Taking Advantage of Multihoming with Session Layer Striping

Ahsan Habib
Siemens TTB, Berkeley
Email: habib@sims.berkeley.edu

Nicolas Christin
Information Networking Institute
and CyLab Japan
Carnegie Mellon University
Email: nicolasc@cmu.edu

John Chuang
School of Information
UC Berkeley
Email: chuang@sims.berkeley.edu

***Abstract*—**
Striping is a resource aggregation technique that can improve application performance for a multihomed host by utilizing multiple interfaces. In this position paper, we argue in favor of decoupling striping primitives from all of the traditional layers of the networking protocol stack, and instead resurrecting the session layer for striping. Session layer striping indeed allows applications to take advantage of multihoming, while avoiding most of the deployability issues traditionally linked with modifiying application layer code. We illustrate our argument with a strawman architecture for a session layer striping protocol, and sketch how such a protocol could be implemented.

I. Introduction

As shown in measurement studies, e.g., [1], [2], simultaneously connecting to multiple service providers, or *multihoming*, can drastically improve the quality-of-service a networked end host experiences. In addition, the differences in peering relationships among different service providers make it possible for multihomed hosts to use significantly different routes to a same destination [3]. In other words, besides circumventing last-mile congestion, multihoming can theoretically allow for dynamically avoiding other points of congestion in the network, by taking advantage of the choice between different routes available at any given time [4], [5].

Furthermore, physical layer support for multihoming is already available in metropolitan areas. Not only can users in urban areas generally subscribe to both cable and DSL services simultaneously, but they may also have access to several wireless networks. Thus, multihoming is both desirable and, at least in some instances, readily available. Yet, very few applications take advantage of the service improvements multihoming potentially enables. Explicitly supporting multihoming within an application is indeed far from straightforward, and can actually result in performance degradation due to blocking phenomena at lower layers [6].

From an economic perspective, application layer support for multihoming requires significant networking expertise from the application developer, which results in higher development costs. Increased development costs for multihoming support are hard to justify, considering the relatively low proportion of multihomed hosts in the network at the present time. The lack of support for multihoming is in turn one of the main reasons why so few end-users have an economic interest in multihoming their hosts.

One way to solve this dilemma is to have network or operating system support for multihoming, thereby freeing the application developer from the burden of implementing multihoming primitives. That is, in an ideal world, any of the layers situated below the application layer should manage the (de)multiplexing of an application layer flow across several physical links in a manner transparent to the application. Such a resource aggregation is known as *striping*. As discussed in [7], striping poses a number of challenges and trade-offs at any of the physical, network, and transport layers.

Our main argument in this position paper is that *session layer striping* allows applications to take full advantage of multihoming, while avoiding the overhead of rewriting most networking primitives at the application layer to support multihoming. We develop our argument by presenting a strawman architecture for a session layer striping protocol, and sketch how such a protocol could be implemented.

More precisely, we attempt to make the case that striping at the session layer has several advantages. First, rather than striping over multiple connections, applications only see a single virtual "pipe," and do not need specific mechanisms to take advantage of multihoming. Second, by decoupling striping from transport layer primitives, multihoming support can be made independent of any specific transport protocol. In fact, presented with a few application layer preferences, a session protocol could even automatize transport protocol selection on behalf of the application.

Existing literature addressing performance improvement through multihoming can roughly be classified in two categories. A number of works (e.g., [2], [8]) look at how multihoming can be used to improve the performance of a whole stub network, dealing with the case where the stub network under consideration is connected to two or more service providers. These works are mostly considering traffic engineering techniques to improve stub network metrics (e.g., overall cost, average latency, resilience to failures...), whereas, in this paper, we are looking at individual application flows.

More closely related to our proposal are works discussing how to multiplex flows over several logical or physical links, e.g., [9], [10], [11], [12], [13]. The main difference between

our proposal and these related works is that we do not make any assumption on the underlying transport protocol or the number of available physical links.

