A Network-Layer Proxy for Bandwidth Aggregation and Reduction of IP Packet Reordering

Kristian Evensen*† Dominik Kaspar*† Paal Engelstad‡*† Audun F. Hansen* Carsten Griwodz*† Pål Halvorsen*†
*Simula Research Laboratory, Norway, †University of Oslo, Norway, ‡Telenor R&I, Norway
email:{kristrev, kaspar, paalee, audunh, griff, paalh}@simula.no

***Abstract*—With today's widespread deployment of wireless technologies, it is often the case that a single communication device can select from a variety of access networks. At the same time, there is an ongoing trend towards integration of multiple network interfaces into end-hosts, such as cell phones with HSDPA, Bluetooth and WLAN. By using multiple Internet connections concurrently, network applications can benefit from aggregated bandwidth and increased fault tolerance. However, the heterogeneity of wireless environments introduce challenges with respect to implementation, deployment, and protocol compatibility. Variable link characteristics cause reordering when sending IP packets of the same flow over multiple paths.**

This paper introduces a multilink proxy that is able to transparently stripe traffic destined for multihomed clients. Operating on the network layer, the proxy uses path monitoring statistics to adapt to changes in throughput and latency. Experimental results obtained from a proof-of-concept implementation verify that our approach is able to fully aggregate the throughput of heterogeneous downlink streams, even if the path characteristics change over time. In addition, our novel method of equalizing delays by buffering packets on the proxy significantly reduces IP packet reordering and the buffer requirements of clients.

***Index Terms*—Wireless networks, heterogeneous systems, traffic analysis, network protocols, scheduling, measurements.**

I. Introduction

The growing deployment of wireless infrastructure and technologies, such as WLAN, HSDPA, Bluetooth, and WiMAX, often places a single user device within coverage range of multiple access networks. At the same time, there is an ongoing trend towards integration of heterogeneous radio interfaces into single end devices. An example scenario of a laptop that communicates over both a WLAN and an HSDPA access network is shown in Figure 1.

Despite the rise of wireless connectivity, from a user's point of view, Internet access is usually provided using a single link at each point in time. Although multiple networks may be accessible simultaneously, users are today still unable to freely combine available connections and are required to choose a default link over which all traffic will flow.

Using multiple concurrent network connections has several potential benefits. Increased throughput by aggregation of bandwidth is the most intuitive advantage. For applications that open many transport sessions at once, this can be achieved on most operating systems by proper configuration of routing tables. Other benefits include increased service reliability, latency reduction, fault tolerance by sending redundant data over different paths, and enhanced mobility when combining coverage areas of independent access networks.

The final target of our work is a multilink solution that allows easy deployment and installation. The main requirement is that any third-party server application, operating system, and infrastructure should remain unchanged. Additionally, the client-side operating systems should not have to be modified. For enabling the concurrent use of multiple access networks at the multihomed clients, a quick and user-friendly configuration of network interfaces, routing tables, traffic rules, etc., would be desirable. A solution is needed that is robust in severely heterogeneous environments, compatible with all end-to-end applications and transparent to any transport protocols.

For meeting the requirements and reaching the goal of transparently striping packets to multiple addresses at a single receiver, many challenges exist. The major difficulties are caused by the heterogeneity of wireless links. As shown in [1], when IP packets that belong to the same flow are sent over highly variable wireless links, it is possible to achieve increased throughput, but at the cost of high packet reordering at the receiver. Packet reordering is crucial for the buffer requirements at the clients, and it has negative effects on the performance of transport protocols that make certain assumptions (such as interpreting reordered packets as lost).

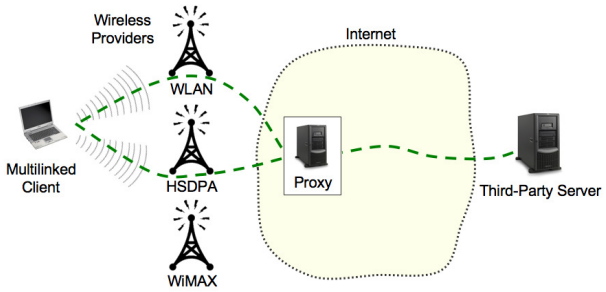

Fig. 1. A multilink proxy is used to schedule traffic from a server to all available interfaces on a client, which is simultaneously connected to a WLAN and an HSDPA access network.

Previously, contributions to multilink transfer (more generally referred to as inverse multiplexing [2]) have been proposed on many layers of the protocol stack (see Section II). However, there has not yet been a breakthrough in allowing a user-friendly and simultaneous utilization of multiple available interfaces. Although there is increased interest in this topic, in practice, it is usually the operating system that selects a default network interface to be used for communication, while any other interface remains idle. The reasons for the lack of a

commercial and technological breakthrough lie in the current Internet architecture and payment models. Even though it is possible to implement the above mentioned benefits on any layer of the protocol stack, no solution has yet been found that is not hindered by deployment issues and that is free of negative effects to mechanisms on other layers.

This paper introduces the design of a multilink proxy, which is able to transparently stripe traffic from servers to multihomed clients, as shown in Fig. 1. We operate on the network layer to achieve transport protocol transparency, and our focus has been on downlink streaming of UDP traffic. Our architecture includes the following functional components:

1) Support for network-layer striping
2) Bandwidth aggregation of multiple client interfaces
3) IP packet queuing for equalizing heterogeneous delays
4) Monitoring of path characteristics

The first component, network-layer multilink support, represents the base functionality, which all other components build on. It is solved with network address translation (NAT). The second component, the scheduler, assigns packets to links according to experienced throughput. The third component, the delay equalizer, is used to delay packets in the proxy in order to reduce packet reordering at the client. The fourth component, the path monitor, measures the different link characteristics and uses them to optimize the scheduler and the delay equalizer.

