A Receiver-Driven Transport Protocol for the Web

Rajarshi Gupta, Mike Chen, Steven McCanne and Jean Walrand
University of California, Berkeley
{guptar, mikechen, mccanne, wlr}@eecs.berkeley.edu

Abstract

In this paper, we present a receiver-oriented, request/response protocol for the Web that is compatible with the dynamics of TCP's congestion control algorithm.

The protocol, called **WebTP**, is designed to be completely receiver-based in terms of transport initiation, flow-control and congestion-control. We propose a dual window-cum-rate based congestion control mechanism that is compatible with parallel TCP flows, and in fact interacts better with a congested network state. In support of our receiver-driven design, we developed a novel retransmission scheme that is robust to delay variations. The resulting flows achieve efficient network utilization and are qualitatively fair in their interaction amongst themselves and even with competing TCP flows. The paper also provides detailed simulation results to support the protocol design.

Keywords: Internet, Transport Protocols, TCP, Simulations, ALF, Congestion Control, Retransmissions.

Presented at
Fifth INFORMS Telecommunications Conference
Session MA-2: IP Measurements
Monday, March 6, 2000. 8:00 - 9:30 am.

Corresponding Author
Rajarshi Gupta, 1822 Francisco St, Apt 21, Berkeley, CA 94703.
Phone: (510) 647-1445, Fax: (510) 647-1441

1 Introduction

The goal of the WebTP project [7] at UC Berkeley is to address a varied suite of problems associated with the current internet model of HTTP layered on TCP. We propose a multi-layered protocol residing on top of IP that allows the Application and the Transport layers to act collaboratively using the principles of Application Level Framing (ALF) [3]. We also aim to utilize the request/response model of the web and introduce the user, with his actions and preferrences, into the transport loop. As part of the transport layer activities for the project, we conceptualized a transport layer for WebTP that would be receiver-driven.

In this paper, we do not present designs for all of the 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. Herein, we address the first crucial and open question for a WebTP transport layer: Is it possible to devise a receiver-oriented, request/response protocol for the Web that is 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 (as defined by Jain's fairness index).

First, we summarize the reasoning behind the choice of a receiver-based approach in our design of WebTP. In the next section, we outline the design of our receiver-driven prototype. We then describe our experimental design and present simulation results, before concluding.

1.1 Why Receiver Driven ?

The motivation for a receiver-driven protocol is led by the granting of more control to the application that interacts with the user. 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. Moreover, by resting the fate of reliable delivery on the receiver rather than the source (as proposed in NETBLT [2]), the client can tailor its reliability requirements as necessary, e.g., by easily aborting a transfer that becomes irrelevant due to dynamic user behavior. Often, the bottleneck link in a connection is the last hop to the user (e.g. a modem conncetion or an wireless link) — in such cases, the receiver is far more aware of the constraints and characteristics of the bottleneck link than the sender, and can apply this knowledge to the transport control decisions. Furthermore, 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. Finally, by shifting the maintenance of the states to 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 [14] without explicitly leaving the connection open at all times.

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 semantics are not violated by a failure to protect this state information. 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).

2 Protocol Details

Rather than design a new protocol from scratch, WebTP follows many of the important lessons gained through decades worth of research on TCP. However, we follow the request-reply model prevalent in the Web to adopt a scheme whereby a flow is initiated, controlled and terminated by requests from the receiver and the data sent in reply. 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. Moreover, 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 section — we describe the various components of WebTP in detail and discuss their effects on the functioning of the protocol.

2.1 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

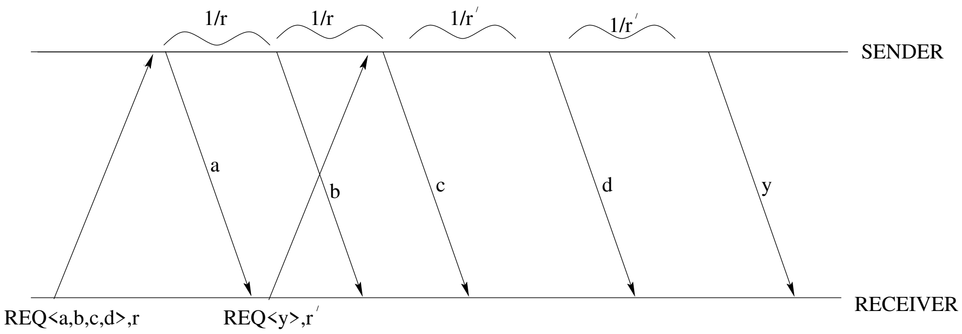

Figure 1: Rate Control at Sender

(Application Data Unit) from that address. 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 requests their delivery.