The remainder of this paper is organized as follows. In Section II, we quickly review the advantages and drawbacks of performing striping at layers other than the session layer. In Section III, we propose an architecture for session layer striping. Finally, we conclude, in Section IV, by discussing implementation issues, limitations, and open problems uncovered by our approach.

II. Why session layer striping?

Striping is a general technique to aggregate multiple resources for better performance. The aggregation of multiple network interfaces can be done at different layers of the protocol stack. Adiseshu et al. [14] propose a general framework for striping as a logical FIFO queue (defined as a channel) at the link, network, or transport layers. IPv6 multihoming can be performed at any layer ranging from the network to the application layer, by assigning multiple provider-dependent aggregatable IPv6 prefixes to each site [15]. In general, striping at lower layers leads to a high striping point utilization, and striping at higher layers leads to less head-of-line blocking [7], i.e., a situation where a multihomed connection throughput is limited by that of its slowest path. To justify the case for session layer striping, we describe the advantages and drawbacks of striping at different layers.

Link layer striping. Link layer striping aggregates available physical links into a single communication path. Transmissions are done on a byte-by-byte basis over the physical interfaces, which improves the utilization of the links. Byte ordering must be preserved in link layer striping, and padding may be necessary when the number of bytes in a datagram is not a multiple of the number of interfaces. Thus, there may be significant overhead in striping at the link layer. In addition, IP datagrams may need to be reconstructed before crossing network boundaries, which makes link layer striping only truly useful for local area communications.

Network layer striping. Even though network layer striping should make multihoming transparent to the transport and higher layers, network layer striping causes poor TCP performance over heterogeneous paths [16]. While performance can be improved by modifying TCP retransmission timers and window sizes, such modifications essentially require changes at the transport layer, which makes network layer striping a relatively unattractive option.

Transport layer striping. Striping IP packets at the transport layer requires a transport protocol that, unlike TCP, can control multiple paths simultaneously. The SCTP protocol [17], for instance, can handle multiple data streams across multiple interfaces. However, SCTP only uses more than one interface in case of failure of the "primary" interface. More recent work on SCTP [10] investigates how one can use multihoming for concurrent transfers. However, the application remains bound to the SCTP semantics.

A few transport protocols, such as pTCP [6], are explicitly designed with transport layer striping in mind. pTCP stripes a connection over a set of (modified) TCP connections (one per interface), and can achieve high throughput aggregation. While pTCP is an excellent option for transport layer striping, the disadvantage of transport layer striping is to impose a specific set of transport layer semantics. The semantics, which, in pTCP's case, are close to those of TCP, may be unsuitable for some applications such as media streaming.

Application layer striping. As applications know the characteristics of the data being transferred, application layer striping can theoretically provide fine-grained performance tuning. For instance, an application may achieve high bandwidth aggregation by sending data via multiple sockets on multiple interfaces [18]. However, due to head-of-the-line blocking phenomena at the transport layer [6], application layer striping can also result in a throughput well below the capacity of the *slowest* path when (1) in-order packet delivery is a must, and (2) the underlying physical paths have a wide range of delay-bandwidth products.

In summary, the main advantage of striping below the application layer is to provide a single virtual "pipe" to the applications, whose performance can then be improved by taking advantage of multihoming without having to explicitly consider multiple underlying physical interfaces. At the same time, striping above the transport layer allows for exploiting the semantics of the transport protocol most suitable for a given application. In other words, a solution that appears like the best of both worlds is to "squeeze in" the striping primitives between the transport and application layers, effectively resurrecting the session layer for striping.

We note that previous proposals, such as the Block Extensible Exchange Protocol (BEEP, [11]), or the Mobile Access Router [12], have investigated related session-layer based architectures. Different from these previous works, we make the case that using a session layer allows for supporting any underlying transport protocol, over any number of channels, even if there is a mismatch in the number of underlying channels between both ends of a connection.

III. A strawman architecture for session layer striping

Because the primary goal of session layer striping is to improve application performance, applications should be able to inform the session layer of their needs (e.g., reliability, throughput maximization, ...). The session layer should accordingly determine which transport layer connections to set up, and how to stripe the data over the different transport layer connections to best meet the performance requirements of the application. The objective of this section is to sketch our vision for an architecture that meets these goals.