We demonstrate that network-layer packet striping is able to fully aggregate the bandwidths from a proxy to multiple client interfaces. In order to optimize for the common case of data downloads, the current focus of our analysis is on pure UDP streams. Experimental results obtained in a proof-of-concept implementation show that the proxy manages to dynamically adapt to changes in throughput and latency. In contrast to related research efforts, the presented solution is able to avoid tunneling overhead and actively reduces the workload at the client by buffering packets on the proxy.

This paper continues with a comprehensive description of related work in Section II. The following Section III introduces the details of our suggested proxy architecture and its implementation in Linux version 2.6.28. After presenting results in Section IV, the final conclusions are drawn in Section V.

II. Related Work

For several years, a lot of contributions to inverse multiplexing, multilink transfer, and bandwidth aggregation have been published. The most distinct way of categorizing the existing work is by the layer in the network protocol stack a solution is positioned. Different approaches require different changes to servers, clients, proxies, or a combination of these. What all approaches have in common, no matter on which layer, is a method to schedule data over different links and a method to later combine the diverted bytes again.

A. Application-Layer Striping

Bandwidth aggregation on the application layer has been proposed in various forms. For instance, the method presented in [3] modifies the FTP protocol to establish several connections when a transfer is initiated. The data to be transferred is divided into fixed-size segments and sent on the first connection that is able to transfer data (i.e., when the socket is not blocking). In [4], a purely client-based method is described, which assigns newly established transport-layer connections to one of the available interfaces and exploits traffic patterns of individual subscribers for optimal flow scheduling.

However, traffic scheduling at the flow level has the disadvantage that concurrent flows must exist for bandwidth aggregation to be exploitable. If only a single flow exists, or if the throughput of one connection dominates all others, the performance increase will be negligible. If both the server and client are controllable, a single flow can be divided into several flows and striped across all the links. This approach is suggested in [5] for dynamic, packet-wise scheduling of TCP streams over multiple paths.

Other research efforts in the highest layers of the protocol stack focus on decoupling striping decisions from the application by providing specialized middleware or a single virtual network socket. For instance, an architecture for session-layer striping is proposed in [6], while a networking middleware for providing network striping capabilities to applications with high demands on uplink throughput is presented in [7].

The drawback of all these network-layer approaches is that they imply software modifications at both endpoints to allow the full potential of bandwidth aggregation. In most cases, the server belongs to a third party, which poses a barrier to immediate deployment.

B. Transport-Layer Striping

Transport protocols build on the unreliable network layer functionality to transparently provide end-to-end connections. Current standardized transport protocols do not support network striping and may experience severe performance degradation from IP packet reordering. Nevertheless, numerous attempts have been made to tune existing transport protocols for multipath capability.

For instance, the work presented in [8] significantly modifies TCP to use multiple paths using mechanisms for handling packet reordering and bottleneck detection. Multiple paths are provided by an overlay network, which guarantees interoperability with arbitrary IP networks. In [9], a transport protocol extension to RTP is described, which is specialized for real-time multimedia transfer over ad hoc networks. It assumes an underlying multipath routing topology and a complementary TCP-variant for session/flow control. In [10], exploiting SCTP's multihoming feature to simultaneously transfer data across multiple end-to-end paths has been suggested. The proposed concurrent multipath modifications to SCTP enable a congestion window evolution very similar to having two separate SCTP connections. A comprehensive overview of recent solutions targeting transport-layer support of multilink striping is provided in [11].

Consequently, approaches targeting the transport layer are very difficult to deploy, because severe changes to operating

systems and already standardized protocols are necessary, with little chance of being widely accepted in the near future.

C. Network-Layer Striping

Several research efforts on network-layer striping exist, as well. The general network-layer approach, as followed in [12]–[14], is to use tunneling mechanisms for transparently redirecting packets from the server or a proxy to all IP addresses at the client. However, the reported solutions are often based on assumptions and numbers that are unavailable in real-life deployments. For example, the solution presented in [13] relies on link-layer feedback from base stations, such as the time the wireless channel requires until it is available for the next transmission. The fact that most approaches have only been tested in simulations, often based on very simple assumptions about heterogeneity, adds to our skepticism on deployability. Both the round-trip time and the throughput of a wireless link are highly dependent on time. Our field measurement results, described in [1] and [15], contradict frequently made assumptions about the characteristics of heterogeneous links, which are often modeled too evenly and not very realistically. For example, the 30 ms delay for *both* WLAN and UMTS reported in [16] does not coincide with the field measurements we have obtained. The delay characteristics of heterogeneous network technologies may exhibit much larger differences and the used packet size additionally affects the delay disparity. For two endpoints that are 8 IP hops apart, Figure 2 depicts the average round-trip times of WLAN and HSDPA.

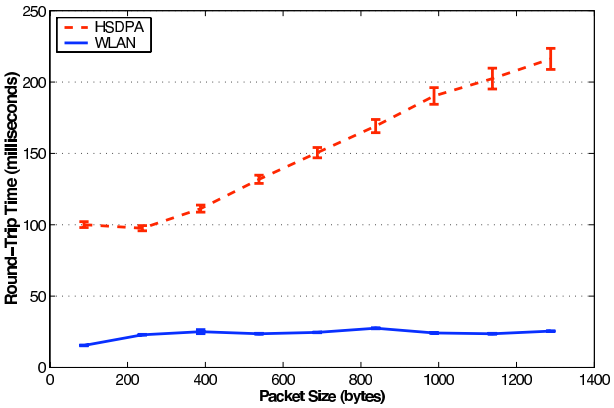

Fig. 2. The heterogeneity of round-trip times is one of the most dominant challenges when aggregating wireless links, such as HSDPA and WLAN. In truly heterogeneous environments, varying the packet size has negligible effect on equalizing the latency of multiple links.

