TPOT: Translucent Proxying of TCP

Pablo Rodriguez[*]
EURECOM, France
rodrigue@eurecom.fr

Sandeep Sibal, Oliver Spatscheck
AT&T Labs – Research
{*sibal,spatsch*}*@research.att.com*

Abstract

Transparent proxies are being widely deployed in the current Internet to enable a vast variety of applications. These include Web proxy caching, transcoding, service differentiation and load balancing. To ensure that all IP packets of an intercepted TCP connection are seen by the intercepting transparent proxy, they must sit at focal points in the network. *Translucent* Proxying of TCP (TPOT) overcomes this limitation by using TCP options and IP tunneling to ensure that all IP packets belonging to a TCP connection will traverse the proxy that intercepted the first packet. This guarantee allows the ad-hoc deployment of TPOT proxies *anywhere* within the network. No extra signaling support is required. In addition to the advantages TPOT proxies offer at the application level, they also generally *improve* the throughput of intercepted TCP connections. In this paper we discuss the TPOT protocol, explain how it enables various applications, address deployment and scalability issues, and summarize the impact of TPOT on TCP performance.

1 Introduction and Related Work

Transparent proxies are commonly used in solutions when an application is to be proxied in a manner that is completely oblivious to a client, without requiring any prior configuration. Recently, there has been a great deal of activity in the area of transparent proxies for Web caching. Several vendors in the area of Web proxy caching have announced dedicated Web proxy switches and appliances [1, 2, 7, 10].

In the simplest scenario, a transparent proxy intercepts all TCP connections that are routed through it. This may be refined by having the proxy intercept TCP connections destined only for specific ports (e.g., 80 for HTTP), or for a specific set of destination addresses. The proxy responds to the client request, masquerading as the remote web server. Scalability is achieved by partitioning client requests into separate hash buckets based on the destination address, effectively mapping web servers to multiple caches attached to the proxy.

In the event of a cache miss, the cache re-issues the request to the web server, and pipes the response it receives from the web server back to the client, keeping a copy for itself (assuming the response is cacheable). Note that, in general, this mechanism may be repeated, where a subsequent proxy along the path may intercept an earlier cache miss, and so on.

The proxy described above is often termed as a Layer-4 switch, or simply L-4 switch, since TCP is a Transport Layer protocol, which maps to Layer 4 in the OSI networking stack. In a variant of the above, the proxy/switch parses the HTTP request and extracts the URL and possibly other fields of the HTTP Request, before deciding what to do with the request. Since such a switch inspects the HTTP Request, which is an Application Layer or Layer 7 function, it is called an L-7 switch [2].

An acute problem that limits the use of transparent L-4 and L-7 Web proxies, is the need to have the proxy at a location that is guaranteed to see *all* the packets of the request [7]. Since routing in an IP network can lead to

[*] He contributed to this work during an internship at AT&T.

situations where multiple paths from client to server can have the lowest cost, packets of a connection may sometimes follow multiple paths. In such a situation a transparent proxy may see only a fraction of packets of the connection. Occasionally it is also possible that routes change mid-way through a TCP connection, due to routing updates in the underlying IP network. For these reasons transparent proxies are deployed exclusively at the edges or *focal* points within the network – such as gateways to/from single-homed clients or servers. This is not always the best place to deploy a cache. In general one would expect higher hit rates for objects cached deeper inside the network [8].

TPOT solves this problem by making an innovative use of TCP-OPTIONs and IP tunnels. A source initiating a TCP connection signals to potential proxies that it is TPOT-enabled by setting a TCP-OPTION within the SYN packet. A TPOT proxy, on seeing such a SYN packet, intercepts it. The ACK packet that it returns to the source carries the proxy's IP address stuffed within a TCP-OPTION. On receiving this ACK, the source sends the rest of the packets via the intercepting proxy over an IP tunnel. The protocol is discussed in detail in Section 3.

The above mechanism will work if the client is TPOT enabled. In a situation where the client is not TPOT enabled, we may still be able to use TPOT. As long as the client is single-homed, and has a proxy at a focal point, we can TPOT enable the connection by having the proxy behave like a regular transparent proxy on the side facing the client, but a TPOT (translucent) proxy on the side facing the server.

