Handheld Routers: Intelligent Bandwidth Aggregation for Mobile Collaborative Communities

Puneet Sharma, Sung-Ju Lee, Jack Brassil, Kang G. Shin
Internet Systems and Storage Laboratory
HP Laboratories Palo Alto
HPL-2003-37 (R.1)
May 20th , 2003*

bandwidth aggregation, link striping, inverse multiplexing, application-awareness

Multi-homed, mobile wireless computing and communication devices can spontaneously form communities to logically combine and share the bandwidth of each other's wide-area communication links using *inverse multiplexing*. But membership in such a community can be highly dynamic, as devices and their associated WAN links randomly join and leave the community. We identify the issues and tradeoffs faced in designing a decentralized inverse multiplexing system in this challenging setting, and determine precisely how heterogeneous WAN links should be characterized, and when they should be added to, or deleted from, the shared pool. We then propose methods of choosing the appropriate channels on which to assign newly-arriving application flows. Using video traffic as a motivating example, we demonstrate how significant performance gains can be realized by adapting allocation of the shared WAN channels to specific application requirements. Our simulation and experimentation results show that collaborative bandwidth aggregation systems are, indeed, a practical and compelling means of achieving high-speed Internet access for groups of wireless computing devices beyond the reach of public or private access points.

* Internal Accession Date Only
Approved for External Publication

Handheld Routers: Intelligent Bandwidth Aggregation for Mobile Collaborative Communities

Puneet Sharma, Sung-Ju Lee, Jack Brassil, and Kang G. Shin
Internet Systems & Storage Lab
Hewlett-Packard Laboratories
Palo Alto, CA 94304

***Abstract*—Multi-homed, mobile wireless computing and communication devices can spontaneously form communities to logically combine and share the bandwidth of each other's wide-area communication links using *inverse multiplexing*. But membership in such a community can be highly dynamic, as devices and their associated WAN links randomly join and leave the community. We identify the issues and tradeoffs faced in designing a decentralized inverse multiplexing system in this challenging setting, and determine precisely how heterogeneous WAN links should be characterized, and when they should be added to, or deleted from, the shared pool. We then propose methods of choosing the appropriate channels on which to assign newly-arriving application flows. Using video traffic as a motivating example, we demonstrate how significant performance gains can be realized by adapting allocation of the shared WAN channels to specific application requirements. Our simulation and experimentation results show that collaborative bandwidth aggregation systems are, indeed, a practical and compelling means of achieving high-speed Internet access for groups of wireless computing devices beyond the reach of public or private access points.**

I. Introduction

An increasing number of multi-homed wireless mobile computing devices are being equipped with two distinct types of wireless communication interfaces: a local area network (LAN) interface such as IEEE 802.11x, and a wide area network (WAN) interface such as a 2.5G or later generation cellular link. The capabilities of these interfaces differ greatly, most notably with the available LAN bandwidth exceeding the WAN's bandwidth by one to three orders of magnitude. For the foreseeable future we anticipate that this bandwidth disparity between local and wide area wireless network connections will remain intact.

Public high-speed Internet connectivity from such devices is now typically achieved by connection via the wireless LAN interface to an access point which is connected to a high-speed, wired connection. It remains unlikely, however, that opportunistic deployment of these access points will ever realize ubiquitous — or even relatively geographically broad — access. Even where access points are densely deployed, seamless roaming between access points remains a technical challenge, and may not serve the business interests of either access point operators, venue owners or service providers. Further, even where access point coverage is rich, the transmission rate of the wired connection — typically 1.5Mb/s — is limited and shared among a possibly large group of users, and unlikely to increase significantly in transmission speed in the foreseeable future.

To overcome the limited geographic coverage of public access points, we envision an alternative, complementary solution to high-speed Internet access through collaborative resource sharing. A group of wireless, mobile computing and communication devices in close proximity can dynamically form communities interconnected through their compatible high-speed LAN interfaces; we call these ad hoc communities *piconets*. Each piconet member independently uses its WAN interface to create a communication *channel* to an inverse multiplexer, and optionally offers to other members (full or partial) access to this channel. The set of participating channels connecting the piconet members to the inverse multiplexer can be logically combined with an inverse multiplexing protocol to yield a higher-speed *aggregated channel* than is available from any one of the individual piconet members. The participating members acting as *handheld routers*, receive some of the packets destined to other members over their WAN links and forward them onto the LAN.

The envisioned bandwidth aggregation mechanism is an enabling technology, as illustrated by the following example. A group of train commuters could spontaneously form a piconet, and all members could receive a video stream delivered at a higher bandwidth — and higher quality — than any one member could receive. Each piconet member would also enjoy higher speed, statistically-multiplexed WAN access, a service often far more desirable than private, but lower-speed access. Indeed, the same technology would apply to Personal Area Networks, where an individual possesses multiple, multi-homed devices.

Striping data across multiple, parallel communication channels is a conventional communications technique used to improve system performance or reliability in relatively statically-configured disk storage systems [7, 9, 31] and fixed, wired LAN–WAN interconnection systems [11, 33, 35]. In stark contrast, due to end-device heterogeneity, mobility, and time-varying link transmission characteristics, the system we consider here is highly dynamic, and must be assembled, administered, and maintained in a decentralized fashion. We present the design of a collaborative bandwidth aggregation architecture that is both practical and readily deployable. A key contribution we make is showing that significant performance gains can be realized by adapting shared WAN link selection to the specific application requirements of the communication flows. As an illustration, we demonstrate how the quality of a hierarchically-layered video

Kang G. Shin is a visiting professor from the Electrical Engineering and Computer Science Department, University of Michigan, Ann Arbor.

stream transmitted over lossy channels can be improved by a priority/application-aware traffic assignment.

The rest of the paper is organized as follows. Section II explores the issues and tradeoffs faced in creating a decentralized inverse multiplexing system. Section III introduces algorithms for the assignment of application flows to heterogeneous WAN channels, and Section IV describes the specific system architecture we chose to study. Performance evaluation results from an *ns*-based simulation are presented in Section V, and Section VI describes the implementation of a prototype system used to corroborate our findings. Related work is summarized in Section VII, and our conclusions are presented in the final section.