We provide a very high-level overview of a session layer striping architecture in Figure 1, for a host with n network interfaces, running k networked applications. Each application starts a session by providing the session layer with a set of

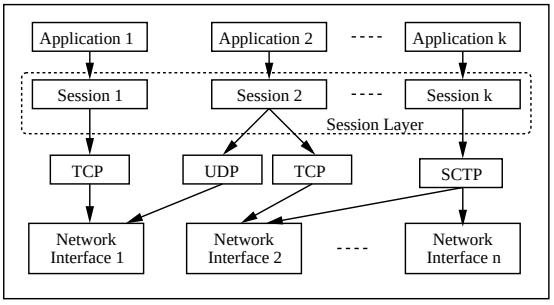

Fig. 1. **Session layer striping architecture.**

preferences. At the session layer, each session in turn selects the appropriate transport protocols and network interfaces to initiate the connection(s). The session layer can act as a mere pass-through filter for a transport layer connection (e.g., Session 1 opens a single TCP connection on interface 1), start multiple transport-level flows on several interfaces (e.g., Session 2 starts an UDP flow on interface 1 and a TCP flow on interface 2), or even rely on a transport protocol that supports multihoming (e.g., SCTP flow over interfaces 2 and n).

In the remainder of this section, we elaborate on the session layer semantics we envision, before discussing how the semantics are realized through transport layer connection management, and sketching a possible implementation and API. We stress that rather than a complete solution, our (significantly more modest) goal is to propose a strawman architecture and outline the objectives for which we believe a session layer striping protocol should strive.

A. Session layer semantics

At the very minimum, the session layer must be able to generalize to multihomed connections the reliability semantics that the transport layer offers to single-homed connections. That is, the session layer should provide a choice between: (1) no guarantees on losses or ordering, (2) lossless delivery without guarantees on ordering, (3) in-order delivery without guarantees on losses, and (4) in-order and lossless delivery. In addition to (1) and (4), which mirror possible transport layer semantics for single-homed connections, (2) can be useful for applications, such as file transfer, that can accommodate out-of-order delivery as long as they can reconstruct packet ordering at the application layer, while (3) may benefit media streaming applications.

Clearly, the above semantics do not ensure that multihoming results in application performance improvement. Thus, in addition, the session layer should select how to stripe traffic over the different (transport layer) connections to meet one or more of the following objectives, averaged over the length of the session: (1) throughput maximization, (2) latency minimization, (3) jitter (i.e., latency variations) minimization, and (4) loss minimization. Note that trying to minimize losses only applies when the session layer is requested to provide unreliable delivery.

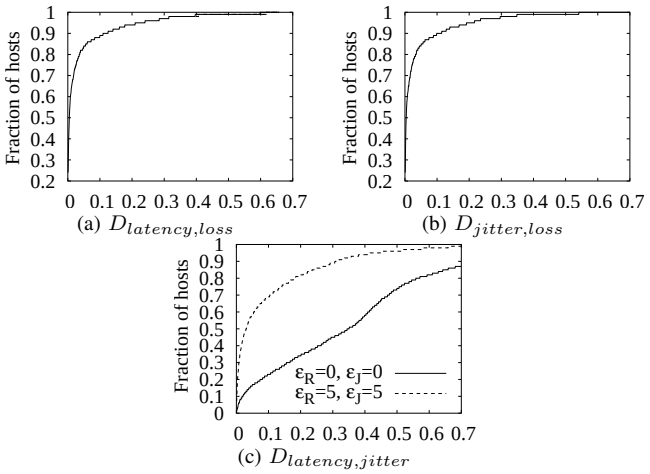

Fig. 2. **Discordance ratios.** Using loss rates (a) as an indicator of end-to-end latency and (b) as an indicator of jitter performance fails only 10% of the time for more than 90% of the hosts we probed. (c) indicates that using a tolerance as small as 5 ms in the optimization greatly reduces the conflicts between end-to-end latency and jitter.