The general idea of using TCP-OPTIONs as a signaling scheme for proxies is not new [16]. However combining this idea with IP tunneling to *pin down* the path of a TCP connection has not been proposed before to the best of our knowledge.

One alternative to TPOT is the use of Active Network techniques [26]. We believe that TPOT is a relatively lightweight solution that does not require an overhaul of existing IP networks. In addition, TPOT can be deployed incrementally in the current IP network, without disrupting other Internet traffic.

The authors of [21] also use the term *translucent* to distinguish their proposed web caching proposal from *transparent caching*. However, their work is distinct and largely complementary to TPOT. In [21], the routers along the path from the client need to be enhanced so that they can provide the next-hop cache information, to the previous-hop cache. This requires routers to know in advance the information of next-hop caches. TPOT on the other hand does not require any such information, and is therefore easier to administer and manage.

The proposal in [21] has a *maximum number of requests* option that can be exploited in TPOT as well, to limit the number of TPOT proxies that can intercept a TCP connection. We could insert this as part of the TCP-OPTION, and decrement it every time the connection is intercepted by a TPOT proxy.

1.1 Paper Overview

Section 2 highlights two classes of applications of TPOT. Section 3 describes the TPOT protocol. In addition to the basic version, a pipelined version of the protocol is also discussed. Pathological cases, extensions, and limitations are also studied. Section 4 discusses deployment issues. Section 5 discusses our approach to solving this problem using a technique that we call TPARTY, which employs a farm of servers that sit behind a front-end machine. The front-end machine only farms out requests to the army of TPOT machines that sit behind it. We address the TCP level performance of TPOT in Section 6. *Due to space limitations, we were unable to cover the details here. A discussion of the performance of TPOT using both analysis and experiments may be found in an extended version of this paper [23]. The technical report also covers a prototype implementation that was used in these experiments.* Finally Section 7 highlights our major contributions, discusses future work, and possible extensions to TPOT.

2 Applications of TPOT

As mentioned in Section 1 TPOT allows the deployment of TCP proxies anywhere in the network. While several applications exist, in this section we describe two that show how TPOT may be applied to important real world problems.

2.1 Hierarchical Caching and Content Distribution Trees

In addition to allowing the placement of transparent Web proxy caches anywhere in the network, TPOT also enables newer architectures that employ Web proxy *networks*. In such architectures a proxy located along the path from the client to the server simply picks up the request and satisfies it from its own cache, or lets it pass through. This, in turn, may be picked up by another proxy further down the path. These incremental actions lead to the dynamic construction of spontaneous hierarchies rooted at the server. Such architectures require the placement of multiple proxies *within* the network, not just at their edges and gateways. Existing proposals [13, 17, 28] either need extra signaling, or they simply assume that all packets of the connection will pass through an intercepting proxy. Since TPOT explicitly provides this guarantee, implementing such architectures with TPOT is elegant and easy. With TPOT no extra signaling support or prior knowledge of neighboring proxies is required.

2.2 Transcoding

Transcoding refers to a broad class of problems that involve some sort of adaptation of content (e.g., [11, 19]), where content is transformed so as to increase transfer efficiency, or is distilled to suit the capabilities of the client. Another similar use is the notion of enabling a transformer tunnel [25] over a segment of the path within which data transfer is accomplished through some alternate technique that may be better suited to the specific properties of the link(s) traversed. Proposals that we know of in this space require one end-point to explicitly know of the existence of the other end-point – requiring either manual configuration or some external signaling/discovery protocol. TPOT can accomplish such functionality in a superior fashion. In TPOT an end-point non-invasively flags a connection, signifying that it can transform content – without actually performing any transformation. Only if and when a second TPOT proxy (capable of handling this transformation) sees this flag and notifies the first proxy of its existence, does the first proxy begin to transform the connection. Note that this does not require any additional handshake for this to operate correctly, since the TPOT mechanism plays out in concert with TCP's existing 3-way handshake.

3 The TPOT Protocol

This section describes the operation of the basic and pipelined versions of the TPOT protocol. Pathological cases, extensions, and limitations are also studied. Before describing the operation of the TPOT protocol, we provide a brief background of IP and TCP which will help in better understanding TPOT. See [24] for a detailed discussion of TCP.

