Hybrid TCP-UDP Transport for Web Traffic

Israel Cidon and Raphael Rom
Sun Microsystems Laboratories
901 San Antonio Road
Palo Alto, CA 94303, USA
{israel.cidon,raphael.rom}@sun.com
and
Dept. of Electrical Engineering
Technion, Haifa, Israel
{cidon,rom}@ee.technion.ac.il

Amit Gupta and Christoph Schuba
Sun Microsystems Laboratories
901 San Antonio Road
Palo Alto, CA 94303, USA
{amit.gupta,christoph.schuba}@sun.com

Abstract

Most of the web traffic today uses the HyperText Transfer Protocol *(HTTP), with* Transmission Control Protocol *(TCP) as the underlying transport protocol. Unfortunately, TCP is poorly suited for the short conversations that comprise a significant component of web traffic. The overhead of setting up and tearing down TCP state amortizes poorly for these small connections. Moreover, emerging modern web server systems employ HTTP* redirection *for server load-balancing and content distribution; such schemes require setting up (and tearing down) multiple TCP connections for servicing a single client request.*

We have designed and analyzed a hybrid scheme to address these issues. The scheme uses either TCP, or the User Datagram Protocol *(UDP) as the underlying transport protocol for carrying web traffic. UDP is used for short transfers (including HTTP redirection), while TCP is used for all other transfers. In this manner, we avoid the extra TCP overhead for short connections, but still benefit from the reliable delivery and congestion control that TCP provides. We ran trace-based simulations to quantify the effects of various network parameters (i.e., packet loss rates) on the performance of the hybrid scheme. We observed performance gains exceeding 20-25% with HTTP/1.1-style persistent connections, and over 40-50% without persistent connections. These gains can be improved with further performance optimizations that we describe.*

1 Introduction

As the *World Wide Web* (WWW) and other Internet applications, such as real-time audio and video become more and more popular, both the Internet itself and the most popular sites are suffering from severe congestion. This congestion is perceived by users as service delays at best, and as a lack of service at worst.

Many web sites today use a variety of approaches, including server and content replication and distribution, to improve service latency and reliability. Current HTTP *redirection* solutions require tearing down and setting up multiple TCP connections. The redirection operation itself is a short lived HTTP connection. Several commercial products such as the BrightTiger[1] use HTTP redirection for content distribution at the added expense of setting up multiple TCP connections.

This paper takes a different approach and tackles the problem at its roots. Today, web services usually use a single transport protocol, namely TCP, for all traffic; dynamically selecting the underlying transport protocol (e.g., TCP or UDP) can substantially improve service latency and reduce network traffic and server load.

This hybrid TCP-UDP scheme combines the best of both worlds: for short transactions it benefits from stateless UDP's low overheads; on the other hand, for large data transfers, it obtains the desired reliability, resequencing, and congestion control from TCP. The hybrid scheme is attractive because it only requires application-level changes, while the operating system kernel code can remain unchanged. Finally, the scheme is incrementally deployable in the current Internet and is fully compatible with the installed base.

Previous research investigated the cost of high TCP overhead for small connections. *TCP for Transactions* (T/TCP) [1] was developed as a transport protocol for request/reply type message passing protocols. For web traffic, T/TCP behavior and performance should be similar to that of persistent-HTTP: the first connection between a pair of hosts requires a three-way handshake while successive connections avoid this overhead. Heidemann et al. [4] relies entirely on using UDP augmented with adaptive re-

[1] http://www.brighttiger.com/

transmission and congestion control mechanisms, thereby mimicking the behavior of TCP [6]. The disadvantage of such a scheme is that we need to develop yet another complex protocol with reliability, resequencing and congestion avoidance, duplicating the long tedious process of TCP development.

2 Background

HTTP defines a request/reply protocol, where client applications can request data from servers by providing a *Universal Resource Locator* (URL). HTTP is used to address many different types of resources, including text, image, audio, video, executable files, index search results, and database query results.