Conflicting semantics. One may wonder what the session layer should do when it is asked to fulfill conflicting objectives. Consider the example of an application that asks the session layer to both minimize packet losses and jitter at the same time. It is conceivable that using a given interface consistently provides a lower loss rate than using the other interfaces, if the interface leads to network paths that traverse routers with larger buffers. At the same time, large router buffers may result in larger variations in queue sizes, which cause the applications to experience larger end-to-end jitter. In such a case, it is unclear which performance metric the session layer should favor.

We attempt to grasp the potential magnitude of the problem by running a very simple set of end-to-end measurements from a dual-homed residential host to over 100,000 remote hosts. For each measurement interval, and for each remote host, we measure the latency, jitter and loss rates experienced between each interface and the remote host. We define the discordance ratio between two metrics as the fraction of time during which it is impossible to select a single interface to optimize for both metrics simultaneously.

In Figure 2 we plot the cumulative probability distribution of the discordance ratios between latency and loss, jitter and loss, and latency and jitter, and we observe that, in an overwhelming number of cases, conflicts resulting from trying to optimize for different metrics can be avoided. Even when the outcome is bleaker, as is the case in Figure 2(c), we see that if we omit optimizations that result in improvements of less than 5 ms in either end-to-end latency or jitter, conflicts remain relatively rare.

This experiment is very simple (a single source host was used), and, as such, by no means provides a definitive answer to the potential problem of conflicting semantics. However, our measurements hint that different semantics are so unlikely to

conflict, that, at this point in our design, we simply advocate to use a static priority order; for instance, if an application wants to optimize for both loss and throughput, loss should be given precedence.

Fairness. Because the operating system (through rate-limiting of individual connections), or the network (through enforcement of TCP-friendliness [19]) can punish flows that are unfair to other flows, the session layer must, in addition to reliability and performance semantics, provide an interface to specify the fairness objectives of the application. Complementary to transport layer congestion control on a single stripe (e.g., [20]), the session layer has to be able to distribute traffic fairly over multiple stripes if so desired. Examples of flow/congestion control mechanisms over multiple links can be found in [6], [16], and could be adapted to a session layer architecture. Likewise, the techniques used in the Congestion Manager [9], or in TCP control block sharing [13], for improving performance over multiple concurrent connections at a single-homed host may form an interesting basis for session-layer multihoming support.

Finally, given the difficulty of designing session layer primitives general enough to satisfy the needs of all applications, we must ensure that means of partially or completely bypassing the session layer exist, even though all layering principles oppose such "overriding mechanisms." For instance, an application may want to rely on TCP connections exclusively, and the session layer should not prevent it from doing so.

B. Connection establishment and management

The session layer implements the semantics discussed above by managing transport layer connections through (1) connection establishment, (2) path evaluation, (3) connection management, and (4) data delivery.

Connection establishment. An application wishing to establish a connection to a remote host sends a message to the session layer instructing it of its desired semantics, and of the destination of the connection (specified by an IP address, and a port number). Using a reliable transport protocol, the session layer accordingly sends a message to the remote host advertising the IP addresses of the m interfaces that it will be using, and its performance objectives, including a proposed list of transport protocols it can use to meet the desired reliability semantics: For instance, when no guarantees on losses are requested, the session layer only needs to demultiplex an application layer flow over multiple unreliable (e.g., UDP) flows. Alternatively, the session layer can use reliable transport-level flows to implement lossless semantics.

If the session semantics match those desired by the remote host, the remote session replies with a list of IP addresses of its own n interfaces, and the list of transport protocols that will be used. Subsequently, each session can establish up to $n \times m$ transport layer connections (or paths) to the remote session.