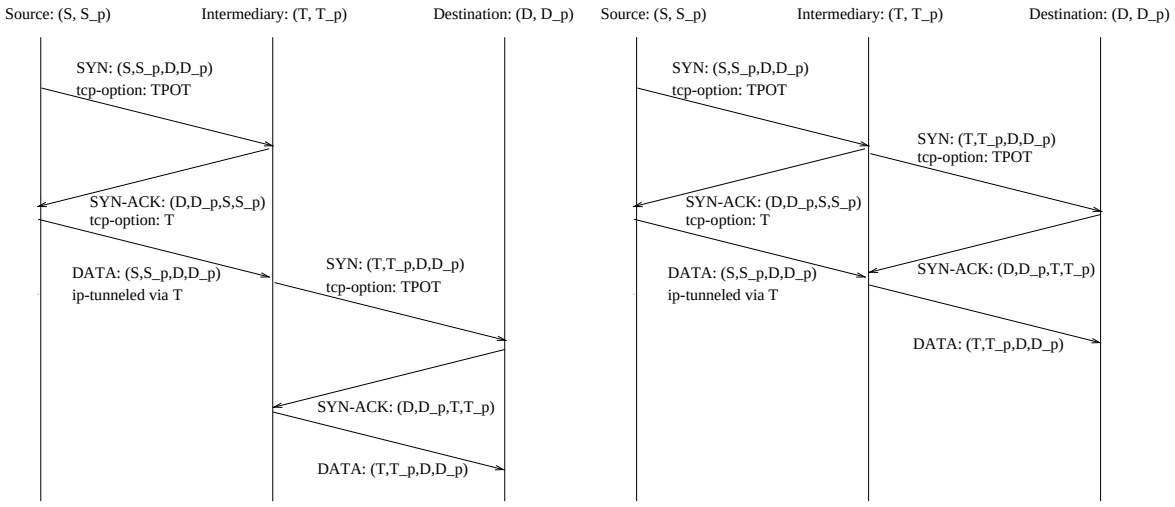
3.1 IP and TCP

Each IP packet typically contains an IP header and a TCP segment. The IP header contains the packet's source and destination IP address. The TCP segment itself contains a TCP header. The TCP header contains the source port and the destination port that the packet is intended for. This 4-tuple of the IP addresses and port numbers of the source and destination uniquely identify the TCP connection that the packet belongs to. In addition, the TCP header contains a flag that indicates whether it is a SYN packet, and also an ACK flag and sequence number that acknowledges the receipt of data from its peer. Finally, a TCP header might also contain TCP-OPTIONs that can be used for custom signaling.

In addition to the above basic format of an IP packet, an IP packet can also be encapsulated in another IP packet. At the source, this involves prefixing an IP header with the IP address of an intermediate tunnel point on an IP packet. On reaching the intermediate tunnel point, the IP header of the intermediary is stripped off. The (remaining) IP packet is then processed as usual. See RFC 2003 [22] for a longer discussion.

3.2 TPOT: Basic Version

In the basic version of TPOT a source **S** that intends to connect with destination **D** via TCP, as shown in Figure 1(a). Assume that the first (SYN) packet sent out by **S** to **D** reaches the intermediary TPOT proxy **T**. **(S,Sp,D,Dp)** is the



(a) Basic Version

(b) Pipelined Version

Figure 1: The TPOT protocol

notation that we use to describe a packet that is headed from **S** to **D**, and has **Sp** and **Dp** as the source and destination ports respectively.

To co-exist peacefully with other end-points that do not wish to talk TPOT, we use a special TCP-OPTION "TPOT," that a source uses to explicitly indicate to TPOT proxies within the network, such as **T**, that they are interested in using the TPOT mechanism. If **T** does not see this option, it will take no action, and simply forwards the packet on to **D** on its fast-path. If **T** sees a SYN packet that has the TCP-OPTION "TPOT" set, it responds to **S** with a SYN-ACK that encodes its own IP address **T** in the TCP-OPTION field. On receiving this packet, **S** must then send the remaining packets of that TCP connection, IP tunneled to **T**. From an implementation standpoint this would imply adding another 20 byte IP header with **T**'s IP address as destination address to all packets that **S** sends out for that TCP connection. Since this additional header is removed on the next TPOT proxy, the total overhead is limited to 20 bytes regardless of the number of TPOT proxies intercepting the connection from the source to the final destination. This overhead can be further reduced by IP header compression [9, 15].

