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

Dhananjay S. Phatak, and 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, cell-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 (congestion, traffic patterns, ...), cost, power consumption, battery life, etc. 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 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 bandwidth of multiple IP links by splitting data flow across multiple 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 when the underlying links have widely different bandwidths and delays thereby demonstrating that the proposed mechanism is applicable in many practical scenarios. Several interesting directions for future work have been identified.

I. Introduction

As wireless networks, services and computing continue their explosive growth, it is clear that multiple transport mechanisms will be available. For instance a mobile host might have access to the Internet via multiple services such as Bluetooth, wireless LAN (802.11, Airport LANs, Ricochet type wireless access provider ...), cell phone-modems, etc. Selecting which service (transport) to use for data transfer is a dynamic optimization problem which should track attributes like bandwidth/throughput, latency, jitter, Quality of Service requirements, cost, power consumption, residual battery life, interference, traffic patterns, etc. Wireless bandwidth is a scarce resource unlikely to keep on improving in synch with the rapid growth in bandwidth available via wired networks. Hence it is likely that users will frequently want to simultaneously stream data across multiple interfaces in order to squeeze every last bit of bandwidth available. Hence, this paper investigates simultaneous use of multiple interfaces (such as Cell-phone-modems, Bluetooth and 802.11) to enhance bandwidth available to a wireless node. An immediate side advantage is reliability (if one interface goes down or one LAN is congested, continue using the other LAN).

In some cases (for example military applications), multiple identical or similar network access mechanisms are provided for the sake of reliability in hostile environments. In such cases it would be nice to stream (distinct) data packets across all interfaces simultaneously whenever possible (even though the original intent behind providing the redundant network access capability is reliability rather than performance). This would be the “performance-enhancement” mode. When the reliability becomes an issue, the data transfer could switch into the “reliability-first” mode wherein two (or more in general) identical copies of the data stream might be sent across two (or more) access points in the hope that at least one makes it through. Ideally there would be no *hard* cutoff between the performance-enhancement and reliability-first modes. A mixture could exist (only transfer lost packets in duplicate across both interfaces). Furthermore, the switching between the performance-enhancement and reliability-first modes should happen dynamically in reaction to the environment.

A. *Problem definition*

Figure 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 style airport LAN (say interface A1), Bluetooth LAN (A2) and

CSEE Dept. University of Maryland Baltimore County (UMBC), 1000 Hilltop Circle, Baltimore, MD 21250. Email: {phatak,tgoff1}@umbc.edu

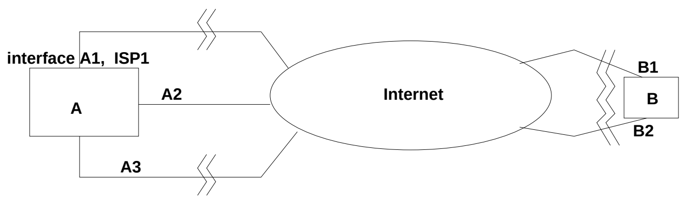

Fig. 1 : Hosts with multiple network connections

a cell-phone modem (A3). In general the IP addresses assigned to interfaces A1, A2, and A3 will be controlled by separate Service Providers (ISP)s who in-turn might implement firewalling to various degrees.

Suppose A wants to stream data to B across multiple interfaces at the same time to enhance bandwidth. How should this be done ? Should the data stream be split into multiple streams at the application level ? In this case the application should open multiple connections across different interfaces and be responsible for splitting data stream on the server end and merging it correctly at the client end. This might be too cumbersome and static (number of connections might have to be decided in advance, to figure out how many flows to split the traffic into). Should the splitting be done at transport layer ? or at the network layer ? In essence we are looking at connection striping for bandwidth aggregation across multiple IP interfaces.

B. Related Work

Bandwidth aggregation across multiple channels has been considered in the literature in different contexts (distinct from the one being considered in this paper) and for different scenarios/applications. For instance, Multi-link PPP [1], [2], [3] is intended to aggregate multiple **logical** data channels into one. Note that PPP was intended to operate at the **Data link** layer, **below IP level** [4], [5], [6]. Hence the multi-link PPP extension also bundles multiple channels into a single link, at the data-link level, **below** the IP level. In the scenario under consideration, the links have different IP addresses assigned and controlled by completely independent Internet Service Providers (ISPs). It is highly unlikely that totally independent ISPs will allow the links under their control to be bundled into “one logical link” for the sake of arbitrary users. Thus, PPP would have to run on-top-of IP links which is not the right approach.