II. Issues, Challenges, and Approaches

Let's first consider a relatively basic channel aggregation system. Assume that each shared channel contains only the single WAN link between a participating piconet member and the inverse multiplexer. Suppose we seek to provide a single, bidirectional, unicast connection between an Internet source and a single piconet node. End-system applications are oblivious to the presence of the aggregated channel in the downstream path; all upstream traffic follows the single WAN link associated with the shortest return path. No cross traffic is present on either LAN or WAN links. Each packet flowing downstream is received by a piconet member, and immediately forwarded to the destination via the receiving device's LAN interface.

At the receiving-end we assume that devices belong to a single, self-organizing piconet. Each device is exactly one wireless LAN hop away from any other device. Piconet membership is dynamic; a newly-arriving device can join an existing community and contribute its (partial or full) WAN channel to the resource pool. Member devices can also leave the community — typically without prior announcement — due to either device failure or movement out-of-range of LAN communications. We will assume that a mechanism exists to detect such departures from the resource pool, though packets may be lost until the system can detect and recover from that resource loss.

Even with this remarkably simple system model we are immediately faced with several intriguing questions. What is the performance loss associated with an unannounced departure of a single, actively-used WAN channel? How does this performance loss vary with the traffic type traversing the channel? What is the minimum time duration that a newly-arriving channel participates in an aggregated channel such that its throughput is increased?

To begin to address this last question, consider the decision of whether to run a TCP connection over a single, persistent link, or an inverse-multiplexed connection comprising that same link *plus* a second link of equal capacity alternating between connected and disconnected states. It is intuitive that though the multiplexed connection might *promise* greater average bandwidth capacity, the fluctuating presence of the second link may result in TCP window size reductions in response to packet losses, such that the two links can have lower throughput than the single persistent link. See the appendix for a more rigorous development of a method for best channel selection.

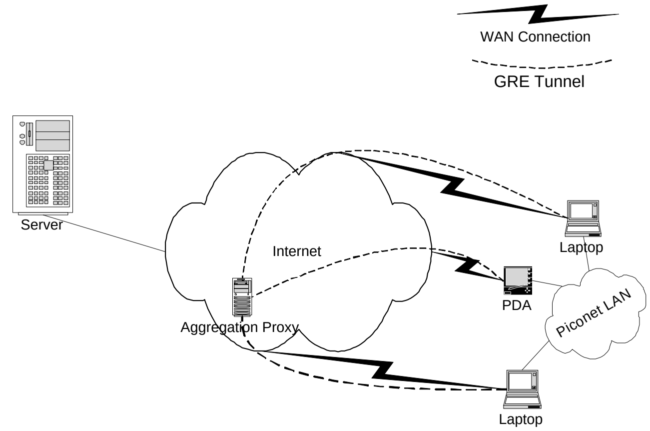

Fig. 1. A bandwidth aggregation service architecture.

The challenge of designing an effective inverse multiplexing system becomes far harder when we recognize that the components are heterogeneous, imperfect, and supporting time-varying workloads. For example, WAN link transmission characteristics (i.e., bandwidth, packet latency, loss) will vary, possibly dramatically as end-devices move around. Links from different service providers may be of dissimilar technologies with different costs, complicating link selection. Links of the same type from a single network operator might have dependent or correlated transmission characteristics or outages.

The potentially large latencies introduced by packet forwarding through power- and processing-limited mobile computing devices is also a challenge. Disparities in the forwarding latency on different paths traversing heterogeneous computing devices with time-varying computing workloads can introduce packet misordering in the end-to-end path that can affect certain applications adversely. For example, non-interactive multimedia streaming applications will typically be lightly affected, though larger client buffer capacities might be desired. Although packet reordering might not reduce multimedia application performance noticeably, it can complicate TCP RTT computation and decrease TCP throughput. Packet reordering is not uncommon in today's Internet [4], and in the event that reordering becomes significant, there are approaches that can mitigate performance degradation [5].

Another key issue in our overall system design is the identification of the preferred protocol layer for the multiplexing function. Since IP performs routing and multiplexing, it is natural to consider a network layer multiplexing implementation. An IP-based solution could be implemented exclusively at the communicating end-systems; in this case any packet scheduling, reordering, and reassembly would occur, as usual, only at the source and the destination. Though such a network layer implementation can be achieved in several ways, each requires end-system kernel modification, restricting the availability of channel aggregation to data transfers between modified end-systems. An additional disadvantage of network layer striping is that it could restrict the channel assignment policies (i.e., the intelligent mappings of flows to available channels) that we might seek to implement, since the network layer is generally not aware of application characteristics and requirements. Performing multiplexing at the network layer does have the advantage that it would not require any changes to existing applications.

An alternative solution is to perform multiplexing at the transport layer. Once again, end-system protocol stacks would require modifications, though transport-layer channel assignment policies could potentially be made more easily aware of application requirements. The obvious deployment issues associated with either network- or transport-layer multiplexing suggest a role for solutions using application-layer multiplexing. Although such an implementation would incur more packet processing overhead, it requires no kernel modification and is easy to install, maintain and monitor. Application layer multiplexing also permits controlling packet scheduling on a per-application, per-connection or per-packet priority basis.

What forwarding mechanism should an inverse multiplexer use to transmit a packet over a chosen channel? Irrespective of a packet's destination, different packets must traverse different routes. There are several means of achieving this. One approach is to change each packet's destination address to the IP address of the appropriate piconet member's WAN interface. When a packet arrives at the piconet, its destination address would be reverted back to the original piconet member destination address. This would, in a sense, be similar to providing a Network Address Translation (NAT) service, albeit in a distributed manner. But packet modification and processing overhead at the forwarding nodes associated with this approach would be prohibitive.

Another packet forwarding approach could use *loose source routing* to forward a packet through the intermediary interfaces associated with the desired WAN channel to traverse. This would avoid the need to provide a special NAT-like packet forwarding service beyond ordinary IP routing itself. However, loose source routing has multiple, well-known weaknesses (e.g., use of IP options, extra router processing) as well as limited router support, making its use largely unworkable.

A preferred packet forwarding implementation would use *tunnels* between the inverse multiplexer and each piconet node. Tunneling has long been used to establish static paths [38], and most operating system network stacks today have built-in support for tunnels. In such a system packet forwarding would operate as follows. Unicast packets sent from an Internet-connected source would be routed normally to the inverse multiplexer, where each would then be forwarded, according to the multiplexer's flow-to-channel assignment policy, to the tunnel corresponding to the appropriate WAN channel. Upon arrival at the piconet node, the packet would be decapsulated and forwarded on the wireless LAN to its intended destination. In this simple case, all upstream traffic would be sent over a single WAN link, typically — but not necessarily — the receiver's own. Figure 1 shows a bandwidth aggregation service architecture using Generic Routing Encapsulation (GRE) [14] tunnels.

