A Novel Mechanism for Data Streaming Across Multiple IP Links for Improving Throughput and Reliability in Mobile Environments

Dhananjay S. Phatak   Tom Goff

Abstract— **With ubiquitous computing and network access now a reality, multiple network conduits will be available to mobile as well as static hosts (for instance, wired connections, 802.11 style LANs, Bluetooth, cellular phone modems, etc). Selection of the preferred mode of data transfer is a dynamic optimization problem depending on the type of application, its bandwidth/latency/jitter requirements, current network status 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 related work identifying schemes that might be applicable to the problem, along with their feasibility, and pros and cons. We then propose a new mechanism to aggregate the bandwidth of multiple IP links by splitting a data flow across multiple network interfaces at the IP level. Our method is transparent to transport (TCP/UDP) and higher layers. We have analyzed the performance characteristics of the aggregation scheme and demonstrated significant gain when the links being aggregated have similar bandwidth and latency. The use of multiple interfaces also enhances reliability. Our analysis identifies the conditions under which the proposed scheme, or any other scheme that stripes a single TCP connection across multiple IP links, can be used to enhance throughput. Several interesting directions for future work have also been identified.

I. 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 LAN (802.11, Airport LANs, Ricochet, etc.), and cellular phone modem, where each technology has a corresponding service provider. Selecting which service to use for data transfer is a dynamic optimization problem which should track attributes which might include bandwidth/throughput, latency, jitter, quality of service (QoS) requirements, cost, power consumption, residual battery life, interference, and traffic patterns. 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 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 interface goes down or one network is 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.

A. 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.

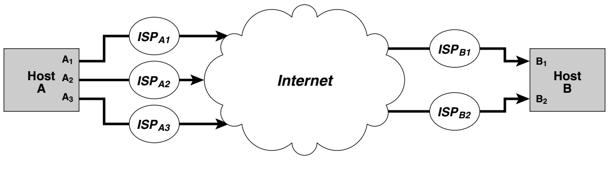

Fig. 1
COMMUNICATING HOSTS WITH MULTIPLE NETWORK CONNECTIONS (MULTI-HOMED).

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.

CSEE Dept. University of Maryland Baltimore County (UMBC), 1000 Hilltop Circle, Baltimore, MD 21250. Email: {phatak,tgoff1}@umbc.edu.
This work was supported in part by Aether Systems Inc.

0-7803-7476-2/02/$17.00 (c) 2002 IEEE.

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 striping a connection across multiple IP interfaces for bandwidth aggregation.

B. Related Work

Bandwidth aggregation across multiple channels has been considered in the literature in varying contexts and for different scenarios and applications, distinct from those considered in this paper. For instance, multi-link PPP [1], [2], [3] is designed to aggregate multiple *logical* data channels into one. Since PPP was intended to operate at the data link layer, below the IP level [4], [5], [6], 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 Internet Service Providers (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 [7], [8], [9], [10], [11], [12], [13], [14], [15], [16]. 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 links. 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 [17], [18], [19], [20]. 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 [21] product provides bandwidth aggregation across multiple Ethernet links. This product targets the extremely high instantaneous bandwidth requirements between a data-server and nodes which services queries in typical database applications. Likewise, a recent IETF draft from the HP storage systems group [20] deals with exploiting ultra wide-band SCSI connections for distributed storage. They simply mention the problems associated with trying to aggregate multiple IP links into a single TCP connection; they don't propose a solution.

The recently proposed "Stream Control Transmission Protocol" (SCTP, RFC 2960, and [22], [23], [24], [25], [26]) comes closest to solving the problem at hand. SCTP is a new transport level protocol which supports multi-streaming and multihoming (a host having multiple IP addresses). A recent extension [27] proposes dynamic addition and deletion of IP addresses at source/destination.

However, in its current form SCTP does not do load-sharing, that is multi-homing is used only for redundancy [25]. A single 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 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 links has not been considered in the context of mobile/wireless networks where we believe it will be a real issue. In the following we propose a new mechanism for bandwidth aggregation across multiple IP links and analyze its performance.

II. Proposed Aggregation Mechanism

We propose a solution at the network (IP) layer. The idea is to manipulate the routing entities (daemons, tables, etc.) so as to send packets across different 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, another 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 "Addr(A_i)", and the IP address associated with interface B_i is denoted "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 daemon at host B logs the source address of the connection as being associated with interface A_1, i.e. Addr(A_1). Likewise the connection identifier at A has Addr(B_1) as the destination address for the connection. Now suppose the application at A requires more bandwidth and packets from the same 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 sent out. Therefore, packets from a connection that was established with source address Addr(A_1) will contain the unrecognized source address 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 tuple (source IP address, source port, destination IP address, destination port). The packet is then dropped by B.

