Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE DIRECT®
Computer Networks 43 (2003) 787–804
COMPUTER NETWORKS
www.elsevier.com/locate/comnet

IP-in-IP tunneling to enable the simultaneous use of multiple IP interfaces for network level connection striping ☆

Dhananjay S. Phatak *, Tom Goff, Jim Plusquellic

Department of Computer Science and Electrical Engineering, University of Maryland Baltimore County (UMBC), 1000 Hilltop Circle, Baltimore, MD 21250, USA

Received 21 March 2003; accepted 26 March 2003

Responsible Editor: I.F. Akyildiz

Abstract

With ubiquitous computing and network access now a reality, multiple network conduits are become widely available to mobile as well as static hosts: for instance wired connections, 802.11 style wireless LANs, Bluetooth, and cellular phone modems. Selection of the preferred mode of data transfer is a dynamic optimization problem which depends on the type of application, its bandwidth/latency/jitter requirements, current network conditions (such as congestion or traffic patterns), cost, power consumption, battery life, and so on. Furthermore, since wireless bandwidth is likely to remain a scarce resource, we foresee scenarios wherein mobile hosts will require simultaneous data transfer across multiple IP interfaces to obtain higher overall bandwidth.

We present a brief overview of existing work which enables the simultaneous use of multiple network interfaces and identify the applicability as well as strengths and weaknesses of these related approaches. We then propose a new mechanism to aggregate the bandwidth of multiple IP paths by splitting a data flow across multiple network interfaces at the IP level. We have analyzed the performance characteristics of our aggregation scheme and demonstrate significant gains when the network paths being aggregated have similar bandwidth and latency characteristics. In addition, our method is transparent to transport (TCP/UDP) and higher layers, and allows the use of multiple network interfaces to enhance reliability. Our analysis identifies the conditions under which the proposed scheme, or any other scheme that stripes a single TCP connection across multiple IP paths, can be used to increase throughput.

Keywords: Network protocols; IP tunneling; Connection striping; Bandwidth aggregation; Multihomed hosts

1. Introduction

As wireless networks, services, and computing continue their explosive growth it is clear that multiple network transport mechanisms will become available to hosts. For instance, a mobile host might have access to the Internet via multiple networking technologies such as Bluetooth, wireless

☆ This work was supported in part by Aether Systems Inc. and NSF grants ECS-9875705 and ECS-0196362. Portions of this work appeared in "D.S. Phatak, T. Goff, A novel mechanism for data streaming across multiple IP links for improving throughput and reliability in mobile environments, in: IEEE INFOCOM, June 2002, pp. 773–781".

* Corresponding author.

E-mail addresses: phatak@umbc.edu (D.S. Phatak), tgoff1@umbc.edu (T. Goff), plusquel@umbc.edu (J. Plusquellic).

1389-1286/$ - see front matter © 2003 Elsevier B.V. All rights reserved.
doi:10.1016/S1389-1286(03)00319-0

LANs (802.11, Airport LANs, Ricochet, etc.), and cellular phone modems where each technology has a corresponding service provider. Selecting which service to use for data transfer is a dynamic optimization problem which should track time varying attributes which might include bandwidth/throughput, latency, jitter, quality of service (QoS) requirements, cost, power consumption, residual battery life, interference, and traffic patterns. Also, wireless bandwidth is a scarce resource which is unlikely to improve at the same pace as the rapid growth in bandwidth available via wired networks. Hence it is probable that mobile users will want to simultaneously stream data across multiple network interfaces in order to best make use of all available bandwidth. Therefore, this paper investigates the simultaneous use of multiple network interfaces to enhance the overall bandwidth available to a wireless node. An immediate secondary advantage is increased reliability, for example if one network interface goes down or one network path becomes severely congested the end-to-end connection is not interrupted.

In some cases, such as military applications, multiple identical or very similar network access mechanisms are provided for the sake of reliability in hostile environments. Although though the primary motivation for providing redundant network access is reliability, it would be desirable to stream distinct data packets across all interfaces simultaneously whenever possible. This would be a "performance-enhancement" mode. When reliability becomes an issue, the data transfer could switch to a "reliability first" mode, where two or more identical copies of the data stream might be sent across multiple access points in the hope that at least one reaches its destination. Ideally there would be no hard cutoff between the performance-enhancement and reliability first modes. In general a mixture could exist, for example only duplicate lost packets for transfer across multiple interfaces. Furthermore, the transition between performance-enhancement and reliability first modes should happen dynamically in reaction to the environment.

1.1. Problem definition

Fig. 1 illustrates the case when hosts A and B want to communicate via the Internet and both have multiple transport conduits. For instance, host A might be a laptop at an airport with access to the Internet via an 802.11 LAN (say interface A_1), a Bluetooth scatternet (A_2), and a cellular phone modem (A_3). In general, the IP addresses assigned to interfaces A_1, A_2, and A_3 will be controlled by separate Internet service providers (ISPs) who might in turn implement firewalling to various degrees.

Suppose A wants to stream data to B across multiple interfaces to enhance the overall bandwidth. Several questions arise when considering the possible ways to achieve this effect. Should the data stream be split into multiple streams at the application level? In this case the application would open multiple connections across different network interfaces and be responsible for splitting the data stream at the server and merging it properly at the client. This approach might be too cumbersome and restrictive since the number of connections might have to be decided in advance in order to figure out how many flows to split the traffic into. Alternatively, should the splitting be done at transport layer or at the network layer? In essence we are considering this general question of

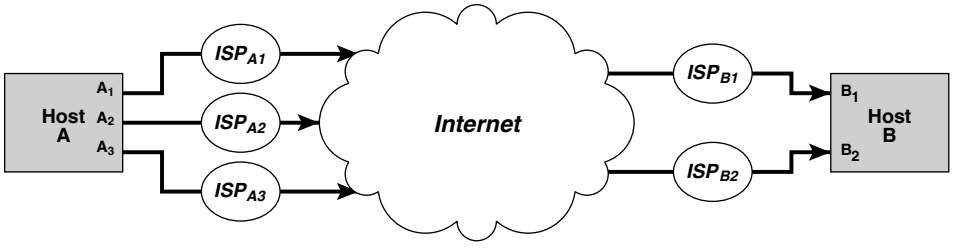

Fig. 1. Communicating hosts with multiple network connections (multi-homed).

striping a connection across multiple IP interfaces for bandwidth aggregation.

1.2. Related work

Bandwidth aggregation across multiple channels has been considered in the literature in varying contexts and for different applications and scenarios, distinct from those considered in this paper. For instance, multi-link PPP [1,2] is designed to aggregate multiple *logical* data channels. Since PPP was intended to operate at the data link layer, below the IP level [3–5], the multi-link PPP extension bundles multiple data-link level channels into a single logical link. In the scenario we are considering, links have different IP addresses assigned and controlled by completely independent ISPs. It is highly unlikely that independent ISPs will allow arbitrary users to bundle their links into "one logical link". Thus, PPP would have to run on top of the IP layer which is not its intended use, and has its own set of difficulties.

Multi-path and QoS routing have been considered in wired as well as wireless networks, a sampling can be found in [6–15]. Multi-path routing in wired networks concentrates mainly on the network layer. The issue of splitting a single connection into multiple streams is not addressed. Multi-path and QoS routing work in mobile ad hoc networks also focuses on routing at the network layer. Here the assumption is that there is a single radio interface which is used to find distinct routes to a destination via multiple neighbors within listening range of the source. This is different from considering multiple interfaces and striping the same connection across those interfaces. We would like to point out that the issues addressed in multi-path QoS routing are relevant to the problem at hand: it would be good to use disjoint shortest path pairs, incorporate QoS attributes, etc. Fundamentally, however, these issues are different from the problem of efficiently striping a single connection across multiple IP paths. Thus the multi-path QoS routing results found in the literature can be used in conjunction with a connection-striping scheme.

Bandwidth aggregation has also been considered in the context of storage systems and high volume data or file servers, for instance [16–19]. Typical storage-server architectures are confined to within a LAN having a single controlling authority. Note that within a LAN it is possible to do bandwidth aggregation at the MAC layer. For instance, Cisco's EtherChannel [20] provides bandwidth aggregation across multiple Ethernet links. This product targets the extremely high instantaneous bandwidth requirements between a data-server and nodes which service queries in typical database applications. Likewise, a recent IETF draft from the HP storage systems group [19] deals with exploiting ultra wide-band SCSI connections for distributed storage. They simply mention the problems associated with trying to aggregate multiple IP paths into a single TCP connection; they do not propose a solution.