For applications such as Web Caching where **T** may be able to satisfy a request from **S**, the response is simply served from one or more caches attached to **T**. In the case of a "cache miss" or for other applications where **T** might connect to **D** after inspecting some data, **T** communicates with the destination **D** as shown in Figure 1(a). Note that the proxy **T** sets the TCP-OPTION "TPOT" in its SYN to **D** to allow possibly another TPOT proxy along the way to again proxy the connection. Note that Figure 1 only shows the single proxy scenario.

3.3 TPOT: Pipelined Version

In certain situations one can do *better* that the basic version of the TPOT protocol. It is possible for **T** to pipeline the handshake by sending out the SYN to **D** immediately after receiving the SYN from **S**. This *pipelined* version of TPOT is depicted in figure 1(b).

The degree of pipelining depends on the objective of the proxying mechanism. In the case of an L-4 proxy for Web Caching, the initial SYN contains the destination IP address and port number. Since L-4 proxies do not inspect the content, no further information is needed from the connection before deciding a course of action. In such a situation a SYN can be sent out by **T** to **D** almost immediately after **T** received a SYN from **S**, as shown in Figure 1(b). In the case of L-7 switching, however, the proxy **T** would need to inspect the HTTP Request (or at a minimum the URL in

the Request). Since this is typically not sent with the SYN, a SYN sent out to **D** can only happen after the first ACK is received by **T** from **S**. This is consistent with Figure 1.

3.4 Pathological Cases

While the typical operation of TPOT appears correct, we are aware of two pathological cases that also need to be addressed.

1. In a situation when a SYN is retransmitted by **S**, it is possible that the retransmitted SYN is intercepted by **T**, while the first SYN is not – or vice versa. In such a situation, **S** may receive SYN-ACKs from both **D** as well as **T**. In such a situation **S** simply ignores the second SYN-ACK, by sending a RST to the source of the second SYN-ACK.

2. Yet another scenario, is a simultaneous open from **S** to **D** and vice-versa, that uses the same port number. Further **T** intercepts only one of the SYNs. This is a situation that does not arise in the client-server applications which we envision for TPOT. Since **S** can turn on TPOT for only those TCP connections for which TPOT is appropriate, this scenario is not a cause for concern.

3.5 Extensions

As a further sophistication to the TPOT protocol it is possible for multiple proxied TCP connections at a client or proxy that terminate at the same (next-hop) proxy, to integrate their congestion control and loss recovery at the TCP level. Mechanisms such as *TCP-Int* proposed in [4] can be employed in TPOT as well. Since the primary focus of TPOT, and this paper, is to enable proxy services on-the-fly, rather than enhance performance we do not discuss this further. The interested reader is directed to [4] and [27] for such a discussion.

Note that an alternative approach is to multiplex several TCP connections onto a single TCP connection. This is generally more complex as it requires the demarcation of the multiple data-streams, so that they may be sensibly demultiplexed at the other end. Proposals such as P-HTTP [18] and MUX [12], which use this approach, may also be built into TPOT.

3.6 Limitations

As shown in Figure 1 the TCP connection that the intermediate proxy **T** initiates to the destination **D** will carry **T**'s IP address. This defeats any IP-based access-control or authentication that **D** may use. Note that this limitation is not germane to TPOT, and in general, is true of any transparent or explicit proxying mechanism.

In a situation where the entire payload of an IP packet is encrypted, as is the case with IPsec, TPOT will simply not be enabled. This does not break TPOT, it simply restricts its use.

The *purist* may also object to TPOT breaking the semantics of TCP, since in TPOT a proxy **T** in general interacts with **S**, in a fashion that is asynchronous with its interaction with **D**. While it is possible to construct a version of TPOT that preserves the semantics of TCP, we do not pursue it here. In defense, we point to several applications that are prolific on the Internet today (such as firewalls) that are just as promiscuous as TPOT.