(ii) To keep B from unnecessarily dropping these otherwise valid packets, suppose the source address Addr(A_1) is inserted into 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 do not 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 on 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 [28], [29], [30], [31]). 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 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 Addr(A_1) and Addr(B_1) respectively) is encapsulated in a packet having source address Addr(A_3) and destination address 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.

Depending upon how many firewalls the packet needs to traverse and how extensive or restrictive the firewall policies are, more than one encapsulating header could be required in a some rare cases. However, a single encapsulation header suffices in most cases.

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 link not multiple links. If the bandwidth/delay characteristics of two links are substantially different, packets sent on the slower link 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 buried in TCP. The net effect could be that the slow link "drags down" the fast link. This would defeat the original purpose of using multiple interfaces simultaneously, in which case using only the fast link would be optimal.

We address each of the above issues. The first three are discussed in the remainder of this section while the last one is considered in the following section.

The first issue is how to make A and B aware of any multiple 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 in SCTP which allows initialization chunks to list multiple addresses. While such modifications would interfere with interoperability and typically break "something else", we believe that using some more of the full header length allowed (by TCP, IPv4 and IPv6) and employing some of the unused fields is a fairly transparent solution.

The second question, as to whether using multiple interfaces is tantamount to using the same spectrum, is beyond the scope of this work and highly technology specific. For example, while Bluetooth and 802.11 occupy the same band, cellular phone modems do not. Neither is the wideband CDMA projected to occupy the same ISM band. In these cases, the radio interference issue is nonexistent. With the rapid growth in wireless technologies and services, the chances are small that all transport conduits will end up using same technology and/or spectrum. Also, 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 likely have a wired Ethernet connection as well as an 802.11 wireless LAN connection, where the wireless cell data rate can 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 IV). 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 fastest (has the highest bandwidth), most of the packets are then free from the IP-in-IP overhead. Furthermore if tunneling is not done, the slower link(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 slower link(s). Thus any added bandwidth utilization is available for free, meaning it is simply not realizable by the conventional single link TCP connection. In this sense the term "overhead" is a misnomer. In addition, the minimal encapsulation proposed in the mobile IP standard [30] along with header compression techniques can be employed as well to mitigate any IP-in-IP overhead.

Now only the last question from the list above remains, the question of TCP performance. This is considered in the following section.

III. 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 across different links 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 links.

As previously mentioned, for TCP, using two links together at the network layer can sometimes lead to worse performance than using a single link alone. This can happen due to the two main mechanisms listed below.

(i) The bandwidth/delay mismatch between links causes packets sent on a lower capacity link to arrive after the sender-side TCP has already 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 link alone. Unfortunately, with each timeout TCP drastically scales back its congestion window and invokes slow-start, thereby under-utilizing the higher capacity link.

(ii) Most TCP implementations include the "fast-retransmit/recovery" mechanism [32]. Even if the TCP timeout values were set high to prevent the scenario described above, fast-retransmit still poses an independent problem. For the purpose of illustration, suppose the ratio of the round-trip-times (RTTs) of two links being used is 4 and suppose the first packet is transmitted on the slower link and the next four packets are transmitted on the faster link. The receipt of packets 2, 3, and 4 at the destination will cause the receiver-side TCP to signal a fast-retransmission of the first packet, thereby wasting the prior transmission on the slow link. 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 links, we present an analytical model for N links. We then derive expressions that show how much bandwidth discrepancy can be tolerated without incurring timeouts.

A. Analysis of TCP Retransmission Timeouts

For the sake of illustration, consider striping TCP packets across two links, a high bandwidth and a low bandwidth link. Timeouts will happen only if the RTTs for packets sent across the two links are substantially different. The RTT can be split into two components, namely

$$\mathrm{RTT} = \frac{p}{b} + \tau, \qquad (1)$$

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

If the RTTs for packets sent across the two links are dominated by their respective $\frac{\mathrm{p}}{\mathrm{b}}$ ratios, then a bandwidth disparity between the links causes a corresponding disparity in the RTTs across the links. This in turn can lead to frequent timeouts degrading the performance. On the other hand if the RTT is 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 is sufficiently large, or the transmission distance and hence propagation delay is sufficiently large. In such cases we expect to see performance gains.