The recently proposed stream control transmission protocol (SCTP, RFC 2960, and [21–25]) comes closest to solving the problem at hand. SCTP is a new transport level protocol which supports multi-streaming and multi-homing (a host having multiple IP addresses). A recently proposed extension [26] would allow the dynamic addition/deletion of IP addresses at both the source and destination. However, in its current form SCTP does not perform load-sharing, that is multi-homing is used only for redundancy [24] as follows. A single IP address is chosen as the primary address for a connection, meaning the destination to which all normally transmitted data chunks are sent. Retransmitted data chunks may use an alternate destination address to improve the probability of reaching a remote endpoint. Persistent send failures to a primary address will ultimately result in choosing a new primary address for the connection.

To the best of our knowledge bandwidth aggregation across multiple IP paths has not been considered in the context of mobile/wireless networks where we believe it will be a real issue. In the following sections we propose a new mechanism for bandwidth aggregation across multiple IP paths and analyze its performance.

2. Proposed aggregation mechanism

We propose a bandwidth aggregation solution at the network (IP) layer. The idea is to manipulate

the routing entities (daemons, tables, etc.) so as to send packets through different network interfaces. This approach has the following advantages:

(i) To transport, application, and higher layers the data stream looks like a single flow. Therefore all existing applications can benefit transparently without being rewritten to explicitly do stream splitting themselves.
(ii) This approach is independent of underlying network technologies, e.g. 802.11, Bluetooth. The prevalence of TCP/IP makes IP support likely for most common network technologies. Note that even if the actual network layer protocol is not IP, we propose implementing aggregation at the network level.
(iii) The proposed approach can be highly dynamic. If the route through an interface becomes congested packets can simply be sent through an alternate interface. In the discussion that follows we show some problems that can arise from this approach.
(iv) Reliability can be enhanced. If one interface goes down an alternate interface can be used.
(v) Likewise, once such a mechanism is available it could be employed to optimize parameters other than reliability, such as power consumption, latency, jitter, etc.

Since TCP is used more predominantly than UDP from inter-LAN communication, we illustrate our method for TCP (although it can also be used for UDP flows). For the purpose of illustration, assume that node A is the source and node B is the destination, as shown in Fig. 1. The IP address associated with interface A_i is denoted $\mathrm{Addr}(A_i)$, and the IP address associated with interface B_i is denoted $\mathrm{Addr}(B_i)$. Assume that A only knows how to reach B using the IP address associated with interface B_1. For now, also assume that A will only send data to B using interface B_1, the more general case where both A and B use multiple interfaces will be considered later.

For the scenario described above, the initial SYN packet for the TCP connection from A to B goes through interfaces A_1 and B_1. The TCP stack at host B notes that the source address of the connection is associated with interface A_1, i.e. $\mathrm{Addr}(A_1)$. Likewise the connection identifier at A has $\mathrm{Addr}(B_1)$ as the destination address for the connection. Now suppose the application at A requires more bandwidth and packets from the single transport layer connection with B are simultaneously routed through an alternate network interface, say interface A_2. This leads to the following problems:

(i) Typically the source address of a packet is the address associated with the interface through which it is transmitted. Therefore, packets from a connection that was established with source address $\mathrm{Addr}(A_1)$ will contain the unrecognized source address $\mathrm{Addr}(A_2)$ when sent through A_2. When the packet reaches its destination, B, it is not recognized as belonging to an established connection since a TCP connection is uniquely identified by the 4-tuple (source IP address, source port, destination IP address, destination port). The packet is then dropped by B.
(ii) Alternatively, suppose the source address $\mathrm{Addr}(A_1)$ is assigned to packets sent out through A_2. As a side effect, these packets now appear spoofed to intermediate routers. For security reasons spoofed packets are almost certainly dropped by ISPs. Therefore packets sent through A_2 still fail to reach the destination application at B.

Our solution to these problems is to employ *tunneling*. At the sender side, A, the transport layer assembles all packets as if they were going through A_1 and addressed to B_1. The packets going out interface A_2 then get encapsulated in a new IP packet with an extra header having destination address B_1 and source address A_2. The protocol field for the outer header should be IP-in-IP (also used for tunneling in the mobile IP standard and implementations [27–30]). The destination, B, can then recognize IP-in-IP packets and strip the outer header. This leaves the original packet with proper source and destination addresses to be delivered up the network stack to TCP in a transparent manner. The TCP stack at B is unaware that A tunneled the packet through an alternate interface.

Note that the same tunneling mechanism can be used when B is accepting data on multiple interfaces. For instance, to send a packet through interfaces A_3 and B_2, the original packet (having source and destination addresses $\mathrm{Addr}(A_1)$ and $\mathrm{Addr}(B_1)$ respectively) is encapsulated in a packet having source address $\mathrm{Addr}(A_3)$ and destination address $\mathrm{Addr}(B_2)$. The receiving IP stack at B strips the outer header, realizes that the inside packet is for an existing TCP connection, and transparently passes it to TCP.

While this scheme presents a feasible solution, it raises the following issues:

(i) How does a source learn multiple IP addresses for a destination and vice versa?
(ii) Many radio technologies operate in the same frequency band, for example Bluetooth and 802.11. Using Bluetooth and 802.11 simultaneously might lead to interference between the two, causing degraded performance.
(iii) The IP-in-IP encapsulation adds some processing overhead.
(iv) TCP performance could be adversely effected. TCP was designed to work well across a single network path, not multiple paths. If the bandwidth/delay characteristics of two paths are substantially different packets sent on the lower bandwidth path will cause timeouts and/or fast retransmits. This will cause the sender to throttle its transmission rate via congestion window reduction, slow start, and other congestion avoidance mechanisms employed by TCP. The net effect could be that the lower bandwidth path "drags down" the higher bandwidth path. This would defeat the original purpose of using multiple interfaces simultaneously, in which case using only the higher bandwidth path would be optimal.

We address each of the above issues in turn. The first three are discussed in the remainder of this section while the following section considers the forth issue.

The first issue is how to make A and B aware of any alternate IP addresses the other may have. To accomplish this, IP and/or TCP headers and their processing would have to be modified. This is similar to the solution found in SCTP which allows connection initialization chunks to list multiple addresses. While such modifications might interfere with interoperability, we believe that adding addition header fields or utilizing unused fields is a fairly transparent solution.

The second question, as to whether using multiple interfaces is tantamount to creating self interference, for example when the same wireless spectrum is used, is beyond the scope of this work and highly technology specific. While Bluetooth and 802.11 occupy the same frequency band, cellular phone modems and the projected wideband CDMA do not. In the latter case the radio interference issue is non-existent. With the rapid growth in wireless technologies and services, various transport conduits are unlikely to use the same technology and/or spectrum. In addition, since our solution is at the IP level it can be applied to wired scenarios as well. A typical desktop at a campus today may have a wired Ethernet connection as well as an 802.11 wireless LAN connection, where the wireless cell data rate might be 11 Mbps or higher. The proposed connection striping scheme could be applied in this case, with some packets being sent across the wired interface and some across the wireless link. In this case there is obviously no interference between the two interfaces.

The third issue deals with the IP-in-IP overhead. Experimental data shows that this overhead is negligible (see Section 5). Note that packets sent on the primary path, i.e. the path over which the TCP SYN packet was sent that established the source and destination addresses, need not be tunneled and do not incur the IP-in-IP encapsulation overhead. Assuming that the primary path is the highest bandwidth path, most of the packets are then free from the IP-in-IP overhead. Furthermore, without tunneling the lower bandwidth path(s) are not used at all. By employing the proposed scheme, one can hope to utilize at least some part of the spare capacity of the lower bandwidth path(s). Thus any added bandwidth utilization is available for free, meaning it is simply not realizable by the conventional single path TCP connection, in this sense the term "overhead" is a misnomer. In addition, the minimal encapsulation proposed in the mobile IP standard [29] along with

header compression techniques can be employed to mitigate any IP-in-IP overhead.

Now only the question of TCP performance remains. This is considered in detail in Section 4, but first a quantitative analysis of data splitting in general is presented.

3. Splitting a data stream

Splitting a single transport layer (TCP) stream into multiple network layer (IP) streams can be accomplished in many ways. In this section we present an analytical model of a network path and find how data should be divided between paths to best make use of available network resources in order to maximize overall throughput. However, an exact packet scheduling algorithm is not considered since many specific methods for packet scheduling could be used, each having various tradeoffs, the study of which is beyond the current scope and a matter for future work.