Another indication of the general underestimation of network heterogeneity can be detected in [14]. In this work by Phatak et al., the conclusion is drawn that packet sizes can be adjusted to equalize round-trip times. While this is true to a certain degree, for truly heterogeneous links it can easily be verified that using different packet sizes per link may have no significant effects on equalizing the RTTs. Figure 2 illustrates that the round-trip times over WLAN and HSDPA may always experience a significant delay difference, no matter how the packet sizes are chosen.

D. Link-Layer Striping

Multiple interfaces of the same technology can also be striped for better performance at the link layer, which is referred to as bonding or trunking. The main idea of bandwidth aggregation on the link layer is to stripe data across a bundle of physical channels, as done in [17] and [18]. A method for channel aggregation in cellular networks is described in [19]. In order to improve resilience, parity codes are applied across channels rather than across packets. Another interesting approach is followed in [20], where it is proposed that users of WLANs should be able to multihome and split their traffic among all available access points, based on obtained throughput and a charged price.

However, a link-layer solution of striping data through heterogeneous networks and to different IP addresses is not feasible because the link layer has no notion of IP. This leaves us with the network layer as the most promising candidate for tackling multilink transfer.

III. Multilink Architecture

This chapter describes the implementation of our multilink proxy and the involved architectural elements used for enabling network-layer packet striping. The final solution is anticipated to consist of a cross-platform software package that users can easily download and install on their end devices. This software creates a connection to the nearest available proxy (most likely over multiple IP hops) and communicates the IP addresses of the various available interfaces. The proxy is also informed when interfaces are added or removed.

A. Using a Proxy

The advantage of a proxy solution is that it is fully controllable and allows servers to remain unchanged. Some modifications on the multilinked clients are still necessary, but they can be reduced to configuration issues. Figure 3 depicts the network-layer striping functionality of the proxy. A transport-layer connection originating at the server is transparently divided at the proxy and recombined at the client, with the transport layer being completely unaware of the fact that IP packets traveled over multiple independent access networks.

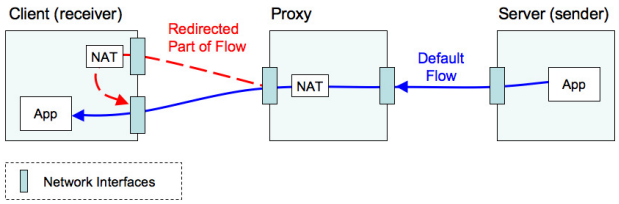

Fig. 3. Network-layer striping – By employing network address translation at the proxy, a downlink packet stream is transparently divided to multiple client interfaces, where the stream is combined again.

There is a tradeoff between the amount of intelligence on the client and on the proxy. Moving functionality to the client's stack, or into its application layer, might severely increase the workload and buffer requirements of thin clients, such as

mobile phones. On the other hand, scalability issues might arise when introducing proxies. For the proof of concept implementation in this paper, scalability issues are not a concern. They will be studied in the future.

The proxy is composed of four functional elements, which have already been outlined in the introduction and will be described more thoroughly below. Figure 4 shows the internals of the proxy with these four elements: NAT, Packet Scheduler, Delay Equalizer, and Path Monitor.

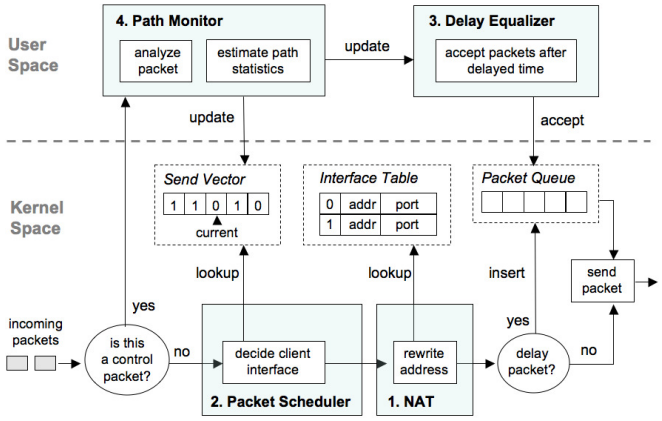

Fig. 4. Internals of the multilink proxy – The destination address of incoming packets is rewritten according to a weighted round-robin schedule. Latency differences are equalized by queuing packets destined to travel over a fast path. Dynamic adaptation to differences in throughput and delay is facilitated through path monitoring feedback from the client.

To the best of our knowledge, the only use of a network proxy for bandwidth aggregation to multihomed hosts has been mentioned by Chebrolu et al. [12], [13]. However, in contrast to their research, we do not assume that the base stations are placed within the same administrative system. Our proxy can be located anywhere in the Internet, preferably close to the clients it serves. In addition, the only feedback the proxy receives is from the clients; it is totally independent of any sort of feedback from base stations or other elements within an access network. We assume the access networks to be black boxes that induce variable path characteristics.