Another key question in the design of our system is the appropriate placement of the inverse multiplexer in the end-to-end connection. In principle, this function can be located at almost any point between the WAN link terminations and the connection end-point (e.g., origin server), including the end-point itself. The preferred location depends on many factors including the type of WAN links, whether collaborating devices agree to connect to a common multiplexing point, and how generally accessible the multiplexing service must be from a wide range of origin servers.

If all the WAN links from a piconet terminate at the same point, a preferred location for the inverse multiplexer is that termination point. It is natural to think of a *proxy* providing this service, and to ease our discussion, we will simply use this term to refer to the location of the inverse multiplexer, regardless of whether a distinct physical component is used to implement the function. If the proxy is located near the WAN link termination points, then it is likely easier and more efficient for a wide range of services to use the proxy to transfer data to the piconet. The proxy can also be located at the network edge close to the origin server, or even at the origin server itself. While this location avoids the potential restriction of requiring a common WAN link termination point, piconet members might have to communicate with different aggregation services to communicate with different servers.

One of the most common but disruptive events affecting a flow traversing an aggregated channel is a sudden, unannounced channel loss. Depending on the reason for channel unavailability, it is necessary for either the proxy or surviving piconet members to detect the departure, and the process of detection and informing the proxy might be slow. In the meantime, a substantial number of packets might be lost. Upon detection of a lost channel a proxy's highest priority is to quickly reassign the flow component associated with the lost channel to other available channels. In general, recovery of lost packets is the responsibility of the affected end-systems, not the proxy.

In the remainder of this paper we will focus on the proper design of a proxy's channel allocation and packet striping algorithms, and show that such a design can achieve significant performance gains. There are many other intriguing issues beyond the scope of this paper, including handling malicious piconet community members, retaining privacy of information transmitted through piconet devices, as well as a host of questions associated with more sophisticated topologies involving devices participating in multiple piconets, connections transmitted over one or more inverse-multiplexed hops (in sequence or in parallel), and fairness issues between flows being assigned to channels.

III. Channel Allocation and Packet Striping

For each active flow a proxy is responsible for two tasks. First, the proxy must select a set of channels on which to forward packets to the piconet destination. Second, the proxy must intelligently stripe arriving packets across those channels. Efficient channel allocation and striping algorithms map or remap the flows to the channels based on both application requirements and the number and the condition of available channels. Hence, the algorithms we examine in this section are both *application-aware* and *channel-adaptive*. As an example, the algorithms we consider would seek to assign a flow from an audio or video source to channels that would maintain that application's stringent delay or delay jitter requirements, while assigning bulk data transfer (e.g., FTP) flows to channels that might incur longer delays but are reliable. Of course, both the number and condition of assignable channels might vary over a given flow's lifetime. Channel allocation and striping algorithms can be categorized along the following orthogonal dimensions:

- **Channel-adaptive**: These algorithms assign packets on different channels according to the channel conditions such as

TABLE I

CATEGORIZATION OF CHANNEL ALLOCATION AND SCHEDULING.

	Application-aware	Application-agnostic
Channel adaptive	Layer Priority Striping	WRR, WFQ
Channel non-adaptive	Not applicable	Random, Round-robin

bandwidth, loss, and delay. For example, a Weighted Round Robin (WRR) algorithm stripes packets to channels in proportion to each channel's bandwidth.

- **Application-aware**: Striping algorithms can also use knowledge or a *profile* of an application flow and its end-system requirements for channel selection and packet striping. Since applications can have different profiles, each application would potentially need a different algorithm. These algorithms promise to provide better performance than application-agnostic algorithms, but they have the burden of obtaining information about a flow's requirements. This information can be obtained explicitly from the traffic source, or may be inferred by examining the flow itself, or some combination of both. For instance, a source might mark its packets (e.g., ToS field in the IP header) or a proxy might infer application type from destination information (e.g., TCP or UDP port numbers) or even the application payload.

A given striping algorithm can be both channel-adaptive and application-aware, as summarized in Table I.[1]

We now (1) define and characterize application requirements, and (2) describe how to monitor and update channel characteristics before presenting illustrative algorithms for intelligently mapping/remapping and scheduling flows to available channels, using video as a motivating example.

A. Application Characteristics

Each application flow can be described by itself (intra-characterization) or against other application flows (inter-characterization). Examples of the former include Multiple Description video Coding (MDC) [2, 37] and the imprecise computation model [20] that is widely used in the real-time computing community. That is, an application flow has multiple representations or versions expressing different degrees of satisfaction (being minimally-to-fully satisfactory). The proxy must allocate and schedule resources to at least guarantee the minimum degree of satisfaction for each given application flow. That is, timely delivery of the base layer or essential part of each application flow must be guaranteed, and the enhancement layer or the optional part receives lower priority.

On the other hand, the inter-characterization deals with relative importance among different applications, rendering their priority order. In general, it is more "beneficial" to give more important application flows priority over less important ones in scheduling their data transmission or allocating bandwidth.

An application flow itself is also characterized by its minimum packet interarrival time, burstiness, multiple QoS levels, bandwidth, loss rate, delay, and jitter requirements.

[1] Application-aware, channel-adaptive algorithms are application-specific. We propose layer-priority striping for hierarchically-layered videos in Section III-D.

B. Channel Characteristics

The number and condition of channels between the proxy and piconet can change with time due to many factors including inter-channel interference, and communication failure due to piconet member departure, device mobility, or power depletion. While a proxy must be continuously aware of channel conditions, it does not have the benefit of observing packet reception or piconet member behavior directly. The proper design of a monitoring system providing such feedback is rather complicated, and deserves to be covered in a separate paper. Here we will assume the existence of a two-sided channel monitor (i.e., one side on the piconet and the other side at the proxy) that is jointly responsible for detecting membership changes, "sensing" channel characteristics (e.g., bandwidth, error rate, latency, security, reliability, cost, etc.) and ensuring that the proxy has reasonably current channel information.