We note the implementation of the connection establishment primitive presents several challenges. In particular, we need to ensure backwards compatibility with single-homed hosts; that is, we need to have a mechanism that determines whether session layer support is available on the remote end. This could be solved by simply sending an initial "HELLO" packet to the remote destination and check whether any reply is coming back. In addition, we need to guarantee IP addresses and connection properties are securely exchanged, to prevent connection hijacking attacks. While discussing the specifics of a handshake mechanism are beyond the scope of this paper, we believe that a mechanism inspired by the SSL/TLS handshake protocol [21] would be suitable. Such a handshake mechanism comes with the added benefit of allowing a cryptographic secret (e.g., symmetric key) to be shared, which allows the subsequent connection is to be encrypted if so desired. The Block Extensible Exchange Protocol uses similar mechanisms.

Path evaluation. Once transport layer paths have been established, the session layer periodically evaluates the service received on each path by assessing network metrics such as round-trip delay, achievable throughput, packet loss, or fraction of link overlap between different paths. Some of these metrics may be evaluated by piggybacking on the packets used for session establishment (e.g., round-trip time evaluation through SYN-ACK measurements for TCP traffic [22]), or can be estimated using transport layer variables (e.g., the congestion window size in TCP gives an indication of the achievable throughput). Alternatively, the session layer may resort to active measurements such as packet-pair dispersion, or even use application feedback regarding the service experienced at the application layer. In an effort to reduce the overhead associated with path evaluation for short-lived connections such as HTTP, the session layer could use a "scoreboard" of recently used paths across all sessions.

Connection management. Selecting an appropriate transport layer protocol can guarantee losslessness and packet ordering across a single stripe. To maintain packet ordering across all stripes, as required by in-order delivery semantics, we need to introduce sequence numbers at the session layer, and reorder packets coming from different interfaces according to their session sequence number, before passing them to the application layer. Thus, the session layer needs to insert a session header in each packet. The session header should at least contain the IP address of the interface used and the session sequence number.

In addition to implementing reordering primitives, the session layer should continuously monitor the state of the underlying transport layer connections. In particular, stateful protocols such as TCP may require peeking into state variables such as the congestion window size to ensure that no head-of-the-line blocking phenomena occur at the transport layer. When the session layer suspects head-of-the-line blocking is occurring at the transport layer, a simple solution is to close the faulty transport layer connection and reopen a new connection on a different path.

Data delivery. The choice of a specific scheduling algorithm to govern how data is sent over the different transport layer

Function	Parameters	Purpose
session_socket	Desired semantics	Create comm. endpoint
session_bind	Session descriptor, Port number	Listen to a local port
session_connect	Session descriptor, remote address and port number	Establish session with remote host
session_read	Session descriptor, blocking flag	Request data from session layer
session_write	Session descriptor, data chunk, blocking flag	Provide data to session layer
session_close	Session descriptor	Terminate session

TABLE I

Example of session layer API.

connections is likely to depend on the desired performance semantics. For the strawman architecture discussed in this paper, we do not advocate a specific (set of) scheduling algorithm(s). We however conjecture that a number of algorithms proposed in the context of adaptive scheduling for service differentiation in the Internet (see [23] for a survey) may apply to session layer scheduling with minor modifications. For instance, to maximize throughput, one could envision a modified deficit-round-robin algorithm [24], where quanta are a function of the achievable throughput on each path.

C. Toward a session layer implementation

For the implementation of our proposed session layer primitives, we have to determine where we implement the session layer primitives (i.e., kernel vs. user space), and to specify the interface (i.e., API) we provide to applications.

User vs. kernel space. A user-space implementation of session layer striping seems more appealing from a deployability perspective, but does not provide access to transport layer state variables, which, as described in Section III-B, may be necessary to guarantee acceptable performance. Hence, we advocate to implement our session layer primitives in kernel space, and more precisely, to implement session layer services (i.e., striping) in a kernel daemon.

API specification. We propose an API between the session layer daemon and applications that is highly inspired by the BSD sockets interface. The API we describe is deliberately simple; indeed, we conjecture that applications interested in a very fine-grained control of multihomed interfaces are probably rare, and probably prefer to directly open connections at lower layers of the network stack.