B. Network Address Translation

While we envision the sender to be any third-party server in the Internet, both the client and the proxy must be configured for transparent traffic redirection. In contrast to all previous research efforts that rely on tunneling mechanisms (such as [14] and [13]), our solution makes use of NAT. For scheduling packets to arbitrary client interfaces, the proxy rewrites the default destination IP address (and port) to the address of a secondary interface. For packets that do not arrive at the default interface, the client does the inverse address translation and forwards them internally.

Even though our approach induces no additional encapsulation overhead and does not suffer from packet fragmentation problems, the main reason for following a NAT solution is its easy configuration on both the proxy and the client. For instance, in Linux, *iptables* commands are sufficient to achieve the desired redirections.

C. Packet Scheduler

NAT alone is insufficient for sending packets over various paths. For bandwidth aggregation, a scheduler is needed that decides for each packet to which client interface it should be sent. On the proxy, this function is implemented as a dynamically loadable kernel module (see Figure 4).

Since the available interfaces at the client experience variable throughput, IP packets should be forwarded at the estimated throughput ratio. We propose the use of a *send vector* for allowing both static and dynamic packet striping. The send vector contains the order of client interfaces (identified by integers) to be picked and is used to decide the forwarding destination of incoming packets at the proxy. A pointer into the send vector is incremented after each lookup and reset when the end is reached or when the vector is updated. During initialization, the send vector V is set to emulate pure round-robin behavior, i.e., $V = \{0, 1, ..., m\}$ for m client interfaces. Once the path monitor (Section III-E) has received new information about the throughput ratio, it updates the send vector accordingly. For the bandwidth estimates of two interfaces, which correspond to two weights w_0 and w_1, the send vector V is constructed as described in Algorithm 1, a fair variant of weighted round-robin scheduling.

Algorithm 1 createSendVector(n, w_0, w_1)

Input: Vector length $n \in \mathbb{N}_{>0}$ and weights $w_0, w_1 \in \mathbb{R}_{\geq 0}$
Output: Send vector V of length n

1: $V = zeros(n)$; {initialize V with n zeros}
2: $r = w_1/(w_0 + w_1)$; {calculate weight ratio}
3: $r = round(r * n)/n$; {adjust r such that $r * n$ is an integer}
4: **for** $i = 1$ **to** $r * n$ **do**
5: $\quad V(\lceil i/r \rceil) = 1$;
6: **end for**

Send vectors for more than two client interfaces can be created by recursively merging send vectors created for two weights. The main idea is illustrated in Figure 5, in which a send vector is constructed based on three weights that correspond to throughput estimates of 2, 3, and 5 Mbit/s. In the first step, Algorithm 1 returns $V_0 = \{0, 1, 1, 0, 1\}$ for the weights $w_0 = 2$ and $w_1 = 3$. In the second step, Algorithm 1 is again called for the combined weight of $w_0 + w_1 = 5$ and $w_2 = 5$, which returns $V_1 = \{0, 2, 0, 2, 0, 2, 0, 2, 0, 2\}$. Overwriting the zeros in V_1 with the elements of V_0 will result in the final send vector $V = \{0, 2, 1, 2, 1, 2, 0, 2, 1, 2\}$. The same procedure can be continued for m interfaces.

Proposition: A binary send vector V of finite length n will often be unable to accurately represent the ratio of two measured bandwidths b_0 and b_1. Errors in approximating the true ratio translate directly into a less efficient aggregation of bandwidth. The error is bounded and never exceeds $\frac{1}{2n}$.

Proof: The approximation error is the difference between the actual bandwidth ratio $r_{BWs} = b_0/(b_0 + b_1)$ and the send vector's approximated ratio $r_V = m/n$, where m is the

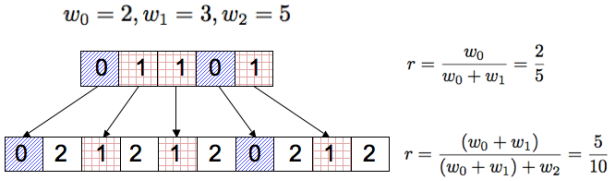

Fig. 5. Example – the recursive creation of a send vector for a client with three interfaces. The weights correspond to measured bandwidth estimates of 2 Mbit/s (w_0), 3 Mbit/s (w_2), and 5 Mbit/s (w_2).

number of zeros in the vector and $n - m$ the number of ones ($n \geq m, m \in \mathbb{N}$). The approximation error Err is defined as:

$$Err = |r_{BWs} - r_V| \tag{1}$$

In the worst case, m will differ from its ideal value $r_{BWs} * n$ by $\pm\frac{1}{2}$. Thus, we can write $m = r_B Ws * n \pm \frac{1}{2}$ and replace it into Equation 1:

$$Err = \left| r_{BWs} - \frac{m}{n} \right| = \left| r_{BWs} - \frac{r_{BWs} * n \pm \frac{1}{2}}{n} \right| \tag{2}$$

For an upper limit on the error, this simplifies to:

$$Err \leq \frac{1}{2n} \tag{3}$$

■

A generalization of this worst-case analysis to more than two links is beyond the scope of this paper.

D. Delay Equalizer

In practical wireless environments, IP packet reordering depends mostly on variances in delay [1], not bandwidth. If all available paths were of constant and equal delay, packets would always arrive in order, no matter which destination interface the scheduler picks. Therefore, for mitigating IP packet reordering, a dynamic scheduler is needed that adapts to both throughput and delay estimates.