The proxy is thus capable of ordering channels in its resource pool according to the application requirements of arriving flows. For example, channels can be sorted according to their delay and reliability characteristics, and then the proxy may choose the n most reliable channels for transporting the base layer (or essential part) of a video flow while choosing less reliable channels for the enhancement layer.

C. Allocation/Reallocation of Channels

Each application flow f_i; $1 \leq i \leq k$, is assumed to have been demultiplexed into an ordered (according to the application characteristics) set of $n_{f_i} \geq 1$ subflows $\{sf_j : j = 1, \ldots, n_{f_i}\}$. The traffic of each subflow sf_j is represented by either a simple token bucket model (ρ_j, σ_j) or a linear bounded arrival process $(p_j, s_j^{max}, b_j^{max})$ [10], where

ρ_j:	average token drain rate,
σ_j:	bucket size,
p_j:	minimum or average time separation between two consecutive packets,
s_j^{max}:	maximum packet size (in bytes),
b_j^{max}:	maximum burst size (in bytes) for subflow j.

Let $C = \{ch_\ell : \ell = 1, \ldots, n_c\}$ be an ordered (according to their condition) set of channels available. Note that the size and ordering of this set changes with time and will be updated by the monitor. The problem is now to select one or more channels from C on which to assign each subflow j. This selection must also be adapted to reflect the changing number and condition of available channels.

We first treat the simple case of only one application flow between a proxy and a piconet, and then the more general case of multiple application flows.

1) Case I: Single Application Flow: We want to map a demultiplexed application flow $f_i = \{sf_j^i : j = 1, \ldots, n_{f_i}\}$ to a dynamically-changing set of channels $C = \{ch_\ell : \ell = 1, \ldots, n_c\}$. Recall that the subflows of f_i are ordered according to their importance to the application, while the channels are ordered according to their relevance to the application requirements. For example, $f_v = \{sf_1^v, sf_2^v\}$ and $C = \{ch_1, ch_2, ch_3\}$ where sf_1^v and sf_2^v represent the base and enhancement layers of a video stream f_v, respectively, and ch_i's are ordered according to their reliability or their signal-to-noise ratio values. In this case sf_1^v may be transported via ch_1 and ch_2, and sf_2^v via ch_3, assuming that the former requires two channels while the latter requires only one channel.

In general, as many topmost (say, k) channels as necessary for transporting sf_1^i are assigned first to sf_1^i, and then repeat the same procedure with the remaining channels for sf_2^i, and so on. If sf_1^i does not need the entire bandwidth of channel ch_k, the remaining bandwidth of this channel is assigned to sf_2^i, and ch_k will transmit the packets of sf_1^i and sf_2^i using a Weighted Round-Robin (WRR) scheduling algorithm where the weights between the two subflows are determined based on the ch_k's bandwidths assigned to sf_1^i and sf_2^i. Also, if there is not enough bandwidth available, the least important subflows are not transported at all, realizing a form of imprecise computation [20].

The actual number of channels to be allocated for each subflow are determined by the subflow's requirements of delivery delay or bandwidth. For example, one can compute the bandwidth and delay requirements of both the base and the enhancement layers for layered videos, and derive the effective bandwidth of each channel from its raw bandwidth and loss-rate information.

2) Case II: Multiple Application Flows: In the case where there are multiple application flows f_i, $i = 1, \ldots, n_f$, the channels between the proxy and the piconet must be shared among these flows according to the relative importance of the flows and the channel condition. We now order applications flows according to their relative importance, and allocate channels to the application flows, exactly in the same way as the channels are allocated to the subflows in the previous subsection. Multiple application flows of the same importance share some channels using WRR where the weights are assigned according to their bandwidth requirements.

If (weighted) fairness is used instead of importance, or if all channels are of the same quality, one can use a weighted round-robin scheduling algorithm to "serve" the different application flows. If multiple flows are multiplexed on a channel, packet transmissions of the multiplexed flows can be scheduled using either WRR or Weighted Fair Queueing (WFQ) to reflect the difference in the flows' bandwidth requirements, or Rate-Monotonic (RM) or deadline scheduling [19] for delay guarantees.

D. Example: Assignment of Video Flows

The potential benefit of application-aware channel assignment is best illustrated by considering the case of video traffic. First, a high-quality video flow might be of sufficiently high bandwidth that it could not be transmitted over a single WAN channel. Second, link transmission characteristics can directly affect the perceived quality of the transmission. We present three 'strawman' algorithms, based on simple heuristics, for striping video packets.

- **Layer-Priority Striping (LPS)**: This algorithm can be used for video streams that are hierarchically layer-coded [24, 36]. This encoding process generates a base layer ℓ_0 containing information required for decoding, and one or more optional enhancement layers ($\ell_i : i = 1, \ldots, n$) in a hierarchical structure of cumulative layers. The reconstruction is progressive (i.e., enhancement layer ℓ_k can only be used if all sublayers $\ell_i : i = 0, \ldots, k-1$ are available). Thus, the layer index i corresponds to the layer priority.

  The LPS algorithm matches the layer-priority to the channel reliability as described in Section III-C.1. For instance, the base layer (ℓ_0) is assigned to the most reliable channels, where the channel loss rate is used as the metric for reliability. The packets for each layer are striped in WRR fashion onto the allocated channels. If a new channel with higher reliability becomes available, allocation of layers is shifted up to channels with higher reliability. Similarly, if the channel with the highest reliability becomes unavailable, the allocation is shifted down.
- **Frame-Priority Striping (FPS)**: This algorithm can be used for MPEG video traffic [17]. The MPEG video stream is separated into three subflows (sf_I, sf_P, sf_B) based on frame types. The priority order for the frames in MPEG Group of Pictures (GoP) is I>P>B. Similar to the LPS algorithm, the channels are allocated according to the subflow priority. The I-frame subflow (sf_I) is sent over the most reliable channels, and so on.
- **Independent-Path Striping (IPS)**: This algorithm is well suited to multiple state video coding [2, 37], where a stream is encoded into multiple *independently* decodeable subflows. Moreover, information from one subflow can be used to correct the errors in another subflow. Hence, it is important for a receiver to successfully receive as many complete subflows or components as possible, and it is desirable to achieve a low correlation of loss across different subflows.