The API we suggest consists of six functions: *session_socket*, *session_bind*, *session_connect*, *session_read*, *session_write*, and *session_close*. Table I summarizes the purpose of each function, and the parameters it takes as input. *session_socket* is used by an application to create a new session layer socket, and to specify the desired performance, reliability, and congestion control semantics. Similar to the *socket* call in BSD sockets, *session_socket* does not initiate any form of communication. To that effect, one uses *session_bind* or *session_connect*. *session_bind* binds an existing session layer socket to a local port number on *all* available interfaces; once a session layer socket is bound to a local port number, incoming connections to that port (on any interface) can be processed by the session layer. *session_connect* is used to connect a session layer socket to a remote session, and takes, as parameters, a local session socket and a remote address (i.e., IP and port number). If the remote session is multihomed, *session_connect* should be able to use any of the multihomed remote IP addresses. As soon as *session_connect* is called, connection establishment is performed as discussed in Section III-B. In other words, *session_connect* is used by hosts to inform each other of their multihoming capabilities, including the absence of any multihoming support, in the case of a single-homed host, and of the parameters of the multihomed session.

Once two sessions have established a communication channel, *session_read*, *session_write* and *session_close* implement the connection management and data delivery primitives we discussed in Section III-B, to write to/read from the session layer socket, in a blocking or non-blocking mode depending on the application preference. Last, *session_close* is used to close a session layer socket and terminate any (transport layer) connection still active within the session.

Last, we point out that the API presented here is only a minimal set of primitives. One could reasonably augment this API with functions allowing applications for providing some feedback to the session, for instance, primitives to convey the level of service experienced by the application back to the session layer.

IV. Discussion

We argued that decoupling striping primitives from all of the traditional layers of the networking protocol stack helps improve application performance for multihomed hosts. We illustrated our case by showing how a session layer striping architecture significantly enhances the connection semantics offered to applications, without requiring drastic changes in application code or transport-layer implementations.

The optimist in us is tempted to argue that, considering the significant service improvements multihoming can provide [1], [2], [3], [4], [5], a deployable striping architecture may finally offer a practical solution to address the absence of quality-of-service support from the network.

The pessimist in us, however, acknowledges that deployment difficulties subside. For instance, even though our session layer architecture only imposes minor rewrites of the communication primitives used at the application layer, the associated deployment cost is not inexistent. However, considering that "zero-cost alternatives," such as intercepting all socket system calls in the kernel to exploit multiple interfaces, only provide marginal performance improvements over single-homed connections, we believe that our solution provides a good compromise between limited deployment cost and potentially significant performance improvement. In addition, session-layer striping may drive forward multihoming support by application developers, thereby giving end users stronger economic incentives to invest in multihoming.

ACKNOWLEDGMENTS

This work is supported in part by the National Science Foundation under grant number ANI-0085879 and ANI-0331659. We thank the anonymous reviewers for a number of insightful suggestions that greatly improved the paper.

REFERENCES

[1] A. Akella, B. Maggs, S. Seshan, A. Shaikh, and R. Sitaraman, "A measurement-based analysis of multihoming," in *Proceedings of ACM SIGCOMM'03*, Karlsruhe, Germany, Aug. 2003, pp. 353–364.

[2] A. Akella, S. Seshan, and A. Shaikh, "Multihoming performance benefits: An experimental evaluation of practical enterprise strategies," in *Proceedings of USENIX'04 Annual Technical Conference*, Boston, MA, June 2004, pp. 113–126.

[3] A. Habib and J. Chuang, "On the effectiveness of residential multihoming," Tech. Rep. TR-SIMS-HC-1104, School of Information Management and Systems, University of California, Berkeley, Nov. 2004, `http://p2pecon.berkeley.edu/pub/TR-SIMS-HC-1104.pdf`.

[4] A. Akella, J. Pang, B. Maggs, S. Seshan, and A. Shaikh, "A comparison of overlay routing and multihoming route control," in *Proceedings of ACM SIGCOMM'04*, Portland, OR, Aug. 2004, pp. 93–106.