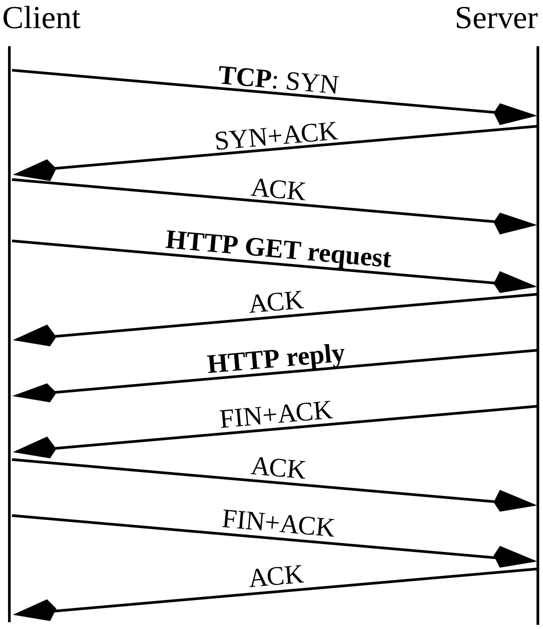

Figure 1: HTTP GET request and reply over TCP.

Figure 1 illustrates a typical packet exchange for an HTTP GET request between a client and a server. In Figure 1, only two packets seem to be *useful* (i.e., carrying data): one is the HTTP GET request, and the other one the HTTP reply. All other packets represent TCP overhead. Under the HTTP protocol Version 1.0, each transfer requires a separate TCP connection. HTTP Version 1.1 introduced *persistent HTTP connections* to address this problem.

One may expect that all pages for a web site (i.e., with a common server identifier) reside on the same server. For example, the URLs `http://www.sunlabs.com/people/index.html` and the URL `http://www.sunlabs.com/research/index.html` share the server identifier `www.sunlabs.com` and we may expect that both URLs will be fetched from the same web server. In practice, many web sites are set up to use different servers for different URLs. In the example described above, the first URL may be served by the server `www.people.sunlabs.com` while the second is served by `www.research.sunlabs.com`. We call this approach *heterogeneous content provisioning*.

The HTTP REDIRECT mechanism is used to support heterogeneous content. HTTP server A hands off HTTP requests from clients to server B by sending a REDIRECT as a response to an HTTP GET request. Note that the HTTP REDIRECT is part of a short HTTP transfer. Using TCP for an HTTP request–HTTP REDIRECT pair requires at least 7 packets (usually 9-10 packets), while two packets suffice if UDP is used as the underlying transport protocol.

3 Design

A good solution for reducing the TCP overhead for small connections should have the following characteristics: transparency to users, backward compatibility, opportunity for incremental deployment, and low runtime overhead in space and time (if any).

3.1 Proposal: Hybrid TCP-UDP Transport

One approach is to use UDP instead of TCP as the transport protocol for HTTP traffic. However, in the presence of unreliable network layer communications, reliability services and congestion management need to be added to UDP to make this a viable proposal. We decided against putting such functionality into UDP.

Instead, we use a hybrid approach where short connections are served using UDP and large connections use TCP as its transport protocol. In this manner, the TCP overhead for short connections is avoided, but the benefits from TCP's well tuned timers, retransmission, congestion control, and error recovery mechanisms are preserved. In this scheme, clients first attempt to use UDP as their transport protocol, and fall back to using TCP, if UDP turns out to be the wrong choice for the requested URL. The fall back mechanism provides the following guarantees:

- If any of the initial UDP packets are lost, the loss is gracefully handled by switching to TCP.
- If the contacted HTTP server does not implement the hybrid scheme, the client will re-try with TCP.

Figure 2 presents this algorithm in more detail. A *hybrid* capable HTTP client sends the HTTP GET request using UDP as the underlying transport protocol. The client starts a timer at the time the request is sent.

