WebTP: A USER-CENTRIC RECEIVER-DRIVEN WEB TRANSPORT PROTOCOL

by

Rajarshi Gupta

Memorandum No. UCB/ERL M98/77

18 December 1998

COVER

WebTP: A USER-CENTRIC RECEIVER-DRIVEN WEB TRANSPORT PROTOCOL

by

Rajarshi Gupta

Memorandum No. UCB/ERL M98/77

18 December 1998

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Abstract

The use of TCP for the Web has caused a variety of performance problems because the interactive request/response nature of Web traffic is incongruent with the sequenced, bi-directional, continuous, byte-stream model of TCP. We believe that these problems can be overcome by abandoning the constraints imposed by TCP and designing a new receiver-oriented transport protocol for the Web that leverages the concept of Application Level Framing (ALF). In this report, we present a receiver-oriented, request/response protocol for the Web that is amenable to ALF and compatible with the dynamics of TCP's congestion control algorithm.

The resulting protocol — **WebTP** — optimizes the transport of a document from the sender to the receiver by taking into account a number of different factors like the contents of the page, the state of the network, the available hardware at the client and even the preferences of the user. We set up schemes to represent all of the above information, and design a system to implement the optimization structure. The computationally feasible methodology adopted at the receiver allows it to determine an optimal order of transport for the objects contained in the document. The resulting transfer is optimized with respect to suitable utility functions and yields maximum satisfaction to the user.

Such a framework for optimization demands a transport layer that is aware of the application and is controlled at the receiver end. Our protocol is designed to be completely receiver-based in terms of transport initiation, flow-control and congestion-control. In support of our receiver-driven design, we developed a novel retransmission scheme that is robust to delay variations and can operate without an explicit "ack clock". The resulting flows achieve efficient network utilization and are qualitatively fair in their interaction amongst themselves and even with competing TCP flows. The report also provides detailed simulation results to support the protocol design.

Contents

List of Figures

Chapter 1

Introduction

The Web is a deceptively simple abstraction: a collection of distributed objects are each named according to simple conventions — the uniform resource locator (URL) — and are stored across an ensemble of object servers that span the global Internet. Clients retrieve objects from servers through a request/response style of interaction embodied in the hypertext transport protocol (HTTP) [F+96]. To avoid the task of designing a new transport protocol from scratch and to facilitate rapid deployment, the original Web architecture simply layered HTTP on top of the existing transport protocol.

Layering HTTP over TCP affords tremendous benefit: TCP is ubiquitous, has been extensively tuned, has a well-developed set of congestion control algorithms, and is well behaved under high degrees of multiplexing. TCP provides a connection-oriented, reliable, bi-directional byte stream, and as such, must transform the underlying unreliable Internet packet service into a reliable byte stream. To do so, TCP establishes a "connection" between the two communicating end-points via a "three-way handshake," breaks the application's byte stream at arbitrary boundaries to frame it in packets, and performs end-to-end automatic repeat/request (ARQ) to implement reliable delivery, all under the covers and hidden from the application using the service.

Because a congestion-controlled transport protocol like TCP is so difficult to design and implement [Pax97], there is a strong incentive to reuse it across multiple applications. Consequently, a "one size fits all" solution to network transport protocol has emerged, where almost every modern network application — e.g., remote terminal access, bulk file transfer, data base update, and the Web — is layered on top of TCP. Of course, this means that TCP must somehow offer a universal service that is well matched to each of these applications. Unfortunately, the conflicting requirements of this broad range of applications cannot be simultaneously satisfied with a monolithic protocol, and instead, performance is sacrificed in favor of the TCP lingua franca.

The use of TCP for the Web, in particular, has caused a variety of performance problems because the interactive request/response nature of Web traffic is incongruent with the sequenced, bi-directional, continuous, byte-stream model of TCP. An example of this *impedance mismatch* between TCP and HTTP is the impact of Nagle's algorithm [Nag84], a scheme to avoid the overhead of small packets (e.g., from key strokes in a telnet session) by coalescing them into a single larger packet when possible. The TCP sender delays transmission of small datagrams in the hope that the application might attempt to transmit more data, trading off delay to reduce per-packet overhead. Unfortunately, while Nagle's algorithm is beneficial to telnet, it is detrimental to interactive Web transfers, where small requests are delayed unnecessarily and thus application interactivity suffers [Hei97, N+97, PK98].

While this particular performance problem can be easily overcome by disabling Nagle's algorithm in TCP, a number of other fundamental performance problems result from pathological interactions between TCP and HTTP:

- the TCP three-way handshake incurs unnecessary connection-setup delay even though the Web communication model requires fundamentally only a single request and response;
- TCP's slow-start congestion control phase can impact interactive performance since it "restarts" after user idle periods [Hei97];
- TCP performance suffers from redundant network adaptation because multiple, independent connections are often used for physically co-located transfers [N+97];
- packet reordering in the network causes TCP to withhold the delivery of received data that is potentially useful but misordered, thus degrading interactivity;
- the framing of application data units (ADUs) onto packets cannot be controlled by the application and thus individual packet losses incur increased performance degradation;
- a Web user's requirements and priorities may vary rapidly and on a fine-grained basis at the receiver (e.g., as a user manipulates browser controls), yet these receiver-driven requirements can have no influence over how packets are sent and prioritized at the server-side of an in-progress TCP-based Web connection; and finally,
- the connection-oriented nature of TCP places a disproportionate share of responsibility on

the sender side of the protocol, leading to unnecessary complexity in Web servers designed to handle very large-scale workloads.

The many solutions suggested for the above problems may be classified into three categories:

1. HTTP Enhancements: These schemes try to improve the performance by altering HTTP without modifying TCP. The advantage lies in the very fast deployment (Browsers get updated every six months) and not having to deal with any alterations in the underlying transport layer. Persistent HTTP (P-HTTP) [PM95] and HTTP1.1 [F+96] are instances of research efforts that fall in this category.

2. TCP Enhancements: These try to enhance the underlying TCP protocol without altering HTTP. Improving the efficiency of various TCP algorithms like loss recovery and congestion control then have favorable effects on all applications that use this service. Transaction TCP (T/TCP) [Bra94] and fast-start [PK98] are various approaches that follow the TCP-enhancement schema.

3. Proxy Enhancements: These schemes involve the creation of a proxy in the middle of the TCP connection — which interferes with the packet flow from the sender to the receiver and alters it suitably to satisfy the needs of both sides. Work in this area include the Snoop protocol [BSAK95] for TCP over wireless and the dynamic distillation proxy scheme [FGBA96].

All these solutions, while effective in their own niche, fail to address the overall problem. Even when composed together, they do not pose a comprehensive solution for the problems of HTTP/TCP. We believe that these problems can be overcome by abandoning the constraints imposed by TCP and designing a new receiver-oriented transport protocol for the Web that leverages the concept of Application Level Framing (ALF) [CT90]. ALF says that to optimize communication paths within the application and across the network, an application's semantics should be explicitly reflected in the design of its network protocol. Although ALF has been assimilated into modern protocols like the Real-time Transport Protocol (RTP) [SF96] and is now part of the network research community's folklore, paradoxically it has not been systematically applied to previously existing applications like the Web.

With an ALF framework, Web data can be intelligently framed onto network packets so that a client browser can process received packet data in an arbitrary order and allow retransmissions of missing packets to be processed "in the background". This resiliency to missing or reordered data would substantially enhance the interactivity and performance of a Web browsing session since connections are never unnecessarily stalled waiting for retransmitted packets. Moreover, by adopting a receiver-driven design, such a protocol could quickly react to changing user requirements (as inferred through the disposition of the browser's user-interface) and modify the priority of pending data requests on the fly. By resting the fate of reliable delivery on the receiver rather than the source (as proposed in NETBLT [CLZ87]), the client can tailor its reliability requirements as necessary, e.g., by easily aborting a transfer that becomes irrelevant due to dynamic user behavior. Finally, a receiver-driven design reduces the complexity of the server implementation and amortizes protocol state management and function across the large number of clients that interact with a centralized server.

To realize the advantages of ALF and a receiver-based design, we have designed a new transport protocol for the Web called *WebTP*. WebTP requires a number of new important protocol components that our design weaves together into a comprehensive protocol framework, including:

- a naming scheme that allows receivers to request data in a semantically-aware, fine-grained fashion from the server;
- an optimization framework that accounts for user requirements in transport protocol;
- data representations for Web content that are progressively-refinable and thus enable fine-grained adaptation (e.g., as a rate-limited packet stream is received, important image content might be progressively refined expediently while background content is filled in more slowly);
- a congestion control framework whose dynamics are compatible with those of TCP;
- session-oriented caching of congestion-control state in order to reuse flow parameters across several connections in a session
- a specific protocol specifications and packet formats; and,
- a flexible API that allows programmer's to reflect application requirements and priorities across the application/protocol boundary.

The inspiration for this project comes from the WebTP proposal [AMTVW98] made by the Department of Electrical Engineering and Computer Science at UC Berkeley to the National Science Foundation in March 1998. Much of this Introduction chapter is also attributed to that proposal, which presented the motivation and the long-term goals of the project.

In this report, we do not present designs for all of these important protocol components nor do we present definitive proof that WebTP will perform fundamentally better than TCP/HTTP. Instead, we argue that a comprehensive and foolproof design of WebTP is an iterative process that will require many interdependent contributions and defer much of this effort to future work. Here, we present the first steps towards a World Wide Web which optimizes Web transfers to suit the user and the current situation. We describe the overall view of the future Web, and define a qualitative optimization scheme that will help us reach the goal.

We also address the first crucial and open question for WebTP: Is it possible to devise a receiver-oriented, request/response protocol for the Web that is amenable to ALF and compatible with the dynamics of TCP's congestion control algorithm? We claim that the answer to this question is "yes," and in the remainder of this paper present evidence to this end. We sketch the design of a prototype protocol for WebTP and, through simulation, study its dynamics both alone and against a competitive TCP traffic load. We find that our receiver-driven protocol is efficient in terms of network utilization and is as qualitatively fair as TCP.

Chapter 2

WebTP View of the Internet

In this chapter we present a view of an Internet in which web transfers are optimized to include the user into the "Transfer Loop". The basic purpose of this chapter is to present, in a conversational way, our driving motivations and the eventual goals of the project as outlined in [AMTVW98]. We will use simple examples and compare with today's standards to describe this utopian view. Needless to say, most of the contents of this chapter lies in the category of Future Work and will be handled as the project progresses.

2.1 Comprehensive Framework for Web Transport

The web transfer framework in use today is significantly rigid and unadaptive — once an user clicks on an URL, all control passes over to the protocol, and the page is transported in the order in which it was set up by its creator. Sometimes, the network performance is outstanding and the entire page is rendered within seconds — at other more frustating times, we have to suffer through long delays. In future, when the protocol utilizing Application Level Framing is aware of the application and its requirements, the transfer should take into account *all* of the following available criterion.

- the contents of the page;
- the hardware at the client;
- the current state of the network; and
- the preferences of the user.

2.1.1 Contents of the Page

When a web page is created, the creator of the page has a fair idea about the relative importance of the different objects on the page. Consequently, the creator should designate 'values' for each of the individual objects on a page (e.g., on a news page like the New York Times, the main headline could get a value of 10, each of the smaller stories get a value of 5, four pictures on the page each have a value of 1 and the six advertisement banners each have value 2). The contents of a page could be easily described by a metadata table which would include attributes like object type, object size, value and the acceptable delay for the object.

A little more sophisticated version of such a table could provision for multiple versions of an object (e.g., different resolutions for a picture). Another important feature is the 'browse time' of an object — the expected time taken by an user to browse that object. The browse time is a function of the size of the display, and the type of the object. Such an attribute is of outstanding value in web transfers because the browsing time of the first few objects may be used for the transport of the remaining objects, whence the user sees no effective delay for the later ones.

2.1.2 Hardware at the Client

The hardware present at the client is a vital component affecting web transfer and display, yet it is completely ignored by today's framework. A couple of intuitive examples make this very obvious. Suppose the machine an user is working on is not equipped with a sound card — in that case it is obviously futile to transfer any audio objects to that machine. Or if the user is browsing the web on a PalmPilot, which has a low-quality, small, black-and-white display — then it would not be efficient to retrieve the large jpeg files for colorful pictures. Again, suppose the client (a laptop) is alternately connected over LAN and a Modem — since it knows its own connectivity status, the client can thus decide whether to ask for the high-resolution image or settle for the lower one.

In a futuristic situation where the client is well aware of the available hardware and the requirements of the application, it could make very well-judged decisions about what objects to transfer, as well as the optimal way to carry out these transfers.

2.1.3 Current State of the Network

A web transfer is of course highly dependent on the current state of the network. One should first acknowledge many distinguished research efforts that have attempted to optimize a web transfer depending on the current network state (e.g., [FGBA96, N+97]). We will indeed leverage such works to account for network variability in web transfer.

Considering that many of the network state variables are continually calculated by the Transport layer of the protocol, it would be useful to utilize this knowledge in the Application layer too. For example, if the protocol is aware of the cumulative bandwidth currently available being only 56.6 kbps, it should be able to make an intelligent choice and not attempt to start a video stream at 150 kbps.

Furthermore, these network variables calculated are also used to compute the expected arrival time of the objects (transport time = size / transfer rate). An estimate of the loss rate being experienced could alert the reliable application to change the level of error protection to give to the data. An important aspect to consider is that while the previous factors mentioned are fairly static, and not expected to change midway through a transfer, the state of the network is prone to rapid deterioration. Consequently, the planned protocol needs to be dynamic and able to adapt quickly to changes in the underlying network state.

2.1.4 Preferences of the User

The preferences of the user are probably the most important component that has been overlooked in web transfer. Currently, the web browsers allow little control to the user — like switching off images, or not starting any java aplets. We want to emphasize the fact that the ultimate goal of any transfer is to satisfy the user, and it is critical to incorporate the preferences of that user into the transport mechanisms. As such, the measure of utility generated by a page depends not only on the contents of the page and when they arrive, but also on the preferences of the user who is viewing the page.