  The IPS algorithm tries to achieve path diversity by allocating a separate channel for each description. Since the video can be reconstructed (albeit at lower quality) even if one or more entire subflows are lost, video reception is protected against one or more complete channel failure(s).

We will later show using simulation that even these simple algorithms based on heuristics can improve video quality significantly in realistic settings.

IV. Architecture

Considering the many systems issues identified in Section II, we chose a channel-aggregation architecture that is both simple and scalable. Figure 1 shows the proposed architecture which permits deployment by various types of network transport and service providers, including content owners, Internet access providers, wireless telecommunication service providers, or content distribution network operators.

The system architecture has three principal components: a dedicated appliance providing channel-aggregation proxy services, standard LAN-based announcement and discovery protocols, and standard protocol tunnels. The dedicated aggregation proxy performs inverse multiplexing at the application layer.

Generic Routing Encapsulation (GRE) [14] tunnels are used to create channels between the proxy and participating piconet members, and support packet forwarding. This approach requires no modification to piconet members, as most operating systems (Linux, FreeBSD, Windows, etc.) today have built-in support for GRE tunnels. Each packet received by a piconet member over a GRE tunnel is automatically decapsulated and forwarded via the wireless LAN to the destination device. Since the destination is oblivious to which piconet node forwarded the data packets,

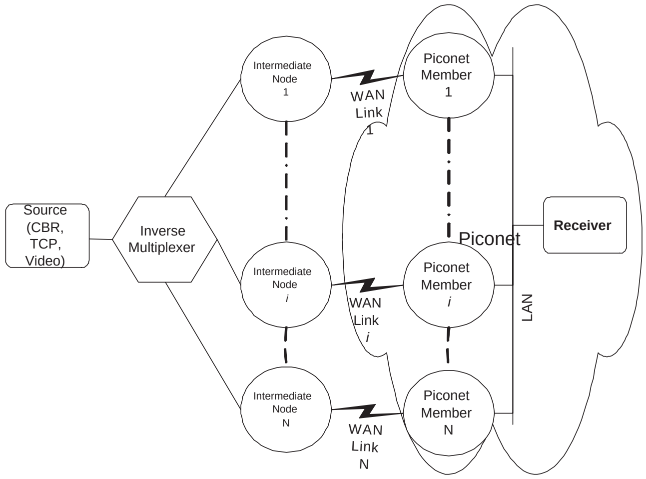

Fig. 2. Simulation topology.

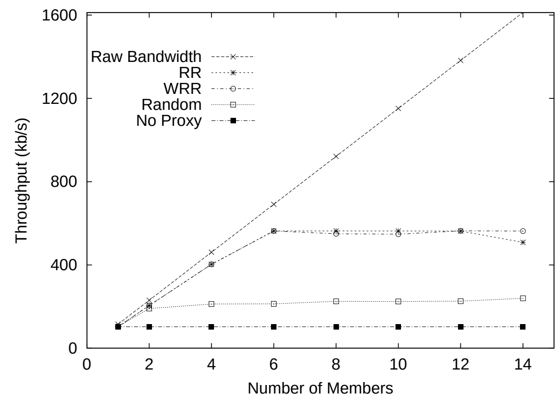

Fig. 3. TCP throughput as a function of piconet size.

no additional data reassembly functionality is required at the receiver.

To participate in piconet formation and channel aggregation, a standard announcement and discovery protocol is required on end-devices. The choice of a standard protocol enables end-devices to participate in other types of resource or service discovery and access. Though the specifics of these protocols are beyond the scope of this paper, Jini, Universal Plug and Play (UPnP), and the Service Location Protocol (SLP) [13] may all be suitable candidates.

The performance gains that our channel aggregation can realize will be explored through simulation and implementation in Sections V and VI. These benefits come at the expense of some computing and communication overhead. Note, for example, that it will not be possible in general to have a proxy on the shortest path between a source and a destination. Clearly, both an application-layer proxy as well as tunneled channels incur packet-processing overhead. However, since the total transmission bandwidth of an aggregated channel will ordinarily be modest ($<$ 2Mb/s), we anticipate that a dedicated proxy will be capable of managing a very large number of incoming flows and outgoing aggregated channels.

V. Performance Evaluation: Simulation

We evaluated the proposed bandwidth aggregation system using the *ns-2* [28] simulator. Figure 2 shows the network topology we used for simulating an entire end-to-end system. The number of piconet members was varied from 2 to 14, and those piconet members were interconnected via an 11Mb/s wireless LAN. In our experiments with homogeneous WAN links, the link bandwidth was set at 115.2kb/s, consistent with currently-available 2.5G cellular services. With the exception of the single dedicated receiver, each piconet member was equipped with both a WAN and a LAN interface. The receiver could only communicate upstream using one of the other members as a gateway. We consider a variety of scenarios with varying link characteristics such as bandwidth, loss, and membership dynamics. We first evaluate the benefits of bandwidth aggregation for different applications: we use (1) bulk file transfer over TCP and measure TCP throughput, and (2) CBR traffic over UDP and measure packet loss rate and delay jitter. We then study how much performance improvements application-aware striping can make using layered video as an example application. For experiments with TCP and UDP traffic we implemented three application-agnostic striping algorithms: random, round-robin (RR), and weighted round-robin (WRR).[2] We implemented the LPS algorithm described in Section III-D for application-aware, channel-adaptive striping algorithms.

A. TCP Throughput

We first evaluate the effect of the addition or deletion of a WAN link in an aggregated channel on TCP throughput. Let's consider the simple case of a fixed membership piconet. We measured TCP throughput by transferring a 1MB file from a data source to a piconet receiver using $2 \sim 14$ identically-configured links aggregated into the shared pool. To provide a baseline for measured TCP throughput, we also performed the experiment with a single channel (i.e., no aggregation).

Figure 3 plots the measured TCP throughput as the piconet size changes. The average throughput achieved with a single link was 103.2kb/s. As expected, the TCP throughput increases nearly linearly as the number of links grows under both RR and WRR policies until saturation occurs with six links. This saturation occurs due to the limit imposed by the receiver's maximum window. As the number of available channels increases, the bandwidth-delay product increases, but TCP cannot utilize all the available bandwidth because of the small receiver window. The TCP throughput continues to increase linearly if the receiver-advertised window is increased to accommodate a larger bandwidth-delay product. The random policy does not perform as well as (W)RR because it causes undesired side effects, such as packet reordering and unstable RTT calculation, thus reducing the TCP throughput.