While the previously described packet scheduler handles bandwidth aggregation, the delay equalizer is implemented as an independent user space module, where packets are delayed for a short while if they are destined to travel over a low-delay path p_i (with $i \in \{1, ..., m\}$). Thus, the delays d_i of m paths are equalized so that packets should experience similar latencies over all paths. In order to always use up-to-date estimates on the path delays d_i, feedback from the path monitor (Section III-E) is used.

If a packet is destined to travel over a path p_i, the total time T_{tot_i} that it should be held in the buffer is the difference between the estimated path delay d_i and the highest measured delay of all paths $d_{max} = max(d_i), i \in \{1, ..., m\}$:

$$T_{tot_i} = d_{max} - d_i \tag{4}$$

Since packets should be guaranteed to travel in-order over individual links, the buffer for each path is implemented as a FIFO queue. Every packet that enters the queue is released a given time offset T_{δ_i} after the previous packet in the queue. This time offset is dominated by the interarrival time T_{IAT_i}, which is measured for each packet coming into the proxy. Additionally, the time offset T_{δ_i} depends on measurement updates from the path monitor. Given the current delay estimate $d_{i_{cur}}$ and a new measurement $d_{i_{new}}$ of path p_i, the wait time offset T_{δ_i} is defined as:

$$T_{\delta_i} = T_{IAT_i} + (d_{i_{new}} - d_{i_{cur}}) \tag{5}$$

For illustrative purposes, assume that there are two paths p_0 and p_1 from the proxy to the client with estimated delays $d_0 = 30\,\text{ms}$ and $d_1 = 10\,\text{ms}$, implying that the packets destined for p_1 will be delayed by 20 ms. The throughput of p_0 is estimated to 3.8 Mbit/s and p_1 to 2.2 Mbit/s. Following Algorithm 1, this results in an 11-element send vector $V = \{0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1\}$. The stream from the server is offered in packets of 1500 bytes at 6 Mbit/s, which ideally translates to a time of 2 ms between incoming packets at the proxy. Table I lists all the values needed in the procedure of equalizing the heterogeneous path delays. Looking at the values for the arrival times at the client makes it clear that in an ideal scenario with predictable delays, the delay equalizer is able to perfectly avoid packet reordering.

TABLE I
EXAMPLE DELAY EQUALIZING PROCEDURE

Packet sequence number	1	2	3	4	5	6	7	8	9	10	11
Send vector	0	0	1	0	0	1	0	1	0	0	1
Arrival time at proxy (ms)	0	2	4	6	8	10	12	14	16	18	20
Wait time offset T_{δ_i} (ms)	0	0	20	0	0	6	0	4	0	0	6
Total wait time T_{tot_i} (ms)	0	0	20	0	0	20	0	20	0	0	20
Departure time from proxy (ms)	0	2	24	6	8	30	12	34	16	18	40
Arrival time at client	30	32	34	36	38	40	42	44	46	48	50

This method allows the reduction of IP packet reordering without any changes to the operating system at the client. A packet buffer at the proxy can also be useful in case the offered load from the server exceeds the aggregated bandwidth of all client interfaces.

E. Path Monitor

The path monitor is a user space tool implemented at the proxy. It collects estimates of path throughput and latency used for optimizing the packet scheduler and delay equalizer. For simplifications in our testbed setup, delay estimates are derived from sending TCP probe packets every 200 ms and measuring the time until an ACK returns, while throughput estimates are obtained by measuring the client's incoming data rate. In order to compensate for high variances in reported feedback, the path monitor reports the average of the 10 most recent measurements. A disadvantage of this approach is that inaccuracies can be caused during congestion and when retransmissions delay later measurements. Thus, for our results we made sure that the network had close to no loss and enough bandwidth to support control traffic.

Packet pairing, a method less affected by congestion, has been described in [21]. The idea is to send consecutive packet

pairs over the same path and to derive the bandwidth from the time difference of arrival. We use packet pairs to check for available bandwidth when the send vector locks to unwanted patterns, such as $V = \{0, 0, 0, 0, ...\}$, and plan to utilize it for all throughput monitoring in the future.

There exists a tradeoff between the monitoring overhead and the accuracy of the send vector. Frequent monitoring updates allow the send vector to become accurate more quickly, but it also introduces more traffic and calculation overhead.

IV. Results

This section addresses the two conflicting main goals of the multilink proxy through experimental verification. Is it possible to aggregate the bandwidth of multiple client interfaces, no matter how heterogeneous the links are? At the same time, is the delay equalizer able to mitigate IP packet reordering and to reduce the buffer requirements at the client?

A. Testbed Setup

For measuring the performance of the multilink proxy, an experimental, basic testbed was set up as shown in Figure 6. This test network contains a client machine with two network interfaces, a server machine that streams packets at a constant bitrate, and the proxy machine that transparently schedules the packets to different client interfaces. All machines are connected with 100 Mbit/s Ethernet links. For varying the test parameters, the *netem* network emulator was used to throttle the bandwidths to the client and to introduce additional latency.

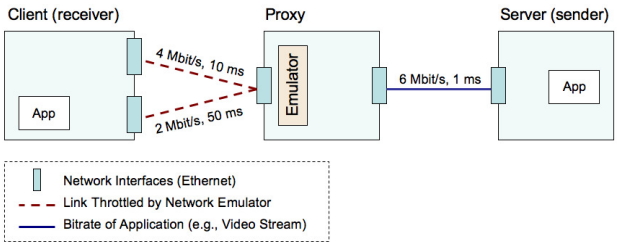

Fig. 6. Multilink test network – The *netem* tool is used to emulate link heterogeneity (bandwidth and latency) in a controlled manner.