3.1. Analytical path model

Consider two network nodes, node A and node B, where node A is sending data to node B. Assume there are N distinct paths from A to B, where the time needed to send a packet along path i is a linear function of packet size. That is, the one-way latency of the kth packet of size p_k, sent from A to B along path i at time t is

$$l_{AB_i}(p_k,t) = \frac{p_k}{b_{AB_i}(t)} + c_{AB_i}(t). \tag{1}$$

Given this model, $b_{AB_i}(t)$ and $c_{AB_i}(t)$ correspond to the effective bandwidth and propagation delay, respectively, of the ith path from A to B at time t. From this one-way latency, the round-trip time (RTT) of the kth packet becomes

$$r(p_k,q_k,t) = l_{AB_i}(p_k,t) + \delta_k + l_{BA_{i'}}(q_k, t + l_{AB_i}(p_k,t) + \delta_k), \tag{2}$$

where δ_k is the delay between when packet k is received and when the corresponding ACK is sent, and q_k is the size of the packet sent from B to A containing the ACK.

The following simplifying assumptions are made to make the ensuing analysis more manageable:

- The effective bandwidth of a path is constant throughout the lifetime of a TCP connection, this allows the time dependency to be dropped from (1) and (2).
- All data packets sent along path i have the same size, p_i.
- There is no delay between when a packet is received and when the corresponding ACK is sent, that is $\delta_k = 0$ in (2).
- All ACKs are sent back on the reverse path through which the TCP SYN packet was sent. We also assume that ACK packets are much smaller than data packets. Since all ACKs are sent back on the same path, the latency of transmission for all ACKs can be assumed to be the same, Δ_{ACK}.

These assumptions simplify (2) and the RTT of a packet sent along path i becomes

$$r_i = l_i(p_i) + \Delta_{\mathrm{ACK}} = \frac{p_i}{b_i} + c_i + \Delta_{\mathrm{ACK}}. \tag{3}$$

3.2. Optimal portion of data sent on each path

To fully utilize the available network resources, data should be sent on all paths between A and B throughout the life of a TCP connection. In other words, the time needed to send n_i packets each of size p_i along path i should equal the total connection time, leaving no path idle during the connection. This means

$$L_1(p_1,n_1) = L_2(p_2,n_2) = \cdots = L_N(p_N,n_N), \tag{4}$$

where $L_i(p_i,n_i)$ is the total transfer time for data sent on path i. Assuming packets are sent in a pipelined fashion, where multiple packets are simultaneously in flight, this total transfer time becomes

$$L_i(p_i,n_i) \approx \frac{p_i \cdot n_i}{b_i} + c_i. \tag{5}$$

The system of equations given in (4) then becomes

$$\frac{n_1 \cdot p_1}{b_1} + c_1 = \frac{n_2 \cdot p_2}{b_2} + c_2 = \cdots = \frac{n_N \cdot p_N}{b_N} + c_N. \tag{6}$$

Alternatively, this can be expressed in terms of the total amount of data, D, transferred from A to B where $D = \sum_i n_i p_i$. Noting that the fraction of total data sent along path i is

$$f_i = \frac{n_i \cdot p_i}{D}, \tag{7}$$

Eq. (6) becomes

$$\frac{D}{b_1} \cdot f_1 + c_1 = \frac{D}{b_2} \cdot f_2 + c_2 = \cdots = \frac{D}{b_N} \cdot f_N + c_N, \tag{8}$$

where

$$f_1 + f_2 + \cdots + f_N = 1. \tag{9}$$

The system of equations defined by (8) and (9) can be represented in matrix form as

$$\mathbf{Bf} = \mathbf{c}. \tag{10}$$

Here $\mathbf{B}$ is an $N \times N$ matrix whose elements are defined by

$$\mathbf{B}_{ij} = \begin{cases} b_i^{-1} & \text{if } j = i \\ -b_{i+1}^{-1} & \text{if } j = i+1 \quad \text{for rows } i < N, \\ 0 & \text{otherwise} \\ \mathbf{B}_{ij} = 1 & \text{for row } i = N, \end{cases} \tag{11}$$

$\mathbf{f}$ is a column vector whose elements are the fraction of data sent along the corresponding path, or $\mathbf{f}_i = f_i$, and $\mathbf{c}$ is a column vector where

$$\mathbf{c}_i = \begin{cases} (c_{i+1} - c_i)/D & \text{if } i < N, \\ 1 & \text{if } i = N. \end{cases}$$

This makes (10)

$$\underbrace{\begin{bmatrix} b_1^{-1} & -b_2^{-1} & & & \\ & b_2^{-1} & -b_3^{-1} & & \\ & & \ddots & \ddots & \\ & & & b_{N-1}^{-1} & -b_N^{-1} \\ 1 & \cdots & \cdots & \cdots & 1 \end{bmatrix}}_{\mathbf{B}} \underbrace{\begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_{N-1} \\ f_N \end{bmatrix}}_{\mathbf{f}} = \underbrace{\begin{bmatrix} (c_2 - c_1)/D \\ (c_3 - c_2)/D \\ \vdots \\ (c_N - c_{N-1})/D \\ 1 \end{bmatrix}}_{\mathbf{c}}. \tag{12}$$

Due to its construction, this system of equations has a unique solution given by

$$\mathbf{f} = \mathbf{B}^{-1}\mathbf{c}. \tag{13}$$

For large D, the terms $(c_i - c_{i+1})/D$ approach zero, leading to the solution

$$f_i = \frac{b_i}{b_1 + b_2 + \cdots + b_N}. \tag{14}$$

In other words, the fraction of data sent along a given path corresponds to the ratio of that path's bandwidth to the total available bandwidth. This can be verified intuitively as follows.

For large D the transfer time, t, will be bandwidth dominated where from the pipelined transfer assumption

$$t = \frac{D}{b_{\text{total}}} \quad \text{and} \quad b_{\text{total}} = b_1 + b_2 + \cdots + b_N. \tag{15}$$

In this time the fraction of data sent along the ith path, D_i, is

$$D_i = b_i \cdot t = \frac{b_i \cdot D}{b_{\text{total}}}. \tag{16}$$

The fraction of total data sent along path i is then

$$f_i = \frac{D_i}{D} = \frac{b_i}{b_{\text{total}}}. \tag{17}$$

Therefore, for large data transfers, data should be partitioned across multiple paths according to (14) in order to minimize the overall transfer time.

4. TCP performance analysis

At the outset we would like to point out that the proposed aggregation mechanism will enhance UDP performance since UDP is connectionless and there are no congestion control mechanisms or retransmissions. Therefore, any asymmetry in bandwidths and/or latency between different network paths will impact the performance of UDP applications much less than TCP applications, making UDP the simpler case of the two. We have therefore concentrated most of our efforts on the more difficult and interesting case of splitting a TCP connection across multiple IP paths.

As previously mentioned, for TCP the use of multiple paths together at the network layer may lead to worse performance than when a single path is used alone. This can mainly be attributed to the two mechanisms listed below.

(i) The bandwidth/delay mismatch between paths causes packets sent on a lower capacity path to arrive after the sender-side TCP has timed out. In this case the late arriving packet needs to be retransmitted. If this were the only drawback one would expect to see only a slight degradation of performance when compared to using a higher capacity path alone. Unfortunately, with each timeout TCP drastically scales back its congestion window and invokes slow-start, thereby greatly under-utilizing the higher capacity path.
(ii) Most TCP implementations include the fast retransmit and fast recovery algorithms [31]. Even if the TCP timeout values were set high enough to prevent the scenario described above, fast retransmit still poses an independent problem. For the purpose of illustration, suppose two paths are being used and the ratio of the path RTTs is 4. In addition, assume the first packet is transmitted on the lower bandwidth path and the next four packets are transmitted on the higher capacity path. The receipt of packets 2, 3, and 4 at the destination will cause the fast retransmission of the first packet (by the receiver-side TCP sending three duplicate ACKs for packet 0), thereby wasting the prior transmission on the lower bandwidth path. Worse yet, the recovery phase following a fast retransmission scales back the congestion window.

Both of these issues are addressed in detail below. We first consider the prevention of timeouts. After a brief overview of the problem in the context of two paths we present an analytical model for N paths. We then derive expressions that show how much of a discrepancy in bandwidth can be tolerated without incurring timeouts. The issue of fast retransmissions is then examined where the number of duplicate acknowledgments that can typically be expected is expressed in terms of path bandwidth ratios.

4.1. TCP retransmission timeouts

For the sake of illustration, consider striping TCP packets across two paths: a high bandwidth path and a low bandwidth path. Timeouts will happen only if the RTTs for packets sent across the two paths are substantially different. Noting from (3) that the RTT can be split into two components, namely

$$\mathrm{RTT} = \frac{p}{b} + \tau, \tag{18}$$

where p is the packet size, b is the bandwidth of the path, and τ includes all other delays such as signal propagation delays of all hops, processing delays at all intermediate routers, all delays associated with acknowledgments (ACKs), etc.