User A might be really concerned about the sports scores on ESPN's page, while user B might care more for the visual excitement captured by the photographs. There is no way for a protocol to guess such personal preferences unless they are captured in the form of User Rules. Such rules could be learned by using a web form wherein the user clicks in his/her choices, these could then be stored as user profile in the client machine. However, it is worthy of noting that preferences vary widely across sites — a user visiting CNN's web site is likely to pay more attention to the news items and less to the accompanying photos; the same user while visiting Louvre Museum's site would care most for quality photographs, and considerably less for the accompanying text. Such choices would then have to be stored as $(user, website)$ pairs, like bookmarks files. The learning

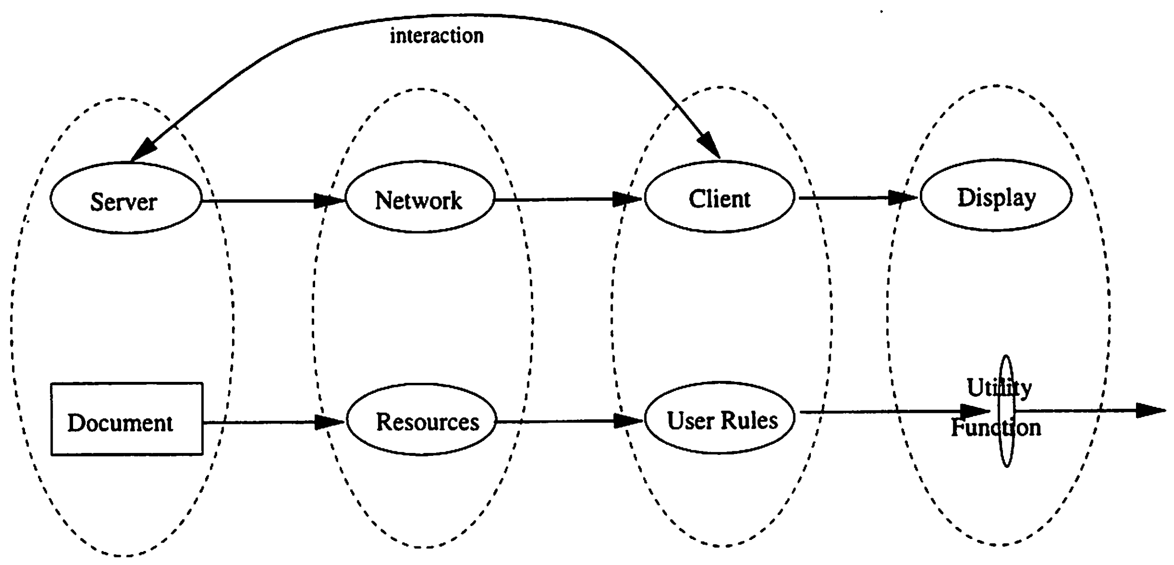

Figure 2.1: Conceptual View of WebTP

process of such schemas would also have to be different, and the protocol would need to "learn" the user preferences by evaluating his/her actions during visits to those pages.

The preferences of the user are also applicable when the document is finally displayed. Each object on the page has some utility for the user, depending on its arrival time. The quantity of utility imparted by each object are calculated using Utility Functions (discussed in detail in Section 3.4). The utility generated by the objects together make up the 'user satisfaction' which is the ultimate quantity that measures the success of a transfer.

2.2 Optimized WebTP Transfer

A transfer optimized by WebTP would occur as described in the following list. A conceptual view of the transfer, emphasizing the interaction between the client and the server is demonstrated in Fig. 2.1.

- Firstly, the receiver asks the sender to send the page;
- The sender begins by sending the document descriptor, containing a description of the contents of the page;
- The receiver takes the descriptor page and applies its set of rules on it. These rules incorporate the hardware constraints, as well as the preferences of the user. The result of this is an

altered version of the metadata table, with changed values for the utilities. Objects deemed unnecessary for transfer have value 0;

- In the background, the first few objects on the page are transferred, and displayed;
- These first transfers are utilized by the client to estimate the state of the network, especially the available bandwidth;
- Once the bandwidth is calculated, $\frac{size}{bandwidth}$ gives the required transfer time for each of the objects;
- Given the value, the transfer time of each object and the acceptable delay (i.e., an object is useful if it appears before its acceptable delay, and of no value if it appears later), the problem is recognizable as an optimization problem.
- The transfer order is optimized with respect to some utility function that captures the satisfaction granted to the user by the contents of the page;
- The receiver then controls the transfer by getting the objects using a request-response model;
- The transferred objects, finally displayed, generate maximum satisfaction for the user.

Chapter 3

WebTP Optimization

The ultimate goal of WebTP is to optimize a transfer so that it takes into account all the factors described in Chapter 2. In this chapter, we attempt to formalize the ideas presented before and formulate the problem in a scientific manner. The ideas presented here are not necessarily the final version of the WebTP scheme. In fact, it is likely that the eventual schemes will be radically different. However, in this report, we present an intuitive and feasible scheme for a solution to the problem, which gives us a direction to focus on and will likely expose the potholes in the road to a viable implementation.

The WebTP system can be characterized by the mathematical expression describing user satisfaction:

$$S = f(D, C, N, U)$$

Here,

$$\begin{aligned} S &= User\ Satisfaction \\ D &= Document\ Descriptor \\ C &= Client\ Set\ of\ Rules \\ N &= Observed\ Network\ State \\ U &= Utility\ Function \end{aligned}$$

To express in words: Each document to be transported includes a descriptor D that summarizes (in the form of metadata) the contents of the page. This descriptor D is initially transmitted

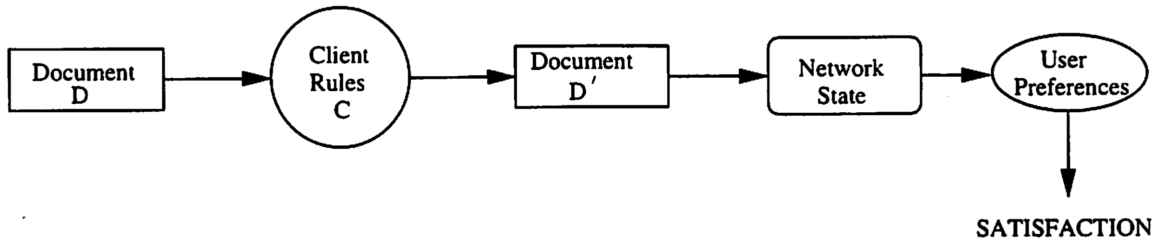

Figure 3.1: Document Transformation in WebTP

to the client. Keeping with the client-based aim of the model, the client applies its set of rules C on the document descriptor D to generate a new scheme D' of prioritizing the transfer of the individual components of the page. The client also has to estimate the parameters to decide the observed network state N. Next it applies the utility function U on D', based on the observed state N, to generate the exact specifications for the transfer of the page. The transfer, which follows this client-generated scheme is optimized to generate the maximum user satisfaction S. Fig. 3.1 explains it in the form of a flow chart.

We will now look at each of the individual components in more detail.

3.1 Document Descriptor D

The document descriptor is described by a metadata table that summarizes the contents of the document. A document is viewed here as a set of items $1, 2, \ldots N$ and each object $O[i]$ has a set of attributes attached to it. The attributes of object i are all or some of the following:

- $O[i].size \rightarrow$ size of the object (in kB). Fragmentation information is also given here to inform the receiver of the preferred mode of fragmentation, if necessary;
- $O[i].type \rightarrow$ type of the object (e.g., text, jpeg, audio);
- $O[i].position \rightarrow$ display position on the page;
- $O[i].value \rightarrow$ value of the object as determined by the creator of the document, using any suitable unit of value;
- $O[i].priority \rightarrow$ display priority of the object (e.g., objects of priority 5 must be displayed before objects of priority 4);

- $O[i].delay \rightarrow$ the acceptable delay for this object i.e., in any transfer — this object must arrive before this time;
- $O[i].browse \rightarrow$ the "estimated" amount of time spent by an user in browsing the object. The idea is that objects that are lower in the list be allowed to incur greater delay while the user is busy browsing the previously transmitted objects.

It is conceivable to have different versions of the same object on the page (like different versions of the same picture) — in which case we can expect

$$\begin{array}{lll} O[2a].value = 20 & O[2b].value = 5 & O[2c].value = 30 \\ O[2a].size = 5Mb & O[2b].size = 1Mb & O[2c].size = 25Mb \end{array}$$

Examples of metadata tables describing the objects on a page may be found in Chapter 5 (Tables 5.1 and 5.2).

The purpose of this document descriptor is to present the client with a complete description of the page. It also provides the initial set of values for each object — it is imperative to include this in D because it is the creator of the document who alone knows the relative values of the objects on the page.

3.2 Client Set of Rules C

The client has its own set of rules C that act upon the descriptor D to produce an intermediate version D'. The set of rules embody the special characteristics of the client state (hardware, software and user) which only the client is aware of. For the machine lacking a sound card — rules C will then give 0 value to any audio object.

The client set of rules considers the size and type of objects and acts on the value and the priority. Currently we are considering a very simplified additive/multiplicative model:

```
if O[i].type = xxx
        set O[i].value = $\alpha$.O[i].oldvalue + $\beta$ ;
        set O[i].priority = O[i].oldpriority + $\gamma$ ;
Here,   0 $\leq \alpha \leq$ max factor K
        $\beta, \gamma$ could be +ve or -ve
OR
```

```
if O[i].size > 2 Mb
        set O[i].priority = O[i].oldpriority - 1;
```

This algorithm enables the client to:
— Prevent transmission of unwanted object types
— Give higher priority to objects that are valued more under the current circumstances
— Delay objects that take too much time and consequently block other objects

A very strong advantage for this scheme is the user involvement. Such a set of rules can be easily stored in the user profile and the transfers will thus be optimized for each user

3.3 Network State N

An integral part of the problem is the estimation of the network state. It is the current state of the network which decides the amount of delay incurred by each object at the client. At a minimum, we require the bandwidth, round-trip-time, delay, delay jitter (for real-time applications), and loss rates.

The methodologies for estimating all of the above at the sender have been studied for a long timenow — in order to effect measures like admission control, congestion control and error recovery. We will try to leverage these knowledge to implement such schemes from the point-of-view of the receiver. Ongoing efforts in this field, like the SPAND project [SSK97], also provide valuable insight into possible solutions.

Inherent advantages and disadvantages of the receiver-based model are described in more detail in Section 6, together with experimental results of a simulated receiver-driven scheme.

3.4 Utility Function U

The structure of the utility function determines the final measure of satisfaction gained from the object. An initial model of the system works as follows:

Model 1

We view a transported document as a series of items $1, 2, \ldots M$ which are actually transported (i.e., this is the subset of objects chosen from the complete list of N objects on the page, so $M \leq N$).

The user receives object i at time $T(i)$, where $T(i)$ is given by

$$T(i) = \frac{\sum_{j=1}^{i} O[j].size}{R}$$

Here, R is the rate. i.e., the time taken for object i to arrive is the sum of the times taken to transmit all the objects up to and including i.

Then, the satisfaction generated for the user is given by

$$S = \sum_{i=1}^{M} O[i].value \times \mathbf{1}\left\{T(i) < O[i].delay\right\}$$

This simple satisfaction model considers that item i has a fixed value $O[i].value$ to the user provided it reaches before $O[i].delay$ and 0 value otherwise.

Model 1b

The objects are still transported in the same order, but the utility of the object now decays linearly rather than be the binary scheme of 1 or 0.

Thus, satisfaction

$$S = \sum_{i=1}^{M} O[i].value \times u$$

where,

$$u = \begin{cases} 1 & \text{if } T(i) < O[i].delay_{min} \\ \frac{O[i].delay_{max} - T(i)}{O[i].delay_{max} - O[i].delay_{min}} & \text{if } O[i].delay_{min} \leq T(i) \leq O[i].delay_{max} \\ 0 & \text{if } T(i) > O[i].delay_{max} \end{cases}$$

Model 2

The above model is extended to include the browsing time described previously.

Here satisfaction

$$S = \sum_{i=1}^{M} O[i].value \times \mathbf{1}\left\{T(i) < B(i)\right\}$$

$T(i)$ is as described above,
while $B(i) = \Delta + \sum_{j=1}^{i-1} O[j].browse$, the total time spent by the user in browsing the objects until this one.
Δ = initial time that a user waits before anything arrives.

Each of the above models may be modified to incorporate multiple versions of the objects.

3.5 Decision

The system characterizes the problem as trying to maximize $S = f(D, C, N, U)$.

Once the maximization is carried out, it generates the list of objects (from the original contents of the page) which are to be transferred, and their ordering. These objects are then sequentially requested from the sender. In a more advanced scheme, the client is also enabled to designate the amount of error protection (e.g., Forward Error Correction) to be provided to each individual object.

Implicit under this setup is the assumption that the network resources N remain constant throughout the transfer — obviously invalid in the face of rapidly changing network parameters. The protocol thus needs to recalculate on the fly all of the above and change its pattern of requests to the sender. We are working on effective algorithms to solve the problem of carrying out these calculations in an incremental and computationally feasible manner.

Chapter 4

Optimization Theorems

A first step to optimize transfers is to think of a priority-based scheme,in which each object $O[i]$ in document D' has a priority field (say from 1 to K, with 1 being the lowest). The simple transfer scheme would then transfer all objects of priority K first, then those of priority $(K-1)$ and so on. Amongst objects with the same priority (say objects $a_1, a_2, ...a_L$ all have priority J, $(J < K)$ — the order is decided by maximizing the function

$$S_J = \sum_{i=1}^{L} O[a_i].value \times \mathbf{1}\{T(a_i) < D(a_i)\}$$

Here,

$$D(a_i) = O[a_i].delay$$

and

$$T(a_i) = \sum_{k=1}^{i} \frac{O[a_k].size}{R} + \sum_{\forall O[b]:O[b].priority>J} \frac{O[b].size}{R}$$

i.e.,

$$\begin{aligned} T(a_i) &= \text{time when the current object gets there} \\ &= \text{Time taken to transmit all objects up to itself of this priority} \\ &+ \text{Time taken to transmit all objects of higher priority} \end{aligned}$$

An analysis of the system for the schemes outlined in Chapter 3 yields some results, as summarized in the following two theorems.

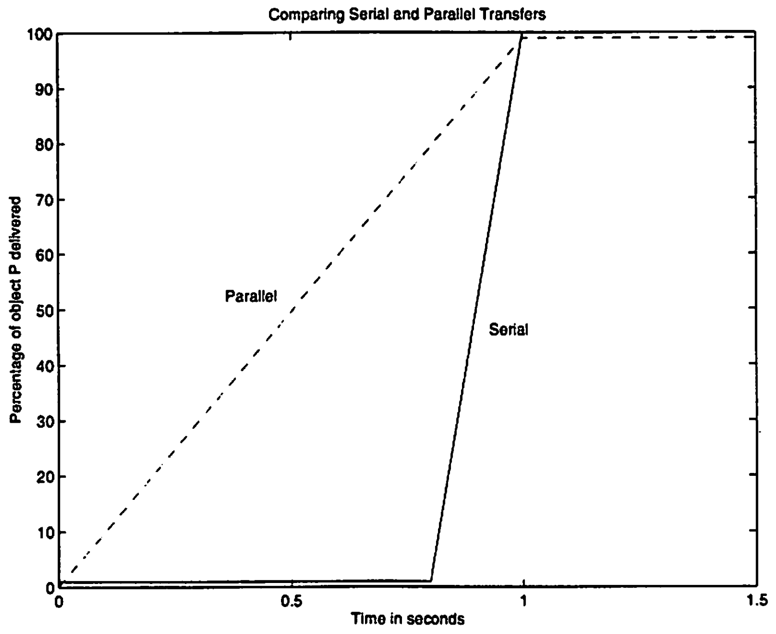

Figure 4.1: Comparing Serial and Parallel Transmissions

4.1 THEOREM I

The utility of the parallel transmission is always less than or equal to that of the optimal serial order.