As an example, consider two links with bandwidths of 160 Kbps and 40 Kbps, giving a bandwidth ratio of 4:1. For full Ethernet sized packets (approximately 1560 bytes per packet) being sent a single hop between adjacent nodes the "other delays" in equation (1) are negligible, since τ is on the order of microsecond whereas the p/b ratios are 100s of milliseconds. In this case the RTTs are dominated by the p/b ratios, implying that using the two links together is worse than using the higher capacity link by itself. Now consider two links with the same 4:1 bandwidth ratio, but with actual bandwidths of 100 Mbps and 25 Mbps and where the destination is several hops away. Now the p/b ratio for each link is on the order of 100s of microseconds, whereas τ can be milliseconds. In this case the RTTs are dominated by τ implying that using the two links together can be expected to yield better overall performance than using the higher capacity link 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 link can be dynamically fragmented to equalize the p/b ratios of both links. Next we present a quantitative analysis of the problem.

B. Analytical 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 k is a linear function of packet size. That is, the one-way latency of the i-th packet of size p_i, sent from A to B along path k at time t is

$$l_{AB_k}(p_i,t) = \frac{p_i}{b_{AB_k}(t)} + c_{AB_k}(t). \tag{2}$$

Given this model, $b_k(t)$ and $c_k(t)$ correspond to the effective bandwidth and propagation delay of path k, respectively, at time t. From this one-way latency, the RTT of the i-th packet becomes

$$r(p_i,q_i,t) = l_{AB_k}(p_i,t) + \delta_i + l_{BA_{k'}}(q_i, t + l_{AB_k}(p_i,t) + \delta_i), \tag{3}$$

where δ_i is the delay between when packet i was received and when the corresponding ACK was sent, and q_i is the size of the packet sent from B to A containing the ACK. From equations (1),(2), and (3) note that

$$\tau = c_{AB_k}(t) + \delta_i + l_{BA_{k'}}(q_i, t + l_{AB_k}(p_i,t) + \delta_i).$$

To make the ensuing analysis more manageable, the following simplifying assumptions are made.

- The effective bandwidth of a path is constant throughout the lifetime of a TCP connection, this allows the time dependency to be dropped from (2) and (3).
- All data packets sent along path k have same size p_k.
- There is no delay between when a packet is received and when the corresponding ACK is sent, that is $\delta_i = 0$ in (3).
- All ACKs are sent back on a single path on 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 link, the latency of transmission for all ACKs can be assumed to be the same: Δ_{ack}.

These assumptions simplify (3), and the RTT of a packet sent along path k becomes

$$r_k = l_k(p_k) + \Delta_{\text{ack}} = \frac{p_k}{b_k} + c_k + \Delta_{\text{ack}} \tag{4}$$

C. Optimal Fraction 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_k packets each of size p_k along path k should equal the total connection time, leaving no link idle during the connection. This means

$$L_1(p_1,n_1) = L_2(p_2,n_2) = \cdots = L_N(p_N,n_N), \tag{5}$$

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

$$L_i(p_i,n_i) \approx \frac{p_i \cdot n_i}{b_i} + c_i + \Delta_{\text{ack}} \tag{6}$$

The system of equations given in (5) then becomes

$$\frac{n_1 \cdot p_1}{b_1} + c_1 = \frac{n_2 \cdot p_2}{b_2} + c_2 = \cdots = \frac{n_N \cdot p_N}{b_N} + c_N. \tag{7}$$

Alternatively, this can be expressed in terms of the total amount of data having size D to be 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

$$f_i = \frac{n_i \cdot p_i}{D}, \tag{8}$$

equation (7) becomes:

$$\frac{D}{b_1} \cdot f_1 + c_1 = \frac{D}{b_2} \cdot f_2 + c_2 = \cdots = \frac{D}{b_N} \cdot f_N + c_N, \tag{9}$$

where

$$f_1 + f_2 + \cdots + f_N = 1. \tag{10}$$

This system of equations defined by (9) and (10) can be represented in matrix form as

$$\mathbf{Bf} = \mathbf{c}. \tag{11}$$

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

$$\mathbf{B}_{ij} = \begin{cases} b_i^{-1} & \text{if } j = i \\ -b_{i+1}^{-1} & \text{if } j = i+1 \quad \text{For rows } 1 \le i \le N-1 \\ 0 & \text{otherwise} \end{cases}$$
$$\mathbf{B}_{ij} = 1 \quad \text{For row } i = N, \tag{12}$$