Multi-path and QoS routing has been considered in wired as well as wireless networks (a sampling can be found in [7], [8], [9], [10], [11], [12], [13], [14]). 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 find distinct routes to the destination via multiple neighbors in the listening range from 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 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/file servers (for instance [15], [16], [17], [18], etc.). Typical storage/server architecture is confined to be within a LAN with 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 [19] product provides bandwidth aggregation across multiple Ethernet links. This product targets the extreme high bandwidth requirements between a data-server node and a node which receives and serves queries (in general queries could be sent to one of multiple nodes that fetch data from a database server). In this scenario, depending on the number of users making queries, the instantaneous bandwidth demand (between the query server and the database server) could be very high and Cisco’s product targets such cases. Likewise, a recent IETF draft from HP storage systems group [18] 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 [20], [21], [22], [23], [24]) comes closest to solving the problem at hand. SCTP is a new transport level protocol which supports multi-streaming and multi-homing (multiple IP addresses for source/destination). A recent extension [25] 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 for redundancy purposes only [23]. A single address is chosen as the primary address and is used as the destination for all DATA chunks for normal transmission. Retransmitted DATA chunks use the alternate address(es) to improve the probability of reaching the remote endpoint, while continued failure to send to the primary address ultimately results in the decision to transmit all DATA chunks to the alternate until heartbeats can reestablish the reachability of the primary path.

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 (A local company which is a commercial vendor of wireless services is sponsoring part of this work because their engineers foresee this multiple interface issue as one of the problems they'll have to face in the near future). 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 several advantages:

(i) To transport, application and higher layers, the data stream looks like a single flow so all existing applications can benefit transparently (without having to be re-written to explicitly do the stream-splitting themselves).

(ii) We don't need to worry about re-doing this for each different type of MAC protocol, ex. 802.11, Bluetooth, etc. Since most protocols are likely to integrate TCP/IP functionality, it is appropriate to assume IP-level support and implement our scheme at the IP level.
Note that even if the actual network layer is not IP but something different, we propose to implement aggregation at that level (between Link layer and Transport layer).

(iii) It can be highly dynamic: if the route through an interface becomes congested, simply don't send packets on that interface. In the following we show some problems that can arise with this approach.

(iv) It can enhance reliability: if one interface goes down, use the other interface.

(v) Likewise, once such a mechanism is available, it could be employed to optimize other parameters such as power consumption, latency, jitter, etc.

Since TCP is more predominantly used than UDP, we illustrate our method for TCP (it is also valid for UDP flows). For the purpose of illustration assume that node A Figure 1 is the source and node B is the destination. IP address associated with interface A*i* is denoted as "Addr(A*i*)". Likewise IP address associated with interface B*i* is denoted as "Addr(B*i*)". Assume that A knows only the IP address associated with the B1 interface of destination B (this is true in practice. Name servers typically don't return multiple addresses in response to a query). For the purpose of illustration assume that A will always send data to B on interface B1 (we discuss a bit later the more general case where both A and B use multiple interfaces simultaneously). In this case, the initial SYN packet for the TCP connection from A to B goes via interfaces A1 and B1. So the TCP daemon at host B logs the source address of the connection to be that associated with interface A1, (i.e., Addr(A1)). Likewise the destination address field in the connection identifier is filled with Addr(B1). Now, to use more bandwidth, A decides to route some packets through say interface A2. Following problems arise:
[i] The source address on any packet is the address associated with the interface on which the packet goes out. Hence the packets sent through interface A2 will contain Addr(A2) as their source address (which is different from Addr(A1)). At the destination B, TCP sees a packet with the right port number and with a valid sequence number in the receiver window. However the source address is Addr(A2) and not Addr(A1). Most TCP implementations drop such packets. So it is desirable to maintain the same source address (of the interface on which the connection was established).
[ii] To achieve this if we put out a packet with source address Addr(A1)) on interface A2, the router maintained by the ISP will most likely drop such a packet. This "denying source routed" packets policy is the most commonly employed packet filtering mechanism in order to safeguard against IP spoofing. The problem is compounded if B wants to receive data on multiple interfaces.

Our solution to all these problems is to employ **tunneling**. At the sender side (A), the transport layer, assembles all packets as if they were going through A1 and addressed to B1. The packets going out on interface A2 get encapsulated in a new packet with an extra header with destination Address B1 and source address A2. The protocol field in the outer header should be IP-in-IP (also used for tunneling in mobile IP standard and implementations [26], [27], [28], [29]). At the destination after seeing the protocol field "IP-in-IP" the outer header is stripped and the original packet with the proper source and destination addresses is delivered up the stack to TCP in a transparent manner. TCP process at B is unaware of the fact that the packet was tunneled through another interface A2 at the source host A.