4 Deployment Issues

The value of any proposed addition to the standard Internet protocols chiefly depends on the deployment issues involved. To make the protocol valuable it has to address four criteria. First, it should be possible to deploy the protocol gradually without negatively impacting the rest of the Internet. Second the new protocol should allow interoperability between the modified and unmodified network elements. Third it should be possible to gain benefits early on in the deployment when only a few network elements understand this new protocol. Fourth the proposed protocol has to scale to the size and speed of the current and future Internet. The remaining part of this section will address the first three criteria while the scalability issues are addressed in section 5.

TPOT addresses the first criterion, viz., gradual deployment without negative impact, by its inherent nature. TPOT utilizes standard TCP and IP tunneling. Therefore the impact on traffic that is not TPOT enabled is minimal. One possible impact is that the round trip times and error rates of individual TCP connections that are proxied is reduced. The other potential negative impact is that since TPOT forces all packets of a TCP connection through an intermediate proxy that is determined during the connection establishment phase, changes in routes during the lifetime of the connection, may result in sub-optimal routes, since the intermediate proxy may no longer be on the direct path to the destination. However, since TPOT will use IP routing to automatically route the next connection on the direct path, such a sub-optimality is not only rare, but also short-lived.

TPOT deals with the interoperability criterion by utilizing TCP options as its signaling mechanism. TCP options have the built in feature that if a receiver of a TCP option does not support or does not want to support a particular option it can just ignore the option.

So far we have argued that TPOT will not negatively impact the performance and connectivity of todays Internet. We now address the third criterion by showing that TPOT will provide substantial benefits in the early stages of its deployment as well.

The applications described in section 2 assumes that the end-clients are TPOT enabled. For the near future this is most certainly an unrealistic assumption. To circumvent the problem of updating the networking stack of all end-client machines with TPOT, we envision that TPOT will be first implemented only in proxies inside the network. The client-side of the edge proxy will behave like a transparent proxy. To circumvent the split-path problem at the edge, the edge proxy has to be placed at a focal point with respect to the client. This seems to introduce the same disadvantage current transparent proxies have. However, deeper in the network, it adds the advantages that cache hierarchies can be built without configuration or additional signaling overhead. Indeed it is deeper in the network where hit rates are higher and the cost of a cache is more justifiable.

Enabling the placement of proxy caches deeper into the network is particularly interesting for ISPs with well-connected networks. In highly dense networks the only focal point where a TCP connection can be intercepted by a transparent proxy is most likely at the point of present (POP). However, the low aggregate traffic at many dial up POPs and the management overhead prohibits the placement of caches in all POPs. Edge proxies solve this problem. A transparent edge proxy with no caching functionality but with the sole purpose of enabling TPOT on outgoing TCP connections can be built in a small plug and play, disk-less and fan-less box at a fairly low cost. This allows the placement of edge proxies within all POPs and the placement of a reduced number of the more expensive and harder to maintain caches deeper in the network.

5 Scalability Issues

As described in Section 4 scalability is a major issue for any protocol which should be deployed in the Internet. Fortunately, TPOT can be easily parallelized. This allows the scaling of TPOT to support a potentially large number of TCP connections using what we call *TPARTY*.

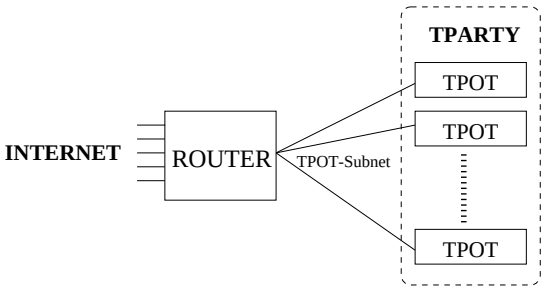


Figure 2: TPARTY: parallelizing TPOT proxies.

TPARTY as shown in figure 2 can be used to scale TPOT in cases when the load cannot be supported by a single

TPOT machine. TPARTY uses a farm of TPOT proxies front-ended by a modified router. In addition to routing, the router forwards TCP SYN packets for certain TCP port numbers which have the TPOT option enabled (and are not received on the line card connected to the TPOT subnet) towards one of the TPOT machines.