For the utility models presented in Section 3.4, we compare the methods of Parallel and Serial transmissions. In a serial transmission, the objects are transmitted one at a time, while in a parallel transmission the available bandwidth is split into N streams, which then retrieve N objects in parallel. An example of the two types of transmission are shown in Fig. 4.1, which shows the transmission for some object P. In the serial transmission, it begins to be transferred at 0.8 seconds, at which time all the bandwidth is utilized for it and it arrives at 1.0 seconds. For the parallel scheme, the transfer of object P starts at 0 seconds itself, but it utilizes only a fraction of the bandwidth, so the transfer occurs far more slowly and continues till 1.0 seconds. The y-axis in the graph shows the percentage of object P received at the destination.

Proof:

For simplicity of notation we hereby denote $O[i].size$ as $S(i)$, $O[i].browse$ as $b(i)$ and $O[i].delay$ as $D(i)$.

When there are N objects sharing a processor of rate R, each gets effective rate $\frac{R}{N}$ initially. When there are N' objects left to be transmitted, the effective rate is $\frac{R}{N'}$.

Consider the largest object that is sent during the transfer — say it is object K.
With the parallel scheme of transfer, this is the last object to reach the receiver, and the time when it finishes is the same as the time when all of the objects finish, i.e.,

$$T(K) = \frac{\sum_{i=1}^{M} S(i)}{R}.$$

where M is the cardinality of the subset of objects actually sent during the transfer ($M \leq N$).

Since object K arrived at the receiver — its utility was non-zero (since we do not send any object that has utility 0). So it arrived within its acceptable delay. Then,

$$T(K) = \frac{\sum_{i=1}^{M} S(i)}{R} \leq D(K)$$

Since $D(K)$ is greater than the time taken to send all the objects, we can safely send object K last using the serial transfer without any decrease in the total satisfaction.

Now suppose the second largest object (say object L) reached at time $T(L)$,

At time $T(L)$, part of object K had already been sent. Consequently,

$$T(K) - T(L) < \frac{S(K)}{R}.$$

So that,

$$T(K) - \frac{S(K)}{R} < T(L).$$

Denote with primes the corresponding figures for the serial rule of transfer. Then,

$$T'(K) = T(K)$$

But,

$$\begin{aligned} T'(L) &= T'(K) - \frac{S(K)}{R} \\ &= T(K) - \frac{S(K)}{R} \\ &< T(L) \end{aligned}$$

Again, $T(L) \leq D(L)$, since object L had arrived in time for the parallel transfer.
Then $T'(L) \leq D(L)$, which implies that object L gets there in time under the serial transfer rule. Thus there is no decrease in the utility by using the Serial rule.

Moreover, since $T'(L) < T(L)$, we may be able to meet the deadline for some other object L' whose value is more than L and send that in stead to generate higher utility.

Extending the same argument to earlier objects, we see that the serial transfer is never worse than the parallel transfer, and might actually do better. □

Looking at the problem in an intuitive manner, we realise that the parallel scheme is suboptimal because a large object of comparatively lesser value occupies a significant part of the bandwidth — thereby delaying the rest.

The next theorem is much less obvious and proves the existence of an universally optimal ordering of the items.

4.2 THEOREM II

There exists an optimal ordering of the items that is independent of the transmission rate

__**Proof:**__

Consider the models 1 and 2 in Section 3.4. Then we claim that there exists an optimal ordering of the items such that the best strategy is only to send a subset of this list — in the predetermined order. Only the subset to be chosen depends on the transmission rate. This is very important for a practical implementation of the scheme. The transfer may begin by sending a few of the most important objects. The client uses these transfers to estimate the network parameters N, and at the same time computes the optimal ordering of the remaining items. Then it can simply use the estimated rate R to pick the optimal subset of the objects to be transferred (Section 5.2).

The proof differs somewhat for each of the two models.

Model 1

Recall that here user satisfaction U is given by

$$U = \sum_{i=1}^{M} O[i].value \times 1\{T(i) < D(i)\}$$

where

$$T(i) = \sum_{j=1}^{i} \frac{O[j].size}{R}.$$

Assume that for some two objects i, j and rate R it is optimal to send i before j, i.e., $U(xijyz; R) > U(s; R)$ where s is any ordering where j precedes i, and x, y, z are other objects being transferred. Then, $U(xiyjz; R) > U(xjyiz; R)$.

Consequently, $U(xiyj; R) > U(xjyi; R)$, since the systems are exactly equivalent while sending z

Had all the objects arrived in time in both schemes, the utilities would be the same. So here j arrives in time in the first scheme, but i does not arrive in time for the second scheme.

This can happen only if $\frac{S(x)+S(i)+S(y)+S(j)}{R} < D(j)$ in the first scheme,
which implies $S(x) + S(i) + S(y) + S(j) < R.D(j)$ ———— (1)

Also, in the second scheme,
$\frac{S(x)+S(j)+S(y)+S(i)}{R} > D(i)$, for object i to not arrive in time.
Then, $S(x) + S(j) + S(y) + S(i) > R.D(i)$ ———— (2)

Using (1) and (2),

$$\begin{aligned} R.D(j) &> R.D(i), \\ i.e.,\ D(j) &> D(i) \end{aligned}$$

Now, assume that for some other rate R', it is optimal to send j before i.

For objects i, j, u, v, the optimal utility is given by $U(ujvi; R')$. This implies that under rate R', the object set $\{u, j, v, i\}$ reaches the destination before the expiry of the time for the last item i. This gives,

$$\begin{aligned} \frac{S(u) + S(j) + S(v) + S(i)}{R'} &< D(i) \\ S(u) + S(j) + S(v) + S(i) &< R'.D(i) \\ S(u) + S(j) + S(v) + S(i) &< R'.D(j), \quad \text{since } D(j) > D(i) \\ S(u) + S(i) + S(v) + S(j) &< R'.D(j) \\ \frac{S(u) + S(i) + S(v) + S(j)}{R'} &< D(j) \end{aligned}$$

Thus, interchanging i and j still allows j to reach on time. Also, u, v, and i are all sent earlier than (or same as) before.

So there is no decrease in the utility by the interchanging.
This means that we can do as good or better by sending i before j.
Hence the optimal order for R also suffices for R'. □

Model 2

Recall that each object has size $S(i)$, value $v(i)$ and browsing time $b(i)$.

We want to maximize

$$U = \sum_{i=1}^{M} O[i].value \times \mathbf{1}\{T(i) < B(i)\}$$

where

$$T(i) = \sum_{j=1}^{i} \frac{O[j].size}{R}$$

and

$$B(i) = \Delta + \sum_{j=1}^{i-1} b(j)$$

Recall that,

$B(i)$ = total time spent by the user in browsing the objects until this one.

Δ = initial time that a user waits before anything arrives.

$b(j) = O[j].browse$ = browsing time for individual object j.

Again, we make the same assumption — that for some rate R there exists an optimal order where i is sent before j and show that this ordering is the same for any other rate R'.

Assume that for objects x, i, y, j, the optimal order involves sending i before j, i.e., the optimal utility is given by $U(xiyj; R)$. This involves being able to send all of the objects $\{x, i, y, j\}$ in that order before the browsing time expires.

Then, $\frac{S(x)+S(i)+S(y)+S(j)}{R} < \Delta + b(x) + b(i) + b(y)$,
which is equivalent to $S(x) + S(i) + S(y) + S(j) < \Delta.R + b(x).R + b(i).R + b(y).R$ ————(3)

Since it is optimal to send i before j, $U(xiyj; R) > U(xjyi; R)$. This can happen only when the objects $\{x, j, y, i\}$ do not arrive in that order before the browsing time expires.

Then, $\frac{S(x)+S(j)+S(y)+S(i)}{R} > \Delta + b(x) + b(j) + b(y)$; had it been otherwise, i too would have reached on time and the initial order would not be the optimal one.
This gives $S(x) + S(j) + S(y) + S(i) > \Delta.R + b(x).R + b(j).R + b(y).R$ ————(4)

Using (3) and (4),

$$\Delta.R + b(x).R + b(j).R + b(y).R < \Delta.R + b(x).R + b(i).R + b(y).R$$

This gives,

$$b(j) < b(i)$$

Assume that for some other R', it is optimal to send j before i Then, by following the exact line of logic as above we can show that $b(j) > b(i)$ (i.e., Contradiction).

Hence, the optimal order for some rate R is also the optimal order for any other rate R'. $\square$

While the theorem is proved for the model we consider, it often belies intuition. An useful counterexample is a page containing large image files (e.g., paintings) and inlaid text (e.g., information about the pictures). If the user is on a very fast link, he would like to receive the pictures first, look at them, and then read the accompanying text. On the other hand, if the link is extremely slow, it would take a very long time for the pictures to download — the user may then prefer to receive the text first, which will give him *some* knowledge about the contents and then try for the pictures later when more bandwidth becomes available. In such a case, the transfer order is *not* independent of the rate. However, this anomaly arises due to the non-linearity of the user satisfaction (his wish for the photos changes depending on how long it takes and on the contents of the text items), and is beyond the scope of the current model. Such non-linear situations too might be taken into account in the future.

Chapter 5

Optimization Algorithms

Note that the existence of the optimal order still leaves two important questions unanswered:

- How to find the optimal order
- Once the order is decided, how to pick the optimal subset depending on the rate

5.1 Picking Optimal Ordering

The problem of picking an optimal order is currently open. We do not currently have a polynomial-time algorithm for this problem, but are working on it and hope to formulate a dynamic programming solution.

The existing brute-force solution of evaluating all the possible arrangements will surely work, but this yields an exponential solution. For a document that can contain up to 50 objects, this is clearly unacceptable. An option is to do priority-based scheduling where we decide to send all objects of a higher priority before sending any from a lower priority. While this does not theoretically lessen the problem, it can substantially decrease the computations required if we limit the number of objects per priority class (e.g., if we divide the 50 objects on a page into 5 priority classes of 10 objects each, then the computation complexity is lessened from 2^{50} to 5×2^{10}).

All these calculations are carried out at the receiver upon the receipt of just the descriptor. The first few objects of the page are always sent first — and this transmission time is utilized by the receiver to calculate the optimal ordering for the remainder of the objects. Also, carrying out all the computations at the client ensures that the system scales very well.

5.2 Picking Optimal Subset

For the sake of clarity, let us begin by restating the problem of picking the optimal subset assuming the optimal order has been generated:

Model 1

We have a set of N objects $1, 2, \ldots, N$ which need to be transmitted to the receiver. Each object has size $S(i)$, value $v(i)$ and acceptable delay $D(i)$. Given a rate R, we need to find the optimal ordering such that the sum of the values is maximized, under the additional constraint that each object i must also arrive before its own acceptable delay $D(i)$.

NP-Completeness: If we let $D(i) = T, \quad i = 1, 2, \ldots N$, i.e., we are only interested in the sum of the values picked under a global delay constraint T and do not care about the individual delay constraints — then this problem can be recognized as the Knapsack Problem. The Knapsack Problem [CLR90] is well-known in the field of Algorithms as an NP-complete problem — so we can clearly hope to do no better.

We have, however, devised a dynamic programming algorithm to tackle the problem.

Define an array $V[i,t]$,
where $0 < i < N$ = number of objects,
and $0 < t < T$ = maximum acceptable delay

Let $V[i,t]$ denote the maximum value that can be accumulated by picking a subset of the objects $\{1, 2, \ldots i\}$ given that the maximum time one is willing to wait for any object is t. The individual delay constraints $D(i)$ imply that object i must be delivered before time $D(i)$.

Then, $V[N,T]$ gives the maximum value for the optimal order of the objects.

Without loss of generality, let $R = 1$ i.e., the time taken taken to send any object = size $S(i)$.

Under this framework we present a dynamic programming algorithm that uses previous values in the array to calculate the value for the current location.

```
Initialize
        V[0,t] = 0 for $0 \le t \le T$
        V[i,0] = 0 for $0 \le i \le N$
for i = 1 to N
```

```
        for t = 1 to T
             leave_val = V[i-1,t]
             take_val = v(i) + V[i-1, min(t,D(i))-S(i)]
             V[i,t] = max (leave_val, take_val)
        next t
next i
```

Explanation: For any object i, we can either send it or leave it.
If we leave it, then the utility generated is the same as the utility from the subset $\{1, 2 \ldots i-1\}$ and is given by $V[i-1, t]$.
If we send object i, we get its value $v(i)$ but are required to send the remaining objects in time $t - S(i)$ or $D(i) - S(i)$, whichever is lower.

This algorithm is of the order $O(NT)$ as seen from the main loop in the pseudo-code.

Typical values can have a document with 50 objects ($N \approx 50$), and the maximum wait time could be 5 minutes. With a granularity of 0.1 seconds, this gives

$$T = 5 \times 60 \times 10 = 3000$$

Even then the number of computations required is $50 \times 3000 = 150000$, which is easy to accommodate.

Once the maximum value is calculated, we need to generate the exact list of items to send. That operation may be carried out as follows:

```
x = maxobjects($x \leq N$)
y = maxdelay ($y \leq T$)
while (x > 0)
        if V[x,y] == V[x-1,y]
             Discard (O[x])
             x = x-1
             y = y
        else
             Transmit (O[x])
             x = x-1
             y = y-S(x)
        end
end while
```

This algorithm will output the optimal list of objects in their reverse order, for any subset of the complete list. Essentially, at each step, it checks if the current object was sent or not and correspondingly moves the pointers x and y.

Model 2

Recall that here,

$$U = \sum_{i=1}^{M} O[i].value \times 1\{T(i) < B(i)\}$$

where $T(i)$ is as described before, while

$$B(i) = \Delta + \sum_{j=1}^{i-1} b(j)$$

The individual delay constraint is made more realistic by replacing the "acceptable delay" in the previous model by the sum of the browse times of the already displayed objects. Basically, we consider the situation where each object transmitted has a browse time, and this time may be utilized for the transport of later objects.

Here each object has size $S(i)$, value $v(i)$ and browse time $b(i)$.

For this variant of the algorithm, we utilize the array $V[N,T]$ as before.

But we also require a parallel array $B[N,T]$, where $B[i,t]$ = the total browsing time accumulated by sending the optimal set of objects given by $V[i,t]$

The main loop of the algorithm now becomes:

```
for i =1 to N
       for t = 1 to T
            leave_val = V[i-1,t]
            take_val = v(i) + V[i-1, min(t-S(i),B[i-1,t-S(i)])
            if (leave_val >= take_val)
              V[i,t] = leave_val
              B[i,t] = B[i-1,t]
            else
              V[i,t] = take_val
              B[i,t] = b(i) + B[i-1,t-S(i)]
            end
        next t
next i
```

Explanation: At each stage, we check if the object may be sent before the accumulated browsing times till the last object expires. Other than updating the value array, we also need to update the browsing time array corrspondingly.

Once again, this is of order $\mathbf{O}(NT)$ and picking the actual list is carried out just as before.

5.3 Experimental Setup

In order to visualize the system better, we setup a realistic scheme that shows the effect of user optimization on a document transfer. We began by taking a typical web page (the DIGEST Daily News page from CNN) and modified it slightly. The experimental page may be accessed at *http://www.path.berkeley.edu/~guptar/SAMPLE/index.html.* Our modifications were simply to pick out some of the audio and video objects linked from the page and embed them into the document itself. These objects are currently linked away from the document since not all users can see them — however this means that people who would care to see them need to click in order to receive each individual object. WebTP would eliminate this problem by the system of user optimization as described earlier in the section. Further, we set up a list of values and acceptable delays for the objects to test our algorithms.