Note that the exact same tunneling mechanism can be used when B is accepting data on multiple interfaces. For instance to send a packet on interfaces A3 and B2 the original packet (with source, destination addresses (Addr(A1),Addr(B1)) is encapsulated in another packet with the addresses (Addr(A3),Addr(B2)). The IP process at B strips the outer header, then realizes that the inside packet is for an existing TCP connection on interface B1 and passes it on transparently to TCP.

Depending upon how many firewalls the packet needs to traverse and how extensive and restrictive the firewall policies are, more than one encapsulation headers 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

(1) How do source and destination learn each other's multiple IP addresses ?

(2) Many radio technologies operate in the same band (for instance Bluetooth and 802.11). So trying to simultaneously use Bluetooth and 802.11 might lead to more interference among the two and lead to performance degradation

(3) IP-in-IP encapsulation overhead ?

(4) TCP performance: TCP was designed to work well across a single link not multiple links. So if the bandwidth/delay characteristics of the links are radically different, the packets sent on slow link(s) will cause timeouts and/or fast-retransmits causing the sender to throttle it's transmission rates (via congestion window reduction, slow start, and other congestion avoidance mechanisms buried in TCP). The net effect could be the slow link "dragging down" the fast link. This would defeat the purpose behind the simultaneous use of multiple interfaces, leaving the use of only the single (faster) link by itself as the optimal solution.

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 next section).

The first issue is how to make A and B aware of each other's multiple IP addresses. For this purpose, the IP and/or TCP headers (and their processing) would have to be modified a bit. This is similar to the solution in the SCTP protocol which has a different header (different from normal TCP header) that allows for a listing of multiple addresses. While most such modifications interfere with inter-operability 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, cell-phone modems do not. Neither is the wideband CDMA projected to occupy the same ISM band. In these cases, the radio-interference issue is non-existent. With the rapid growth in wireless technologies and services, the chances that all transport conduits will end up using same technology and/or spectrum are small. We would like to point out that our solution is at the IP level and therefore applies even to wired scenarios as well: a typical desktop at a campus today is likely to 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 (if for instance, the wired link had too much traffic and collisions).

The third issue deals with the IP-in-IP overhead. Experimental data (see section 4) shows that this overhead is negligible. Note that packets sent on the primary path (i.e., the path over which the TCP SYN packet was sent, establishing the source and destination addresses) need not be tunneled and do not have the IP-in-IP encapsulation overhead at all. Assuming that this primary path is the fastest (i.e., has highest bandwidth), majority of packets are free of the IP-in-IP overhead. Furthermore if tunneling is not done, the slower link(s) don't get utilized at all. By employing the proposed scheme, one can hope to utilize at least a part of the spare capacity of the slower link(s). Thus any added bandwidth utilization is available for free (simply not realizable by the conventional single link TCP connection). Hence, the term "overhead" is a misnomer. Furthermore minimal encapsulation proposed in the mobile IP standard [28] along with header compression techniques can be employed as well, which takes care of the IP-in-IP overhead issue.

That leaves the last question in the list above, which is addressed in the next 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 does not care about retransmits (unlike TCP). Hence any asymmetry in bandwidths and/or latency across different links will have a significantly lower impact on the performance of UDP applications (than on TCP applications). Thus, UDP is the simpler case and therefore we have concentrated most of our efforts on the harder and more interesting case: splitting a TCP connection across multiple IP links.

As mentioned above, with TCP, using two links together at the network layer can sometimes lead to worse performance than using the higher bandwidth (i.e., fatter) pipe alone. This can happen due to two main mechanisms:

(1) The (bandwidth, delay) mismatch between the links causes packets sent on the lower capacity link to arrive so late that by the time they reach the destination, sender-side TCP has already timed out. In this case the late arriving packet needs to be re-transmitted (on the fatter pipe). If this were the only drawback one would expect to see only a slight worsening of performance (compared to using the fatter pipe alone). Unfortunately, with each timeout, TCP drastically scales back the congestion window and invokes slow-start, thereby under-utilizing the capacity of the fatter pipe.
(2) Most TCP implementations include the "fast-retransmit/recovery" mechanism [30]. Even if the TCP timeout values were set high to prevent the scenario stated in item (1), fast-retransmit still poses an independent problem. For the purpose of illustration, suppose the ratio of the round-trip-time (RTT) delays of the two links being used is 4 and suppose the 1st packet is transmitted on the slow link and the next 4 packets are transmitted on the fatter/faster pipe. The receipt of packets 2,3,4 at the destination will cause the receiver side TCP to signal a fast-retransmission of packet 1, thereby wasting the prior transmission on the slow link. Wore yet, the recovery phase following a fast retransmission scales back the congestion window.