While the performance of bandwidth aggregation can be expressed in Mbit/s, getting a notion of the amount of reordering in a packet sequence is less straightforward. Various metrics for IP packet reordering have been defined in the literature (e.g., [22]) and by standardization efforts in the IETF. Where appropriate, we report the number of packets that arrived with an out-of-order sequence number, as defined in RFC 4737 [23]. However, counting the number reordered packets is often insufficient to express the magnitude of disorder and its implications, such as buffering requirements.

On the client, packet reordering can be reduced through the use of a buffer, where every incoming packet is stored if its sequence number is greater than the currently expected. If more out-of-order packets arrive than the buffer can hold, they must be discarded. The intuitive measure of a buffer's load, the *average buffer occupancy*, has been defined in RFC 5236 [24]. However, the average buffer occupancy does not give much insight on the number of packets that were discarded due to overflows. In addition to the share of reordered packets, we report the 95th (or 99th) percentile values of the buffer occupancy to express the buffer size needed to guarantee at most 5% (or 1%) packet discards.

B. Bandwidth Aggregation

In a first experiment, it is shown that a send vector-based approach is able to fully aggregate two links, regardless of their bandwidth ratio. For this, the incoming bitrate was set to a constant 6 Mbit/s, while the two links from the proxy to the client were throttled so that their bandwidths summed up exactly to the offered load of 6 Mbit/s. The send vector was limited to 64 elements and no additional latency was introduced. Table II shows that for all chosen bandwidth ratios, the aggregated throughput at the client was measured as 5.99 Mbit/s and at most 0.12% traffic was lost. These results are statistics of 5 experiment batches, each lasting 10 min.

TABLE II
Throughput Aggregation for various Bandwidth Ratios

Bandwidth ratio	1:11	2:10	3:9	4:8	5:7	6:6
Avg. aggregated throughput (Mbit/s)	5.99	5.99	5.99	5.99	5.99	5.99
Out-of-order packets (%)	1.56	1.98	1.68	1.29	2.03	0.09
95th percentile buffer occupancy (packets)	0	0	0	0	0	0
99th percentile buffer occupancy (packets)	1	1	1	1	1	0
Lost packets (%)	0.10	0.12	0.11	0.05	0.04	0.01

For the stable links used in this experiment, the observed packet reordering is comparable to the 0.01 - 1.65 % [25] and 1 - 1.5 % [26] reported for wired, high-speed networks. Additionally, our low buffer occupancy values show that in 99% of the time a buffer with capacity for only 1 packet is able to avoid packet discards.

Small amounts of packet loss are caused during an experiment's initialization phase, when the round-robin send vector $V = \{0, 1, 0, 1...\}$ is unable to correctly represent the actual bandwidth ratio. Therefore, one link becomes congested for a short time until the first feedback messages from the path monitor are able to stabilize the send vector to a pattern that is able to represent the true ratio.

Even when the bandwidth characteristics are unstable and change over time, the multilink proxy is able to adapt dynamically. Figure 7 depicts this behavior with results obtained from an experiment in which the emulated links were abruptly changed at constant intervals of 30 seconds. Systems without support for packet striping can only achieve the throughput of either Path 1 or Path 2, while our approach is able to aggregate the throughput of both paths. The visible spikes in the curve of the aggregated throughput are caused by the path monitor's needed time to adjust the send vector to a new pattern. Therefore, the bandwidth estimates are briefly imprecise, which leads to short spikes in throughput.

C. Reduction of Packet Reordering at the Client

In a second experiment, we investigated how proxy-based packet buffering affects the client workload. We used the

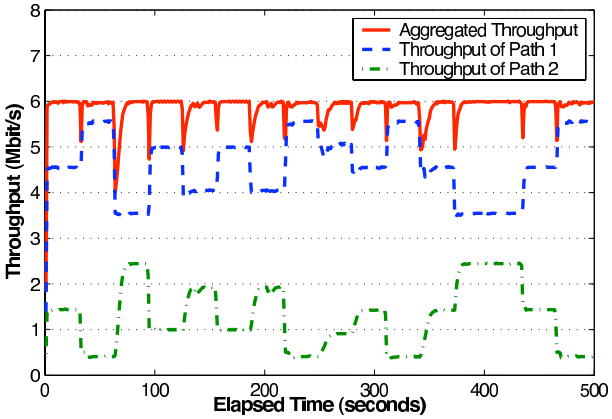

Fig. 7. Adaptive bandwidth aggregation – A send vector based on path monitoring feedback is able to adapt to arbitrary bandwidth changes. In this example, the bandwidths of Link 1 and Link 2 always sum up to 6 Mbit/s.

testbed described in Section IV-A and the server streamed packets at 6 Mbit/s. However, in this experiment, the two links between the proxy and the client had no bandwidth limitations, so that a static send vector $V = \{0, 1\}$ was used.

Figure 8 shows that a packet buffer on the proxy is able to significantly reduce the number of reordered packets. When the delay equalizer module is disabled, the buffer occupancy grows as the delays become increasingly heterogeneous. When the delay equalizer is enabled, the buffer occupancy at the client is decreased. Under stable network conditions, a buffer with the capacity of a single packet would be enough to guarantee less than 1% packets to be lost due to overflows.