We next explore TCP performance for the highly-dynamic case where the channels were frequently added or removed from the pool. It is difficult to predict the likely rates of joins and leaves in a piconet, as the behavior will likely change dramatically with the actual setting (e.g., a bus or a conference room).

[2] To be precise, since packets are not fragmented in the proxy we have implemented the Surplus Round Robin approximation of bit-WRR.

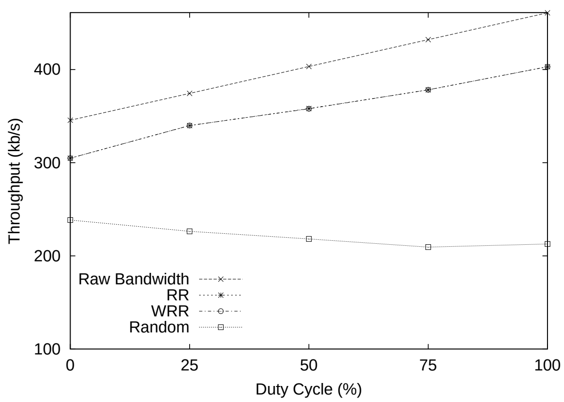

Fig. 4. TCP throughput with 3 persistent links and 1 transient link.

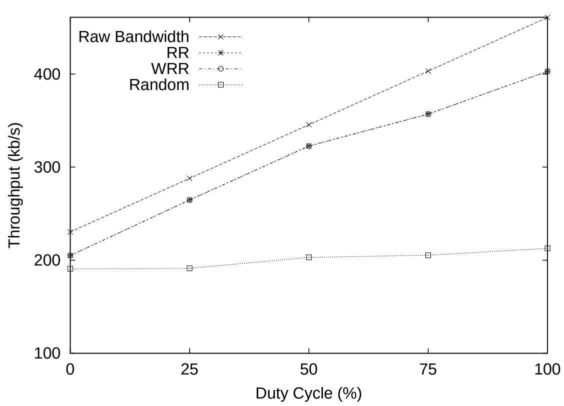

Fig. 5. TCP throughput with 2 persistent links and 2 transient links.

Hence, we conducted a variety of experiments to study join and leave dynamics, repeating the file-transfer scenario described earlier and measuring the TCP throughput. In this set of experiments there was no significant difference in the achieved throughput for RR and WRR striping. Hence it is difficult to distinguish between the two in the figures presented here.

1) Case I: 3 persistent links, 1 transient link: In this scenario, three links always remain active in the pool. The fourth link periodically joins the pool for *up-time* and leaves for *down-time*. The sum of *up-time* and *down-time* was kept constant at 20 seconds. That is, an *up-time* of 20 seconds is same as striping continually over four links (i.e., 100% duty cycle) and a *down-time* of 20 seconds is the same as continually striping over only three links. Figure 4 shows that as the duty cycle increases, the TCP throughput increases for RR and WRR schemes, whereas the random striping cannot effectively utilize the available bandwidth of the transient link.

2) Case II: 2 persistent links, 2 transient links: This scenario is identical to the previous one, except that there are two links remaining active and two links being simultaneously added and removed from the pool. Figure 5 shows that as the duty cycle increases, the average TCP throughput increases for RR and WRR. Even though two of the links in the pool are rather short-lived, channel-adaptive striping is able to utilize their capacity to improve the transfer rate.

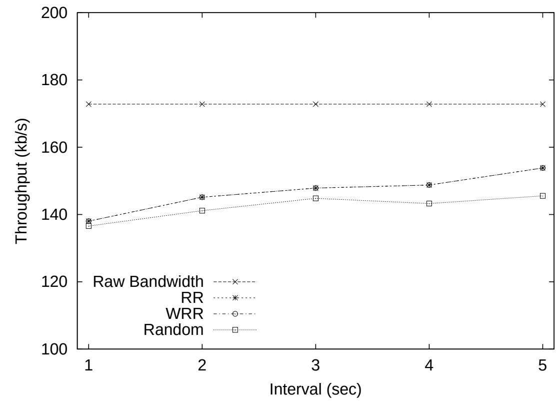

Fig. 6. TCP throughput as a function of *up-time* interval.

3) Case III: 1 persistent link, 1 transient link: In this scenario only one link is persistent and one link is periodically added and removed from the pool. We varied the length of the *up-time* interval from one second to five seconds. The duty cycle was kept constant at 50% by using same value for *down-time* and *up-time* intervals. Figure 6 shows the TCP throughput as the interval is varied. Although the duty cycle is constant, the TCP throughput slightly increases with the length of *up-time* interval. Thus, we observe that TCP throughput varies with not only the frequency of change in the number of links, but also with the length of the change intervals.

We also measured the TCP throughput by transferring a 1MB file over an aggregated channel consisting of four links with unequal bandwidths of 128kb/s, 64kb/s, 32kb/s, and 16kb/s. The throughput achieved by Random, RR, and WRR striping was measured at 41.2kb/s, 44kb/s, and 55.6kb/s, respectively. It is interesting to note that —even for WRR— the throughput for the aggregated channel is less than the highest bandwidth of a single link. Since the proxy does not fragment packets and, instead, uses an approximation of bit-WRR, there is frequent packet misordering if the link bandwidths vary greatly. The effect of link bandwidth disparity in TCP throughput is explored in [30]. Several techniques as weighted packet fragmentation [34] and multiple parallel TCP connections [16] can be adopted to address this problem.

B. *CBR Media Traffic over UDP*

Many media applications generate CBR traffic carried over UDP. We studied the loss and jitter observed for a 920(=8×115)kb/s CBR stream from a video source to a piconet destination. The RTP delay jitter as described in RFC 1889 [32] was measured at the receiver. The topology used for this set of experiments was the same as the one for the TCP throughput experiments.

TABLE II

CBR LOSS RATE (%) AS A FUNCTION OF PICONET SIZE.