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

2.2 Sender

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 r. 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-sends the requested packet(s).

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 explicitly 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.1 the receiver requests that ADUs a through d be sent at rate r and later requests that ADU y be sent

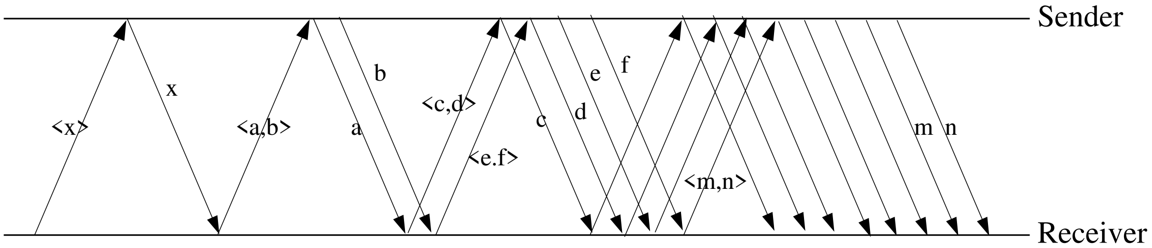

Figure 2: Slow-Start Controlled by the Receiver

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.

2.3 Congestion Control

The congestion control mechanism employed by WebTP is a hybrid of a window-based and a rate-based mechanism. To explain qualitatively — we use a window mechanism for coarse grained control, and fine tune the performance using a rate-based control.

2.3.1 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 [8]. 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.

Fig. 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 begins 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, without sending any rate information yet. 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. Upon the first loss, the receiver shifts to the Congestion Avoidance mode, whereupon it sends explicit rate information with its requests.

2.3.2 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. Over and above, we implement an additional control method based on the rate during the Congestion Avoidance phase. Instead of sending an entire window of data into the network at once, we pace out the packets at the desired rate (Section 2.2).

The rate control system is implicitly based on the window control scheme and the estimated round trip time (RTT) of the flow. The desired rate is calculated as

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

where $srtt$ is a smoothed estimate of the round-trip-time. $srtt$ is calculated using a first-order linear filter as in TCP [15] using the formula $srtt_{new} = \alpha \times srtt_{old} + (1 - \alpha) \times rtt$, where $0 \leq \alpha \leq 1$. 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 2.2, this is the rate information used by the sender to pace out its packets.

In order to estimate $srtt$, we need to first calculate the RTT at the receiver. This mechanism is not entirely new and has been used in other protocols like NTP's time synchronization algorithm [10] and SRM [6]. To this end, we include explicit timestamps in each packet as described in [9]. 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 writes 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

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

The system for calculating the RTT at the receiver is demonstrated in Fig.3.

2.4 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 them. The naming scheme for the objects, and metadata describing them is by itself a whole different research topic and is not addressed in this paper.

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

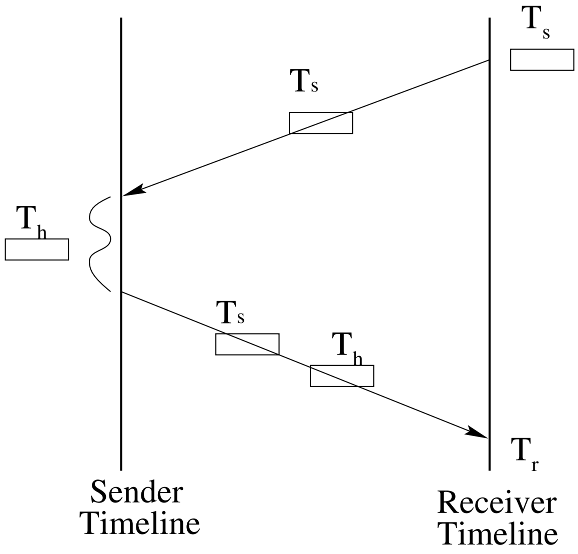

Figure 3: Calculating RTT at the receiver

the sender, it removes the corresponding entry from the EOQ. The size of this EOQ controls the window mechanism (as specified in Section 2.3.1) 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.