$\mathbf{f}$ 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

$$\mathbf{c}_i = \begin{cases} \frac{c_{i+1} - c_i}{D} & \text{for } 1 \le i \le N-1 \\ 1 & \text{for } i = N \end{cases}.$$

This makes (11)

$$\underbrace{\begin{bmatrix} b_1^{-1} & -b_2^{-1} & & & \mathbf{0} & \\ & b_2^{-1} & -b_3^{-1} & & & \\ & & b_3^{-1} & -b_4^{-1} & & \\ & \mathbf{0} & & \ddots & \ddots & \\ & & & & b_{N-1}^{-1} & -b_N^{-1} \\ 1 & \cdots & \cdots & \cdots & \cdots & 1 \end{bmatrix}}_{\mathbf{B}} \underbrace{\begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_{N-1} \\ f_N \end{bmatrix}}_{\mathbf{f}} = \underbrace{\begin{bmatrix} \frac{c_2 - c_1}{D} \\ \frac{c_3 - c_2}{D} \\ \frac{c_4 - c_3}{D} \\ \vdots \\ \frac{c_N - c_{N-1}}{D} \\ 1 \end{bmatrix}}_{\mathbf{c}}. \tag{13}$$

Due to its construction, this system of equations is guaranteed to have a unique solution, given by

$$\mathbf{f} = \mathbf{B}^{-1}\mathbf{c}. \tag{14}$$

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

$$f_i = \frac{b_i}{b_1 + b_2 + \cdots + b_N}. \tag{15}$$

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

For large D the transfer time, t, is going to be bandwidth dominated, under the pipelined transfer assumption, where

$$t = \frac{D}{b_{\text{total}}} \quad \text{and } b_{\text{total}} = b_1 + b_2 + \cdots + b_N. \tag{16}$$

In this time, the fraction of data sent along path i, D_i, using the full bandwidth of the path is

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

The fraction of total data sent along path i is then

$$f_i = \frac{D_i}{D} = \frac{b_i}{b_{\text{total}}}. \quad (18)$$

Therefore, for large data transfers, data should be partitioned across multiple paths according to (15) to minimize the overall transfer time.

D. Conditions for Preventing TCP Retransmission Timeouts

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

$$\text{RTO}_i = R_i + K \cdot V_i, \quad (19)$$

where R_i is the current smoothed estimate of RTT, V_i is the current smoothed estimate of the deviation in RTT, and K is a constant factor (typically $K = 4$). R_i and V_i are defined by the following recursive equations.

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

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

where RTT_i is the sampled RTT of the i-th packet. Equations (20) and (21) act as a low-pass filter on the sampled RTT, which smoothes out 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 (15).

From relations (8) and (15), the fraction of *packets* sent along each path is given by

$$\frac{n_i}{n_{\text{total}}} = \frac{b_i/p_i}{\sum_{k=1}^{N} b_k/p_k}. \quad (22)$$

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

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

The average RTT is then

$$\overline{r} = \sum_{i=1}^{N} r_i \cdot \frac{n_i}{n_{\text{total}}} = \frac{N}{\sum_{k=1}^{N} 1/r_k}. \quad (24)$$

Similarly, the average deviation in RTT is

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

From (19), no timeouts will occur if

$$r_i < \overline{r} + K \cdot \overline{v} \quad \text{for } 1 \le i \le N. \quad (26)$$

E. A Two Path Example

While (26) ultimately determines if timeouts will occur, a simple two path example is now considered to examine how much difference between 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 $r_1 \ge r_2$ and that both are bandwidth dominated. The average RTT is then

$$\overline{r} = \frac{2 \cdot r_1 \cdot r_2}{r_1 + r_2}, \quad (27)$$

and the average deviation in RTT is

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

Then from (26), no timeout will occur if

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

From the assumption $r_1 \ge r_2$, it follows that $r_2 \le \overline{r}$, and (26) is always satisfied for $r_i = r_2$. Therefore, a timeout could only occur when $r_i = r_1$.

If the ratio of RTTs is expressed as κ, where

$$\kappa = \frac{r_1}{r_2} \quad \text{or} \quad r_2 = \frac{r_1}{\kappa}, \quad (30)$$

no timeouts will occur if

$$r_1 < 2 \cdot r_1 \left[\frac{1}{\kappa+1} + \frac{K(\kappa-1)}{(\kappa+1)^2}\right]. \quad (31)$$

This restricts κ to

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

In other words, for two paths using equal packet sizes and the typical value of $K = 4$, the RTTs 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:

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

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 40 Kbps cellular phone modem connection along with a 1 Mbps wireless LAN. In this case the ratio of bandwidths of the two links is 25. Hence, to avoid time-outs, the packet size ratio must be set to of 25/7:1 (or higher) as dictated by constraint (33).