Next, we address both these issue. First we consider the prevention of timeouts. After a brief overview of the problem in the context of two links, we present 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's 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 RTT delays for packets sent across the fatter and thinner pipes are substantially different. The RTT can be split into two components:

$$\mathrm{RTT} = \frac{\mathrm{p}}{\mathrm{b}} + \tau \qquad \text{where } p = \text{packet size}, \tag{1}$$

b = bandwidth of the link and τ includes all other delays (signal propagation delays on all links, processing delays at all intermediate routers, all delays associated with the ACK packet, etc). If the RTTs (for packets sent across the two links) are dominated by their $\frac{\mathrm{p}}{\mathrm{b}}$ ratios then a bandwidth disparity between the links causes a big disparity in the RTT delays 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 (as might be the case if the number of hops is sufficiently large, or the transmission distance and hence delay is sufficiently large), then a bandwidth disparity does not lead to a disparity in the RTT delays In such cases, we can expect to see performance gains.

As an example, consider two links with bandwidths of 160 Kbps and 40Kbps, i.e., a bandwidth ratio of 4:1. For full Ethernet sized packets (approximately 1560 bytes per packet) being sent between adjacent nodes (no hops), the "other delays" τ in equation (1) are negligible (order of microsecond whereas p/b ratios are 100s of milliseconds). Hence RTT is 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 bandwidth ratio of 4:1 but actual bandwidths of say 100 Mbps and 25 Mbps and where the destination is several hops away. Now the p/b ratio is of the order of 100s of microseconds whereas τ can be milliseconds. In this case, the RTTs are dominated by τ so that using the two links together can be expected to yield better overall performance than using the fatter pipe alone.

Note that reducing the packet size P will help mitigate the effect of the (p/b) ratio (and hence the Bandwidth) on the RTT. In fact the packets queued for transmission on the thinner pipe can be dynamically fragmented to equalize the p/b ratios of both links. Next we present present a quantitative analysis of the problem.

A.1 *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 round-trip-time (RTT) of the i-th packet becomes

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

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

note that $\tau = c_{AB_k}(t) + \delta_i + l_{BA_{k'}}(y_i, t + \delta_i)$

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 one single path on which the TCP SYN packet got sent, thereby establishing the source and destination addresses. 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

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

A.2 *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, so that no link is 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

$$L_i(p_i,n_i) \approx b_i^{-1} \cdot p_i \cdot n_i + c_i + \Delta_{\text{ack}} \tag{6}$$

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 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 $f_i = \frac{n_i \cdot p_i}{D}$, (7) becomes

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

$$\text{where} \quad f_1 + f_2 + \cdots + f_N = 1 \tag{9}$$

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

$$\mathbf{Bf} = \mathbf{c}. \quad \text{Here } \mathbf{B} \text{ is an } N \times N \text{ matrix whose elements} \tag{10}$$

are defined by

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

$$\mathbf{B}_{ij} = 1 \quad \text{For row } i = N, \tag{11}$$

$\mathbf{f}$ 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

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

This makes (10)

$$\underbrace{\begin{bmatrix} b_1^{-1} & -b_2^{-1} & & & & \\ & b_2^{-1} & -b_3^{-1} & & \mathbf{0} & \\ & & b_3^{-1} & -b_4^{-1} & & \\ & \mathbf{0} & & \ddots & \ddots & \\ & & & & b_{N-1}^{-1} & -b_N^{-1} \\ 1 & \cdots & \cdots & \cdots & \cdots & 1 \end{bmatrix}}_{\mathbf{B}} \underbrace{\begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_{N-1} \\ f_N \end{bmatrix}}_{\mathbf{f}} = \underbrace{\begin{bmatrix} \frac{c_2 - c_1}{D} \\ \frac{c_3 - c_2}{D} \\ \frac{c_4 - c_3}{D} \\ \vdots \\ \frac{c_N - c_{N-1}}{D} \\ 1 \end{bmatrix}}_{\mathbf{c}}. \tag{12}$$

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

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

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

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

This means 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

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

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

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

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)to minimize the overall transfer time.