# of members	Random	RR	WRR	No proxy
2	75.15	75.15	75.15	87.57
4	50.31	50.3	50.32	87.57
6	25.48	25.45	25.5	87.57
8	1.14	0.61	0.59	87.57
10 or more	0	0	0	87.57

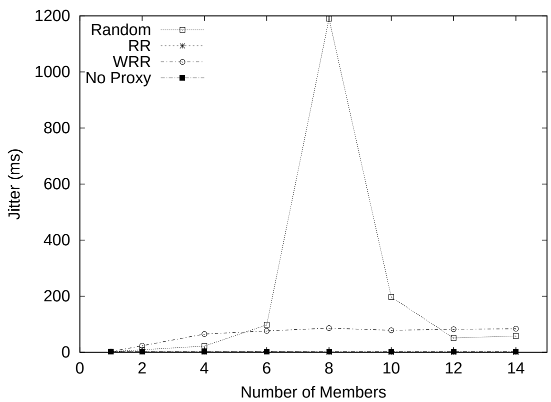

Fig. 7. CBR jitter as a function of piconet size.

Table II shows the packet loss rate as a function of the piconet size. Without channel aggregation we observe 87.5% loss as the CBR stream rate was eight times the bandwidth of a single link. As more links are pooled, the loss rate decreases. Figure 7 shows that except for random striping, the jitter values remain largely unaffected by channel aggregation. With random striping, the jitter increases as the number of piconet members increases to eight. The maximum jitter value of 425ms was observed with eight members, i.e., when the offered CBR traffic load is equal to the sustainable throughput of the pool. When there are more than eight piconet members, the jitter decreases as the number of piconet members increases because packets will be delivered to piconet members with less queueing delays over (assumed) homogeneous channels.

We also studied the performance of different striping algorithms for UDP streaming over four heterogeneous links of 128kb/s, 64kb/s, 32kb/s, and 16kb/s, respectively. Table III shows the loss rates when a CBR stream of 256kb/s is sent over the aggregated channel. Random and RR algorithms do not adapt to channel bandwidth and allocate an equal number of packets to each channel. Hence, the lower bandwidth links drop larger amounts of traffic, resulting in higher total loss rates. In contrast, WRR achieves a low overall loss rate by assigning packets proportionally to the bandwidths of various links and distributing the loss uniformly over different links. A small penalty is paid through a very slight increase in jitter under the WRR algorithm as shown in Table IV, but this small increase in jitter can be easily absorbed in the receiver buffer.

We also evaluated how CBR streaming over UDP is affected

TABLE III

CBR LOSS RATE (%) OVER FOUR HETEROGENEOUS LINKS.

	Random	RR	WRR
Link 1 (128kb/s)	0	0	14.1
Link 2 (64kb/s)	6.93	7.9	13.75
Link 3 (32kb/s)	53.67	53.95	13.06
Link 4 (16kb/s)	77.15	76.97	11.54
Total	34.18	34.4	13.25

TABLE IV

CBR JITTER (MS) OVER FOUR HETEROGENEOUS LINKS.

Protocols	Random	RR	WRR
Jitter	7.24	2.45	13.25

by the dynamics of piconet membership. Under the same join and leave dynamics as for the TCP throughput experiments, the loss rate decreased with the increase of duty cycle.

C. Application-Aware Striping

We now present the results from the application-aware striping experiments. We experimented with the application-aware, channel-adaptive *LPS* algorithm introduced in Section III-D. The scenarios were chosen so as to elucidate the key benefits of application-aware mechanisms in comparison with application-agnostic schemes.

1) Availability of Extra Channels: Let's consider a scenario where the proxy has 10 channels available for striping data. All the channels are identical except for having different error rates that vary from 1 to 10%. The error rate e_i for channel ch_i was set at i%. The traffic source generated CBR traffic at 30kb/s and the bandwidth of each channel was 20kb/s. Thus, at least two channels are required for the transfer. Table V shows the average loss rates for different striping algorithms. If the proxy is unaware of the application profile/requirements, then it will use all the available channels indiscriminately. Hence, the observed loss rate is higher for the application-agnostic striping algorithms. But a proxy using an application-aware algorithm achieves better performance by striping data over only the two channels with minimum loss. Hence, even minimal information, such as the bandwidth requirements of the application, can make a significant improvement in the system performance.

2) Priority-awareness: As we discussed in Section III, different packets in an application flow can have higher priority than others, such as base layer or I-frame packets. We now present the results for striping a hierarchically-layered video stream with a base layer ℓ_0 and two enhancement layers ℓ_1 and ℓ_2. Each layer was modeled as a 15kb/s CBR stream. The topology consists of three piconet members, each with a 20kb/s WAN link. The error rate on the channels was 1, 5 and 10%, respectively. Table VI shows the percentage loss rate suffered by each layer. As expected, the random striping indiscriminately distributes the loss over all the layers. Since all the layers are constant bit-rate with equal bandwidth and the number of channels is same as the number of layers, the RR algorithm stripes all the packets from one

TABLE V

LOSS RATE (%) WITH EXTRA AVAILABLE CHANNELS.

Protocols	Random	RR	WRR	LPS
Loss rate	5.56	5.58	5.68	1.41

TABLE VI

LOSS RATE (%) FOR LAYERED VIDEO WITH STATIC CHANNELS.

	Application-agnostic			Application-aware
	Random	RR	WRR	LPS
Layer ℓ_0	5.07	9.97	6.05	1
Layer ℓ_1	5.28	1.02	4.89	4.96
Layer ℓ_2	5.53	4.81	5.16	9.72

layer to one channel. Instead of the loss being spread over all the layers equally, the layer sent over the most unreliable link suffers the most loss. The loss rate for the base layer is significantly less with the LPS algorithm. LPS uses priority-awareness to assign the base layer to the most reliable link, and the highest enhancement layer to the link with the highest error rate.

The striping algorithms utilize application-awareness to intelligently drop lower-priority subflows when an insufficient amount of resource is available. To demonstrate this benefit of application, we simulated a scenario with two piconet members connected to the Internet via 20kb/s WAN link. The error rate of the channels was 1% and 5%, respectively. Note that the offered traffic rate exceeds the aggregated channel bandwidth. Table VII shows the loss experienced by different layers while streaming the same video traffic as described above. Since the two available channels cannot handle the offered load of all the three video layers, the LPS algorithm drops the layer ℓ_2 entirely, improving the loss suffered by the base layer ℓ_0 and the enhancement layer ℓ_1. The application-agnostic mechanisms end up spreading the loss over all the layers.