We would like to point out that in many cases it is possible to appropriately size packets sent along different links 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 links in the right proportions so that constraint (33) is satisfied. This fragmenting is slightly more involved than the normal IP fragmentation needed when forwarding between networks with different MTUs.

Note that the restrictions stipulated in (33) apply *only if the links are bandwidth dominated*, i.e. if $p/b >> \tau$ in (1). Propagation and queuing delays at intermediate routers as well as the ACK reception time all contribute to τ. If all packet sizes, including the largest packet size which will be sent on the fastest link, are such that $p/b < \tau$ then the links are no longer bandwidth dominated. In this case, RTTs are dominated by τ which should be about the same, independent of the link used. In other words, the RTTs are approximately equalized thereby making it possible to stripe the connection across all links without incurring timeouts, irrespective of the bandwidth ratios involved.

F. The Effect of Using Smoothed RTT and RTO Values

Note that use of the exact mean values $\bar{r}, \bar{v}$ in (27) and (28) is an idealized approximation. In reality the smoothed estimates for RTT and deviation in RTT from (20) and (21) will play a role in determining what difference in RTTs will cause timeouts. We extracted the TCP retransmission timer management code (henceforth abbreviated 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 timer-code. This sequence was generated by assuming two links with bandwidths b_1 and b_2. Packets were probabilistically split across the two links, where the probabilities corresponded to bandwidth fractions in accordance with (22). Packets sent on the first and second links 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 link it was sent on. If the RTT exceeded the smoothed RTO from the timer-code, defined in (19), it was regarded as a timeout.

The fraction of packets that lead to timeouts is show in Fig. 2 as a function of the ratio of link bandwidths, γ. The curve labeled “Ideal” in the figure corresponds to ideal behavior as predicted by (33). That is, no timeouts occur for bandwidth ratios of $\gamma < 7$ and timeouts occur for 100% of the packets for $\gamma \geq 7$, resulting in a step-function. The curve labeled “Actual” shows what the timer-code did with its default values of $\alpha = 7/8$ and $\beta = 3/4$ for the smoothing coefficients, defined 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. The effect of having these high smoothing coefficients is to track the long-term averages more closely, while ignoring any short-term transients.

As previously mentioned, packet sizes can be adjusted to equalize RTTs. To test this hypothesis, we conducted the same set of experiments with a packet size ratio of 5:1. The results are shown in Fig. 3.

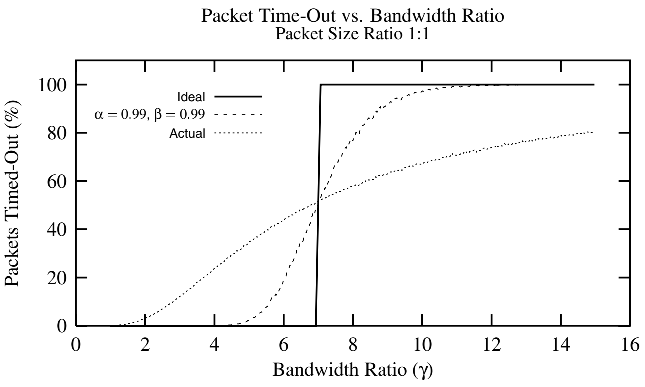

Fig. 2

PACKETS TIMED-OUT FOR PACKET SIZE RATIO 1:1.

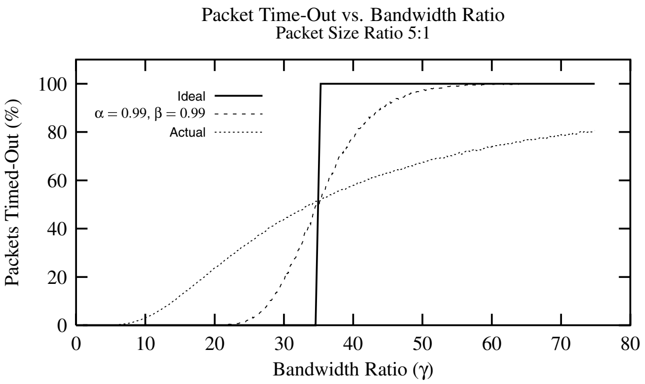

Fig. 3

PACKETS TIMED-OUT FOR PACKET SIZE RATIO 5:1.