When a TPOT-enabled SYN arrives at the TPOT machine, the TPOT machine decides if it can handle an additional request. If it cannot handle the request, the SYN is sent back to the router and the packet is routed as usual to the final destination. In either case, all subsequent packets on the router are routed as plain IP packets. Therefore, the additional processing on the router is limited to detecting and forwarding of TPOT enabled SYN packets.

5.1 Buffer Size

The buffering required on the TPOT machines is another cost of performing proxying in the middle of the network. Each proxy requires a send and receive buffer to terminate the TCP connection. In theory all buffering will happen before the worst link (some combination of the highest RTT and highest packet drop rate), and little or no buffering would be required thereafter. In general, buffering on the order of the advertised window by the remote receiver of the TCP connection on the proxy is required. More studies need to be conducted for estimating the buffer requirements for the TPOT proxy.

5.2 Port Numbers

Another scaling problem arises from the fact that demultiplexing of TCP connections is based on source and destination IP addresses and port numbers. The destination address and port number are usually fixed. The source address seen by the final destination will be the address of the last TPOT proxy. This limits the last proxy to open at most 64K connections (range of the port number space) to a single server. Since ports may time out substantially after a connection is closed, the number of active connections the last TPOT proxy can handle is far lower. The exact number depends on the TCP implementation, but clearly this is a potential bottleneck. Solutions to this problem are multiplex several TCP connections onto a single TCP connection and multihoming the TPOT machine.

6 Proxy Performance

Due to the size limitation of the workshop submission we refer the interested reader to the extended version of this paper [23] for a detailed performance analysis.

Contrary to what we initially expected, TPOT typically *improves* the performance of TCP connections. This apparent counter-intuitive result has been observed before [3, 6, 14], though in somewhat different contexts. In [3] a modified TCP stack called *Indirect TCP* is employed for mobile hosts to combat problems of mobility and unreliability of wireless links. Results show that employing *Indirect TCP* outperforms regular TCP. In [14] similar improvements are reported for the case when TCP connections over a satellite link are split using a proxy. Finally, in [6], the authors discuss at length how TCP performance may be enhanced by using proxies for HFC networks. The notion of inserting proxies with the sole reason of enhancing performance has recently led to the coining of the term *Performance Enhancing Proxies* (PEP). An overview is provided in [5]. As shown in [23], TPOT does indeed enhance performance, but unlike PEP, this is *not* the motivation behind TPOT.

The extended paper [23] also details a prototype implementation of TPOT in Scout [20], which is used in the experiments we conducted for assessing TPOT's performance.

7 Conclusions and Future Work

In this paper we evaluated the benefits of using TPOT – a *translucent* mechanism for proxying TCP connections. In general, transparent proxies do not always see all the packets of a TCP connection, unless they are placed at focal points within the network. TPOT proxies do not suffer from this limitation because of a novel way in which TCP-OPTIONs and IP tunneling are used to *pin down* TCP connections.

Web proxy caches built using TPOT thus have the freedom of being placed *anywhere* in the network, which in turn enables new architectures such as spontaneous Web proxy *networks*, where proxy caching hierarchies are *dynamically* constructed. In addition, transcoding and several other applications can benefit from TPOT.

We believe the results of our work indicate that TPOT is viable, and can be practically deployed in a high-speed network to enable a variety of applications. We are currently investigating its use in building *zero-maintenance* proxy caching networks for the Web.

Acknowledgments

We thank Misha Rabinovich, Adam Buchsbaum, and Fred Douglis of AT&T Labs – Research, for their comments on an earlier draft of this paper.

References

[1] Alteon Networks. Webworking: Networking with the Web in mind. Technical report, San Jose, CA, USA, May 1999.

[2] ArrowPoint Communications. What is Web Switching? Technical report, Westford, MA, USA, 1999.

[3] A. Bakre and B. R. Badrinath. Indirect TCP for mobile hosts. In *International Conference on Distributed Computing Systems (ICDCS)*, May 1995.

[4] H. Balakrishnan, V. Padmanabhan, S. Seshan, R. H. Katz, and M. Stemm. TCP behavior of a busy internet server: Analysis and improvements. In *Proceedings of IEEE INFOCOM*, San Francisco, CA, March 1998.