When the server processes the client's request, it can choose from one of the following alternatives:

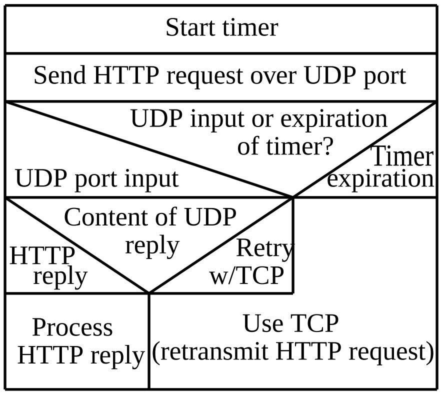

Figure 2: Algorithm executed at HTTP client application to receive HTTP service over either TCP or UDP.

1. If the response is small enough (say, it fits into one datagram), the server returns it using UDP. Small web pages and most cache validation and HTTP REDIRECT responses fall into this category.
2. The server can ask the client to re-try using TCP if the reply is too large to fit into a single UDP packet. At this time, the server can also ask the client to try a different URL. This approach is useful if the server generated the reply dynamically and attempts to avoid generating the reply a second time for the subsequent re-try with TCP. This extension can be implemented with a new HTTP return code.

At the client side, one of the following three events can happen:

- The client gets a response from the server. If the reply contains the desired HTTP reply, the client processes the data. If the server asks the client to re-try (a different URL and/or using TCP), the client does so.
- If the server does not handle HTTP packets sent over UDP, the client may get an ICMP (*Internet Control Message Protocol* [9]) error message (destination unreachable/protocol unreachable). In this case, the client should re-try using TCP.
- If the timer expires, the client should re-try with TCP.

Figure 3 illustrates the packet exchange in case the timer expires. This time-out feature provides reliability and backwards compatibility with servers that do not use the hybrid TCP-UDP scheme. We recommend that this timeout interval be set the same as the corresponding TCP initial timeout interval. As the upgraded clients can work work with servers that do not implement the hybrid scheme, this fall-back mechanism supports gradual, incremental deployment in the Internet.

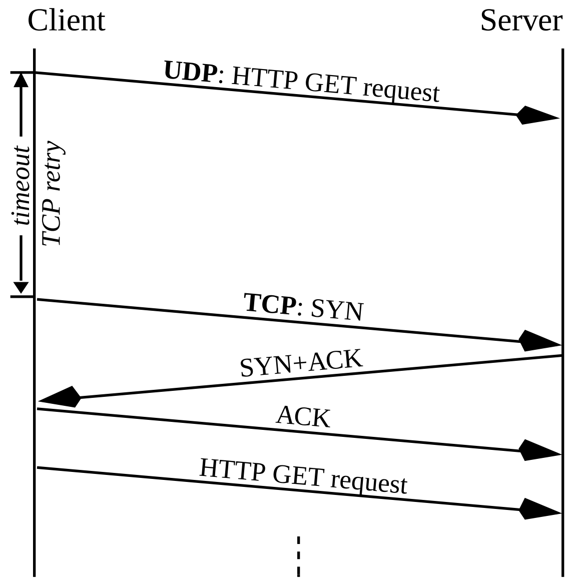

Figure 3: Implicit fall back from UDP to TCP. One of the UDP request/response packets got lost or the server is not UDP capable.

Figure 4 demonstrates the packet exchange where the server requests the client to resend the HTTP request over TCP.

For heterogeneous content, our hybrid scheme offers a new mechanism to redirect requests; this new mechanism is transparent to clients. When the HTTP server receives an HTTP request over UDP, the request can be forwarded and served by another server without further client involvement. The server that sends the HTTP response must masquerade as the first server, or the client would not accept the HTTP reply. It is interesting to see that we can forward a single HTTP request through a chain of servers before we find one that can respond to the client, and it would still be completely transparent to the clients.

The use of UDP makes this optimization possible without any kernel-level changes in the server operating systems. This scheme can be used to improve web cache validation, robust anycasting, and transparent proxies.

Furthermore, we can use HTTP proxies to incrementally deploy the hybrid TCP-UDP scheme without modifying the installed client browsers (please see [2] for details).