Note that both experimental plots in Fig. 3 indicate that the timeout threshold has been scaled by the packet size ratio of 5:1. In other words, the RTO given by (19) is effectively increased by a factor of 5. This is because packets sent on the slower link are 5 times smaller in size than packets sent on the faster link.

G. Avoiding Fast Retransmission

As mentioned at the beginning of this section, the second mechanism that can enable a slow link to drag down a fast link is the fast-retransmission and recovery algorithms found in almost every TCP implementation today. These algorithms are independent of the regular timeout mechanisms.

Fast retransmissions can be avoided by out-of-order transmission. For instance, assume that there are two links between the source and destination with a bandwidth ratio of 4. As per equation (15), the optimal load distribution in this case is 1 packet on the slow link for every 4 packets transmitted on the fast link. If packet 1 is transmitted first on the slow link followed by packets 2,3,4,5 on the fast link, the reception of packet 4 will cause a fast-retransmission of packet 1. The solution to this problem is to send packet number 5 on the slow link first and then transmit packets 1,2,3,4 on the fast link. Assuming that TCP pushes segments down to the network layer in order, the proposed out-of-order transmission can be done transparently by the network layer. Although, the network layer must then understand TCP segment numbers. This scheme also requires sufficiently large sender and receiver windows to buffer the required packets.

H. *TCP Performance Summary*

We conclude this section by summarizing ways of tuning the behavior of TCP. Based on the above discussion it is seen that TCP can be optimized in several ways, including the following.

(i) Set large window sizes to accommodate all the packets when using links with high bandwidth ratios.

(ii) Reduce the packet size p appropriately (to reduce the bandwidth domination of RTTs).

(iii) Set high values for Timeout intervals. As shown above, the default value of K allows for a bandwidth ratio of 7 when equal sized packets are used. A value of $K = 13$ covers links with bandwidth ratios up to 25.

(iv) Use TCP Vegas and 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.

IV. Implementation and Results

We have tested a proof-of-concept implementation where the sender and receiver were connected by two logical links. We implemented the above mentioned tunneling mechanism on a FreeBSD sender and receiver. The "dummynet" facility [33] was used in our experiments. This allows the kernel to route packets through logical links based on specified probabilities.

For our experiments, we created two logical links over a single physical link. We would like to emphasize that it would be just as easy to run the experiment across two different physical links. The logical implementation we chose offered a greater flexibility in controlling both the bandwidth of each link and the probabilities with which each link was chosen, which in turn determined load 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 links. Similarly, 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. 4 and Fig. 5. These Figures show application bandwidth as a function of the raw bandwidth, which refers to the total bandwidth of all the links between the source and the destination. For instance, if there are two links connecting the source and destination, where each link has a bandwidth of 1 Mbps the raw bandwidth is 2 Mbps. TCP application bandwidth was measured using the netperf benchmarking package, `http://www.netperf.org`.

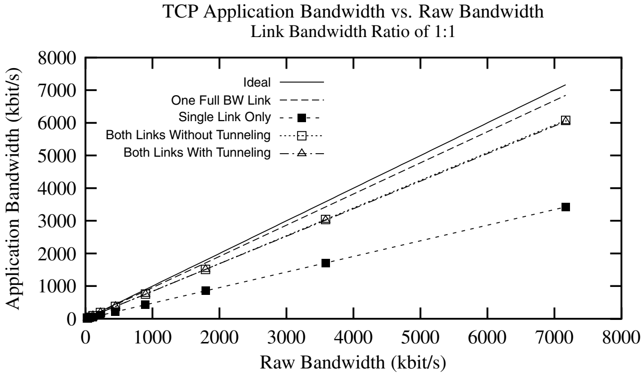

Fig. 4

Application throughput for identical bandwidth links.

Fig. 4 shows the case when the bandwidths of both links are identical. The plot includes 5 curves. The curve labeled "Ideal" is simply the $y = x$ straight line since ideally the application would like the entire raw bandwidth available. However, this is never possible because of headers from various protocol levels, processing overhead, collisions in Ethernet, etc. The second curve labeled "One Full BW Link" shows the TCP application bandwidth when there was a single link with the same raw bandwidth as the abscissa value. For instance, in the above example, instead of configuring two links each having a 1 Mbps capacity, we configured a single link with a 2 Mbps capacity and ran the same TCP application across that link and measured the application level bandwidth. This is slightly less than ideal, as expected, due mainly to headers and processing overhead since the physical network was a private subnet where collisions and congestion were not a factor. The curve labeled "Single Link Only" corresponds to the case when only a single link was used for the connection. Continuing the example scenario, even if 2 links each having a bandwidth of 1 Mbps were available, only one link was used for the TCP connection. This illustrates the performance of a standard TCP connection.