Firstly we simulated the web transfer of the document over a real link (the link was on the local network and was getting a bandwidth of about 500 kbps) — the (blue) curve shown in Fig. 5.1 denotes the arrival of the bytes over time. The vertical (red) lines describe the completion of transfer of each object, each vertical line denoting the arrival of a single object. When the object sizes are small, these lines are bunched closely together (as near the 17 second mark) while the large movie object (No. 22 in Table 5.1) takes up a long interval (from 33 to 55 seconds).

Sl No	Type	Size	Value	Delay
		in kB		in sec
1	text	36	10	5
2	gif	6	8	10
3	au	516	5	50
4	gif	9	4	20
5	gif	10	5	20
6	au	465	5	50
7	gif	6	5	20
8	jpg	100	12	30
9	gif	12	4	45
10	gif	10	5	45
11	gif	5	4	45
12	text	5	6	30
13	gif	5	6	30
14	au	516	3	60
15	au	535	3	60
16	gif	1	1	60
17	gif	4	4	60
18	gif	6	5	60
19	text	10	8	60
20	gif	5	4	60
21	gif	5	4	60
22	mov	1436	3	120
23	gif	3	2	120
24	gif	11	2	120
25	gif	8	2	120

Table 5.1: Typical Web Transfer

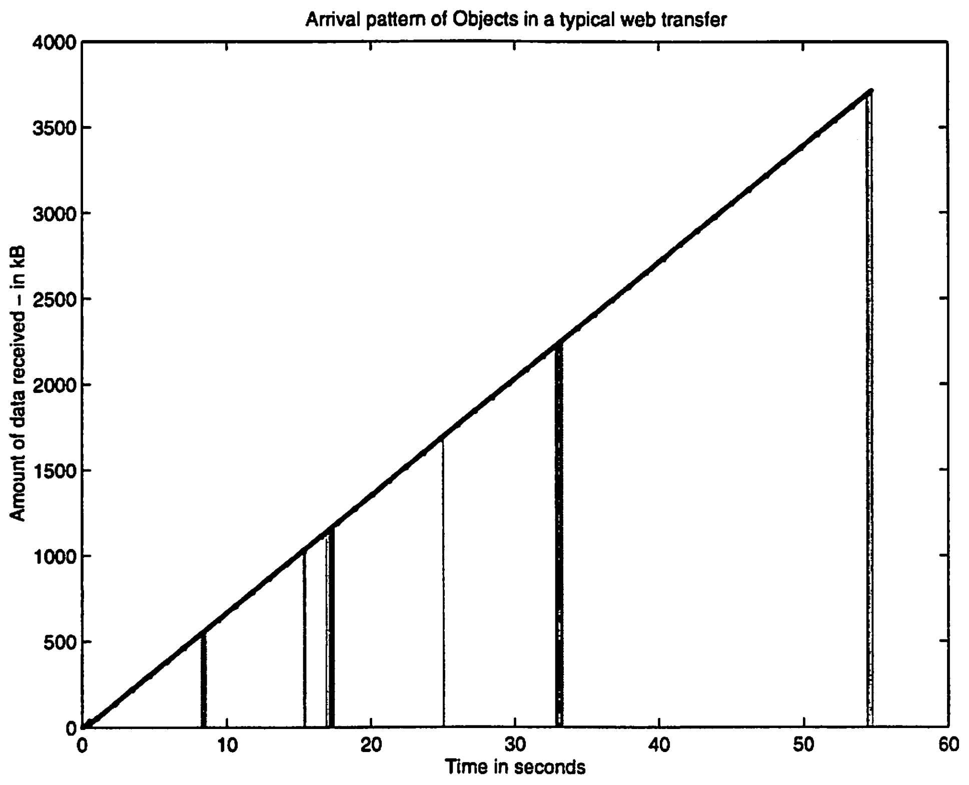

Figure 5.1: Typical Web Transfer

The utility function used for the model was given by the user satisfaction

$$S = \sum_{i=1}^{M} O[i].value \times \mathbf{1}\,\{T(i) < O[i].delay\}$$

The utility of this transfer was compared with the WebTP mode of transfer. We ran the optimization algorithm on the set of objects to find the optimal order — the brute force algorithm sufficed since there were only 25 objects on the page. The algorithm outlined in Section 5.2 was then used to process the list of objects. For each object, we entered its size, value and acceptable delay. The output (for a given input rate) was the actual subset of the objects to transmit. Letting the transfer run its full course (under the same link rate as before), we are able to transmit all the objects — albeit in a new order as shown in Table 5.2.

The (blue) curve shown in Fig. 5.2 is identical to the one in the previous figure — since we allow the transfer to run its full course, at the same rate. The vertical (red) lines however show the reordered arrival of the objects in a WebTP transfer.

As seen from the graph, a large number of the smaller objects having higher value get transmitted first — noticed by the larger number of the vertical (red) lines near the zero mark. Object number

Sl No	Type	Size	Value	Delay
		in kB		in sec
1	text	36	10	5
2	gif	6	8	10
3	gif	6	5	20
4	gif	10	5	20
5	jpg	100	12	30
6	text	10	8	60
7	gif	9	4	20
8	text	5	6	30
9	gif	5	6	30
10	gif	10	5	45
11	gif	5	4	45
12	gif	12	4	45
13	au	465	5	50
14	au	516	5	50
15	gif	6	5	60
16	gif	4	4	60
17	gif	5	4	60
18	gif	5	4	60
19	au	516	3	60
20	au	535	3	60
21	gif	1	1	60
22	gif	3	2	120
23	gif	8	2	120
24	gif	11	2	120
25	mov	1436	3	120

Table 5.2: Optimized Transfer Order for WebTP

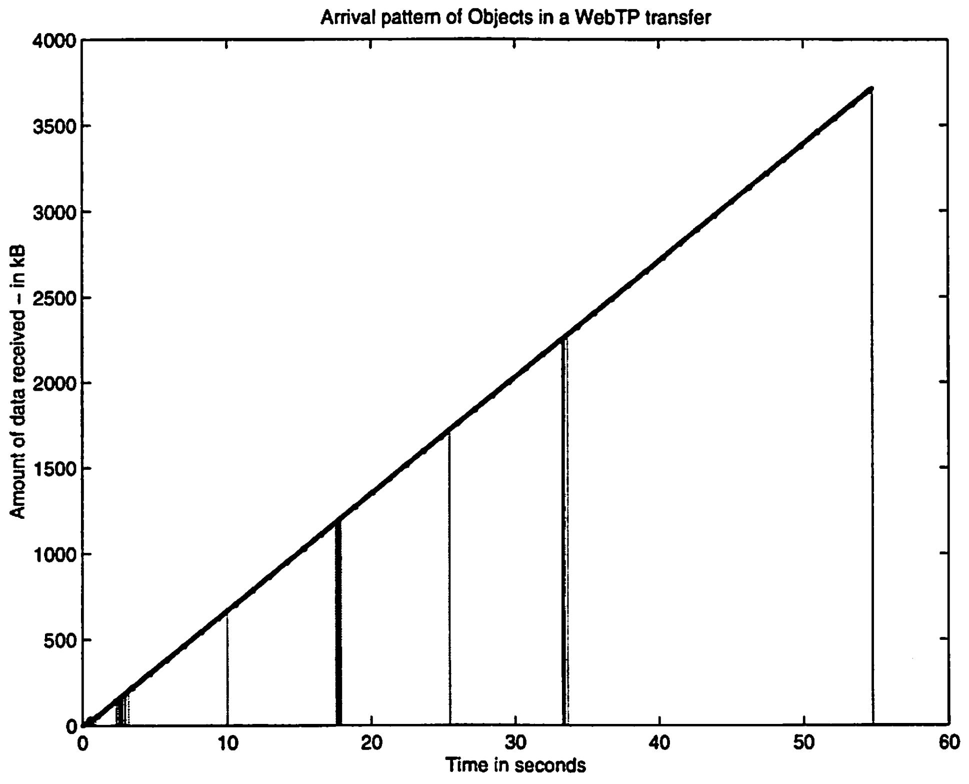

Figure 5.2: Optimized WebTP Transfer

8 (in Table 5.1), having a large utility value, is now transmitted much earlier (as object 5 in Table 5.2). It now arrives over 1 to 3 seconds, while previously it was sent only over 15 to 17 seconds.

We then evaluated the utilities for both schemes by varying the total transfer time. Here we attach the additional constraint of requiring the entire transfer to be completed within T seconds, and ranged T from 0 to 56.2 seconds (the total time required to transmit all the objects). The resulting comparison plot is shown in Fig. 5.3.

At 0 seconds, neither scheme is able to transmit anything — so the utility of the transfer is 0. At 56.2 seconds, all the objects are transferred — so both schemes have 100% utility. The comparison of the curves need to be carried out at the intermediate points. The optimized transfer transmits the more valuable objects first, thereby achieving a high percentage of the utility at a fairly early stage in the transfer. The normal transfer on the other hand, relies on the inherent ordering of the objects and lags behind at all times. Note the situation at 34 seconds, when the two curves meet. This happens since both the normal and the optimized order want to send the remaining few objects at the end. Immedialtely afterwards, WebTP again sends the smaller objects first, gaining more utility than the unoptimized version, which tries to send the large movie object (No. 22 in

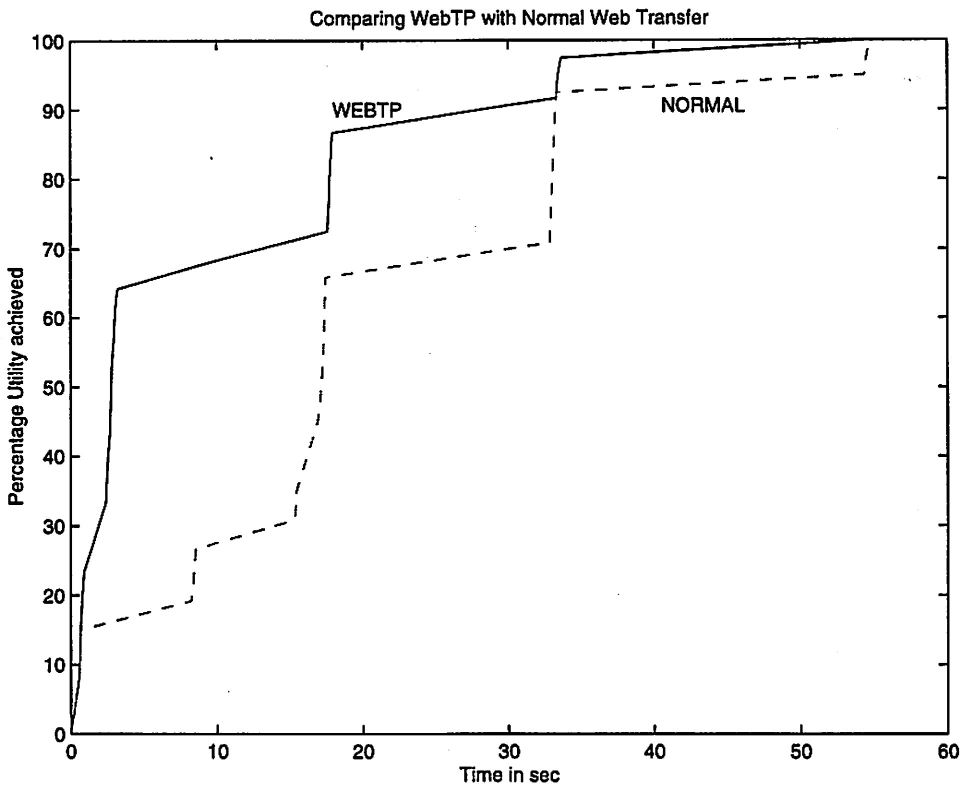

Figure 5.3: Utility of WebTP Transfer vs Normal Web Transfer

Table 5.1).

While the plots shown apply only to the specific instance shown here, the Optimal curve will always lie above the Normal one (the worst we could do is perform no optimization and send in the unoptimized order). The distance between the two curves (instantaneous or average) thus gives the gain from using the Optimal schemes.

Summary of User Optimization

The details of the user optimization scheme used for WebTP were presented in Chapter 3. In Chapter 4, we showed the superiority of using a serial transfer scheme over a parallel scheme. We then claimed and proved that for a document containing a set of objects, there exists an optimal ordering which is independent of the rate of transfer. Given the optimal ordering, we formulated the problem of picking the optimal subset for the transfer (in Chapter 5). This problem, shown to be NP-complete, was solved using a dynamic programming methodology. We also presented an experimental setup using a sample webpage to demonstrate the user optimization schemes. This setup allowed us to compare a normal scheme to the optimized WebTP scheme for the experimental page. Finally, we were able to formulate a quantitative measure for comparing the two schemes, and demonstrated the advantages gained by using WebTP's User Optimization methods.

In the next two chapters, we change focus and try to answer the question about the feasibility of a protocol that shifts the onus of flow control and reliability to the receiver. We show this by presenting the framework of a receiver-driven WebTP protocol, and support it with extensive simulation results.

Chapter 6

Receiver-Driven WebTP Protocol

Rather than design a new protocol from scratch, WebTP follows many of the important lessons gained through the decades worth of research on TCP. We have attempted to bring to bear the more modern concept of ALF on the design of transport protocol for the Web, but at the same time, leverage the well-established and highly successful TCP congestion control algorithms [Jac88].

We follow the request-reply model prevalent in the Web to adopt a scheme whereby a flow is initiated, terminated and controlled by requests from the receiver and the data sent in reply. The object-based nature of the Web is further reflected in WebTP, as it uses ALF-based naming schemes to handle individual objects during transfer.

WebTP is a completely receiver-driven protocol and carries out all its calculations and controls at the receiver end. Consequently, the responsibility of reliable delivery of data lies at the receiver. In carrying out receiver-reliable ARQ we make use of a novel retransmission scheme wherein a "Expected Object Queue" at the receiver is utilized to keep track of outstanding data objects. This queue is handled in an intelligent manner to detect losses, out-of-order packets and retransmission requests.

The flow control scheme of WebTP follows the TCP model of an initial slow-start phase when the rate is increased rapidly to occupy the available bandwidth, and a stable Congestion Avoidance phase whence it does additive increase/multiplicative decrease to attain stability and fairness. During Congestion Avoidance, we employ a hybrid window-based and rate-based scheme. The flow follows the rule of never exceeding one window size of outstanding data — thus ensuring compatibility with TCP. Furthermore, we impose an additional rate control by which the objects to be transmitted are not put on the wire all at once, but are instead paced out at the desired rate.

In the remainder of this chapter — we describe the various components of WebTP and discuss their effects on the functioning of the protocol.

6.1 Flow Initiation and Termination

Since WebTP is a receiver-reliable protocol, and web requests are fundamentally client oriented, it is the WebTP client which initiates contact with the WebTP sender (i.e., the web server). Since the receiver is aware of the address (URL) it is requesting data from, connection is initiated simply by asking for the first ADU (Application Data Unit) from that address. This initial ADU is a standard feature of a page (e.g., a WebPage header) and the receiver always asks for it first. Such an ADU contains metadata about the contents of a page — including the size of the objects and their fragmentation information. Based on the information contained in the metadata, the receiver decides on the exact order of transporting the objects and is also able to ensure reliable delivery.