2.4.1 Timeout

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

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

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

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

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. 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.

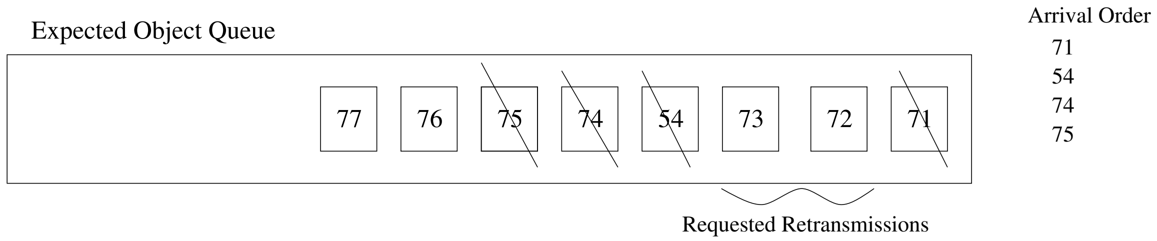

Figure 4: Detecting Missing pkts due to Out-of-Order Arrivals

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 for $\frac{1}{r} + M\sigma$. If a timeout does occur then the timer is rescheduled to $\frac{1}{r_{new}} + M\sigma$ with $r_{new} < r_{old}$ since the drop packet would have induced a reduction in rate. This change of the timer from $\frac{1}{r_{old}}$ to $\frac{1}{r_{new}}$ is the advance back-off scheme of WebTP. While the new rate r_{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.

2.4.2 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. Suppose a single request is lost while all the other requests arrive correctly. Then the sender naively sends only the packets it received requests for. If we do not have a detection mechanism for the missing request, these packets will all arrive in time (thereby not triggering any timeouts) but the missing packets will never be recovered. 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.

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 before removing the packet from the EOQ. 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 (Fig. 4).

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.