If the RTTs for packets sent on all paths are dominated by their respective p/b ratios, then a bandwidth disparity between paths causes a corresponding disparity in RTTs. This in turn can lead to frequent timeouts degrading performance. On the other hand if the RTTs are dominated by τ, then a bandwidth disparity does not lead to a significant difference in RTT. This may be the case if the number of hops between the source and destination is sufficiently large, or if the transmission distance and hence propagation delay is sufficiently large. In such cases we expect to see performance gains when multiple network interfaces are used simultaneously.

As an example, consider two paths with bandwidths of 160 and 40 Kbps, giving a bandwidth ratio of 4:1. For full Ethernet sized packets (around 1500 bytes per packet) being sent a single hop between adjacent nodes the "other delays" in (18) are negligible, since τ is on the order of microseconds whereas the p/b ratios are hundreds of milliseconds. In this case the RTTs are dominated by the p/b ratios, implying that using the two paths together may be worse than using the higher capacity path by itself. Now consider two paths with the same 4:1 bandwidth ratio, but with actual bandwidths of 100 and 25 Mbps, and where the destination is several hops away. Now the p/b ratio for each path is on the order of hundreds of microseconds, whereas τ can be milliseconds. In this case the RTTs are dominated by τ implying

that using the two paths together can be expected to yield better overall performance than using the higher capacity path alone.

Note that reducing the packet size p will help mitigate the effect of the p/b ratio on the RTT, and hence allow a greater disparity in bandwidths. In fact the packets queued for transmission on the lower capacity path can be dynamically fragmented to equalize the p/b ratios of both paths. Next we present a quantitative analysis of the problem.

In current versions of TCP, the retransmission timeout (RTO) for the ith packet is set as

$$\mathrm{RTO}_i = R_{i-1} + K \cdot V_{i-1}, \tag{19}$$

where R_{i-1} and V_{i-1} are the current smoothed estimates of RTT and the deviation in RTT respectively (just prior to sending the ith packet), and K is a constant factor (typically $K = 4$). R_i and V_i are defined by the following recursive equations:

$$R_i = \alpha \cdot R_{i-1} + (1-\alpha)\mathrm{RTT}_i, \tag{20}$$

$$V_i = \beta \cdot V_{i-1} + (1-\beta)|\mathrm{RTT}_i - R_i|, \tag{21}$$

where RTT_i is the sampled RTT of the ith packet. Eqs. (20) and (21) act as a low-pass filter on the sampled RTT which smoothes transient variations.

The exact value of RTO will depend on the sequence of RTT samples. In our multiple path scenario, this will depend on the sequence of paths chosen to send packets on. As an approximation, the average RTT and average deviation in RTT will be considered, when packets are sent along paths according to (14).

From (7) and (14), the fraction of *packets* sent along path i is given by

$$\frac{n_i}{n_{\mathrm{total}}} = \frac{b_i/p_i}{\sum_{k=1}^{N} b_k/p_k}. \tag{22}$$

Assuming that the RTTs for each path are bandwidth dominated (i.e., $r_i \approx p_i/b_i$) this becomes

$$\frac{n_i}{n_{\mathrm{total}}} \approx \frac{1}{r_i\left(\sum_{k=1}^{N} 1/r_k\right)}. \tag{23}$$

The average RTT across all paths is then

$$\bar{r} = \sum_{i=1}^{N} r_i \cdot \frac{n_i}{n_{\mathrm{total}}} = \frac{N}{\sum_{k=1}^{N} 1/r_k}. \tag{24}$$

Similarly, the average deviation in RTT is

$$\bar{v} = \sum_{i=1}^{N} |r_i - \bar{r}| \cdot \frac{n_i}{n_{\mathrm{total}}} = \sum_{i=1}^{N} \frac{|r_i - \bar{r}|}{r_i\left(\sum_{k=1}^{N} 1/r_k\right)}. \tag{25}$$

Then from (19), no timeouts will occur if

$$r_i < \bar{r} + K \cdot \bar{v} \quad \text{for } 1 \leqslant i \leqslant N. \tag{26}$$

4.1.1. A two path example

While (26) ultimately determines if timeouts will occur, a simple two path example is now considered to examine how much of a difference in RTTs can be tolerated before timeouts occur. In this scenario there are two paths between A and B, meaning $N = 2$. The RTTs of the paths are r_1 and r_2 respectively, where we will assume that $r_1 \leqslant r_2$ and that both are bandwidth dominated. The average RTT is then

$$\bar{r} = \frac{2 \cdot r_1 \cdot r_2}{r_1 + r_2}, \tag{27}$$

and the average deviation in RTT is

$$\bar{v} = \frac{2 \cdot r_1 \cdot r_2 (r_2 - r_1)}{(r_1 + r_2)^2}. \tag{28}$$

Then from (26) no timeouts will occur if

$$r_i < \frac{2 \cdot r_1 \cdot r_2}{r_1 + r_2} + \frac{2 \cdot K \cdot r_1 \cdot r_2 (r_2 - r_1)}{(r_1 + r_2)^2} \quad \text{for } r_i \in \{r_1, r_2\}. \tag{29}$$

From the assumption that $r_1 \leqslant r_2$, it follows that $r_1 \leqslant \bar{r}$, and (29) is always satisfied for $r_i = r_1$. Therefore, a timeout can only occur when $r_i = r_2$.

If the ratio of RTTs is expressed as ρ, where

$$\rho = \frac{r_2}{r_1} \quad \text{or} \quad r_1 = \frac{r_2}{\rho}, \tag{30}$$

no timeouts will occur if

$$r_2 < 2 \cdot r_2 \left[\frac{1}{\rho + 1} + \frac{K(\rho - 1)}{(\rho + 1)^2} \right]. \tag{31}$$

This restricts ρ to

$$1 < \rho < 2 \cdot K - 1 \quad \text{or} \quad 1 < \frac{r_2}{r_1} = \frac{b_1 \cdot p_2}{b_2 \cdot p_1} < 2 \cdot K - 1. \tag{32}$$

In other words, for two paths using equal packet sizes and the typical value of $K = 4$, the path bandwidths can vary by up to a factor of 7 and no timeouts will occur.

When packet sizes are not equal, (32) places the following upper bound on the ratio of path bandwidths, denoted ϕ:

$$\phi = \frac{b_1}{b_2} < \frac{p_1}{p_2}(2 \cdot K - 1). \tag{33}$$

The significance of constraint (33) is that the ratio of packet sizes effectively scales the timeout threshold, which then allows a greater bandwidth disparity to be tolerated. For example, consider using a 1 Mbps wireless LAN connection together with a 40 Kbps cellular phone modem. In this case the ratio of bandwidths of the two paths is 25. Therefore, to avoid timeouts, the packet size ratio must be set greater than (25/7):1 as dictated by (33).

We would like to point out that in many cases it is possible to appropriately size packets sent along different paths in a manner completely transparent to TCP by fragmenting at the IP layer. Assuming sufficiently large packet sizes, each packet which TCP sends down to IP could be fragmented into the appropriate number of pieces, each of correct size. These fragments could then be distributed across paths in the right proportions so that (33) is satisfied. This fragmenting is, however, slightly more involved than the normal IP fragmentation needed when forwarding between networks with different MTUs and interferes with the path MTU discovery mechanism used by TCP.

Note that the restrictions from (33) only apply if the paths are bandwidth dominated, i.e. if $p/b \gg \tau$ in (18). Propagation and queuing delays at intermediate routers as well as the ACK reception time all contribute to τ. If the size of all packets, including the largest packet size which will be sent on the highest bandwidth path, are such that $p/b < \tau$ then the paths are no longer bandwidth dominated. In this case, RTTs are dominated by τ which is probably similar for all paths. In other words, the RTTs are approximately equalized thereby making it possible to stripe the connection across all paths without incurring timeouts, irrespective of the bandwidth ratios involved.

4.1.2. The effect smoothed values have on RTO

Note that use of the exact mean values, $\bar{r}$ and $\bar{v}$, in (26) is an idealized approximation. In reality the smoothed estimates for RTT and deviation in RTT from (20) and (21) play an substantial role in determining what difference in RTTs will cause timeouts. We extracted the TCP RTO timer code from the FreeBSD kernel to evaluate the effect of this smoothing.

A sequence of 100,000 RTT values was given as input to the RTO timer code. This sequence was generated by assuming two paths with bandwidths b_1 and b_2. Packets were probabilistically striped across the two paths, where the probabilities corresponded to bandwidth fractions in accordance with (22). Packets sent on the first and second paths were assumed to have sizes p_1 and p_2 respectively. The RTT value for a packet was then determined based on its size and the bandwidth of the path it was sent on. If the RTT exceeded the smoothed RTO from the RTO timer code, defined in (19), it was considered a timeout.