[5] D. Goldenberg, L. Qiu, H. Xie, Y. R. Yang, and Y. Zhang, "Optimizing cost and performance for multihoming," in *Proceedings of ACM SIGCOMM'04*, Portland, OR, Aug. 2004, pp. 79–92.

[6] H.-Y. Hsieh and R. Sivakumar, "A transport layer approach for achieving aggregate bandwidths on multihomed mobile hosts," in *Proceedings of ACM MOBICOM'02*, Atlanta, GA, Sept. 2002, pp. 83–94.

[7] B. Traw and J. Smith, "Striping within the network subsystem," *IEEE Network*, vol. 9, no. 4, pp. 22–32, July 1995.

[8] A. Dhamdere and C. Dovrolis, "ISP and egress path selection for multihomed networks," in *Proceedings of IEEE INFOCOM 2006*, Barcelona, Spain, Apr. 2006, In press.

[9] H. Balakrishnan, H. Rahul, and S. Seshan, "An integrated congestion management architecture for Internet hosts," in *Proceedings of ACM SIGCOMM'99*, Boston, MA, Aug. 1999, pp. 175–187.

[10] J. Iyengar, K. Shah, P. Amer, and R. Stewart, "Concurrent multipath transfer using SCTP multihoming," in *Proceedings of SPECTS'04*, San Jose, CA, July 2004.

[11] M. Rose, "The blocks extensible exchange protocol core," IETF RFC 3080, March 2001.

[12] P. Rodriguez, R. Chakavorty, J. Chesterfield, I. Pratt, and S. Banerjee, "MAR: a commuter router infrastructure for the mobile Internet," in *Proceedings of ACM MobiSys'04*, Boston, MA, June 2004, pp. 217–230.

[13] J. Touch, "TCP control block interdependence," IETF RFC 2140, April 1997.

[14] H. Adiseshu, G. Parulkar, and G. Varghese, "A reliable and scalable striping protocol," in *Proceedings of ACM SIGCOMM'96*, Palo Alto, CA, Aug. 1996, pp. 131–141.

[15] C. de Launois, B. Quoitin, and O. Bonaventure, "Leveraging network performances with IPv6 multihoming and multiple provider-dependent aggregatable prefixes," in *Proceedings of QoS-IP 2005*, Catania, Italy, Feb. 2005.

[16] D. Phatak, T. Goff, and J. Plusquellic, "IP-in-IP tunneling to enable the simultaneous use of multiple IP interfaces for network level connection striping," *Computer Networks*, vol. 43, no. 6, pp. 787–804, Dec. 2003.

[17] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, and V. Paxson, "Stream control transmission protocol," IETF RFC 2960, October 2000.

[18] H. Sivakumar, S. Bailey, and R. Grossman, "PSockets: The case for application-level network striping for data intensive applications using high speed wide area networks," in *Proceedings of IEEE SC 2000*, Dallas, TX, Nov. 2000.

[19] S. Floyd and K. Fall, "Promoting the use of end-to-end congestion control in the Internet," *IEEE/ACM Transactions on Networking*, vol. 7, no. 4, pp. 458–472, Aug. 1999.

[20] M. Allman, V. Paxson, and W. Stevens, "TCP congestion control," IETF RFC 2581, April 1999.

[21] T. Dierks and C. Allen, "The TLS protocol v1.0," IETF RFC 2246, January 1999.

[22] H. Jiang and C. Dovrolis, "Passive estimation of TCP round-trip times," *ACM Computer Communication Review*, pp. 75–88, July 2002.

[23] V. Firoiu, J.-Y. Le Boudec, D. Towsley, and Z.-L. Zhang, "Theories and models for Internet quality of service," *Proceedings of the IEEE, Special Issue on Internet Technology*, vol. 90, no. 9, pp. 1565–1591, Aug. 2002.

[24] M. Shreedhar and G. Varghese, "Efficient fair queueing using deficit round-robin," *IEEE/ACM Transactions on Networking*, vol. 4, no. 3, pp. 375–385, June 1996.