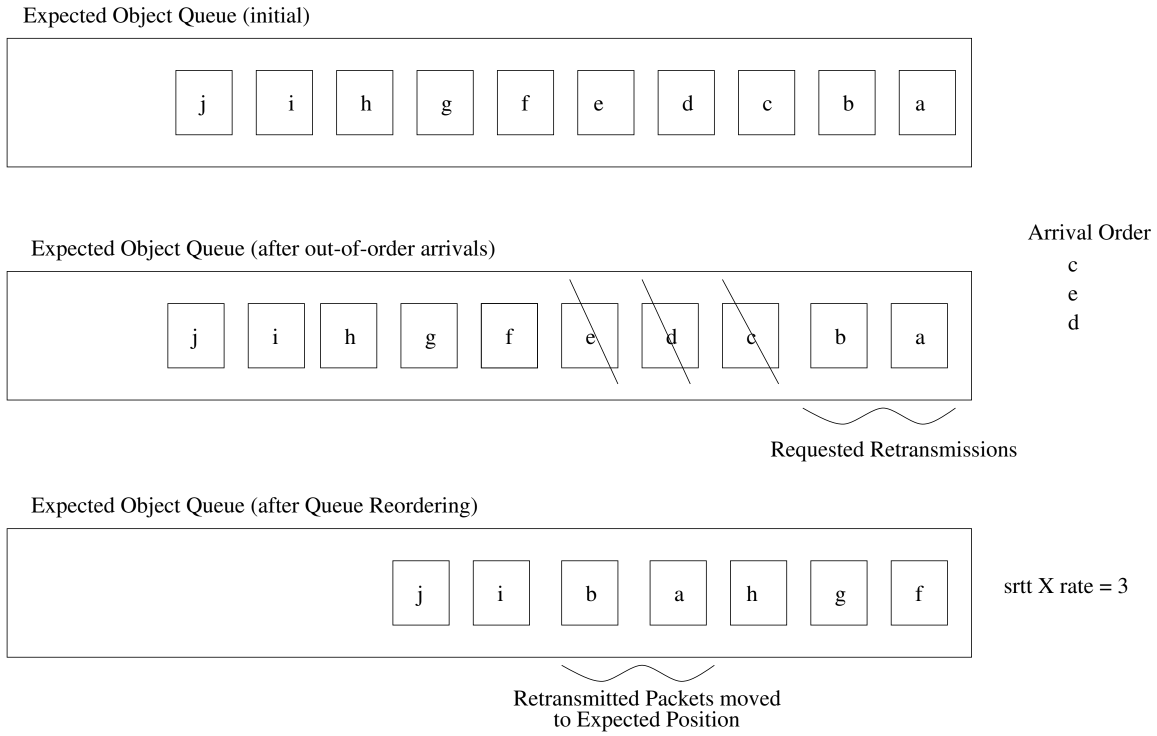

Figure 5: Queue Management for Retransmission Requests

2.4.3 Queue Management

The EOQ at the receiver attempts to mirror the Sender Object Queue to detect out-of-order packets by locating them within the EOQ. 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 time when these retransmitted packets will arrive. This time is calculated as $\lceil srtt \times rate \rceil$.

Fig. 5 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 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 $\lceil rate \times one\text{-}way\text{-}time \rceil$, 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

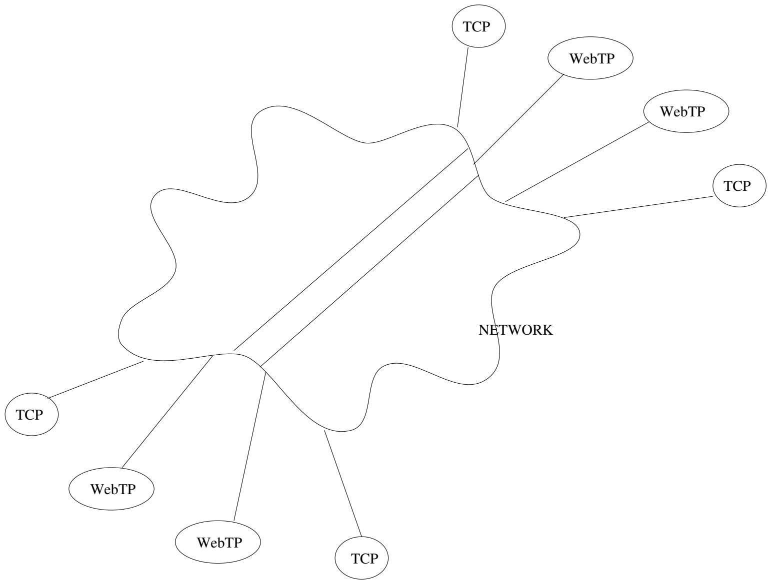

Figure 6: Experimental Setup

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.

3 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* [12] network simulator, is shown in Fig.6. 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 and 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 number of WebTP and TCP flows (*ns* provides a variety of built-in TCP “message agents”) to study their interaction. These flows were often introduced into the network at different times, to study the transient effects.

In the subsequent graphs, rate is calculated using a sampling interval of 0.05 seconds, and considering a window size of 0.5 second. Specifically, every 0.05 seconds, we count the number of bytes received during the last 0.5 second.

To evaluate fairness amongst the interacting flows, we use the criterion outlined by Chiu and Jain [1]. The fairness criterion is given by the following formula:
At any time, for n flows $X_1, X_2, \ldots X_n$ 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 [5]. 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 [11] if its overall sending rate is bounded above by T.

4 Simulation Results

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. 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 surely not the final version of this protocol, it gives an indication of the performance characteristics of WebTP.

4.1 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.