The last two curves, which are almost coincident, correspond to using both links simultaneously, with and without tunneling. The fact that these two curves almost coincide demonstrates that tunneling adds minimal overhead. The plot corresponding to "Both Links With Tunneling" demonstrates the proposed mechanism at work. Note that the application bandwidth of a standard TCP connection is about half of what is achieved when using the proposed tunneling mechanism and routing packets through both links. This demonstrates that the proposed mechanism can, in principle, deliver additional bandwidth to the application level.

Finally, to test the efficacy of the proposed mechanism we carried out experiments on two links having different bandwidths. As per the previous analysis, even if the bandwidths are asymmetric and the links *are* bandwidth dominated, the default value of $K = 4$ in most TCP implementations ideally allows connection striping across links with bandwidth ratio of up to 7 without incurring timeouts. For our experiments we selected a bandwidth ratio of 3:1. The reason for selecting the low ratio, 3 as opposed to 7, was to avoid fast retransmissions which are independent of timeout prevention. The results are illustrated in Fig. 5, with the curves being analogous to those in Fig. 4.

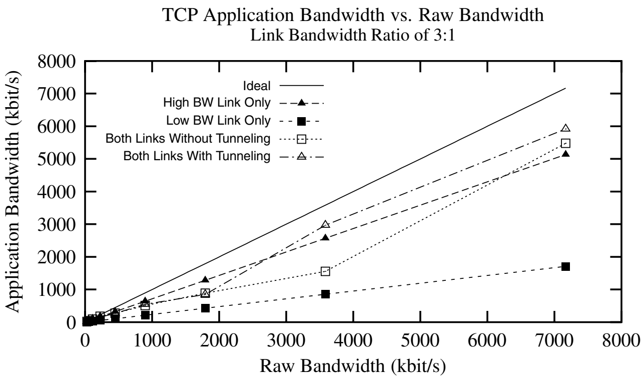

Fig. 5

APPLICATION THROUGHPUT FOR A BANDWIDTH RATIO OF 3:1.

The data shows that despite a 3:1 bandwidth disparity, simultaneous use of both links can yield better performance. We would like to point out that there are many data points below 1000 Kbps (starting from 14 Kbps, 28, 33, 40, 56, ...) that are grouped in a small interval on the *X* axis. For data points at low raw bandwidth values, about 3 Mbps and below, timeouts do occur. This is consistent with the data from Fig. 2 and Fig. 3. This result is due to the effect of using smoothed RTTs as previously explained. The timeouts do cause the slower link to drag down the faster link somewhat.

For raw bandwidths above about 3 Mbps, the RTTs of the links drop to sufficiently low values, which in turn causes the RTO values to drop below 1 second. The FreeBSD kernel imposes a lower limit of 1 second for RTO, below which timeouts will not occur. Note that the links are still bandwidth dominated, i.e. the RTT values are still dominated by the bandwidth, but the timeout values are now sufficiently large to accommodate the variation in RTT, which prevents timeouts. The net result is that the connection striping across two links provides higher bandwidth than using the faster link by itself, demonstrating that the proposed mechanism works even for bandwidth dominated links, as long as TCP timeouts can be mitigated.

V. CONCLUSIONS

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

Future work includes an implementation of all the TCP modifications and optimizations mentioned above and more detailed simulations in wireless and mobile scenarios. Another approach is to implement the load sharing capability in SCTP and compare it with the mechanism proposed herein. Lastly, dynamically adding or deleting IP addresses from a connection involving end-points with multiple interfaces could lead to security hazards which warrant further investigation.

REFERENCES

[1] G. Malkin, RFC 2701: Multi-link Multi-node PPP Bundle Discovery Protocol, Sept. 1999.

[2] K. Sklower, B. Lloyd, G. McGrego, D. Carr, and T. Coradetti, RFC 1990: The PPP Multilink Protocol (MP), August 1996.

[3] K. Sklower, B. Lloyd, G. McGrego, and D. Carr, RFC 1717: The PPP Multilink Protocol (MP), November 1994.

[4] W. Simpson, Editor, RFC 1661 : The Point-to-Point Protocol (PPP).

