order of their priorities.

6.4.2 User Options

The prototype allows three options that users can mix and match to affect bandwidth allocation:

- *Progressive Refinement*

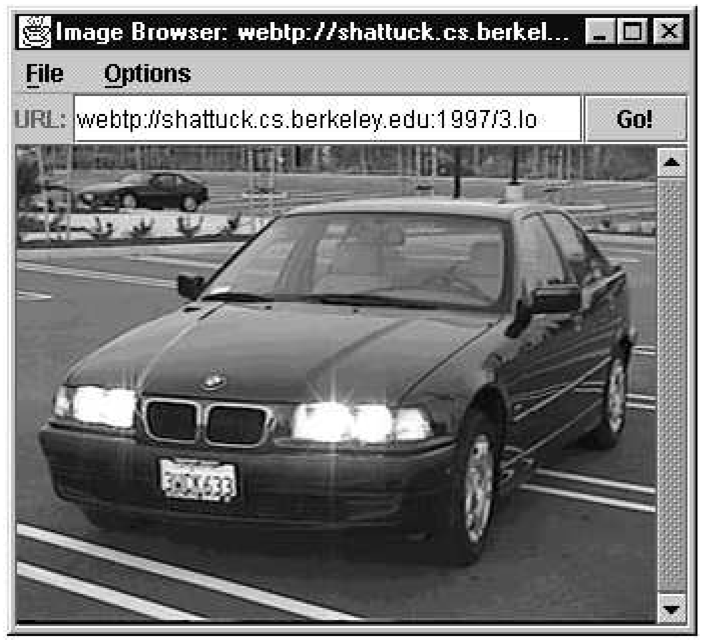
This option, when turned on, causes the browser to render lower quality versions of a picture as soon as possible, rather than waiting for the whole picture to finish downloading.

- *Round-robin Loading*

This option, when turned on, causes the browser to load the first scan of each picture on the page, then load the second scan of each picture, and so on. The user will see a blurred version of the page very quickly. Each picture will refine in a round-robin order as more ADUs arrive.

- *Preset Prioity*

This option, when turned on, causes the priority of each object specified by the page designer to be considered when deciding the ADU download order. The rationale is that the page designer should have a good idea of the semantic values of each object. Nonetheless, an object can have a different value to the page designer than to the user.



(Fig.3 A screen shot of the image browser showing a page downloaded from the URL webtp://shattuck.cs.berkeley.edu:1997/3.lo)

7. Evaluation

To recap, the main purpose of building the first prototype is to let us understand the requirements of each of the four components better. Next, we will discuss the lessons we learnt from building the prototype.

7.1 ADU Naming Scheme

Is the ADU naming scheme <URL, offset, len> expressive enough to capture the structure of various data formats used in web pages? Our simple scheme <URL, offset, len> seems to handle JPEG and GIF without problems. However, the structure of HTML is much more difficult to express using our naming scheme. For example, if the user wants to see only the headings of each paragraph of an HTML file, our scheme does not provide a simple way to specify, say, just the portions of text surrounded by <H1> and </H1>.

Another problem with our naming scheme is the inflexibility in handling dynamically generated pages. For example, if an URL refers to a CGI script that gets executed on the fly, there is no way to tell the exact length of the reply at time of the request. Hence, the client is forced to treat the reply as a single (potentially huge) ADU and send a request of the form <URL, 0, *>.

Nonetheless, this scheme has not failed completely. Ideally, each ADU is small enough to fit in an UDP datagram that fits in a single IP packet. If an UDP-wrapped ADU is larger than the Maximum Segment Size defined by IP, IP will fragment the UDP datagram into smaller IP packets. However, depending on the OS implementation, there is a limit on the largest UDP packet that IP will fragment. The maximum UDP datagram size is therefore limited to somewhere between 500 bytes and 20 kilobytes. Alas, there is no guarantee that each ADU is always smaller than a fixed number, say 500 bytes, and so WebTP must handle ADU fragmentation. Each ADU fragment must be tagged with the length and offset of the fragment with respect to the ADU or with respect to the entire object. Our simple ADU naming scheme handles ADU fragmentation seamlessly by breaking large ADUs into smaller ADUs. For example, <URL, 0, 900> can be broken into <URL, 0, 500> and <URL, 500, 400>. The only difference is that WebTP now has to remember not to deliver these artificially generated ADUs. In conclusion, regardless of how to improve naming scheme to add more expressive power, the simple naming scheme using <URL, offset, len> is still necessary to handle ADU fragmentation.