A flow is terminated when the receiver does not request any further packets. We are thus able to do away with the three-way handshake scheme for connection setup and breakdown and save on this overhead.

6.2 Sender

A chief design goal of WebTP is to minimize the state variables required at the sender. This frees up valuable resources at the sender and enables WebTP to process more flows, as the number of flows being serviced increases. It is of course debatable whether the reduction of workload at the sender is significant and further experimentation will bear more light on this open research question.

Keeping with the principle of minimizing state at the sender, we model the sender side of the WebTP protocol simply as a single queue, with a processor serving it at an adjustable rate ρ. When the sender gets a '*Request*' packet from the receiver, it processes the Request by creating the ADUs requested and placing them into the queue. The priority of the specific ADUs requested may be specified by the client. Currently we designate higher priority to retransmission packets — so these are placed at the head of the queue, while normal packets are placed at the tail.

The sender is not responsible for retransmissions. Once it sends a packet from its queue, the packet is removed from the buffer. The receiver has to specifically ask for a retransmission, upon which the sender re-generates the requested packet(s).

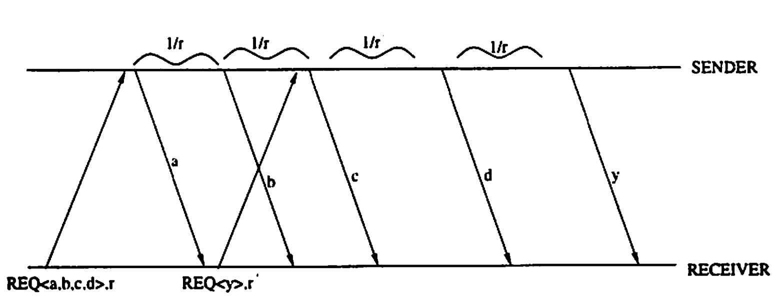

Figure 6.1: Rate Control at Sender

The sender keeps sending packets until its queue is empty. However, it paces the transmission of packets by inserting gaps between consecutive packets to conform to the rate imposed by the processor. The receiver controls the transmission rate by conveying the rate information within the '*Request*' packets. When the sender receives a '*Request*' packet, it checks for updated rate information and alters its processor rate correspondingly. A modified rate does not change the transmission time of the packet at the head of the queue, which had already been scheduled; but affects all the subsequent packets. Note that this rate protocol may be implemented using a single timer that schedules the next packet every time a packet is sent off. This timer value is computed from the rate information and the size of packets.

In Fig. 6.1 the receiver requests that ADUs a through d be sent at rate r and later requests that ADU y be sent with the new rate r'. The sender receives the new rate information when it has already sent ADU b and ADU c has been scheduled $1/r$ time units after b. So while the new rate information r' does not affect the sending of c, the subsequent ADUs (d and y) are scheduled $1/r'$ time units apart. In this example we have assumed that each ADU spans a single packet.

6.3 Window Control

As in TCP, WebTP employs a "congestion window" which limits the maximum number of outstanding packets in the network. WebTP maintains this congestion window (cwnd) at the receiver in contrast to TCP, which maintains it at the sender. Our congestion-control algorithm, like TCP, manipulates cwnd in two distinct phases [Jac88]. Initially the flow undergoes "Slow-Start" when the window is increased exponentially. This period is utilized to increase the rate quickly in order to occupy available bandwidth. Thereafter the flow goes into a "Congestion Avoidance" phase whereby we adopt an additive increase/multiplicative decrease scheme. This allows the flow to continually try to raise the rate to utilize available bandwidth, yet cut back quickly upon detecting congestion.

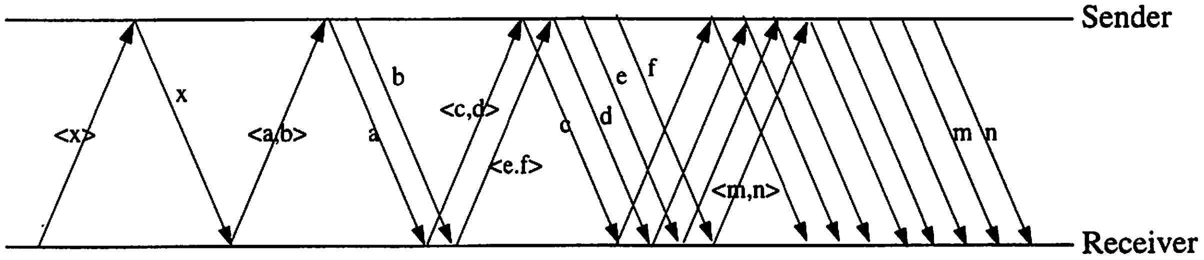

Figure 6.2: Slow-Start Controlled by the Receiver

We thus re-use TCP's congestion control algorithm, but modify it to be run from the receiver end. This control is achieved by explicitly conveying the rate information to the sender using the Request packets, and allows us to aim at compatibility with TCP flows. The window information is not relayed to the sender but used to limit the number of packets requested.

Fig. 6.2 explains the slow-start methodology when controlled from the receiver side. Since a flow begins with the slow-start phase, it does not yet have any information about the RTT or the desired rate. Therefore it may begin by requesting the URL x from the receiver, with infinite (highest possible) rate. Upon receiving the metadata for x, it now asks for packets a and b. As it receives a and b — it increases its window size by 1 every time and consequently asks for two packets each time it gets one, still without sending any rate information. Progressing in the same fashion, we observe that the sender keeps sending packets as fast as it can while the congestion window at the receiver increases exponentially. The slow-start phase is exited when there is a lost packet — signified by a timeout or a gap in the received packets. This initial phase is also utilized by the receiver to evaluate the rate and the RTT suitable for the connection.

One of the problems inherent with the current window-based flow control mechanism of TCP is that it is biased in favor of flows that have a smaller RTT. To observe this, note that after the slow-start phase, the congestion window is allowed to increase by 1 every RTT. Consequently, a flow with a smaller RTT increases its rate faster than one with a larger RTT [Flo91]. In our version of WebTP, we intend to solve this problem by making the window increment proportional to the RTT.

Thus, instead of using $cwnd += \frac{1}{cwnd}$, we may use $cwnd += \frac{1}{cwnd} \times \alpha \times rtt \times rtt$ [HSMK98] Here, the factor rtt biases the increase to neutralize the RTT effect and the factor α is the slope of the increase and will be decided as a design parameter.

6.4 Rate Control

WebTP incorporates a hybrid rate and window based scheme to control its flow. As described in the previous subsection, the windowing mechanism is used to control the maximum number of outstanding packets. We also implement an additional control method based on the rate.

The rate control system used for WebTP is implicitly based on the window control scheme and the desired rate is calculated as

$$rate = \frac{cwnd}{srtt}$$

This formula is based on our TCP-like assumption that we should receive one window worth of data every round-trip-time. As described in Section 6.2, this rate information is used by the sender to pace out its packets.

6.5 Retransmission

Because the responsibility for reliable retransmission of packets in WebTP lies completely with the receiver, it needs to have knowledge about the objects it requests. The objects are requested according to some agreed naming convention that allows the receiver to specify the relevant details. The objects being requested are identified by individual ALF names designating ADUs but may possibly be fragmented into packets, which are sequenced by sequence numbers. The size and fragmentation information about the objects, contained in the metadata, allows the receiver to control its requests and pace them at the desired rate.

The receiver maintains a queue called the *Expected Objects Queue (EOQ)* which is the list of all the data objects it has asked for but not received yet. Whenever the receiver successfully retrieves an object from the sender, it removes the corresponding entry from the EOQ. The size of this EOQ controls the window mechanism (as specified in Section 6.2) and prevents overflowing the network. If the receiver gets an object that it does not expect (e.g., a duplicate packet) it simply discards it.

Furthermore, the receiver uses the EOQ to detect dropped packets in order to ask for retransmissions. To perform this detection, the receiver uses the EOQ to emulate the sender queue at all times.

The receiver detects missing packets in a number of ways:

- from a timeout, or
- by noting gaps in the sequence space of fragmented objects, or
- by noting a missing object

6.5.1 Timeout

Since the receiver knows the current rate (call it ρ packets/sec) at which the sender sends packets, it expects a packet every $\frac{1}{\rho}$ seconds. A timeout occurs when there is no packet arrival within

$$\frac{1}{\rho} + M\sigma$$

where σ is the deviation in the interarrival time and is updated using a filter

$$\sigma_{new} = \beta\sigma_{old} + (1-\beta)|iat - \frac{1}{\rho}|$$

where $0 \leq \beta \leq 1$, *iat* is the interarrival time between arriving packets and M is a multiplicative factor that represents the receiver's tolerance to jitter in packet arrivals. This factor M is currently set as a design parameter but we hope to make it adaptive.

When a timeout occurs, the receiver assumes that the network has dropped a packet. In that case, the receiver cuts the rate and window size and asks for a retransmission for the first object in the EOQ. The retransmission request as well as the new rate information is conveyed to the sender in the form of a '*Request*' packet. This form of a timeout scheme can however function only when the receiver has some rate information. So during the initial slow-start phase (when the receiver is still calculating the rate), we rely on the coarse RTT-based timeout mechanism used for TCP.

The above scheme is implemented using a single timer whose purpose is to keep track of the next expected packet. If the packet arrives ahead of schedule, the timer is nullified and a new timer started at $\frac{1}{\rho} + M\sigma$. If a timeout does occur then the timer is rescheduled to $\frac{1}{\rho_{new}} + M\sigma$ with $\rho_{new} < \rho_{old}$ since the drop packet would have induced a reduction in rate.

This change of the timer from $\frac{1}{\rho_{old}}$ to $\frac{1}{\rho_{new}}$ is the back-off scheme of WebTP. While the new rate ρ_{new} does not become effective at the sender until it gets the request packet (which takes approximately a one way trip time), the receiver suspects that congestion caused the packet loss and so increases its expected time for the next packet anyway.

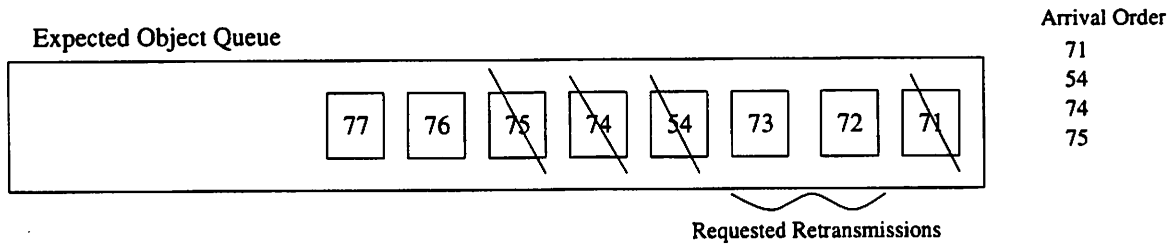

Figure 6.3: Detecting Missing Pkts due to Out-of-Order Arrivals

6.5.2 Out-of-Order Packets

While timeouts are used to signal congestion in the network and missing packets, we also need a mechanism to detect a lost request. A lost request may be detected by a gap in the EOQ, but this mechanism is affected by out-of-order packets, which confuses the detection of a missing packet. Furthermore, suppose a single request is lost while all the other requests arrive correctly. Then the sender sends all the packets in its queue. These packets arrive in time (without triggering timeouts) but the missing packets are never recovered.

So when the receiver gets an object, it notes its position in the EOQ before removing it. Because the arriving packet is typically found at the head of the queue, or near it, this search and remove operation is very fast. If the arriving packet is not the front packet in the queue, the receiver increments an out-of-order counter. In a system analogous to the TCP scheme of "three-dup-acks" — if the receiver gets three such consecutive out-of-order packets, then it labels the unreceived object(s) at the head of EOQ as 'missing' and requests them again.

Note that getting out-of-order packets without timeouts indicates that the link is still functioning as expected and that the current rate is suitable. Accordingly, when the receiver asks for retransmissions because of out-of-order packets, it does not cut back the rate and window.

Packet re-ordering is a connection-specific artifact and varies widely across connections. In an extreme case when packets are *never* re-ordered, any gap in the sequence space signals a missing packet. On the other hand, when there are extended re-orderings, the protocol should be tolerant of gaps in the sequence space to account for re-ordering. We are considering an adaptive scheme based on past history of re-ordering, which allows the protocol to learn from past records and suitably alter its tolerance for re-ordered packets. This scheme will be studied in more detail and implemented in the future.

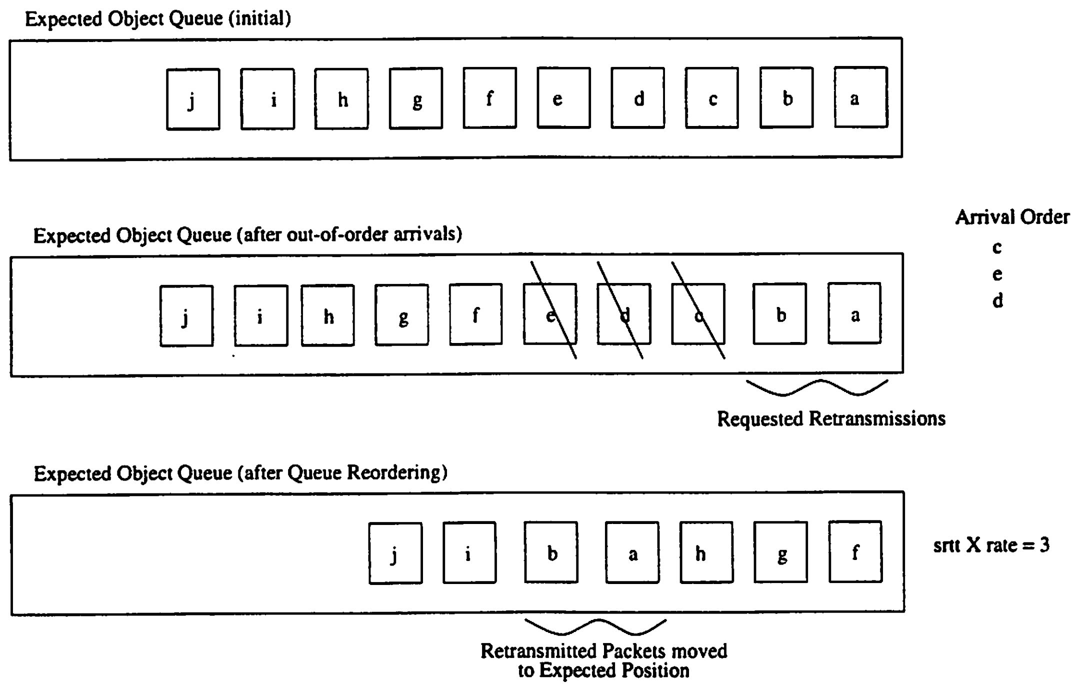
Expected Object Queue (initial)
j
i
h
g
f
e
d
c
b
a
Expected Object Queue (after out-of-order arrivals)
Arrival Order
c
e
d
Requested Retransmissions
Expected Object Queue (after Queue Reordering)
srtt X rate = 3
Retransmitted Packets moved to Expected Position

Figure 6.4: Queue Management for Retransmission Requests

6.5.3 Queue Management

EOQ at the receiver attempts to mirror the Sender Object Queue to detect out-of-order packets by locating them within the EOQ. When the receiver asks for a retransmission, it assumes that the packet was lost and requests it again. However, the time when the sender sends the packet depends on the priority policy in use. If retransmitted packets have the same priority as the rest of the packets, then they would be placed at the end of the queue at the sender. Correspondingly, the retransmission packets are taken off the head of EOQ and placed at the tail.