The hybrid TCP-UDP scheme improves the utility of

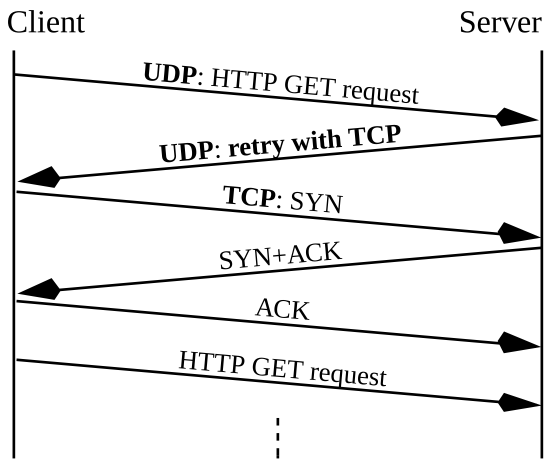

Figure 4: Explicit fall back from UDP to TCP. Server is UDP capable, but the requested resource warrants use of TCP.

HTTP for all three entities involved in HTTP transfers: clients, servers, and the network. The scheme achieves

- reduced browsing latency for clients because latency introduced by the 3-way handshake for TCP connection establishment is avoided,
- reduced load for servers because fewer TCP connections are set-up, maintained, torn down, and because fewer packets are processed, and
- reduced network traffic because fewer TCP control packets are needed.

4 Performance Evaluation

In this section, we quantify the gains obtained by using the hybrid TCP-UDP scheme; our analysis concentrates on answering the following questions:

- How do the use of persistent HTTP and the choice of connection parameters (e.g., max keep-alive time, number of concurrent connections) affect the savings resulting from the use of the hybrid scheme?
- How do other network-related parameters, including network loss rates and MTU size affect the savings?
- How are these savings then affected by various parameter choices for the hybrid TCP-UDP scheme, as described in Section 4.2? We are especially interested in the *max-udp* parameter - UDP is used only when the HTTP response is small enough to fit within *max-udp* packets.

Answering these questions, the following sections present the results from several simulations on our C++/Tcl based simulator. Our goal was to make the experiments realistic so that the results obtained can be transposed to implementations. For this reason, we decided against using synthetic workloads. Instead, we relied on HTTP trace data to drive our simulations. This trace data consists of 18 days' worth of HTTP traces gathered from U.C. Berkeley's Home IP service [3] and of traffic traces obtained within Sun Microsystems' internal network. The traffic for port 80 (HTTP) was recorded. All other protocols (or ports) were excluded from these traces. The traces [3] amounted to 9 million connections over 18 days, thereby providing us with about 500K HTTP transfers per day.

To get an initial feeling for the relevance of the approach we first derived some basic statistics from the traces. First, we derived the Cumulative Distribution Function (CDF) of the size of the replied data in the servers' response. A curve-fitting experiment indicated that a shifted exponential distribution of the type $1 - exp[-.00035(x - 80)]$ (where x is the reply size in bytes, and 80 is the size of the shortest reply) provides a very close approximation, in particular for response sizes smaller than 4KBytes. These data are depicted in Figure 5. A noteworthy result of this distribution is that approximately 40% of the replies would fit into a single 1500 Byte packet. These results are in line with those reported in [5].