For equal sized packets, the fraction of packets sent on the lower bandwidth path that lead to timeouts as a function of path bandwidth ratio, ϕ, is show in Fig. 2(a). The curve labeled "Ideal" in the figure corresponds to ideal behavior as predicted by (33). That is, no timeouts occur for bandwidth ratios of $\phi < 7$ and timeouts occur for 100% of the packets for $\phi \geqslant 7$, resulting in a step-function. The curve labeled "Actual" shows the results from the RTO timer for $K = 4$ and the default values of $\alpha = 7/8$ and $\beta = 3/4$ for the smoothing coefficients in (20) and (21). The plot shows that using smoothed RTT values for the RTO estimation causes more timeouts than predicted by the ideal curve. This demonstrates that smoothing has a significant impact on the number of timeouts. To further clarify the impact of smoothing we have also included the curve labeled "$\alpha = 0.99,\ \beta = 0.99$" which approximates the idealized step function much better than the default values. This curve is included only to confirm the analytical results, where by having such high smoothing coefficients the long-term averages are tracked more closely and the influence of short-term transient fluctuations is diminished.

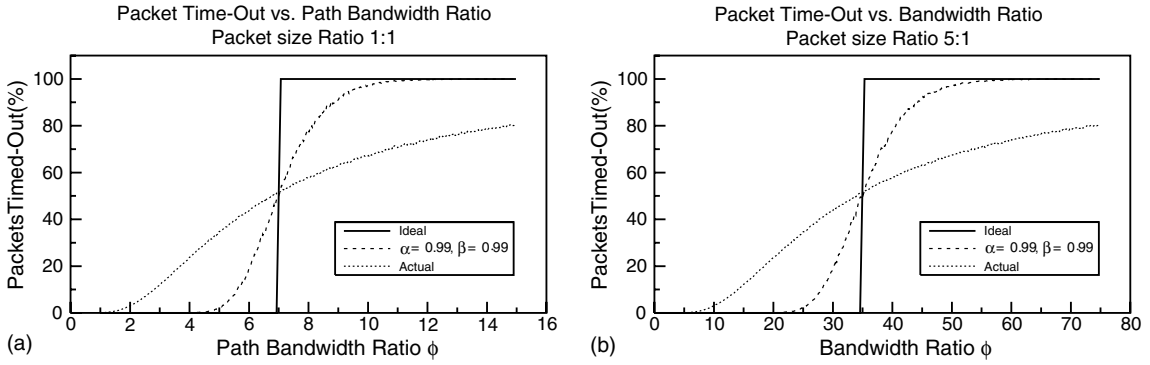

Fig. 2. Lower bandwidth path packet timeout percentage.

As previously mentioned, packet sizes can be adjusted to equalize RTTs. To test this hypothesis we conducted the same set of experiments for a packet size ratio of 5:1 with the results shown in Fig. 2(b). Note that both experimental plots in Fig. 2(b) indicate that the bandwidth ratio threshold has been scaled by the packet size ratio of 5:1, thereby confirming (33).

4.1.3. Periodic smoothed RTT and RTO values

As Fig. 2 illustrates, the use of smoothed RTT and deviation in RTT values in the retransmission timeout calculation leads to a significant discrepancy between the predicted and actual bandwidth ratios that avoid timeouts. When packets are striped non-deterministically according to a given probability distribution, the empirical limit on the maximum tolerable bandwidth ratio is much less than what (33) predicts. In order to see how this might affect an actual packet striping implementation we now consider the special case when sampled RTT values form a repeating periodic sequence. This can occur when packets are striped among available paths by a deterministic scheduling algorithm which distributes data in approximate accordance with (14), for example deficit or surplus round-robin (DRR, SRR) [15,32].

In such a scenario suppose the sampled RTT values repeat every T packets, that is for the ith sampled RTT value r_i, $r_{i+T} = r_i$. This in turn makes the steady-state smoothed RTT values repeat with a period of T, or $R_{i+T} = R_i$. In this case exact expressions for the steady state smoothed values of RTT and deviation in RTT can be found as follows.

From (20), a sequence of T sampled RTT values $r_1, r_2, \ldots, r_T$ leads to the system of equations given below:

$$\begin{aligned} R_1 &= \alpha \cdot R_0 + (1-\alpha) r_1 \\ &\vdots \\ R_T &= \alpha \cdot R_{T-1} + (1-\alpha) r_T \end{aligned} \quad \text{or} \quad \begin{aligned} R_1 - \alpha \cdot R_0 &= (1-\alpha) r_1 \\ &\vdots \\ R_T - \alpha \cdot R_{T-1} &= (1-\alpha) r_T. \end{aligned} \tag{34}$$

Noting that $R_0 = R_T$, this system of equations can be expressed in matrix form as

$$\mathbf{AR} = \mathbf{r}, \tag{35}$$

where $\mathbf{R}$ corresponds to the smoothed RTT values, $\mathbf{r}$ is the sampled RTT sequence, and $\mathbf{A}$ is a $T \times T$ matrix determined by

$$\mathbf{A}_{ij} = \begin{cases} 1 & \text{if } j = i, \\ -\alpha & \text{if } j = (i - 2 \bmod T) + 1, \\ 0 & \text{otherwise.} \end{cases} \tag{36}$$

Expression (35) then becomes

$$\underbrace{\begin{bmatrix} 1 & & & -\alpha \\ -\alpha & 1 & & \\ & \ddots & \ddots & \\ & & -\alpha & 1 \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} R_1 \\ R_2 \\ \vdots \\ R_T \end{bmatrix}}_{\mathbf{R}} = (1-\alpha) \underbrace{\begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_T \end{bmatrix}}_{\mathbf{r}}. \tag{37}$$

Since $\mathbf{A}$ is non-singular, the steady-state smoothed RTT values can be found as

$$\mathbf{R} = (1-\alpha)\mathbf{A}^{-1}\mathbf{r}. \tag{38}$$

The structure of **A** leads to the following solution for the smoothed RTT values given by (38):

$$R_i = \frac{1-\alpha}{1-\alpha^T}\sum_{k=1}^{T} r_k \cdot \alpha^{(i-k \bmod T)}. \tag{39}$$

The steady-state smoothed deviation in RTT values, from (21), can be expressed in a similar manner as

$$V_i = \frac{1-\beta}{1-\beta^T}\sum_{k=1}^{T} |r_k - R_k| \cdot \beta^{(i-k \bmod T)}. \tag{40}$$

Therefore from (19), (39), and (40), a periodic sequence of sampled RTT values will not result in any steady-state timeouts if

$$r_i < R_{i-1} + K \cdot V_{i-1} \quad \text{for } 1 \leqslant i \leqslant T, \tag{41}$$

where $R_0 = R_T$ and $V_0 = V_T$.

4.1.4. A periodic two path example

From (41) the exact distribution of timeouts can be efficiently found for any periodic sequence of sampled RTTs. To see how this actually manifests itself, an example two path scenario is considered in an analogous manner to Section 4.1.1. In this case packets are split between two paths in a deterministic periodic fashion. Assume $T-1$ consecutive packets are sent on the first path having a RTT of r_1 followed by a single packet sent on the second path with RTT r_2, where $r_1 \leqslant r_2$. In other words, the first path is assumed to be of higher bandwidth than the second, and the packets sent generate a periodic pattern of RTT values which repeats every T packets. In this periodic two path case a retransmission timeout will only occur for a packet sent on the lower bandwidth path when $r_2 > \mathrm{RTO}_T$. From (41), the condition necessary to avoid timeouts is then

$$r_2 < R_{T-1} + K \cdot V_{T-1}, \tag{42}$$

which can be determined explicitly from (39) and (40).