A.3 *Conditions for Preventing TCP Retransmission Timeouts*

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

$$RTO_i = R_i + K \cdot V_i, \tag{18}$$

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.

$$R_i = \alpha \cdot R_{i-1} + (1-\alpha) rtt_i \tag{19}$$

$$V_i = \beta \cdot V_{i-1} + (1-\beta)|rtt_i - R_i|, \tag{20}$$

where rtt_i is the sampled RTT of the i-th packet. Equations (19) and (20) 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 (14). The average RTT is then

$$\bar{r} = \sum_{i=1}^{N} r_i \cdot f_i = \mathbf{rf}, \tag{21}$$

where $\mathbf{r}$ is a row vector of RTTs for each path from (4) and $\mathbf{f}$ is given by (13). Similarly, the average deviation in RTT is

$$\bar{v} = \mathbf{r}'\mathbf{f}, \tag{22}$$

where $\mathbf{r}'$ is a row vector defined by $\mathbf{r}'_i = |r_i - \bar{r}|$.

From (18), no timeouts will occur if

$$r_i < \bar{r} + K \cdot \bar{v} \quad \text{for } 1 \le i \le N. \tag{23}$$

A.4 *Two Path Example*

While (23) ultimately determines if timeouts will occur, a simple two path example is now considered to examine how much 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 $r_1 \ge r_2$. The average RTT is then

$$\bar{r} = \frac{r_1 \cdot b_1 + r_2 \cdot b_2}{b_1 + b_2}, \tag{24}$$

and the average deviation in RTT is

$$\bar{v} = \frac{2 \cdot b_1 \cdot b_2 (r_1 - r_2)}{(b_1 + b_2)^2}. \tag{25}$$

Then from (23), no timeout will occur if

$$r_i < \frac{r_1 \cdot b_1 + r_2 \cdot b_2}{b_1 + b_2} + \frac{2 \cdot K \cdot b_1 \cdot b_2 (r_1 - r_2)}{(b_1 + b_2)^2} \quad \text{for } r_i \in \{r_1, r_2\}. \tag{26}$$

From the assumption $r_1 \ge r_2$, it follows that $r_2 \le \bar{r}$, and (23) is always satisfied for $r_i = r_2$. Therefore, a timeout could only occur when $r_i = r_1$.

Assuming the RTTs of the paths are bandwidth dominated, $r_i \approx \frac{p_i}{b_i}$, and letting

$$\kappa = \frac{r_1}{r_2} = \frac{b_2 \cdot p_1}{b_1 \cdot p_2} \ge 1, \tag{27}$$

the RTT and bandwidth of the second path can be expressed as

$$r_2 = \frac{r_1}{\kappa} \quad \text{and} \quad b_2 = \frac{\kappa \cdot b_1 \cdot p_2}{p_1}. \tag{28}$$

Using these relationships, no timeout will occur if

$$r_1 < r1 \left[\frac{1+\lambda}{1+\kappa \cdot \lambda} + \frac{2 \cdot K \cdot \lambda (\kappa - 1)}{(1 + \kappa \cdot \lambda)^2} \right] \quad \text{where } \lambda = \frac{p_2}{p_1}. \tag{29}$$

This restricts κ to

$$1 < \kappa < \frac{p_1}{p_2}(2 \cdot K - 1) \tag{30}$$

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. Equation (30) in turn implies

$$1 < \gamma = \frac{b2}{b1} < (2 \cdot K - 1) \tag{31}$$

If the paths are bandwidths dominated, constraint (31) implies that a bandwidth ratio of up-to 7 can be tolerated without incurring timeouts, (given the default value $K = 4$ in all TCP implementations).
In case the bandwidth ratio of the two links is higher (for instance a 40Kbps cell-phone modem versus about 1 Mbps wireless LAN link yielding a bandwidth ratio of 25) the K value can be increased appropriately. $K = 13$ covers the highly asymmetric combination of cell-phone and wireless LAN links.

Note that the restrictions stipulated in constraint (31) above apply *only if the links are bandwidth dominated*, i.e., if $\frac{p}{b} >> \tau$ in equation (1). Propagation delays, queuing delays at intermediate routers and the ack reception time all contribute to τ. If all packet sizes (including the largest packet size which will be used on the fattest pipe) are such that $\frac{p}{b} < \tau$ then the links are no longer bandwidth dominated. In this case, RTTs are dominated by τ which is same no matter which link is used. In other words, the RTTs are approximately equalized, thereby making it possible to stripe the connection without incurring timeouts irrespective of the bandwidth ratios of the links involved.

A.5 *Effect of Using Smoothed RTT and RTO values*

Note that use of exact mean values $\bar{r}, \bar{v}$ (in equations (24),(25)) is an idealized approximation. In reality the smoothed estimates for RTT and deviation in RTT from (19) and (20) will play a role in determining what difference in RTTs will cause timeouts. To evaluate the effect of smoothing, we did a set of experiments.