[5] W. Simpson, Editor, RFC 1662 : PPP in HDLC-like Framing, July 1994.

[6] D. Rand, RFC 1663 : PPP Reliable Transmission, July 1994.

[7] R. Ogier, V. Ruenburg, and N. Shacham, "Distributed algorithms for computing shortest pairs of disjoint paths," *IEEE Transactions on Information Theory*, vol. 39, no. 3, pp. 443–455, Mar. 1993.

[8] I. Cidon, R. Rom, and Y. Shavim, "Analysis of multi-path routing," *IEEE/ACM Transactions on Networking*, vol. 7, no. 6, pp. 885–896, Dec. 1999.

[9] N. Taft-Plotkin, B. Bellur, and R. Ogier, "Quality of Service Routing Using Maximally Disjoint Paths," in *Proceedings of the IEEE IWQoS'99, London, UK.*, June 1999, pp. 119–128.

[10] A. Nasipuri and S. R. Das, "On-Demand Multipath Routing for Ad-Hoc Networks," in *Proceedings of the IEEE ICCCN,99, Boston*, October 1999, pp. 64–70.

[11] J Raju and J.J. Garcia-Luna-Aceves, "A New Approach to On-Demand Multipath Routing," in *Proceedings of the IEEE ICCCN,99, Boston*, October 1999, pp. 522–527.

[12] M.R. Pearlman, Z.J. Haas, P. Scholander, and S.S. Tabrizi, "On the Impact of Alternate Path Routing for Load Balancing in Mobile Ad Hoc Networks," in *Proceedings of ACM MobiHOC'00, Boston*, August 2000, pp. 119–128.

[13] Mario Gerla Sung-Ju Lee, "Split Multipath Routing with Maximally Disjoint Paths in Ad hoc Networks ," in *Proceedings of ICC'01, Helsinki, Finland*, June 2001.

[14] 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.

[15] A. Snoeren, "Adaptive Inverse Multiplexing for Wide-Area Wirelss Networks," in *IEEE Globecom*, 1999, pp. 1665–1672.

[16] H. Adiseshu and G. Parulkar and G. Varghese, "A Reliable and Scalable Striping Protocol," in *ACM SIGCOMM*, 1996, pp. 131–141.

[17] Darrell C. Anderson, Jeffrey S. Chase, and Amin M. Vahdat, "Interposed request routing for scalable network storage," in *Proceedings of the Fourth Symposium on Operating System Design and Implementation (OSDI)*, October 2000.

[18] "Sun StorEdge Network Data Replicator White Paper," http://www.sun.com/storage/white-papers/sndr.html.

[19] "Filer Deployment Strategies for Evolving LAN Topologies," http://www.netapp.com/tech_library/3009.html.

[20] Randy Haagens, "iscsi requirements," http://www.ietf.org/proceedings/00jul/SLIDES/ips-iscsi-reqs.pdf.

[21] Cisco Systems, http://www.cisco.com/warp/public/779/largeent/learn/technologies/fast_echannel.html.

[22] "SCTP site," www.sctp.org.

[23] "SCTP site in Germany," www.sctp.de.

[24] "SCTP for beginners," http://tdrwww.exp-math.uni-essen.de/pages/forschung/sctp_fb.

[25] "SCTP: An Overview," http://sctp.chicago.il.us/sctpoverview.html.

[26] "Protocol engineering lab at university of delaware cis dept.," http://www.cis.udel.edu/ iyengar/research/SCTP.

[27] R. R. Stewart, M. A. Ramalho et. al., "SCTP Extensions for Dynamic Reconfiguration of IP Addresses and Enforcement of Flow and Message Limits," http://www.ietf.org/internet-drafts/draft-ietf-tsvwg-addip-sctp-02.txt, June 2001.

[28] Perkins, C., RFC 2002: IP Mobility Support, October 1996.

[29] Perkins, C., RFC 2003: IP Encapsulation within IP, October 1996.

[30] Perkins, C., RFC 2004: Minimal Encapsulation within IP, October 1996.

[31] Solomon, J., RFC 2005: Applicability Statement for IP Mobility Support, October 1996.

[32] V. Jacobson, "Modified TCP Congestion Avoidance Algorithm," end2end interest group mailing list, April 1990.

[33] L. Rizzo, "Dummynet: a simple approach to the evaluation of network protocols," *ACM Computer Communication Review*, vol. 27, no. 1, pp. 31–41, Jan. 1997.