The ith smoothed RTT from (39) can be simplified and expressed as

$$R_i = r_1 + \frac{\alpha^{i \bmod T}(1-\alpha)(r_2 - r_1)}{1-\alpha^T}. \tag{43}$$

The smoothed RTT just prior to when the Tth packet is sent on the lower bandwidth path is then

$$R_{T-1} = \frac{(1-\alpha^{T-1})r_1 + (\alpha^{T-1} - \alpha^T)r_2}{1-\alpha^T}. \tag{44}$$

The smoothed deviation in RTT needed for (42), V_{T-1}, can then be found from (40) and (43) and by noting that the smoothed RTT is bounded by $r_1 \leqslant R_i \leqslant r_2$. This then becomes

$$V_{T-1} = \frac{(1-\beta)(r_2 - r_1)}{(1-\beta^T)(1-\alpha^T)}\left[\frac{(1-\alpha)(\alpha^T - \alpha\beta^{T-1})}{\alpha-\beta} + \beta^{T-1}(\alpha - \alpha^T)\right]. \tag{45}$$

To ensure that the fraction of data sent on each path is consistent with (14), the period T must depend on both the packet sizes used for each path and the path bandwidths. From (14), the fraction of data sent on each path equals the ratio of that path's bandwidth to total bandwidth, or if d_i is the amount of data sent on path i:

$$\frac{d_1}{d_1+d_2} = \frac{b_1}{b_1+b_2} \quad \text{or}$$
$$\frac{n_1 \cdot p_1}{n_1 \cdot p_1 + n_2 \cdot p_2} = \frac{p_1/r_1}{p_1/r_1 + p_2/r_2}, \tag{46}$$

where b_i, n_i, and p_i are respectively the bandwidth, number of packets, and packet size for path i. This results in

$$\frac{n_1}{n_2} = \frac{r_2}{r_1}. \tag{47}$$

Therefore, for every packet sent on the lower bandwidth path ($n_2 = 1$) there are $n_1 = r_2/r_1$ packets sent on the higher bandwidth path, or by letting $\rho = r_2/r_1$ the period of the repeating sequence becomes

$$T = \rho + 1. \tag{48}$$

From (44), (45), and (48) the retransmission timeout condition necessary to avoid timeouts, (42), can be expressed equivalently as

$$(\rho - 1)\left(1 - \frac{K\alpha(1-\beta)}{1-\beta^{\rho+1}}\left[\frac{(1-\alpha)(\alpha^\rho - \beta^\rho)}{(1-\alpha^\rho)(\alpha-\beta)} + \beta^\rho\right]\right) < 0. \tag{49}$$

This can be solved numerically for the default values of $\alpha = 7/8$, $\beta = 3/4$, and $K = 4$ which, for the range of interest ($\rho \geqslant 1$), gives

$$1 < \rho < 4.386\cdots \quad \text{or} \quad 1 < \frac{b_1 \cdot p_2}{b_2 \cdot p_1} < 4.386\cdots. \tag{50}$$

Expressed in terms of path bandwidth ratio, ϕ, the upper bound becomes

$$\phi < \frac{p_1}{p_2} 4.386\cdots. \tag{51}$$

In other words for this simple periodic case, smoothed RTT and deviation in RTT values reduce the tolerable path bandwidth disparity from the factor of 7 predicted by (32) to approximately a factor of 4. (When equal sizes packets are sent on both paths and typical values are used for the TCP constants.)

This is verified experimentally in the manner described in Section 4.1.2, with the results shown in Fig. 3. In this case packets are split deterministically between the two paths using the SRR scheduling algorithm [15]. As in Fig. 2, the tolerable bandwidth ratio is scaled by the ratio of packet sizes used. However, due to the deterministic nature of the sampled RTT values, there is a strict maximum bandwidth ratio which will not cause spontaneous retransmission timeouts in steady state. Note that in some cases the SRR scheduling algorithm gives rise to a more complicated periodic sequence than the $T-1$ packets followed by 1 packet used in the analytical model, for example when the ratio of path RTTs, ρ, is non-integer. For this reason the data plotted in Fig. 3 differs slightly from what (51) predicts.

4.2. TCP fast retransmissions

As mentioned at the beginning of this section, the second mechanism that can cause a lower bandwidth path to drag down a higher bandwidth path is the fast retransmission and recovery algorithms found in almost every TCP implementation today. These algorithms are independent of the regular timeout mechanisms. A TCP sender using the fast retransmit algorithm infers that a packet was lost after receiving a sufficient number of duplicate ACKs, typically 3. Since a lost packet implies congestion, congestion avoidance procedures are then invoked. This mechanism can be falsely triggered, i.e. when no packet loss has occurred, when out-of-order packets are received by the destination. This can happen frequently in our bandwidth aggregation scheme, since multiple packets may be received from higher bandwidth paths while a packet sent on a lower bandwidth path remains in flight. Note that even when delayed ACKs are used, a TCP receiver will immediately generate an ACK for each out-of-order packet it receives.

The maximum number of legitimate duplicate ACKs (not caused by packet loss) that a TCP sender using our bandwidth aggregation scheme can

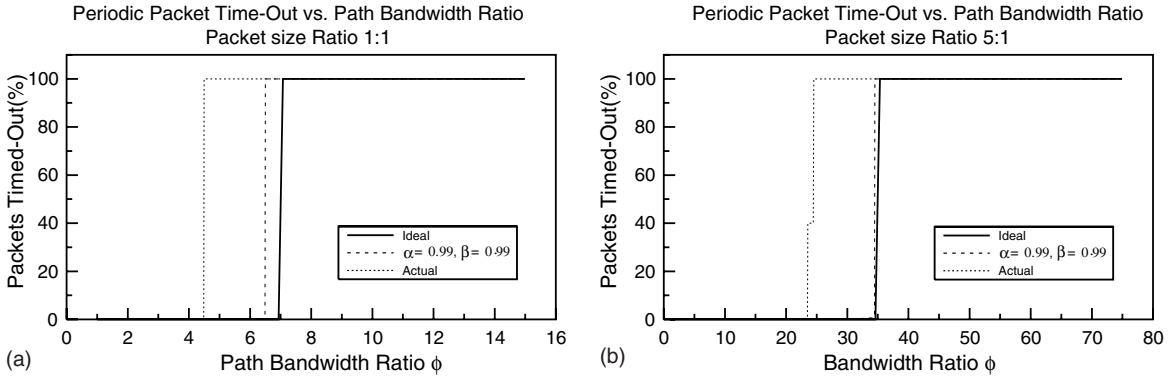

Fig. 3. Periodic lower bandwidth path packet timeout percentage.

expect is characterized as follows. Assuming a TCP receiver processing packets in the order of reception, the number of duplicate ACKs generated will equal the number of packets received on all other paths while a single packet is in flight on the path with the largest one-way latency. More specifically, let l_k be the one-way latency of path k, which is greater than or equal to the one-way latency of any other path, or for bandwidth dominated paths

$$l_k = \max\left(\frac{p_1}{b_1}, \frac{p_2}{b_2}, \ldots, \frac{p_N}{b_N}\right). \quad (52)$$

The maximum number of duplicate ACKs are then generated when packets are continually streaming in from all other paths. Noting that the number of packets received from path i in time l_k is $l_k \cdot b_i/p_i$, the number of duplicate ACKs in this case, DupACKs, is then

$$\text{DupACKs} = \sum_{i=1}^{N} \left\lfloor l_k \frac{b_i}{p_i} \right\rfloor - 1 \leqslant l_k \sum_{i=1}^{N} \frac{b_i}{p_i} - 1. \quad (53)$$

When equal sized packets are used for all paths, this upper bound becomes

$$\text{DupACKs} \leqslant \frac{1}{b_k} \sum_{i=1}^{N} b_i - 1, \quad (54)$$

where b_k is the minimum path bandwidth. For the two path scenario, the right side of (54) simply becomes the ratio of path bandwidths making $\text{DupACKs} \leqslant \phi$.

The impact that the fast retransmit and recovery algorithms have on TCP performance when persistent packet reordering occurs is illustrated in Fig. 4. This plot presents simulation data generated by the ns-2 network simulator [33]. The two path scenario was considered, where the total raw bandwidth between the two communicating nodes was 20 Mbps and the bandwidth ratio of the two paths varied from $1 \leqslant \phi \leqslant 15$ in integral steps. Equal sized packets of 1000 bytes were split deterministically between the paths using the SRR algorithm to conform with (14). Each data point shown is the average of 10 runs, each lasting 4 simulated minutes. The portion of total raw bandwidth realized by TCP is shown, as well as the number of fast retransmissions, as a function of bandwidth ratio. The number of fast retransmissions is represented as a percentage found by dividing the number of fast retransmissions that occurred by the number of packets sent on the lower bandwidth path. Clearly TCP experiences a significantly performance degradation corresponding to the sudden increase in fast retransmissions (due to duplicate ACKs) for bandwidth ratios of $\phi \geqslant 3$.

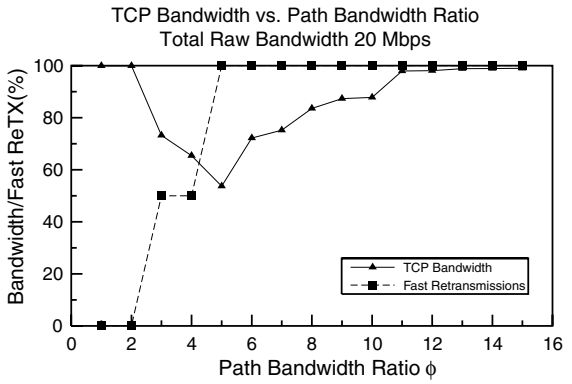

Fig. 4. Simulated TCP throughput for fixed raw bandwidth.