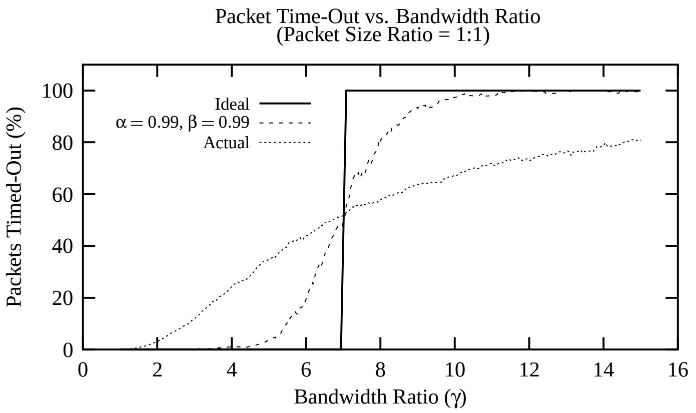

Fig. 2 : Fraction of packets timed out as a function of the bandwidth ratio of the links γ, for packet-size ratio $\lambda = 1$.

We extracted the TCP retransmission timer management code (henceforth abbreviated timer-code) from the (real) FreeBSD kernel and encapsulated it in a simulator which hands it a large sequence (100000 samples) of RTT values. The timer-code provides the smoothed values of RTT and the deviation in RTT which are then used to determine the number of timeouts that are plotted on the Y axes in Figures 2 and 3. The sequence of RTT values is generated by assuming that there are two links with bandwidths b_1 and b_2. Packets are probabilistically split across the two links where the probabilities correspond to bandwidth fractions in accordance with equation (14) (i.e., $b_1/(b_1+b_2)$ and $b_2/(b_1+b_2)$). Packets on link 1 are assumed to have a size p_1 and those on link 2 are assumed to have a size p_2 and RTTs values are based on the bandwidths and packet sizes. If the RTT exceeds the current smoothed value of RTO (defined in (18)) returned by the timer-code, it is deemed to cause a timeout. The fraction of packets that lead to timeouts is plotted as a function of the ratio γ of link bandwidths. The plot labeled “ideal” in Figure 2 corresponds to ideal behavior as predicted by relation (31): no timeouts for bandwidth ratio $\gamma < 7$ and 100% timeouts for $\gamma \geq 7$, resulting in a step-function. The plot labeled “actual” shows what the timer-code does with it’s default values of $\alpha = \frac{7}{8}$ and $\beta = 0.75$ for the smoothing coefficients (defined in equations (19), (20)). The plot shows that using smoothed RTT values for RTO estimation causes more timeouts than predicted by the ideal curve demonstrating that the way smoothing is done has a big impact. To further clarify the impact of smoothing we have also included the plot labeled $\alpha = 0.99\beta = 0.99$ which approximates the idealized step-function a lot better than the default values.

As mentioned at the beginning of this section, packet sizes can be adjusted to equalize RTTs. To test this hypothesis, we conducted the same set of experiments with packet-size ratio $\lambda = 5$, the results are shown in Figure 3

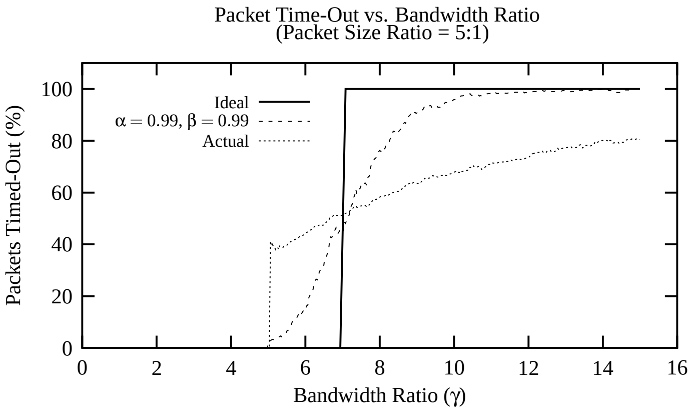

Fig. 3 : Fraction of packets timed out as a function of the bandwidth ratio of the links γ, for packet-size ratio $\lambda = 5$

Note that both the experimental plots in Figure 3 indicate a complete absence of timeouts below $\gamma = 5 = 1/\lambda$. For $\gamma \leq 5$ the RTT on the slower link $\leq$ RTT on faster link (since packets on the slower link are 5 times smaller in size) and hence will not cause timeouts. The interesting point is that this ideal behavior is realized despite using the smoothed versions of RTT and the deviation in RTT. The sharp rise beyond $\gamma = 5$ can be explained as follows: at $\gamma = 5$ the RTTs for both links are approximately equal so that the deviation in RTT is almost 0. This sets up a very tight tolerance around the current smoothed value of RTT which is dominated by the RTT of the faster link (since 5 times more packets go out on that link). Once γ exceeds 5 even by a small amount, the RTTs on the slower link are larger than the RTT values on the faster link and cause timeout because deviation-tolerance is very small.