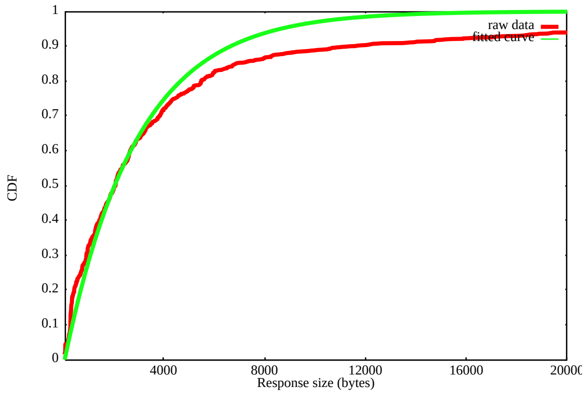

Figure 5: CDF of reply size

We also examined the statistics of HTTP transactions with respect to persistence. Figures 6 and 7 depict the histogram and CDF of the HTTP transactions that took place over the entire measurement period. Noteworthy here is the dominance of the occasional transaction. For example, 88% of the source destination pairs conducted 40 or fewer transactions over the entire period and 98% conducted fewer than 100 transactions. Examining the inter-arrival times of these pairs reveals that these transactions where randomly distributed over the entire measurement period.

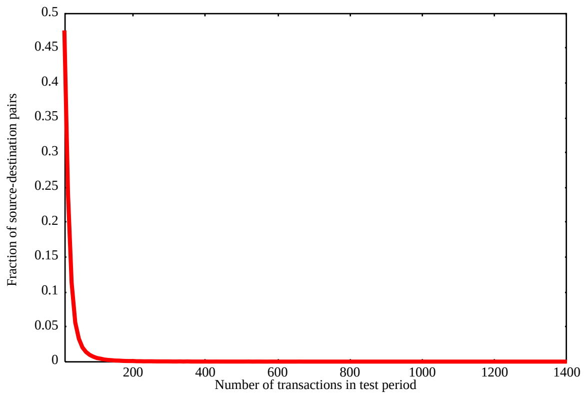

Figure 6: Histogram of transaction densities

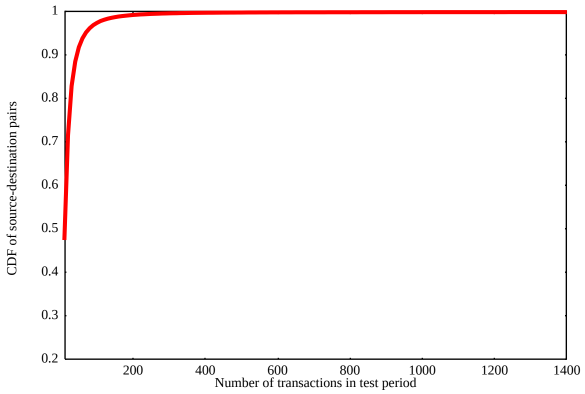

Figure 7: CDF of transaction densities

These results are very encouraging. They indicate that for a vast majority of the transactions persistence would not help because the underlying TCP connection would not stay open long enough to allow its reuse. Therefore, most connections would have to be reestablished, and, given the distribution of the reply size, most connections would complete with a very small number of reply packets.

With these encouraging initial results we conducted more elaborate experiments testing the approach under a variety of parameter changes as indicated above.

4.1 Evaluation Metrics

The following metrics are used for the performance evaluation:

Number of bytes/packets transferred This metric captures the network load, as well as one aspect of the load on the servers and the clients.

Browsing latency This metric captures the performance as observed by the clients.

Number of connections set up This metric captures one aspect of the load on the servers (extra work required to set up and maintain TCP connection state, as well as extra cost in searching *Process Control Block* (PCB) lists).

The choice of our third metric, number of connections set up, was motivated by the following reasons. Our packet traces were HTTP logs. It was impossible for us to obtain the inter-packet traffic patterns that are needed for the first two metrics. Furthermore, the number of connections is the appropriate metric for evaluating HTTP redirection (as well as cache validation). These requests, and the corresponding responses, tend to be small enough to fit into the minimum-MTU IP packets[2]. Also, this metric is linked to the other two metrics. If each HTTP transfer requires a new TCP connection, more bytes/packets will be seen on the network (connection setup and teardown packets) and clients will observe extra latency (due to connection setup delays as well as slow-start latency).

We convert this metric to a unitless ratio (fraction of connections saved) by performing each experiment twice. If an instance requires 100 distinct TCP connections under the current system, and it requires only 80 distinct TCP connections under the proposed hybrid TCP-UDP scheme, we obtain the resulting fraction of $(100 - 80)/100 = 20\%$.