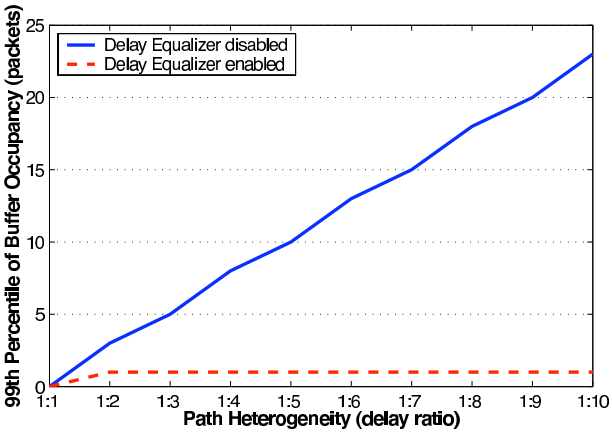

Fig. 8. Reducing IP packet reordering – At the proxy, a buffer is used to delay packets over the low-latency link by putting them into a buffer. This is useful to compensate for latency differences and to reduce the client workload.

A conventional packet striping solution, without the functionality of a delay equalizer, requires buffering at the client that increases with the degree of delay heterogeneity. In contrast, multihomed clients connected to our proxy can benefit significantly from the reduction in IP packet reordering.

Even when the links are unstable and exhibit variances in delay, our multilink proxy is able to compensate for part of the heterogeneity. For illustrating this by means of an arbitrary scenario, we have modeled the two paths from the client to the proxy to roughly resemble the heterogeneity between WLAN and HSDPA. The delay of the first path was set to an average of 10 ms with ±5 ms jitter (normally distributed) and the delay characteristics of the second path were set to 50 ms and ±15 ms jitter (uniformly distributed).

Figure 9 illustrates the influence of the delay equalizer on packet reordering. When the delay equalizer was disabled, almost all packets experienced reordering with peak displacements of -10 and $+10$ sequence numbers. However, with the delay equalizer enabled, these peaks were merged into a single peak at 0, implying that the largest share of the packets exhibited no reordering at all. This shift results in a reduction of the 99th percentile value of the buffer occupancy from 18 packets to 9 packets; in other words, a 50% decrease in buffer requirements.

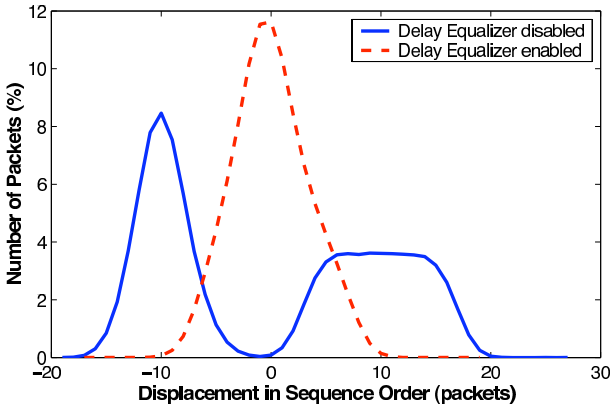

Fig. 9. Buffering packets at the proxy is able to considerably reduce IP packet reordering even if the used links exhibit variable delays.

D. Coping with Dynamic Bandwidth and Delay

In a last experiment, we compared the performance of the multilink proxy to pure round-robin packet striping, while both the throughput *and* delay characteristics were dynamic. The parameters used were a combination of the two previously conducted experiments. The incoming bitrate at the proxy was set to 6 Mbit/s, while the bandwidth of the two paths from the proxy to the client were given random values that add up to 6 Mbit/s (as in Fig. 7). The paths to the client were modeled with the approximate latency characteristics of WLAN and HSDPA, analogous to the values used in Section IV-C.

TABLE III
COPING WITH COMBINED BANDWIDTH AND DELAY VARIATION

	Round-robin striping	Multilink proxy
Average aggregated throughput	4.49 Mbit/s	5.86 Mbit/s
99th percentile buffer occupancy	26 packets	9 packets
Packet loss	25%	2.2%

The results of this experiment are summarized in Table III. They clearly show the combined benefits of the packet scheduler and the delay equalizer modules. The multilink proxy

outperforms pure round-robin packet striping in all aspects: it achieves higher aggregated throughput, a lower packet loss ratio, and it requires less buffer capacity at the client.

V. Conclusions and Future Work

In this paper, we have introduced a modular architecture of a multilink proxy that is capable of striping IP packets to multiple network interfaces at a client. The proof-of-concept implementation presented in this paper operates on the network layer and utilizes NAT rules for transparent traffic redirection, a packet scheduler for bandwidth aggregation, a delay equalizer to reduce packet reordering, and a path monitor that enables adaptation to changing network characteristics.

The experiments conducted in our network testbed have shown that packet scheduling based on a send vector, which is dynamically updated according to throughput estimates, is able to adapt to variations in bandwidth, i.e., the proxy is able to fully aggregate the throughput of multiple client interfaces, regardless of their bandwidth ratio.

In addition, we have experimentally verified that it is possible to reduce the buffer requirements at the client by introducing a delay equalizing buffer at the proxy. The workload can therefore be shifted from the client to the proxy, which is useful for low-performance end devices. A packet buffer at the proxy will become even more interesting when placed into an application-specific scenario. For instance, it is conceivable to improve the quality of video streams by making the scheduling decision dependent on the number of packets in the buffer, their type and content.

The currently implemented testbed has shown very promising results in a controlled environment. However, several technical challenges have slowed down the process of deploying our multilink proxy for field measurements into real wireless access networks. In the near future, we will tackle these challenges and enhance the system to also support uplink streams. Our main short-term goal is to evolve the current code so that it can be installed onto a handheld system, such as a Linux-enabled cellphone, that can be taken out of the lab and demonstrated in public. In addition, our current results were obtained for a client with two interfaces; thus, it would be of high interest to expand our experiments to three and more access networks.