In our scheme, the receiver assigns a higher priority to retransmitted packets and the sender places them at the head of the queue when it receives the retransmission request. Since this placement does not occur until the sender gets the request, it may already have sent a number of packets. Accordingly, the receiver needs to estimate the order in which these retransmitted packets will arrive. This order is calculated as

$$\lceil srtt \times rate \rceil.$$

Fig. 6.4 explains this scheme in more detail. Initially there are 10 packets in the EOQ, labeled a through j. The first three packets to arrive are c, e and d. The receiver therefore labels packets a and b as 'missing' and asks for their retransmission. The receiver knows that these retransmitted packets cannot be sent until the request reaches the sender. The estimate of $\lceil srtt \times rate \rceil$ is equal to

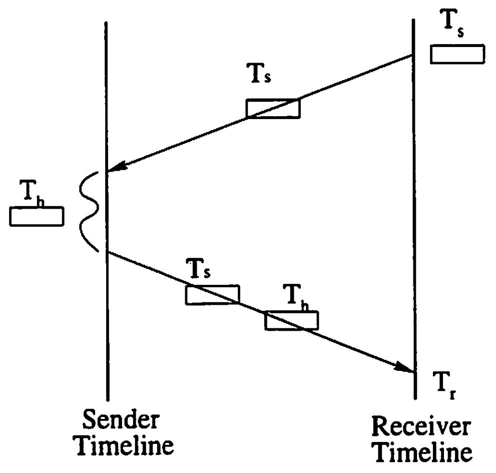

Figure 6.5: Calculating RTT at the Receiver

3. Consequently, the receiver expects the retransmitted packets to arrive after three other packets. So it removes packets a and b from the head of the queue and re-inserts them behind packet h.

Although the accurate position in which these packets arrive is $\lceil rate \times \textit{one-way-time} \rceil$, the receiver does not have any means of estimating the one-way-time. We decided to be conservative and use *srtt* to estimate the expected arrivals. (Another sensible estimate would be *srtt*/2, based on the assumption that the path is symmetric). If the packets arrive ahead of time, not too much harm is done since the receiver then requests at most one extra packet for every three out-of-order packets. Simulation results validate this small loss of efficiency by demonstrating that there are very few (of the order of a few per 1000) duplicate packets caused by false retransmission requests.

6.6 Calculating RTT at the Receiver

Because we are discussing a new protocol, we can take the liberty of introducing new header fields to facilitate protocol machinery like round-trip-time estimation. To this end, we include explicit timestamps in each packet as proposed in [KP91].

As shown in Fig. 6.5, the current time at the receiver (T_s) is stamped on to the request when the request is sent. For every packet the sender sends, it calculates Held Time (T_h) — the time elapsed from the arrival of the last request to the packet sending time. The sender copies both the Receiver Timestamp T_s and the Held Time T_h onto the packet. When the receiver gets the packet at receiver time T_r, it calculates the round-trip-time as

$$rtt = (T_r - T_s) - T_h.$$

A smoothed estimate of the round-trip-time (called *srtt*) is calculated using a first-order linear filter as in TCP [Pos81] wherein

$$srtt_{new} = \alpha \times srtt_{old} + (1 - \alpha) \times rtt$$

where $0 \leq \alpha \leq 1$.

Calculating RTT at the receiver is not entirely new and has been used in other protocols like NTP's time synchronization algorithm [Mil92] and SRM [F+97]. One advantage of this scheme is that, unlike Karn's algorithm [KP91], we do not need to discard the values calculated due to retransmitted packets. The inclusion of the Held Time implies that every RTT calculated denotes the current state of the network, and is a reliable sample.

6.7 Soft State at Sender

A very desirable fallout of such a receiver-based scheme is the replacement of the necessary 'hard' state at the sender by a 'soft' state. Even beyond the fact that the sender needs to maintain a minimal amount of state, that state too is 'soft' and the protocol performance is not affected critically by a failure to protect this state information. If the sender goes down while a connection is in progress, it loses its current packet queue and stops sending packets. The receiver becomes aware of the sender's failure as its packet flow stops — causing it to cut its rate repeatedly and ask for retransmissions. Eventually the sender recovers and receives the latest request, and starts sending packets. The receiver can then figure out which packets were missed and uses its retransmission scheme (as described in Section 6.5) to request the packets again. We are thus able to do away with a critical constraint of a connection — that of requiring *both* parties to remain functional throughout the entire duration. In fact, the notion of 'connection' itself is somewhat moot since we have a simple request-reply model without either side having to acknowledge the beginning or end of data-flow (connection).

Furthermore, by shifting the maintenance of the states at the receiver, we enable the receiver to perform session-oriented congestion control by re-using these states across connections. If the receiver has to make repeated and multiple connections to the same address, it can remember the state variables of the last connection and begin the new flow using the stored parameters, thereby doing away with the initial slow-start phase. This can give us the advantages of schemes like P-HTTP [PM95] without explicitly leaving the connection open at all times.

Chapter 7

Simulation Results

7.1 Experimental Setup

To test our receiver-driven implementations of WebTP, we used a controlled testbed where we could alter the network parameters and also experiment with variations in the control algorithms. Our experimental setup, modeled using the *ns* [NS] network simulator, is shown in Fig. 7.1.

For the experiments, a network cloud is modeled as a single bottleneck link whose characteristics are under our control. By varying the characteristics of the link, we emulate different forms of networks. We are then able to vary any or all of the following parameters of the bottleneck link:

- bandwidth;
- latency;
- loss rate;
- queue size;
- queue management policy.

In our simulation implementation of WebTP, the sender and the receiver incorporate all the algorithms outlined in the previous section. The sender is nearly stateless and follows directives from the receiver. All decisions about rate control, window control and retransmissions are taken at the receiver.

We were interested not only in evaluating the performance of WebTP flows, but also in their interaction with competing TCP flows. Accordingly, our simulation environment incorporated a

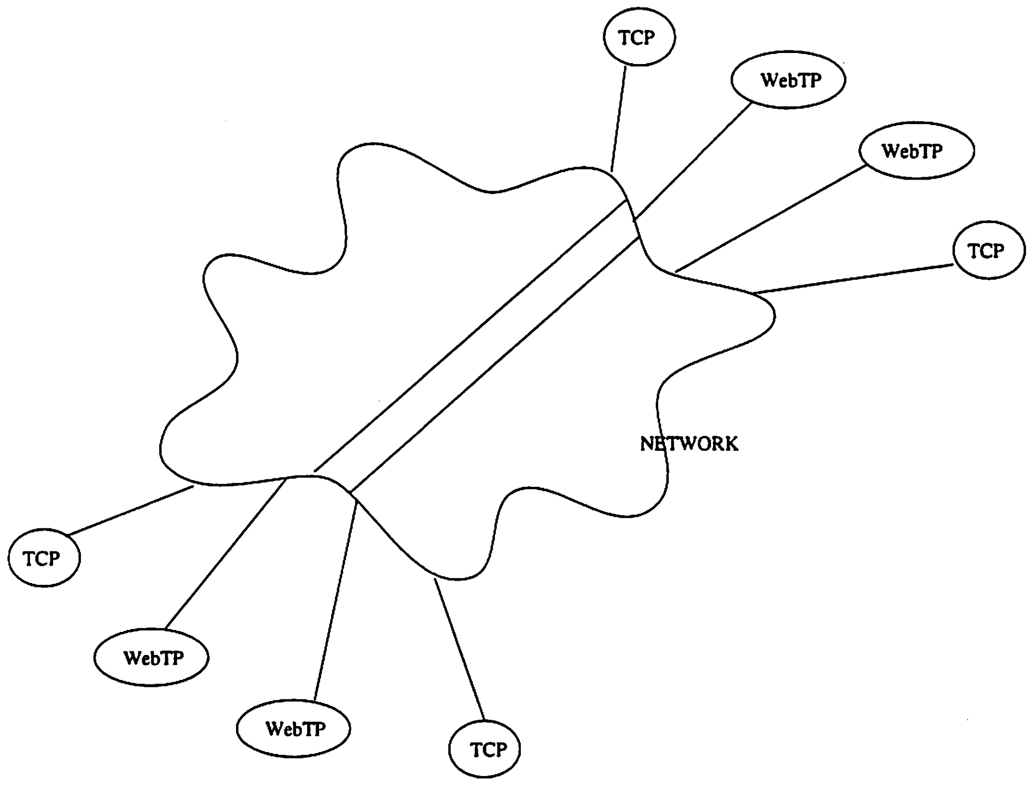

Figure 7.1: Experimental Setup

number of WebTP and TCP flows to study their interaction. These flows were often introduced into the network at different times, to study the transient effects.

In the subsequent graphs, we plot the rate at which packets are received vs. time. Rate is calculated by post-processing trace files that record all arrivals and departures of packets. We use a sampling interval of 0.05 seconds, and consider a window size of 0.5 second while calculating the rate. Specifically, every 0.05 seconds, we count the number of bytes received during the last 0.5 second. By plotting the average rates over time, we observe the individual flow characteristics and also their interaction.

The notable points of comparison are the following:

- *Efficiency:* How the total bandwidth is utilized by all the connections;
- *Stability:* How much each flow deviates from its allotted share;
- *Fairness:* Whether the bandwidth is allocated fairly among the competing links;
- *TCP-Friendliness:* Since WebTP flows are expected to interact with TCP flows over the Internet, it is important to ensure that these flows do not adversely affect existing TCP flows.

To evaluate the fairness, we use the criterion outlined by Chiu and Jain [CJ89]. The fairness criterion is given by the following formula:
At any time, for n flows $X_1, X_2, \dots X_n$ competing across a bottleneck link,

$$Fairness := \frac{(\sum_{i=1}^{n} X_i)^2}{n \sum_{i=1}^{n} X_i^2}.$$

A widely used measure of 'TCP-Friendliness' is the test outlined by Floyd and Fall [FF98]. Given a non-bursty packet drop rate of p, minimum round trip time R and maximum packet size B, a bound on the sending rate (in bits per second) for a TCP connection is given by

$$T := \frac{1.5\sqrt{2/3} \times B}{R \times \sqrt{p}}$$

A flow is deemed TCP-Friendly [MF97] if its overall sending rate is bounded above by T.

The initial challenge in developing a new protocol is to ensure that all the different pieces of the protocol work and interact together, without destabilizing the system. A later challenge arises in the form of the various design parameters, which are critical to the efficient functioning of the protocol. TCP itself has a number of such parameters which were refined through years of experience. Needless to say, WebTP too is encumbered with its own set of design parameters, and being at an early stage of development, many of these have been chosen intuitively. We will fine-tune the parameters and provide a comprehensive analysis of their effects on WebTP performance. Currently, our aim is to present a protocol that functions in an efficient and fair manner and interacts well with other flows. While this is not the final version of this protocol, it gives an indication of the performance characteristics of WebTP.

With the large number of variables in our experimental system (connections, parameters, link characteristics), it is not possible to present all the possible combinations of results. We therefore summarize the chief results obtained, presenting them in logical groups. Graphs denoting the progress of the connections, together with the fairness of the system are presented alongside to illustrate the conclusions.

7.2 WebTP Achieves Efficient Utilization of the Link

The first test of a protocol is to evaluate its performance over a link when a single connection is in progress. As hoped, the WebTP flow performs well and achieves efficient utilization of the link.

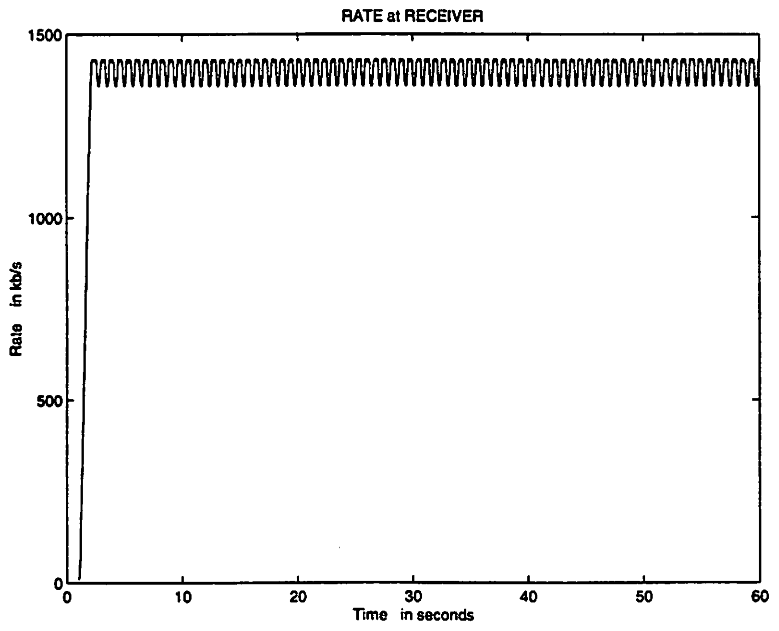

Figure 7.2: A Single WebTP flow across a 1.5Mbps link

The plot shown in Fig. 7.2 shows the performance of a single WebTP flow starting at 1 sec. The link was modeled as a 1.5 Mbps link with a drop-tail queuing policy. The flow achieves 90% utilization of the link, staying between 1.3 and 1.5 Mbps. Furthermore, the rate does *not* cross the 1.5 Mbps limit but is seen to be cut at a value close to it. This effect is firstly due to the fact that we calculate rate based on a window scheme, and so instantaneous rates exceeding the limit are smoothed out. Also, when there is a large queue buildup, causing some packets to be delayed considerably, WebTP times out and cuts the rate. The point at which this cutback occurs (roughly at 1.4 Mbps) depends on the timeout algorithm, and on the chosen design parameters.