B. 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 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 and that the bandwidth ratio of the links is 4. As per equation (14), 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. This scheme also requires sufficiently large sender and receiver windows to buffer the required number of packets.

C. TCP Performance Summary

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

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

(2) Reduce the packet size P appropriately (to reduce the bandwidth domination of RTTs).

(3) Set high values for Timeout intervals. As shown above, the default value of K allows for a bandwidth ratio of 7. $K = 13$ covers links with bandwidth ratios up to 25.

(4) Use TCP Vegas and other enhanced TCP versions that mitigate the effect of slow-start(s).

(5) Allow out-of-order sending of packets 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 interoperate against any standard TCP receiver.

IV. Implementation and Results

We have tested a proof-of-concept implementation where the sender and receiver are connected by two links. We implemented the above mentioned tunneling mechanism on a FreeBSD sender and receiver. FreeBSD has powerful networking options: the dummynet facility [31] provided by FreeBSD was used in our experiments. The kernel lets the packets to be routed through one route or the other based on some primitive policies (weighted queuing where one can specify *probabilities* with which a packet can go out on each of the interfaces). While these policies are not as dynamic as they could be, they are sufficient for a proof-of-concept implementation. For our experiments, we created two "logical" links over one physical link and configured their bandwidth using the "dummynet" mechanism in FreeBSD. We would like to emphasize that it is equally easy to run the experiment across two physically different links. Implementing it this way offers a lot more flexibility over controlling
(i) bandwidth
(ii) Probabilities (which in turn determines load distribution)

FreeBSDS's Firewalling rules were exploited on the sender side in order to achieve proper routing across the two links and to un-tunnel at the receiver. Further details regarding the experimental setup have been omitted for the sake of brevity.

The experimental results are illustrated in Figures 4–5.

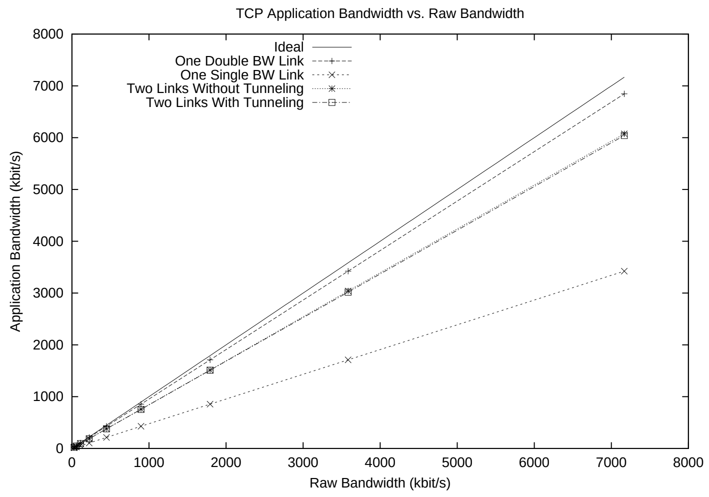

Fig. 4 : Application throughput as a function of raw bandwidth, when the two links have identical bandwidths

The 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 a source to the destination and each link has a bandwidth of 1 Mbps, the raw bandwidth is 2 Mbps). The Y axis shows TCP application bandwidth (as measured by netperf, http://www.netperf.org).

Figure 4 shows the case when the bandwidths of both the links are identical. The figure includes 5 plots. The plot labeled “Ideal” is simply the $y = x$ straight line because ideally the application would like to have the entire raw bandwidth. This is not possible because of the overhead (associated with extra bytes added on as headers at TCP, IP and Link layers and their processing), collisions in the Ethernet, etc. The second plot labeled “One Double BW link” shows the TCP application bandwidth if 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 with 1 Mbps capacity, we configured a single link with 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 header generation/transmission/processing overhead. (the physical network was a private subnet so that collisions and congestion were not a factor). The plot labeled “one single BW link” corresponds to not using multiple interfaces at the same time. For instance, in the example scenario even if 2 interfaces each with a bandwidth of 1 Mbps are available, only a single one is used for the TCP connection (which is how TCP operates today, it cannot avail itself of multiple interfaces if they have distinct IP addresses ...).