One way fast retransmissions might be avoided is by sending packets out of order. For instance, assume that there are two paths between the source and destination with a bandwidth ratio $\phi = 4$. As per (14), the optimal distribution in this case is 1 packet on the lower bandwidth path for every 4 packets transmitted on the higher bandwidth path. If packet 1 is transmitted first on the lower bandwidth path followed by packets 2, 3, 4, and 5 on the higher bandwidth path, the reception of packet 4 will generate a third duplicate ACK causing the fast retransmission of packet 1. The solution to this problem is to first send packet 5 on the lower bandwidth path and then transmit packets 1, 2, 3, and 4 on the higher bandwidth path. Assuming that TCP sends an adequate number of segments to the network layer, the proposed out-of-order transmission can be done transparently by the network layer. However, the network layer must then understand TCP segment numbers.

4.3. TCP performance summary

We conclude this section by summarizing ways of tuning TCPs behavior. From the discussion

above, TCP can be optimized in several ways which include the following:

(i) Use large window sizes to accommodate all packets when using multiple paths which have high bandwidth ratios.
(ii) Reduce the packet size p appropriately for each path to reduce the bandwidth domination of RTTs.
(iii) Use high RTO values by either allowing a greater variation in RTT or by configuring somewhat high absolute minimum values. As shown above, the default value of K allows for a bandwidth ratio of 7 when equal sized packets are used. However, a value of $K = 13$ covers paths with bandwidth ratios up to 25.
(iv) Use TCP Vegas or other enhanced TCP versions that mitigate the effect of slow-start.
(v) Allow sending packets out of order to avoid fast retransmissions.

Note that all the above modifications and optimizations can be confined to the sender side alone. They preserve the end-to-end TCP semantics and will transparently inter-operate with any standard TCP receiver.

5. Implementation and results

We have tested a proof-of-concept implementation where the sender and receiver were connected by two logical paths. We implemented the previously described tunneling mechanism on a FreeBSD sender and receiver. The dummynet facility [34] was used in our experiments. This allows the kernel to route packets through logical paths based on specified probabilities.

For our experiments we created two logical paths over a single physical path. We would like to emphasize that it would be just as easy to run the experiment across two distinct physical paths. The logical implementation we chose offered a greater flexibility in controlling both the bandwidth of each path and the probabilities with which each path was chosen, which in turn determined packet distribution.

The firewalling mechanisms provided by FreeBSD were exploited on the sending side to set the proper source address for each packet and to encapsulate packets going through one of the paths. In this case equal sized 1500 byte packets were split probabilistically between two paths. The receiving side was configured to strip encapsulating headers as needed. Further details regarding the experimental setup have been omitted for the sake of brevity.

The experimental results are illustrated in Fig. 5(a) and (b). These figures show TCP bandwidth as a function of available raw bandwidth, which refers to the total bandwidth (from both paths) between the source and the destination. TCP bandwidth was measured using the netperf benchmarking package [35].

Fig. 5(a) shows the case when the bandwidth of both paths is equal. The three curves illustrate the

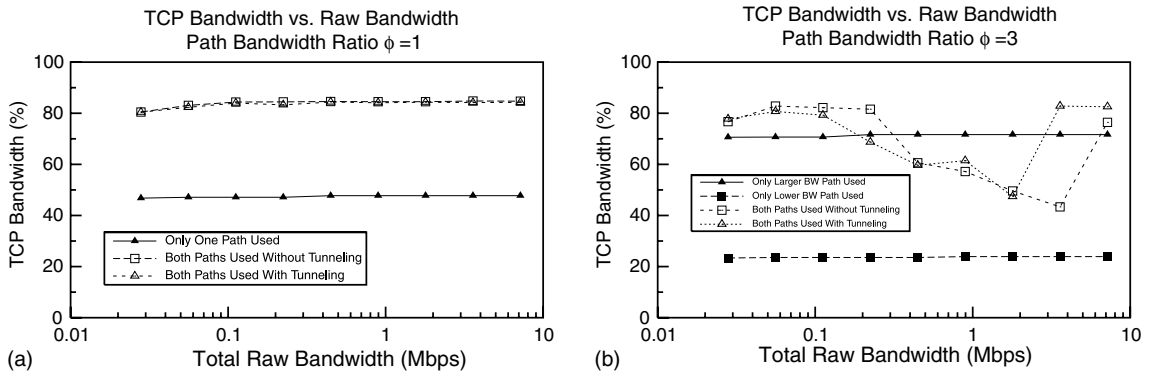

Fig. 5. TCP throughput for fixed bandwidth ratios.

bandwidth TCP achieves as a percentage of the total raw bandwidth when only one path is used, when both paths are used without tunneling, and when both paths are used with tunneling. As expected, over several orders of magnitude TCP realizes roughly twice the bandwidth when both paths are used simultaneously compared to when only a single path is used. The similarity between the tunneling and non-tunneling curves shows that the impact of tunneling on TCP performance is negligible. Since a standard TCP connection is equivalent to using only a single path, this demonstrates that the proposed bandwidth aggregation mechanism can, in principle, deliver additional bandwidth to the application level.

To test the more general efficacy of the proposed mechanism we carried out experiments using two paths of different bandwidths. From the previous analysis for bandwidth dominated paths that have disparate bandwidths, the default value of $K = 4$ in most TCP implementations will ideally allow connection striping across paths with bandwidth ratio of $\phi < 7$ without incurring timeouts. For our experiments we selected a bandwidth ratio of $\phi = 3$ in order to have a reasonable difference in path bandwidths while at the same time pushing the upper limit of the default tolerance for duplicate ACKs before fast retransmissions might occur. The results are illustrated in Fig. 5(b) with the curves being analogous to those in Fig. 5(a).

The data shows that despite a 3:1 bandwidth disparity, the simultaneous use of both paths can yield better performance. For data points at low raw bandwidth values, about 3 Mbps and below, some timeouts do occur which is consistent with the data from Fig. 2(a). This is due to the probabilistic splitting and the effect of using smoothed RTT values as previously explained. In this case, timeouts do cause the lower bandwidth path to drag down the higher bandwidth path somewhat.

For raw bandwidths above about 3 Mbps the RTTs of both paths drop to sufficiently low values, which in turn causes the RTO values to drop below 1 s. The version of the FreeBSD kernel used for these experiments imposed a lower limit of 1 s for RTO, below which timeouts will not occur. Note that the paths are still bandwidth dominated, i.e. path RTTs are still dominated by bandwidth, but the timeout values are now sufficiently large to accommodate the variation in RTT which prevents timeouts. The net result is that connection striping across the two paths provides greater overall bandwidth than when using the higher bandwidth path by itself. This demonstrates that the proposed mechanism can work for even bandwidth dominated paths, assuming TCP timeouts can be mitigated.

To further investigate the limitations of our bandwidth aggregation scheme, TCP performance was investigated over a range of bandwidth ratios while the total raw bandwidth available was held constant. These experiments were again performed with two network paths between communicating hosts, however tunneling was omitted due to its negligible impact. This time equal sized 1500 byte packets were split deterministically between the

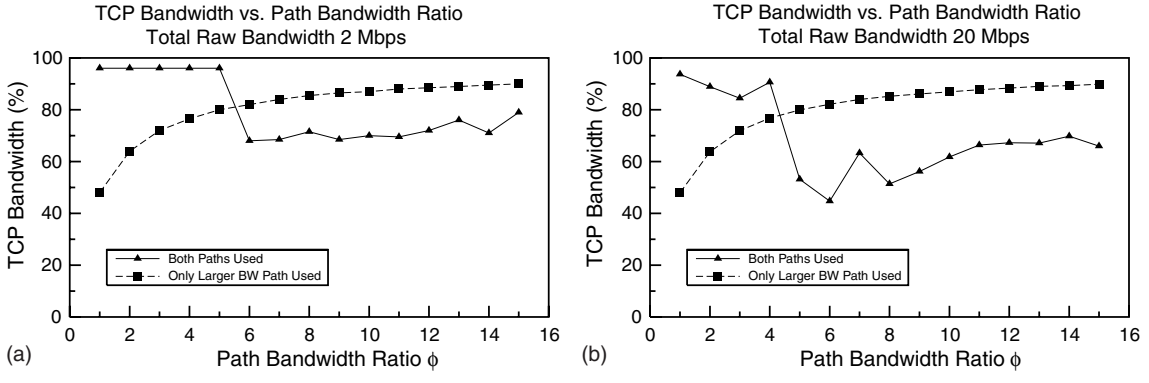

Fig. 6. TCP throughput for fixed raw bandwidth.

two paths. The results are shown in Fig. 6(a) and (b). Both plots show the TCP bandwidth achieved when both paths are used, and when only the higher bandwidth path alone is used. These results agree closely with (51) and Fig. 3(a), showing that a bandwidth ratio of around 4 can be tolerated when deterministic packet splitting is employed.