4.2 Experiments

We performed many sets of experiments, each time varying one of four workload parameters:

Persist This parameter describes the fraction of requests from the clients that support persistent HTTP. The parameter value ranges from 0.0 to 100.0 - a value of 70.0 implies that 70% of requests originate from clients that support persistent HTTP.

Loss-rate This parameter describes the fraction of packets that are lost in transit. The parameter value ranges from 0.0 to 1.0 - a value of 0.10 implies that with

[2] The Internet Protocol requires the underlying network to support IP packets at least as large as 576 bytes ([8, §3.1]).

10% probability a packet will be dropped in the network. For these simulations, we assumed that packet losses are independent (not bursty - zero correlation). This is a pessimistic assumption because it overstates the number of connections that will lose at least one packet, thereby diminishing the overall benefits of the hybrid TCP-UDP scheme.

Max-udp This parameter controls the server policy for choosing between use of UDP and TCP in the hybrid TCP-UDP scheme. A value of max-udp=4 implies the server will try to use UDP for all conversations where it can send the entire data in up to 4 packets. Otherwise, it directs the client to use TCP.

MTU This parameter describes the *Maximum Transmission Unit* (MTU) for packets in the network. An MTU size of 1460 implies that the network MTU allows for up to 1460 bytes payload (packet size minus TCP header, IP header, and link-level headers). We assume that the end hosts send MTU-sized packets whenever possible. The larger the MTU, the more likely it is that the hybrid TCP-UDP scheme would avoid setting up a TCP connection (we avoid a TCP connection when HTTP data size is less than MTU · max-udp, and the network does not lose any of these packets).

In our simulations, we included further parameters for controlling the behavior of persistent HTTP. A keep-alive parameter was set to 60.0, meaning that a persistent HTTP connection would be closed after 60 seconds of inactivity. Also, a max-connect parameter was set to 1024. Thus, the server will only support 1024 concurrently active connections. Inactive connections were closed according to a *Least Recently Used* (LRU) policy.

4.3 Results

The simulation results were reasonably similar for the various traffic traces. For all graphs in this Section, the x-axis quantifies the varied parameter (e.g., loss rate) and the y-axis quantifies the evaluation metric (e.g., fractions of connections saved, expressed as a percentage). Due to space limitations, we only describe the results of two sets of experiments; please see [2] for more simulation results, as well as expanded discussion of the design issues.

Loss Rate and Max-Udp

The graph in Figure 8 shows the effect of the network packet loss rate and the setting of max-udp parameter in the server on the overall performance of the hybrid TCP-UDP scheme. For this experiment, we assumed an MTU of 1460 bytes, as well as full persistence (a very conservative assumption), i.e., all clients support persistent HTTP, and all requests are multiplexed on existing TCP connections wherever possible. As the graph shows, with the very conservative policy of setting max-udp value to 1 (use UDP only if all data from server fits into one packet), the hybrid scheme gains as much as 18%-19% with the typical (around 1%) loss rate observed in many intranets, as well as on the Internet. Even with very high packet losses (around 10% loss rate), the hybrid scheme outperform the traditional mode by 12%-13%. The graph also depicts the additional benefits that using a higher max-udp value would provide: up to 85% improvement over the current schemes. Even with a max-udp setting of 4 to 10, we can expect to see benefits of 35% to 66%, even under the assumption that persistent HTTP is used exclusively.