While the single WebTP flow is observed to achieve efficient utilization of the link, we would like to ensure that this flow is TCP-friendly. Using the Floyd and Fall definition [FF98], we calculate the maximum sending rate for a TCP flow under these circumstances. Here,

- packet size $p = 1kBytes = 8kb$;
- minimum rtt $R = 25.5ms = 0.0255s$;
- packet loss probability $p = \frac{78\ dropped\ packets}{5076\ total\ packets} = 1.537 \times 10^{-2}$;

Consequently, the bound T on the TCP flow rate is computed as follows:

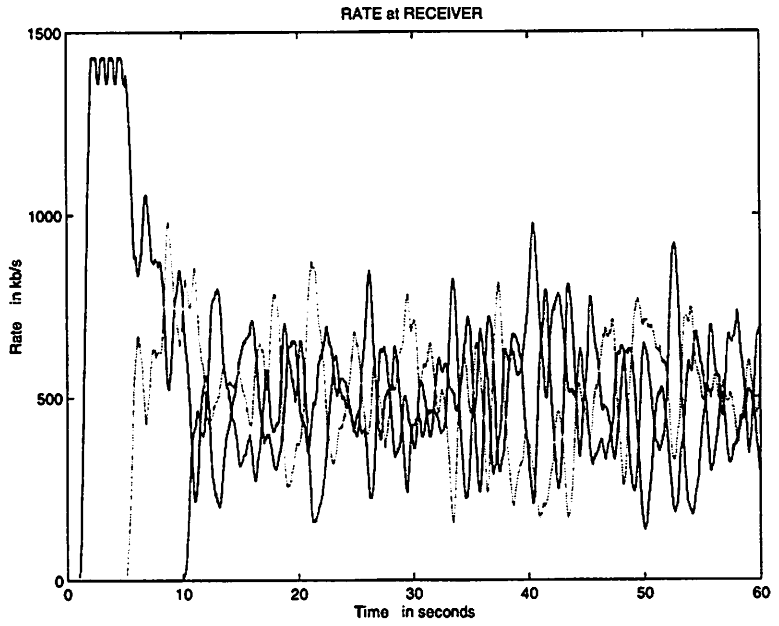

Figure 7.3: Three WebTP flows across a 1.5 Mbps link with Drop-Tail queue

$$\begin{aligned} T &= \frac{1.5\sqrt{2/3} \times B}{R \times \sqrt{p}} = \frac{1.5\sqrt{2/3} \times 8}{0.0255 \times \sqrt{1.537 \times 10^{-2}}} \\ &= \frac{1.225 \times 8}{0.0255 \times 0.12396} = 3100.30\ kbps \\ &= 3.1\ Mbps \end{aligned}$$

The bandwidth limit is thus 3.1 Mbps, and the WebTP flow is seen to be limited well below that figure. The weak bound given by the estimate may be explained by the fact that Floyd and Fall assume that only a dropped packet reduces the rate. This is not the case for WebTP, which can react before a drop. The TCP-friendly bandwidth we get is thus considerably higher and does not give us an effective measure for comparison.

7.3 WebTP Flows Interact Well

As we had hoped, a number of WebTP flows (with no competing TCP flows) competing for the bottleneck link interact well over different link characteristics and different queue management policies. The flows achieve good utilization of the link, are quite stable and fair.

The plots shown in Fig. 7.3 show the interaction between 3 WebTP flows starting at 1, 5 and 10 seconds respectively and competing across the 1.5 Mbps link. As each new flow is introduced,

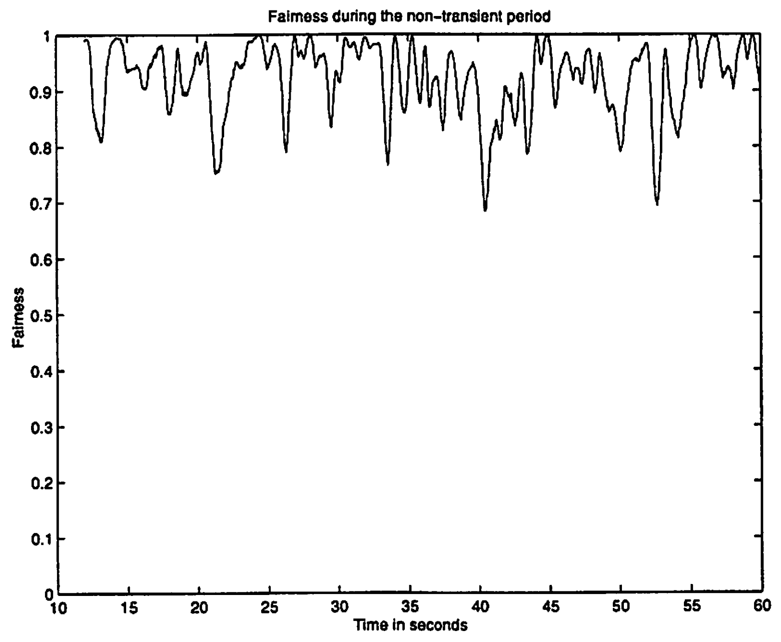

Figure 7.4: Fairness for Three competing WebTP flows

the existing ones are seen to cut their rate and achieve roughly its fair share of the bandwidth. Eventually, all three flows stabilize around 500 kbps, exploiting the full bandwidth of the link. The exponential growth during the slow-start phase is clearly visible for the first two flows. However, the queueing policy used here was the Drop-Tail scheme. The third flow comes in when the link is already under heavy utilization (at 10 second) and experiences an early packet loss. Therefore it goes out of slow start earlier than its peers and moves into the congestion avoidance phase. However, it is also able to compete fairly with the others, showing that WebTP has good resilience to initial packet losses.

Fig. 7.4 plots the min-max fairness of the scheme. The plot begins after 10 second (when all the flows are in progress) and charts the fairness achieved by the competing flows. As seen from the plot, we achieve a very fair performance (with fairness value above 0.8 at most times) — validating WebTP's claim of being fair. Arguing intuitively, WebTP incorporates an additive increase/multiplicative decrease scheme and as shown by Chiu and Jain [CJ89], this leads to a fair sharing of the allocated bandwidth across the bottleneck link.

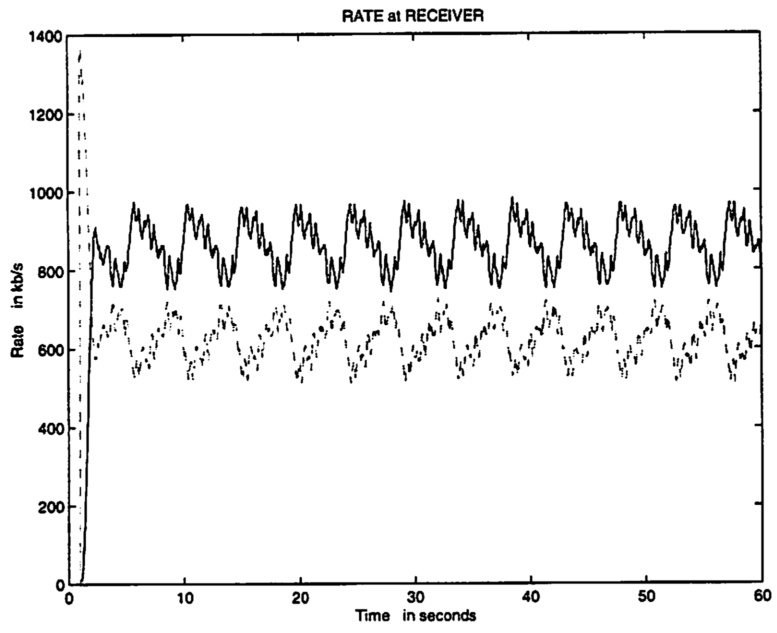

Figure 7.5: One WebTP (solid line) and One TCP (dashed line) flow across 1.5 Mbps link

7.4 WebTP and TCP Interact Well

Fig. 7.5 shows the interaction between a WebTP and a TCP flow competing for the bottleneck link. As seen from the plots, both flows stabilize quite early and stay near their mean values. The variation in the rates is surprisingly small. The symmetry in the two rate plots comes from the simplified simulation model since there are only two flows, and is seen to vanish for more complicated setups. The fairness plot for the simulation (Fig. 7.6) shows the fairnes to remain above 0.9 for the entire period — suggesting a satisfactory interaction between the two protocols. However, the WebTP flow is seen to occupy a slightly greater share of the bandwidth and gets about 800 kbps on an average while the TCP flow can get only about 700 kbps.

We are able to gain a couple of insights by analyzing this interaction. Firstly, the control mechanism of WebTP ensures that it does not blow away other competing protocols and is hence quite suitable for incremental deployment over the Internet. Secondly, albeit getting a slightly higher share of the pipe, WebTP nonetheless maintains a steady stream and further fine-tuning of the parameters should provide improved performance that compete even more fairly with TCP.

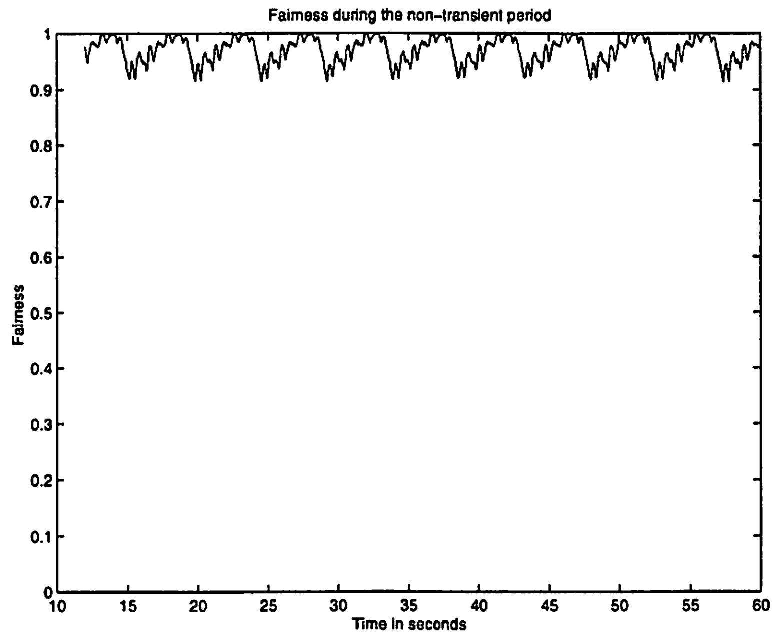

Figure 7.6: Fairness for One WebTP and One TCP flow across 1.5 Mbps link

7.5 Improved Queue Management

The aim of this simulation exercise was to evaluate the effect on WebTP flows of the queue management policies being used. We performed this exercise by comparing the interaction of one WebTP and one TCP flow across a 1.5 Mbps bottleneck link. The queue management policy was changed from DropTail (with a maximum queue size of 10) to a Random Early Detection (RED) policy (with the minimum and maximum parameters set to 3 and 10 respectively). The RED queuing policy drops packets randomly even before the queue overflows, by marking packets with a probability that is linear in the length of the queue when it is between the minimum and maximum values [FJ93]. As a result of this policy, the flows can adjust their rates and avoid overflows. RED queues are also known to limit loss bursts that penalize some connections excessively.

Fig. 7.7 shows the plot for the experiment described above. Comparing with the DropTail scheme (Fig. 7.5) we notice a more equitable distribution of bandwidth in the RED queue, with both flows getting a similar share. There are a few large deviations from the mean (e.g., around 9.5 sec and around 15 sec) — which are caused when RED randomly picks a few consecutive packets from the same flow. However, in all these cases, both WebTP and TCP recover to return to their mean values — indicating a stability in the protocol in the face of repeated packet drops.

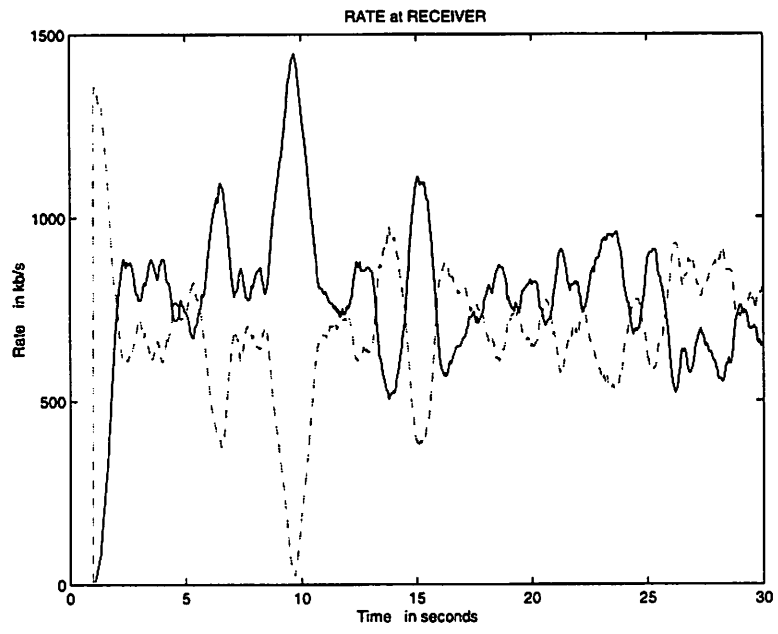

Figure 7.7: One WebTP (solid lines) and One TCP (dashed line) flow across a RED Queue

7.6 WebTP Performance Across a Slow Link

In this set of experiments, the link had a capacity of only 56 kbps (typical modem), with a latency of 100 ms. We ran similar experiments as before across this link to evaluate the performance of WebTP through a slow link, facing successive drop packets and delayed feedback (due to the high latency of the link).

Fig. 7.8 shows the action of a single WebTP flow across this link. The optimal bandwidth of 56 kbps is achieved for most of the duration of the flow, thus ensuring the efficiency of the protocol. Occasional packet drops (at 9, 22, 37 and 50 seconds) cause the rate to be cut, but the rate is able to soon return to the desired level. The slow feedback mechanism does not have any drastic undesirable effects on the functioning of the scheme.

Next we experimented with the interaction of one WebTP and one TCP flow across the slow link. As seen in Fig. 7.9, the WebTP and TCP flow interact with each other to utilize the full bandwidth of the link at all times. The fairness plot presented in Fig. 7.10 shows the interaction to be acceptably fair, remaining above 0.7 for most of the period.

The scheme though is less stable than over the faster link used in the previous subsections, and this may be attributed to the slow feedback mechanism. An illustrative instance is the period near the 10 second mark, when the TCP flow is trying to increase its rate. Although the link limit is

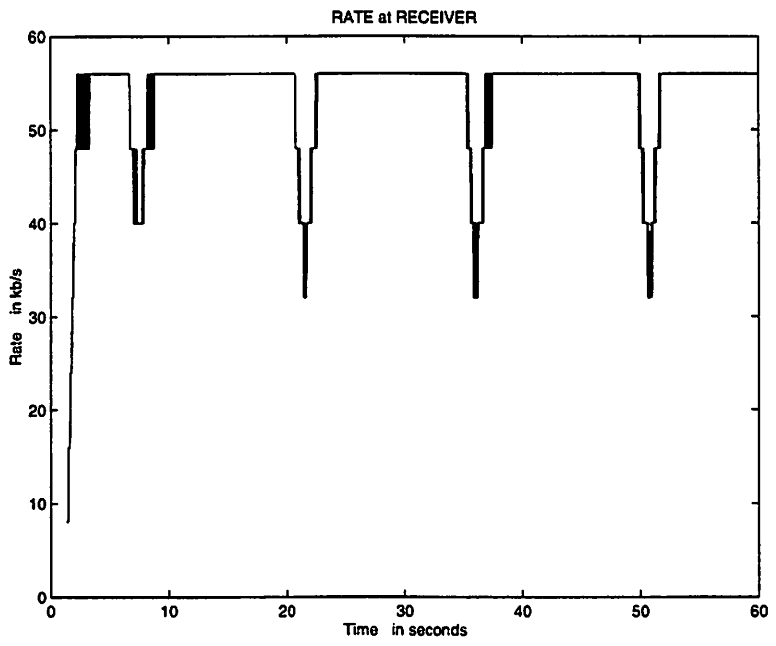

Figure 7.8: One WebTP flow across a 56 kbps link

reached, the TCP flow is slow to realize it — causing repeated WebTP packet drops and WebTP is forced to cut its rate to almost zero. The reverse happens soon afterwards when the WebTP flow is slow to cut its rate. This artifact is however present in any scheme across a slow link, and it is heartening to see that WebTP reacts reasonably fast and works well with TCP. We can thus hope that WebTP will be able to work across different network characteristics (including the prevalent modem connections).