The plot shown in Fig.7 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 timeouts and cuts the rate. The point at which this cutback occurs (roughly at 1.4 Mbps) depends on the timeout algorithm, and 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 definition given in [5], we calculate the maximum sending rate for a TCP-friendly flow under these circumstances. Here,

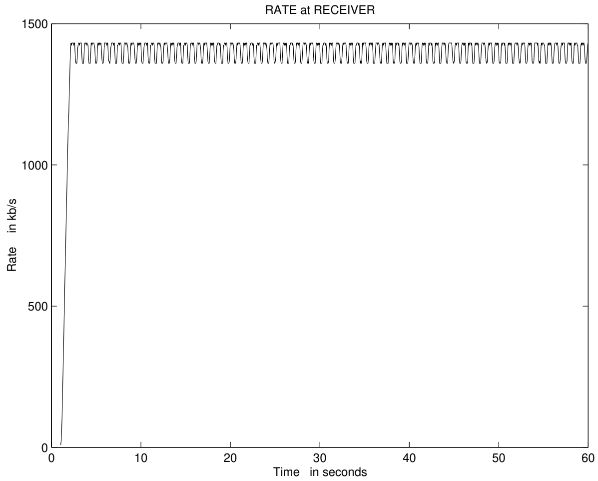

Figure 7: A Single WebTP flow across a 1.5Mbps link

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

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.

4.2 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. 8 show the interaction between three WebTP flows starting at 1, 5 and 10 seconds respectively and competing across the 1.5 Mbps link. As each new flow is introduced, 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 queuing 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

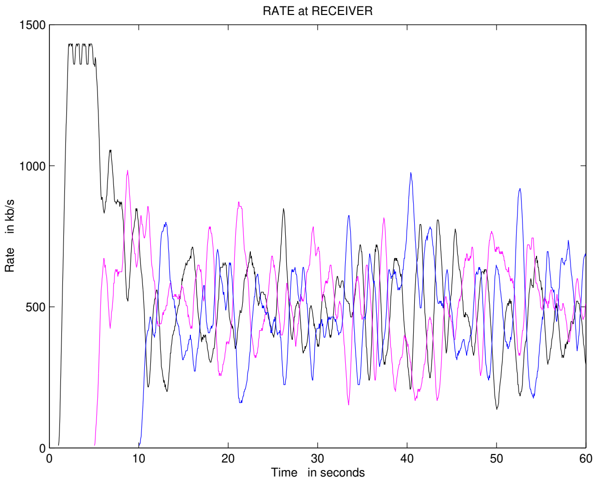

Figure 8: Three WebTP across 1.5 Mbps droptail

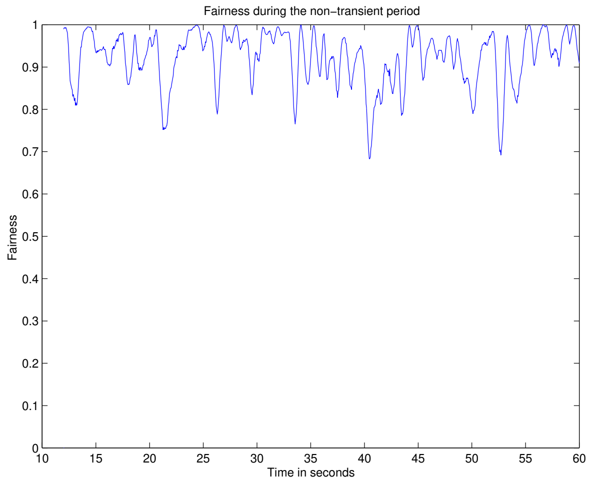

Figure 9: Fairness

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. 9 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 [1], this leads to a fair sharing of the allocated bandwidth across the bottleneck link.