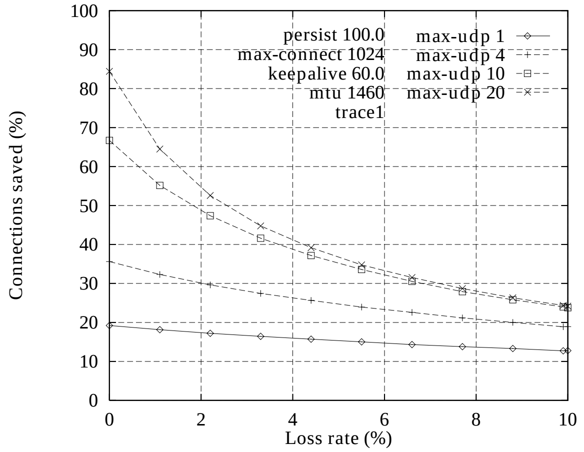

Figure 8: Effect of loss rate and max-udp parameter

Persistent HTTP and Loss Rate

The graphs in Figures 9 and 10 show the effect of persistent-HTTP and network packet loss rates on the overall performance of the hybrid TCP-UDP scheme. The max-udp parameter is set to the value 1 in Figure 9 and the value 4 in Figure 10; the MTU remains 1460 bytes. We performed this experiment to evaluate the performance in the presence of heterogeneous clients (some support persistent HTTP, others do not), as well as to see the effect of the max-udp parameter in this system. As both figures show, the performance is dominated by the presence of non-persistent clients - even with 80% persistence (i.e., only 20% do not support persistence), the results are close to the performance with zero persistence. As expected, the results are a lot better for max-udp of 4 (as compared to max-udp value of 1). As Figure 9 shows, with max-udp set to 1 and with zero persistence, the hybrid scheme provides

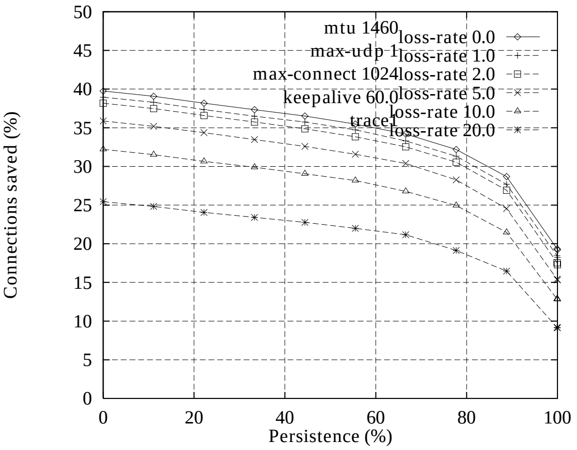

Figure 9: Persistence and loss rate (a)

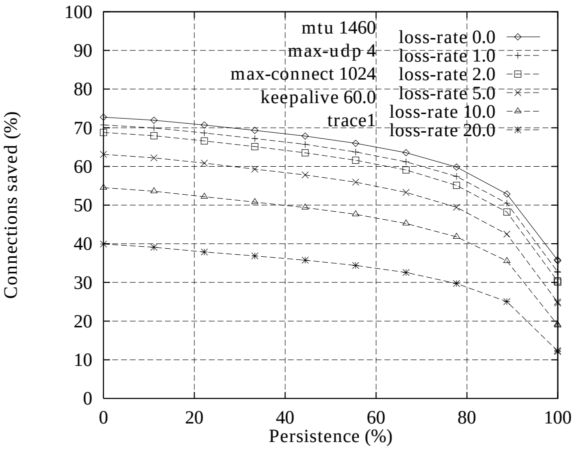

Figure 10: Persistence and loss rate (b)

a gain of 25%-40% over the traditional schemes, depending on the overall packet loss rate. The gains decrease very slightly as the persistence parameter increases to well beyond 80%. Even in the presence of 90% to full persistence, the hybrid scheme provides performance gains of 15% to 30%, except when the packet loss rate goes up to 20% (i.e., pathological conditions). With the max-udp set to 4, the performance gains tend to be around 70% for moderate loss rates, decreasing to around 60% for 70%-80% persistence. Even with 10% packet loss rate, the hybrid scheme gains around 40% at 80% persistence.

4.4 Summary of Simulation Results

The hybrid TCP-UDP scheme provides significant performance gains over the traditional use of TCP as the only transport protocol for HTTP traffic. We observed performance gains exceeding 20-25% with persistent HTTP clients, and over 40-50% with clients without persistent HTTP.