The last two plots (which are almost coincident) correspond to two links with and without tunneling. They illustrate the performance when both links are used. The plot corresponding to “two links with tunneling” demonstrates the proposed mechanism at work. Note that the application bandwidth over “one single BW link” is about half of what is achieved by using the proposed tunneling mechanism and routing packets through both links. This demonstrates that the proposed mechanism can, in principle delivers additional bandwidth to the application level.

The next question is tunneling (i.e., extra header) overhead. The fact that the two plots (two links with and without tunneling) almost coincide demonstrates that tunneling leads to minimal extra overhead. The plot labeled “two links with tunneling” represents the case where the two links were configured as separate logical links with distinct IP addresses. As mentioned above, tunneling was required in order to use both links in this case (IP within IP encapsulation was done for each packet on one of the links).

The plot labeled “two links without tunneling” corresponds to the case where the two logical links were configured as before, but both had the same IP address, thereby obviating the need for tunneling. This was done to isolate the overhead due to tunneling (IP over IP encapsulation). As the Figure shows, the tunneling overhead is negligible demonstrating that the proposed mechanism works nicely.

Finally, to test the efficacy of the proposed mechanism we carried out experiments on two links with different bandwidths. As per the analysis in the previous section, even if the bandwidths are asymmetric and the links *are bandwidth dominated*, the default value of $K = 4$ in most TCP implementations allows connection striping across links with bandwidth ratio of up-to 7 without incurring timeouts. For the experiments we selected a bandwidth ratio of 3:1. The reason for selecting the low ratio (3 as opposed to 7) is to avoid the fast retransmissions which are independent of the timeout preventions (we are currently implementing the out-of-order sending mechanism which is required to prevent fast-retransmissions for link-bandwidth ratios higher than 3:1). The results are illustrated in Figure 5 (the plots in this figure are analogous to those in Figure 4.)

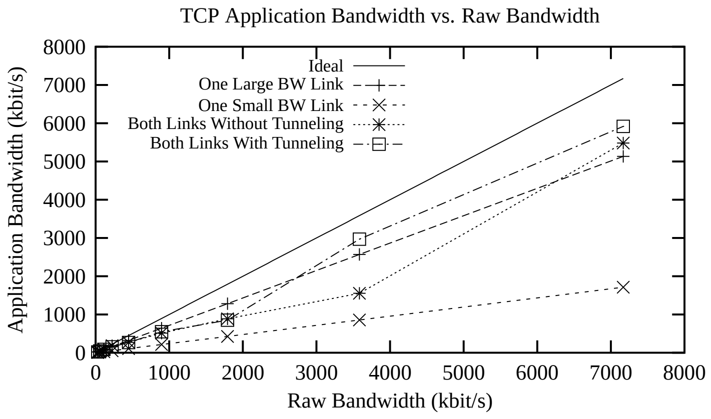

Fig. 5 : Application throughput as a function of raw bandwidth, when the two links have a bandwidth ratio of 3:1

The data shows that despite a 3:1 bandwidth disparity, simultaneous use of both links can yield better performance. We would like to mention that there are a lot of data points below 1000 Kbps (starting from 14 Kbps, 28, 33, 40, 56 ....) that got crammed into a small interval on the X axis (plotting the X axis on a log scale is not a viable alternative either, it distorts the data plots). For data points at low raw-bandwidth values (about 3 Mbps and below on the X axis), timeouts do occur, consistent with the data from Figures 2 and 3. This is due to the effect of using smoothed RTTs as explained in Section III.A.5 above. The timeouts cause the slower link to drag-down the faster link a bit.

For raw bandwidth above about 3 Mbps, the RTTs of the links drop to sufficiently low values which in turn cause the RTO (retransmit timeout interval) values to drop below 1 second. The FreeBSD kernel imposes a lower limit of 1 second on the RTO. 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 any RTT variations, which prevents timeouts. The net result is that the connection striping across two links provides higher bandwidth than using the fatter pipe 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 (this work is funded in part by a company which is primarily a commercial vendor of wireless services): 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. The issue of utilizing multiple interfaces at the same time to enhance bandwidth utilization and reliability has not been adequately addressed in the literature, which motivated us to investigate this problem. Our analysis and data demonstrate that the proposed mechanism works well in many cases 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 the in the SCTP protocol (SCTP currently lacks this vital capability) and compare it with the mechanism proposed herein. Lastly, dynamically adding/deleting IP addresses for end-points (multiple interfaces) could lead to some 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 ICCN,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] 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.

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

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

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

[19] Cisco Systems, ," www.cisco.com/warp/public/779/largeent/learn/technologies/fast_echannel.html.

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

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

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

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

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

[25] 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.

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

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

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

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

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

[31] 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.