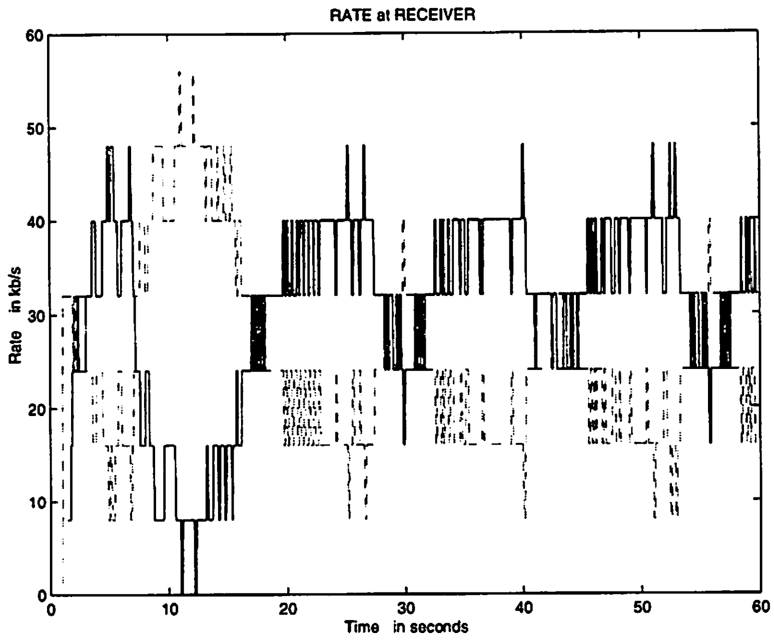

Figure 7.9: One WebTP flow (solid line) and One TCP flow (dashed line) across a 56 kbps link

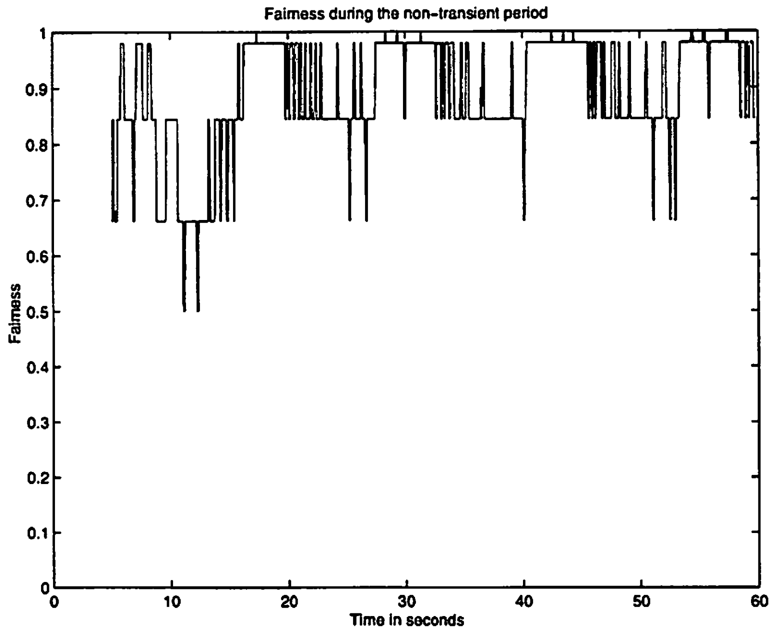

Figure 7.10: Fairness for One WebTP and One TCP flow across a modem link

Chapter 8

Summary

8.1 Conclusion

In this report we have outlined the WebTP research effort which attempts to tackle the complete suite of problems relating to Web transport today by presenting a new protocol suited for the web. This protocol is designed to optimize the transport by using Application Level Framing. We presented the first steps in the process — an attempt to create a receiver-oriented request-response protocol for the web that is optimized to include the user into the transfer loop.

We presented our goals by describing an ideal interactive web transport that is able to utilize the information contained in the overall system — the contents of the page, the state of the network, the current situation at the client and the preferrences of the user. This report presented the framework that can make this work, by defining the requisite structures for the various components. We discussed a sequential scheme to incorporate all the above factors and settle upon an optimal order to transfer the contents of the page. While these structures are not final, and are open to modifications, they provide an intuitive solution which can serve as a guide to more elaborate future schemes. we developed theoretical results for the optimizations, which will allow us to choose the direction to proceed, and we also presented a suite of algorithms to determine the optimal order of transfer for a large class of web interactions.

However, in order to make user-centric optimization happen, it is critical to transfer the control of the flow to the receiver end, thereby breaking away from the current model of a sender-oriented world. Therefore, we tried to create a protocol whose control is largely centred at the receiver end. The resulting protocol — WebTP — is completely receiver-based in terms of connection setup, flow-control and congestion-control. It also incorporates a novel retransmission scheme based at the receiver. The protocol is able to achieve stable, efficient and fair flows which interact well amongst themselves. Furthermore, these flows are also seen to interact in a fair manner with competing TCP-flows, supporting the claims of being TCP-compatible.

8.2 Future Work

The WebTP project being at a fairly initial stage — there is indeed a lot of progress left to be done. This report presents a first step in the solution for WebTP by outlining the requirements and the goals of the project. We also describe a simple set of techniques to handle the issue of user optimization for a large class of web transfers. The work remaining involves studying these techniques at a much greater detail and choosing the appropriate ones — and also to present proofs of the optimality of the chosen schemes.

- We need to devise a suitable ALF-based naming scheme and data representation methodology that enables application control.
- Standardized packet formats are necessary to allow effective communication between the sender and the receiver.
- The simplistic Utility Functions (Section 3.4) are presently chosen intuitively. It is quite possible that the true shape of utility functions on the internet are quite different and far more complex. We not only need to find these functions, but also tailor our schemes to suit them.
- The overall expression denoted by $S = f(D, C, N, U)$ is quite complex, and it will be challenging to prove the optimality of our suggested solutions.
- One outstanding issue is dealing with situations where the server generates the data on the fly (e.g. using a database query) and therefore has no metadata available at the beginning of the connection — so measures need to be taken to convey the relevant information to the receiver.

We also have work to do with the functioning of WebTP as a receiver-driven protocol. WebTP, like any other protocol, includes a number of design parameters which have been chosen intuitively and need to be studied in further detail to analyze their effects, and their optimal values. Many of these can be optimized only after extensive experimentation and real-life experience with the system — hence it has to remain open and adaptive to changes.

- Under the current setup, WebTP is "receiver-reliable" as opposed to "client-reliable", which means that the onus of the control lies with the receiver, even if it is the server. In order to take advantage of the scalability arguments, we would like to convert the model to a client-oriented sytstem — irrespective of who the sender or receiver may be.
- For situations when the client is weak (e.g., a Palm Pilot on a wireless link), it is obviously unwise to burden it with flow control computations. In such cases, we will need to fall back to a TCP-derivative sender-oriented model; or enable a proxy (e.g., the wireless base station) to handle the flow.
- For web transfers where a small amount of the transaction works against the flow (e.g., sending your credit card number while browsing through Amazon.com), the current system is simply to create a *new* flow with the roles of the receiver and sender reversed. While this solution too is acceptable for WebTP, we are considering schemes whereby such transactions could be carried out within the confines of the existing flow itself.
- We are experimenting with an adaptive scheme to deal with packet re-ordering that can learn from the history of the connection and consequently adapt its tolerance to the amount of reordering observed.
- We need to ensure graceful performance under lossy circumstances, especially in the cases when the flow path is asymmetric.
- We have proposed a session-oriented congestion-control mechanism where the receiver re-uses knowledge across connections, and suitable semantics have to be developed to implement this.
- We also need to address all the issues of compatibility to allow parallel deployment of WebTP with other existing protocols.

Looking at the larger picture, the whole issue of incorporating a user-centric optimization scheme into flow control is to be solved since the current protocol only provides a carrier to implement them. The final vital component to be tackled in the future is the actual deployment of WebTP servers and clients and testing the performance of the flows over the Internet.

8.3 Acknowledgements

I would first like to thank my adviser Prof. Jean Walrand for suggesting this exciting project to me and getting me involved. Thanks for his continuous support and superb guidance throughout my M.S. I was privileged to work closely with Prof. Steven McCanne — thanks to him for all the comments and corrections during the experimentations. Thanks also to Mike Chen for his collaboration in the study of WebTP as a receiver-driven protocol.

I would also like to thank Prof. Venkat Anantharam, Prof. David Tse and Prof. Pravin Varaiya for their helpful comments and suggestions and their invaluable contribution towards the planning of WebTP. And thanks to my friend and colleague Kiran for all his help with matters technical and otherwise.

This research was supported by NSF Grant No. ANI-9872764. The views expressed here do not reflect the position or policy of the U.S. Government.

Bibliography

[AMTVW98] V. Anantharam, S. McCanne, D. Tse, P. Varaiya, and J. Walrand, *Group meetings on WebTP*, Spring 1998.

[BPK97] H. Balakrishnan, V. Padmanabhan, and R. H. Katz, "The effects of asymmetry on TCP performance," *Proc. Third ACM/IEEE Mobicom Conference*, Budapest, Hungary, Sep 1997.

[BSAK95] H. Balakrishnan, S. Seshan, E. Amir, and R. H. Katz, "Improving TCP/IP Performance over Wireless Networks," *In Proceeding of the 1st ACM Conf. on Mobile Computing and Networking*, Berkeley, CA, November 1995.

[BPSSK98] H. Balakrishnan, V. Padmanabhan, S. Seshan, M. Stemm, and R. H. Katz, "TCP behaviour of a busy web server: Analysis and improvements," *Proc. Infocom'98*, March 1998.

[Bra94] R. T. Braden, "Enhancing TCP for transactions," *Request for Comments RFC-1379*, Nov 1994.

[CJ89] D. Chiu and R. Jain, "Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks," *Computer Networks and ISDN Systems*, v.17, 1-14, 1989.

[CLR90] T.Cormen, C. Leiserson, and R. Rivest, "Introduction to Algorithms," *McGraw Hill and MIT Press*, 1990.

[CLZ87] D. Clark, M. Lambert, and L. Zhang, "NetBLT: A high throughput transport protocol," *Proc SIGCOMM '87 Conf. ACM*, pp 353-359, 1987.

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

[Flo91] S. Floyd, "Connections with multiple congested gateways in packet-switched networks, Part 1: One-way traffic," *ACM Computer Communications Review*, 21(5):30-47, October 1991.

[FF98] S. Floyd and K. Fall, "Promoting the use of End-to-End Congestion Control in the Internet," submitted to *IEE/ACM Transactions on Networking*, February 1998.

[FGBA96] A. Fox, S. D. Gribble, E. A. Brewer, and E. Amir, "Adapting to network and client variability via on-demand dynamic distillation," *Proc. Seventh Intl. Conf. on Arch. Support for Prog. lang. and Oper. Sys. (ASPLOS-VII)*, Oct. 1996, Cambridge, MA.

[FJ93] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," *IEEE/ACM Transactions on Networking*, Vol. 1, No. 4, August 1993, pp. 397-413.

[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.

[F+97] S.Floyd, V. Jacobson, C.Liu, S.McCanne, and L.Zhang, "A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing, Scalable Reliable Multicast (SRM)", *IEE/ACM Transactions on Networking*, 1997.

[GB98] J. Gilbert and R. W. Brodersen, "Globally Progressive Interactive Web Delivery," *submitted to INFOCOM'99*, available at *http://infopad.eecs.berkeley.edu/~gilbertj/JMG_INFOCOM99.ps.*

[Hei97] J. Heidemann, "Performance interactions between P-HTTP and TCP implementations," *ACM Computer Communication review*, v. 27, no. 2, April 1997, pp. 65-73.

[HSMK98] T. R. Henderson, E. Sahouria, S. McCanne and R. H. Katz, "On improving the fairness of TCP congestion avoidance," accepted for publication for *Globecom 1998 Internet Mini-Conference.*

[Jac88] V. Jacobson, "Congestion avoidance and control," *Proceedings of SIGCOMM'88*, Palo Alto, CA, Aug 1988.

[KP91] P. Karn and C. Partridge, "Improving Round-Trip Time Estimates in Reliable Transport Protocol," *ACM Transaction on Computer Systems (TOCS)*, vol. 9, no. 4, pp. 364-373, November 1991.

[Mil92] D. L. Mills, "Network Time Protocol (Version 3)," *Internet Request for Comments RFC 1305*, March 1992.

[MF97] J. Mahdavi and S. Floyd, "TCP-Friendly unicast rate-based flow control," *Technical note sent to the end2end-interest mailing list*, Jan 8, 1997.

[MLAW98] J. Mo, R. La, V. Anantharam and J. Walrand, "Analysis and comparison of Reno and Vegas," *available at http://www.path.berkeley.edu/~jhmo*

[MW98] J. Mo and J. Walrand, "Fair end-to-end window-based congestion control," *available as http://walrandpc.eecs.berkeley.edu/Mo1.pub.ps.*

[Nag84] J. Nagle, "Congestion control in IP/TCP internetworks," *Internet request for Comments RFC-896*, Jan 1984.

[NS] "Network Simulator — ns," information available at *http://www-mash.cs.berkeley.edu/ns*

[N+97] H. F. Nielsen, J. Gettys, A. Baird-Smith, E. Prud'hommeaux, H. W. Lie, C. Lilley, "Network performance effects of HTTP/1.1 CSS1, and PNG," *available as http://www/w3/org/TR/note-pipelining.*

[Pos81] J. Postel (Ed.), "Transmission Control Protocol," *Internet Request for Comments RFC 793.* SRI International, Menlo Park, Ca., September 1981.

[Pax97] V. Paxson, "Automated packet trace analysis of TCP implementations," *ACM SIGCOMM '97*, Sep 1997, Cannes, France.

[PK98] V. Padmanabhan and R. H. Katz, "Addressing the challenges of web data transport," *Submitted for publication*, Jan 1998.

[PK98a] V. Padmanabhan and R. H. Katz, "TCP fast start: a technique for speeding up web transfers," accepted for publication for *Globecom 1998 Internet Mini-Conference.*

[PM95] V. Padmanabhan and J. Mogul, "Improving HTTP Latency," *Computer Networks and ISDN Systems*, v.28, nos. 1 & 2, Dec 1995, pp. 25-35.

[SF96] H. Schulzrinne and G. M. D. Fokus, "RTP profile for audio and video conferences with minimal control," *Internet request for Comments RFC-1890*, Jan 1996.

[SSK97] S. Seshan, M. Stemm, and R. H. Katz, "SPAND: Shared passive network performance discovery," *Proc. USENIX Symposium on Internet technologies and Systems Proceedings*, Dec 1997.