In heterogeneous environments, the system performance was dominated by the presence of non-persistent clients. Even with 70% to 80% persistence, the performance gains of the hybrid scheme were close to that of a system with zero persistence.

The hybrid TCP-UDP scheme's performance gains were reduced somewhat by network packet losses, but the gains remained significant even under pathological network behavior (10% to 20% packet loss rate).

It is beneficial to try sending more than 1 packet via UDP, especially if the network MTU is small (around 500 bytes). However, we must appropriately consider and trade-off the increased likelihood of network congestion due to the increase in the max-udp parameter.

5 Conclusions and Outlook

This paper introduced and analyzed a hybrid TCP-UDP transport layer scheme for HTTP. The hybrid TCP-UDP scheme combines the low cost of using UDP with the high reliability and congestion control features of TCP. It benefits from the low overhead of using UDP for short transfers, and for large transfers, it is able to use TCP for reliable delivery and good congestion behavior. Our simulation experiments verified these gains. We observed performance gains exceeding 25% with persistent HTTP clients, and over 50% for clients without persistent HTTP, even in the presence of high network packet losses. The performance gains for mixed environments (i.e., clients using persistent HTTP, as well as traditional HTTP) were similar to those of a system where no clients used persistent HTTP. The hybrid scheme is attractive because it only requires application-level changes, while the operating system kernel code can remain unchanged. Finally, the scheme is incrementally deployable in the current Internet and it is fully compatible with the deployed base.

We are currently exploring some promising optimizations to further improve system performance. First, the clients can use some heuristics to guess (on a per-transfer or per-session basis) if they can use UDP for successful transmission. For example, the clients can keep track of whether the server supports the hybrid scheme at all, or predict the file size based on the size of the (expired) cached copy of a previously retrieved version, or assume that the Postscript and PDF files tend to be large, while README files tend to be small. Second, the servers can adapt the max-udp parameter based on the observed network loss rates. Third, the clients can avoid redundant data retransmissions by using sub-ranges appropriately.

We also expect that further performance gains can be observed, if recent trends in web traffic continue. The increasing use of style sheets will lead to smaller HTTP transfers, and so will the increasing use of cache validation and HTTP redirection. The performance gains can be further increased by setting the server policy to use UDP more aggressively (for example, when the data can fit into 4 packets, analogous to some recent proposals to increase the initial TCP window size[7]).

References

[1] Bob Braden. *RFC-1644 T/TCP - TCP Extensions for Transactions*. Network Working Group, July 1994.

[2] Israel Cidon, Amit Gupta, Raphael Rom, and Christoph Schuba. Hybrid TCP-UDP Transport for Web Traffic. Technical Report TR-98-71, Sun Microsystems Laboratories, Palo Alto, California, December 1998. Available at http://www.sunlabs.com/projects/hsn/papers/hybrid.ps.

[3] Steven D. Gribble. UC Berkeley Home IP HTTP traces. July 1997. Available at http://www.acm.org/sigcomm/ITA/.

[4] John Heidemann, Katia Obraczka, and Joe Touch. Modeling the Performance of HTTP Over Several Transport Protocols. *IEEE/ACM Transactions on Networking*, 5(5):616–630, October 1997.

[5] Bruce Mah. An Empirical Model of HTTP Network Traffic. In *Proceedings of Infocom'97*, pages 593–600, Kobe, Japan, 1997.

[6] B. Clifford Neuman. *The Virtual System Model: A Scalable Approach to Organizing Large Systems*. PhD thesis, University of Seattle, Washington, 1992.

[7] Kedarnath Poduri and Kathleen Nichols. *RFC-2415 Simulation studies of increased initial TCP window size*. Network Working Group, September 1998.

[8] Jon Postel, editor. *RFC-791 Internet Protocol*. Information Science Institute, University of Southern California, September 1981.

[9] Jon Postel, editor. *RFC-792 Internet Control Message Protocol*. Information Sciences Institute, University of Southern California, September 1981.