6. Conclusions

We have proposed a novel mechanism to simultaneously stream data across multiple IP paths in order to aggregate their bandwidth. It is applicable to connectionless communication (UDP) as well as for striping data from a TCP connection across multiple IP paths. We have investigated its performance and experimentally validated the analytical results. This work was motivated by practical considerations, since ubiquitous network access is becoming a reality as the number of wireless communication technologies continues to grow. It is therefore clear that in the near future there will be multiple transport conduits available to many users. Our analysis and supporting data demonstrate that the proposed mechanism works well in most scenarios of practical interest.

Future work includes implementing all of the TCP modifications and optimizations mentioned above and further detailed simulations in wireless and mobile scenarios. Another approach to bandwidth aggregation would be to implement load sharing in SCTP and compare its performance with the currently proposed mechanism. Lastly, the ability to dynamically add or delete IP addresses from an active connection between endpoints having multiple network interfaces could lead to security hazards which warrant further investigation.

References

[1] G. Malkin, Nortel networks multi-link multi-node PPP bundle discovery protocol, RFC 2701, September 1999.
[2] K. Sklower, B. Lloyd, G. McGregor, D. Carr, T. Coradetti, The PPP multilink protocol (MP), RFC 1990, August 1996.
[3] E.W. Simpson, The point-to-point protocol (PPP), RFC 1661, July 1994.
[4] E.W. Simpson, PPP in HDLC-like framing, RFC 1662, July 1994.
[5] D. Rand, PPP reliable transmission, RFC 1663, July 1994.
[6] R. Ogier, V. Ruenburg, N. Shacham, Distributed algorithms for computing shortest pairs of disjoint paths, IEEE Transactions on Information Theory 39 (1993) 443–455.
[7] I. Cidon, R. Rom, Y. Shavim, Analysis of multi-path routing, IEEE/ACM Transactions on Networking 7 (1999) 885–896.
[8] N. Taft-Plotkin, B. Bellur, R. Ogier, Quality of service routing using maximally disjoint paths, in: Proceedings of the IEEE IWQoS'99, London, UK, June 1999, pp. 119–128.
[9] A. Nasipuri, S.R. Das, On-demand multipath routing for ad-hoc networks, in: Proceedings of the IEEE ICCCN'99, Boston, MA, October 1999, pp. 64–70.
[10] J. Raju, J. Garcia-Luna-Aceves, A new approach to on-demand multipath routing, in: Proceedings of the IEEE ICCCN'99, Boston, MA, October 1999, pp. 522–527.
[11] M. Pearlman, Z. Haas, P. Scholander, S. Tabrizi, On the impact of alternate path routing for load balancing in mobile ad hoc networks, in: Proceedings of ACM MobiHOC'00, Boston, MA, August 2000, pp. 119–128.
[12] M. Gerla, S.-J. Lee, Split multipath routing with maximally disjoint paths in ad hoc networks, in: Proceedings of ICC'01, Helsinki, Finland, June 2001.
[13] W.-H. Liao et al., A multi-path QoS routing protocol in a wireless mobile ad hoc network, in: Proceedings of the IEEE ICN'00, CREF, Colmar, France, July 2001.
[14] A. Snoeren, Adaptive inverse multiplexing for wide-area wireless networks, in: IEEE Globecom, 1999, pp. 1665–1672.
[15] H. Adiseshu, G. Parulkar, G. Varghese, A reliable and scalable striping protocol, in: SIGCOMM: ACM Special Interest Group on Data Communication, Palo Alto, CA, August 1996, pp. 131–141.
[16] D.C. Anderson, J.S. Chase, A.M. Vahdat, Interposed request routing for scalable network storage, in: Proceedings of the Fourth Symposium on Operating System Design and Implementation (OSDI), October 2000.
[17] Sun Microsystems, Sun StorEdge network data replicator white paper, http://www.sun.com/storage/white-papers/sndr.html.
[18] A. Watson, Network Appliance, Inc., Filer deployment strategies for evolving LAN topologies, http://www.netapp.com/tech_library/3009.html.
[19] R. Haagens, iSCSI requirements, http://www.ietf.org/proceedings/00jul/SLIDES/ips-iscsi-reqs.pdf.
[20] Cisco Systems, Etherchannel technologies, http://www.cisco.com/warp/public/779/largeent/learn/technologies/fast_echannel.html.
[21] SCTP reference implementation, http://www.sctp.org/.
[22] GPL SCTP prototype implementation, http://www.sctp.de/.
[23] SCTP for beginners, http://tdrwww.exp-math.uni-essen.de/pages/forschung/sctp_fb.
[24] SCTP: an overview, http://sctp.chicago.il.us/sctpoverview.html.

[25] Protocol Engineering Lab at University of Delaware CIS Department, http://www.cis.udel.edu/iyengar/research/SCTP.
[26] R.R. Stewart, M.A. Ramalho, et al., SCTP extensions for dynamic reconfiguration of IP addresses and enforcement of flow and message limits, June 2001, http://www.ietf.org/internet-drafts/draft-ietf-tsvwg-addip-sctp-02.txt.
[27] E.C. Perkins, IP mobility support, RFC 2002, October 1996.
[28] C. Perkins, IP encapsulation within IP, RFC 2003, October 1996.
[29] C. Perkins, Minimal Encapsulation within IP, RFC 2004, October 1996.
[30] J. Solomon, Applicability statement for IP mobility support, RFC 2005, October 1996.
[31] V. Jacobson, Modified TCP congestion avoidance algorithm, end2end interest group mailing list, April 1990.
[32] M. Shreedhar, G. Varghese, Efficient fair queuing using deficit round robin, in: SIGCOMM: ACM Special Interest Group on Data Communication, Cambridge, MA, September 1995, pp. 231–242.
[33] S. McCanne, S. Floyd, NS network simulator, http://www.isi.edu/nsnam/ns/.
[34] L. Rizzo, Dummynet: a simple approach to the evaluation of network protocols, ACM Computer Communication Review 27 (1997) 31–41.
[35] Information Networks Division, Hewlett–Packard Company, Netperf: A network performance benchmark, 15 February 1996, http://www.netperf.org/.

Dhananjay S. Phatak received the B.Tech. degree in Electrical Engineering from the Indian Institute of Technology, Bombay, in 1985; M.S. in Microwave Engineering in 1990 and Ph.D. in Computer Systems Engineering in 1993, both from the Electrical and Computer Engineering Department, University of Massachusetts, Amherst. From 1994 till 2000 he was Assistant Professor of Electrical Engineering at the State University of New York, Binghamton. Since Fall 2000 he has been an Associate Professor in the Computer Science and Electrical Engineering Department at the University of Maryland Baltimore County (UMBC).

His current research interests and activities span the areas of (1) mobile and high performance computing and networks; (2) computer arithmetic algorithms and their VLSI implementations, digital and analog VLSI design and CAD; and (3) neural networks.

In the past he has worked on microwave and optical integrated circuits. Dr. Phatak has published articles in the IEEE Transactions in several diverse areas (Computers, Neural Networks, and Microwave Theory and Te+9chniques), as well as in other premier journals and conferences in his areas of research. He has been active in the technical program committees of premier conferences in his areas of research (including INFOCOM, IEEE's Biannual International symposium on Computer Arithmetic and IJCNN). He has obtained research support from the NSF, Lockheed Martin and AetherSystems Inc. He is a recipient of the National Science Foundation's CAREER award in 1999. His high speed CORDIC algorithm has recently been awarded a patent. He is currently an Associate Editor for the IEEE Transactions on Computers.

Tom Goff received the B.S. degree in Electrical Engineering and the M.S. degree in Computer Science from the State University of New York at Binghamton, in 1997 and 2000 respectively. He is currently pursuing a Ph.D. degree in computer science at the University of Maryland, Baltimore County. His research interests include computer arithmetic algorithms and their implementations, wireless and mobile networking, and distributed computing.

Jim Plusquellic received the B.S. degree in Biology from Indiana University of Pennsylvania in 1983 and the M.S. and Ph.D. degree in Computer Science from the University of Pittsburgh 1995 and 1997, respectively. He is now at the University of Maryland, Baltimore County, as an assistant professor in Computer Engineering. Professor Plusquellic is a board member of the ACM Special Interest on Design Automation (SIGDA) and is on the program committee for the International Test Conference. He is also appointed as program co-chair of the Current and Defect Based Testing workshop associated with the 2003 VLSI Test Symposium. He is a member of both the IEEE and ACM.