4.3 WebTP and TCP Interact Well

Fig.10 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.11) shows the fairness to remain above 0.9 for the entire period — suggesting a satisfactory interaction between 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

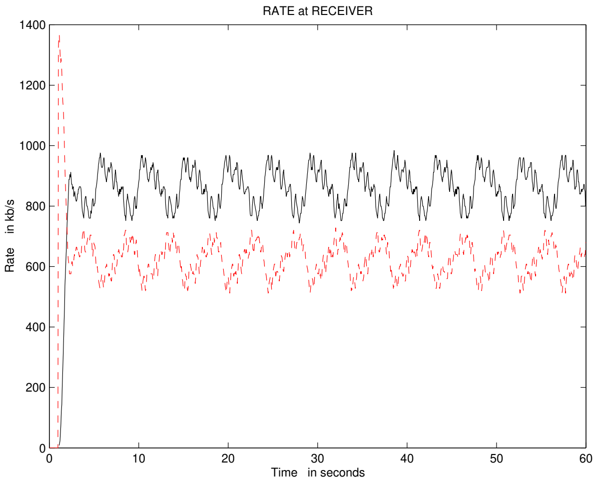

Figure 10: One WebTP One TCP flow across 1.5 Mbps

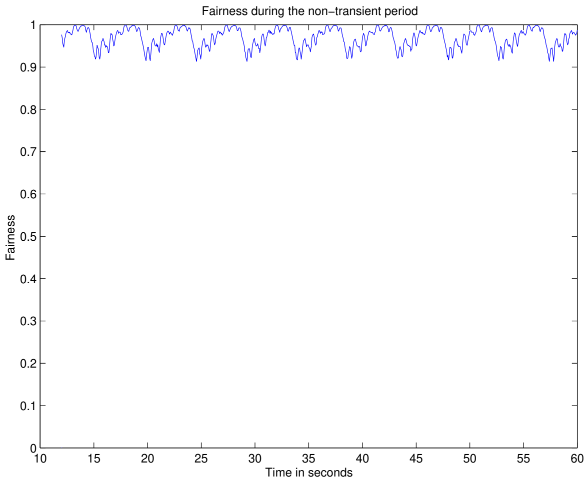

Figure 11: Fairness

nonetheless maintains a steady stream and further fine-tuning of the parameters should provide improved performance that compete even more fairly with TCP.

4.4 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.12 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.13, 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.14 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 experiments, 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 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

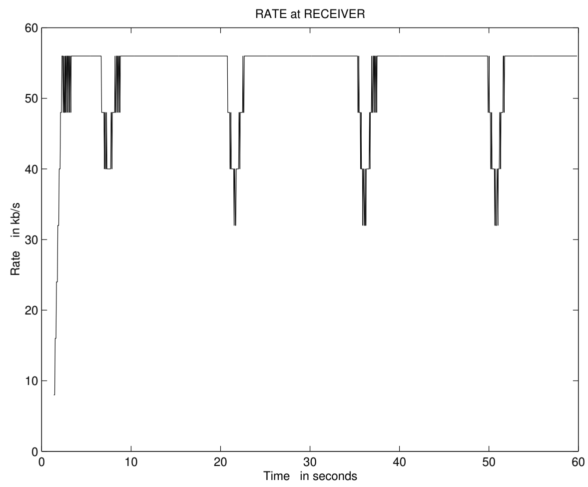

Figure 12: One WebTP flow across a 56 kbps link

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).

5 Conclusion

5.1 Summary

The WebTP research effort 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 incorporate the user into the connection loop and optimize the transport by using ALF. The result is a vertical cross-layer protocol that sits astride the IP layer and incorporates the functionalities of the application and transport layers, maintaining efficient interaction between the application and transport processes.

In this paper, we presented a potential transport layer solution for WebTP that is completely receiver-based in terms of connection setup, flow-control and congestion-control. It includes a window-cum-rate based congestion control mechanism which is friendly to competing TCP flows, and in fact has a superior reaction to congested network state. 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.