A very important analysis that we will carry out in the future is the impact of transparent packet striping on the performance of transport- and application-layer protocols. Will the compensation of delay heterogeneity at the proxy be sufficient for keeping a transport connection intact? How can control messages, such as acknowledgements, be exploited to optimize the path monitoring and the scheduling decisions?

Furthermore, in the future, we plan to enhance our system with support for mobile clients and allowing the use of multiple interfaces for optimizing seamless handover across heterogeneous wireless technologies. In an environment with many mobile clients, scalability issues will become apparent and will have to be studied.

References

[1] D. Kaspar, K. Evensen, A. F. Hansen, P. Engelstad, P. Halvorsen, and C. Griwodz, "An analysis of the heterogeneity and IP packet reordering over multiple wireless networks," in *IEEE Symposium on Computers and Communications (ISCC)*, 2009.

[2] J. Duncanson, "Inverse multiplexing," *IEEE Communications Magazine*, vol. 32, no. 4, pp. 34–41, 1994.

[3] M. Allman, H. Kruse, and S. Ostermann, "An application-level solution to TCP's satellite inefficiencies," in *Proceedings of the First International Workshop on Satellite-based Information Services (WOSBIS)*, Rye, New York, USA, 1996.

[4] N. Thompson, G. He, and H. Luo, "Flow scheduling for end-host multihoming," in *INFOCOM*, April 2006.

[5] B. Wang, W. Wei, Z. Guo, and D. Towsley, "Multipath live streaming via TCP: scheme, performance and benefits," in *ACM CoNEXT*. New York, NY, USA: ACM, 2007, pp. 1–12.

[6] A. Habib, N. Christin, and J. Chuang, "Taking advantage of multihoming with session layer striping," in *INFOCOM*, 2006.

[7] A. Qureshi, J. Carlisle, and J. Guttag, "Tavarua: video streaming with WWAN striping," in *MULTIMEDIA '06: Proceedings of the 14th annual ACM international conference on Multimedia*. New York, NY, USA: ACM, 2006, pp. 327–336.

[8] M. Zhang, J. Lai, A. Krishnamurthy, L. Peterson, and R. Wang, "A transport layer approach for improving end-to-end performance and robustness using redundant paths," in *ATEC '04*, 2004.

[9] S. Mao, D. Bushmitch, S. Narayanan, and S. Panwar, "MRTP: A multi-flow real-time transport protocol for ad hoc networks," *IEEE Transactions on Multimedia*, 2006.

[10] J. R. Iyengar, P. D. Amer, and R. Stewart, "Concurrent multipath transfer using SCTP multihoming over independent end-to-end paths," *IEEE/ACM Trans. Netw.*, vol. 14, no. 5, 2006.

[11] I. Gapanova, "Concurrent multipath transfer for heterogeneous networks," Master's thesis, TU Braunschweig, 2008.

[12] K. Chebrolu, B. Raman, and R. R. Rao, "A network layer approach to enable TCP over multiple interfaces," *Wireless Networks*, vol. 11, no. 5, pp. 637–650, 2005.

[13] K. Chebrolu and R. R. Rao, "Bandwidth aggregation for real-time applications in heterogeneous wireless networks," *IEEE Transactions on Mobile Computing*, vol. 5, no. 4, pp. 388–403, 2006.

[14] D. S. Phatak, T. Goff, and J. Plusquellic, "IP-in-IP tunneling to enable the simultaneous use of multiple IP interfaces for network level connection striping," *Computer Networks*, 2003.

[15] D. Kaspar, A. F. Hansen, and C. Griwodz, "Multilink transfer over heterogeneous networks," in *IEEE International Conference on Network Protocols (ICNP), Student Poster Session*, 2008.

[16] C.-M. Huang and C.-H. Tsai, "WiMP-SCTP: Multi-path transmission using stream control transmission protocol (SCTP) in wireless networks," in *AINA Workshops*, 2007, pp. 209–214.

[17] H. Adiseshu, G. Parulkar, and G. Varghese, "A reliable and scalable striping protocol," *ACM SIGCOMM*, 1996.

[18] A. C. Snoeren, "Adaptive inverse multiplexing for wide-area wireless networks," in *GLOBECOM*, 1999.

[19] J. Chesterfield, R. Chakravorty, I. Pratt, S. Banerjee, and P. Rodriguez, "Exploiting diversity to enhance multimedia streaming over cellular links," in *INFOCOM*, March 2005.

[20] S. Shakkottai, E. Altman, and A. Kumar, "The case for non-cooperative multihoming of users to access points in IEEE 802.11 WLANs," in *INFOCOM*, 2006.

[21] S. Keshav, "The packet pair flow control protocol," International Comp. Sci. Institute, Berkeley, CA 94704, Tech. Rep. 91-028, May 1991.

[22] N. M. Piratla and A. P. Jayasumana, "Metrics for packet reordering - a comparative analysis," in *International Journal of Communication Systems*, 2007.

[23] A. Morton, L. Ciavattone, G. Ramachandran, S. Shalunov, and J. Perser, "Packet reordering metrics," IETF RFC 4737, 2006.

[24] A. Jayasumana, N. Piratla, T. Banka, A. Bare, and R. Whitner, "Improved packet reordering metrics," IETF RFC 5236, 2008.

[25] L. Gharai, C. Perkins, and T. Lehman, "Packet reordering, high speed networks and transport protocol performance," *ICCCN*, 2004.

[26] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley, "Measurement and classification of out-of-sequence packets in a Tier-1 IP backbone," *ACM SIGCOMM*, 2002.