[5] J. Border, M. Kojo, J. Griner, and G. Montenegro. Performance enhancing proxies. Technical report, IETF, June 1999. Internet Draft draft-ietf-pilc-pep-00.txt.

[6] R. Cohen and S. Ramanathan. TCP for high performance in hybrid fiber coaxial broad-band access networks. *IEEE/ACM Transactions on Networking*, 6(1):15–29, Feb. 1998.

[7] P. Danzig and K. L. Swartz. Transparent, scalable, fail-safe Web caching. Technical report, Network Appliance, Santa Clara, CA, USA, 1999.

[8] P. B. Danzig, R. S. Hall, and M. F. Schwartz. A case for caching file objects inside internetworks. In *Proceedings of the ACM SIGCOMM Conference*, September 1993.

[9] M. Degermark, B. Nordgren, and S. Pink. RFC 2507: IP header compression, Feb. 1999. Status: PROPOSED STANDARD.

[10] Foundry Networks. Transparent cache switching primer. Technical report, Sunnyvale, CA, USA, 1999.

[11] A. Fox and E. A. Brewer. Reducing WWW latency and bandwidth requirements by real-time distillation. *Computer Networks and ISDN Systems*, 28(7–11):1445–1456, May 1996.

[12] J. Gettys. Mux protocol specification. Technical report, W3C, October 1996. wd-mux-961023.

[13] A. Heddaya and S. Mirdad. Webwave: Globally balanced fully distributed caching of hot published documents. In *International Conference on Distributed Computing Systems (ICDCS)*, Baltimore, USA, May 1997.

[14] T. R. Henderson and R. H. Katz. Transport Protocols for Internet-compatible Satellite Networks. *IEEE Journal on Selected Areas in Communications*, 1999. To appear.

[15] V. Jacobson. RFC 1144: Compressing TCP/IP headers for low-speed serial links, Feb. 1990. Status: PROPOSED STANDARD.

[16] B. Knutsson. Proxies and signalling. *Extendable Router Workshop*, August 1999.

[17] P. Krishnan, D. Raz, and Y. Shavitt. Transparent en-route cache location in regular networks. In *DIMACS Workshop on Robust Communication Networks: Interconnection and Survivability*, DIMACS book series, New Brunswick, NJ, USA, November 1998.

[18] J. C. Mogul. The case for persistent-connection HTTP. In *Proceedings of the SIGCOMM'95 conference*, Cambridge, MA, August 1995.

[19] J. C. Mogul, F. Douglis, A. Feldmann, and B. Krishnamurthy. Potential benefits of delta encoding and data compression for HTTP. In *Proceedings of the ACM SIGCOMM Conference*, September 1997.

[20] D. Mosberger and L. Peterson. Making paths explicit in the Scout operating system. In *Proceedings of OSDI '96*, October 1996.

[21] K. Obraczka, P. Danzig, S. Arthachinda, and M. Yousuf. Scalabale, highly-available web caching. Technical Report TR 97-662, CS Department, University of Southern California, 1997.

[22] C. Perkins. RFC 2003: IP encapsulation within IP, Oct. 1996. Status: PROPOSED STANDARD.

[23] P. Rodriguez, S. Sibal, and O. Spatscheck. TPOT: Translucent proxying of TCP. Technical Report TR 00.4.1, AT&T Labs - Research, 2000.

[24] W. R. Stevens. *TCP/IP Illustrated, The Protocols*, volume 1. Addison-Wesley, 1994.

[25] P. Sudame and B. R. Badrinath. Transformer tunnels: A framework for providing route-specific adaptations. In *USENIX Annual Technical Conference*, New Orleans, Louisiana, USA, June 1998.

[26] D. L. Tennenhouse, J. M. Smith, D. Sincoskie, D. J. Wetherhall, and G. J. Minden. A survey of Active network research. *IEEE Communications Magazine*, 35(1):80–86, January 1997.

[27] J. Touch. TCP control block interdependence. Technical report, IETF, April 1997. Interent RFC 2140.

[28] L. Zhang, S. Floyd, and V. Jacobson. Adaptive Web caching. In *NLANR Web Caching Workshop*, June 1997.