7.2 Object Request API / Transport Protocol

Thanks to ALF, the Transport Protocol can operate directly on ADUs. There is no need to introduce another protocol on top of the transport protocol. The object request protocol degenerates into an API. Because the Object Request API and the Transport Protocol are so tightly coupled due to ALF, we will discuss the lessons learnt from them together.

The biggest question that arises is the precise semantics of the "priority" of objects. Here we run into some of the traditional problems of scheduling based on priority. In our current implementation, all the bandwidth goes to serving requests of highest priority. Requests of lower priority can starve. For example, if our browser supported background audio clip, and the user had specified that music should have a higher priority than pictures or text, the browser would first download the entire audio clip before downloading pictures or text. What could be worse is that the music is actually a continuous feed, and as a result, the pictures never get downloaded because pictures have a strictly lower priority than the streaming music.

What we learnt is that allocating bandwidth by sorting the requests according to the priorities of the ADUs is insufficient. One possible solution is to use Class-based Queueing (CBQ) which solves the starvation problem and gives us fine-grained control over bandwidth allocation.

7.3 Image Browser

The rationale for choosing Java as the language for implementing the image browser is that Java is cross-platform, and so we can write the program on either UNIX or Windows and run on both. In fact, Java does live up to the promise, "write once, run everywhere." We have also written the prototype WebTP stack in Java to allow easy integration with the front end. However, in retrospect, we regret the latter decision, and are revisiting the first decision.

The first problem with Java networking code is that it does not offer the same level of flexibility as C. This is not a problem with the language itself, but rather with JDK. For example, there are a lot of UDP socket options that we cannot get access to. In the next prototype, we will write the WebTP protocol stack in C or C++. We are currently using JFC 1.0 for the GUI of the browser. However the package is still immature and it makes development and debugging very difficult. We are currently looking into possibility of using another language to rewrite the image browser.

The most important lesson learnt from building the browser is that the feature set of the browser may not be as important as its ability to update pictures quickly and without flickering. Users of the prototype browsers have complained seriously about the flickering during screen updates.

8. Conclusions

We first pointed out that, traditionally, web browsers ignore users needs and preferences in the deciding how to allocate bandwidth among various object transfers. Second, we proposed a new infrastructure with four components: ADU naming scheme, Object

Request API, Transport Protocol, and Browser that takes user-requirements into account. Third, we presented the design and implementation of a prototype which includes the four components mentioned above. Next, we evaluated each component of the prototype. And finally, we summarized the lessons learnt from the prototype and hinted at how we can improve the next prototype. The WebTP user-centric network infrastructure will take years to develop, but our first prototype has provided us with some valuable insights on how to proceed and build the next prototype.

9. References

[CT90] D. Clark and D. Tennehouse "Architectural considerations for a new generation of protocols," *Proceedings of SIGCOMM '90* (Philadelphia, PA, Sep. 1990).

[CNN] CNN Homepage http://www.cnn.com/ Contains 17 GIFs (51.8KB total), 7 JPEGs (33.1KB total), and the HTML file itself is 55.9KB (Date: December 6, 1998.)

[DDS95] John M. Danskin, Geoffrey M. Davis, Xiyong Song, "Fast Lossy Internet Image Transmission" ACM Multimedia '95, San francisco.

[F+96] R. Fielding , H. Frystyk, T. Berners-Lee, J. Gettys, and J. Mogul, "Hypertext transfer protocol – HTTP/1.1," *Internet Request for Comments RFC*, June 1996.

[GCMW 98] R. Gupta, M. Chen, S. McCanne, and J. Walrand. "WebTP: A Receiver-Driven Web Transport Protocol"

[Jac88] V. Jacobson, "Congestion avoidance and control," *Proceedings of SIGCOMM'88*, Palo Alto, CA, Aug 1998.

[JPEG] William B. Pennebaker, Joan L. Mitchell, *JPEG still image data compression standard*,. New York : Van Nostrand Reinhold, 1993.

[Lamp83] Butler Lampson, "Hints for Computer System Design", ACM 1983, 0-89791-115-6/83/010/0033