This paper studies the feasibility of a receiver-driven model for WebTP, and the work presented here is easily applicable to any other research effort that approaches the transport layer from a receiver's point of

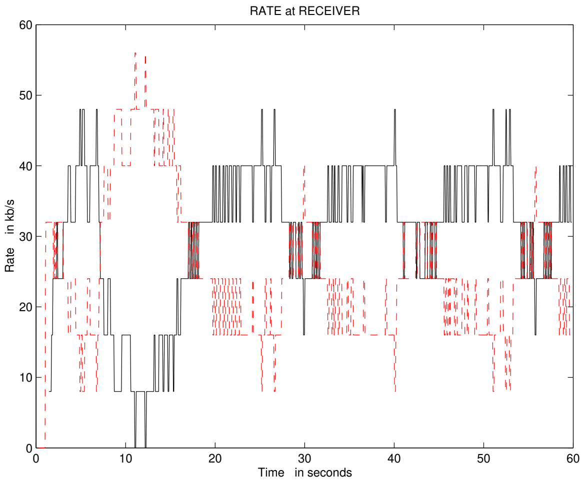

Figure 13: One WebTP One TCP flow across 56 kbps

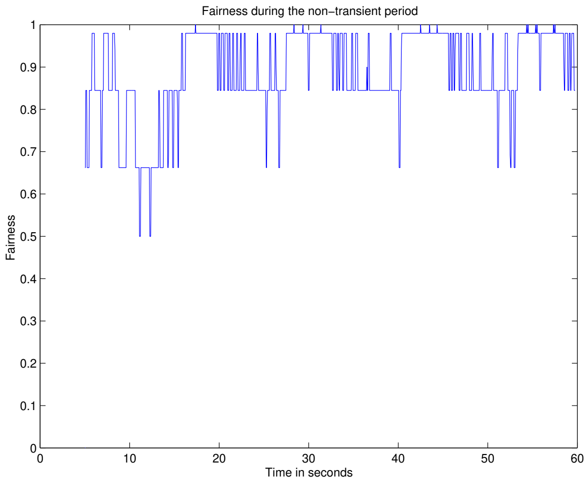

Figure 14: Fairness

view.

5.2 Future Work

There are several important issues still to be tackled before the receiver-driven transport protocol becomes a reality. The initial implementation of WebTP has several design parameters which were chosen based on a combination of intuition and trials. We are working to tune these to optimize performance and evaluate their effects on the protocol behavior. So far, the WebTP model delegates much of the responsibility to the receivers, and we need to create safeguards against malicious receivers launching denial-of-service attacks. We also realize the need for an alternative request mechanism for pages with dynamically generated content (where the receiver is unaware of the objects to request) and have worked out a meta-data representation of such content.

Even apart from the transport aspects, there are several other areas of the WebTP project which have not been addressed at all in this paper. These include user-centric optimization schemes to incorporate the preferrences of the user in the transaction loop, an ALF-based ADU naming mechanism for object transport, an effective set of APIs to allow the application to interact with the transport layer mechanisms, and a browser front-end to provide a user-level handle to the WebTP features. We are also working on a versatile layer between the application and transport which will enable sharing of state across flows and a weighted fair sharing of the bandwidth between competing applications.

The progress of the different aspects of the project and related publications and presentations may be found at *http://webtp.eecs.berkeley.edu*.

5.3 Acknowledgements

We would like to thank Venkat Anantharam, David Tse and Pravin Varaiya for their helpful comments and suggestions and their invaluable contribution towards the planning of WebTP. Thanks are also due to the rest of the WebTP team for their collaboration and comments on this effort. This research is funded in part by a grant from the National Science Foundation.

References

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

[2] D. Clark, M. Lambert and L. Zhang, NetBLT: A high throughput transport protocol, in: *Proceedings of SIGCOMM '87*, 1987, pp 353-359.

[3] D. Clark and D. Tennenhouse, Architectural considerations for a new generation of protocols, in: *Proceedings of SIGCOMM '90*, Philadelphia, PA, September 1990.

[4] 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.

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

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

[7] R. Gupta, WebTP: A User-Centric Receiver-Driven Web Transport Protocol, Master's Thesis, University of California, Berkeley (1998).

[8] V. Jacobson, Congestion avoidance and control, in: *Proceedings of SIGCOMM '88*, Palo Alto, CA, August 1988.

[9] 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.

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

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

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

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

[14] V. Padmanabhan and J. Mogul, Improving HTTP Latency, Computer Networks and ISDN Systems, v.28, nos. 1 & 2, (1995) pp. 25-35.

[15] J. Postel (Ed.), Transmission Control Protocol, Internet Request for Comments RFC 793, September 1981.