3) Dynamic Channel Adaptation: What happens if in the above scenarios the link error rates change dynamically? Let us assume that each link has an error rate of 1% for 100 seconds and then 10% for 50 seconds, repeating this cycle several times during the lifetime of the flow. The changes in error rates are distributed such that at any instant two links have error rate of 1% and one link has error rate of 10%. Thus, the total error rate is the same throughout the experiment. Table VIII shows the measured loss rates for this experiment. Once again, in the case of application-agnostic schemes, lack of application knowledge leads to uniform loss rates for all the layers of the flow. In contrast, LPS is able to protect the base layer from loss, and instead increase the loss rate of enhancement layers.

We also simulated the limited channel scenario described earlier with varying channel error rates. At any instant one link experiences 1% error rate and the other 10%. Layer-wise measured loss has been shown in Table IX. In this case too, with random, RR and WRR striping, all the layers suffers similar loss. As before, LPS entirely drops the enhancement layer ℓ_2 due to limited channel availability, to shield layers ℓ_0 and ℓ_1 from loss. Also, it remaps the base layer to the more reliable channel as the channel error rates change. Hence, the loss suffered by the base layer is

TABLE VII

LOSS RATE (%) FOR LAYERED VIDEO IN LIMITED STATIC CHANNELS.

	Application-agnostic			Application-aware
	Random	RR	WRR	LPS
Layer ℓ_0	18.99	23.57	18.76	0.96
Layer ℓ_1	19.64	12.44	20.53	5.15
Layer ℓ_2	19.89	22.4	19.25	100

TABLE VIII

LOSS RATE (%) FOR LAYERED VIDEO IN DYNAMIC CHANNELS.

	Application-agnostic			Application-aware
	Random	RR	WRR	LPS
Layer ℓ_0	3.87	4.09	4.04	0.91
Layer ℓ_1	3.99	3.93	4.18	1.08
Layer ℓ_2	4	4.24	3.97	10.11

lower and similar to the static case.

It is important to note that in these experiments we assumed that the monitoring agent is continuously measuring the link conditions and the proxy is informed instantaneously of any change. Due to resource limitations and propagation delays this assumption may not be true in practice, and we would expect to see somewhat higher loss rates. The design of a monitoring agent for our system that balances the need to keep the proxy informed of current link state while efficiently using bandwidth and processing resources is an ongoing work. Similarly, the performance of application-aware mechanisms is subject to accuracy of application traffic profile.

VI. IMPLEMENTATION, EXPERIMENTS AND RESULTS

We now present a detailed description of the channel aggregation testbed we built and the experiments we performed. The principal goal of the testbed was to corroborate our proposed architecture and explore deployment issues that might not readily emerge from our simulations.

A. Testbed Implementation

Figure 8 shows a block diagram of the prototype channel aggregation system we constructed, with dark arrows representing control messages and light arrows representing data traffic. Each piconet member runs a compact *Client Connection Manager* (CCM) application. The CCM participates in the announcement and discovery of piconet members (and their associated WAN links). Though we anticipate that standard announcement and discovery protocols would be used in an actual system, resource discovery was done manually in our testbed. This gave us precise control over piconet membership, facilitated automated testing, and allowed us to modify resource availability on very short time scales.

The CCM communicates the addition or deletion of links to the *Server Connection Manager* (SCM) which resides on the proxy and maintains the channel resource pool. The CCM also monitors link transmission characteristics such as bandwidth and delay that is provided to the striping proxy. The CCM can also request the striping algorithm to be used for an aggregated channel.

TABLE IX

LOSS RATE (%) FOR LAYERED VIDEO IN LIMITED DYNAMIC CHANNELS.

	Application-agnostic			Application-aware
	Random	RR	WRR	LPS
Layer ℓ_0	18.88	22.9	18.9	1.01
Layer ℓ_1	19.73	12.78	20.75	4.97
Layer ℓ_2	19.96	22.76	18.88	100

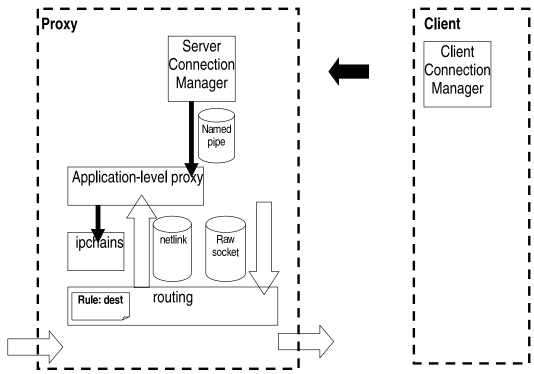

Fig. 8. Linux-based implementation of an aggregation proxy.

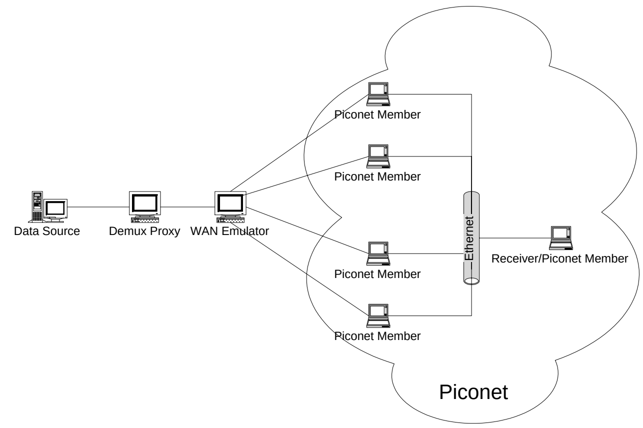

Fig. 9. Experimentation testbed topology.

The SCM and CCM together also coordinate setup and teardown of the GRE tunnels [14] between the proxy and the piconet members.

We implemented a Linux-based inverse multiplexing proxy. The proxy intercepts each packet destined for a piconet and forwards it to the GRE tunnels corresponding to each active channel. Packet interception at the proxy is handled by *Netfilter* [26], a packet filtering subsystem in Linux that is primarily used for building firewalls and NATs. For each channel aggregate, the proxy sets up Netfilter's forwarding rules to intercept appropriate data traffic and passes it to the proxy's user-layer forwarding engine. The forwarding engine currently implements both random and round-robin data striping policies. Use of the IP address of a piconet member's WAN interface to set up the tunnel ensures that each packet sent over the GRE tunnel traverses the desired WAN channel.

Data reassembly at the receiving side is automatic and straightforward. Packet forwarding is enabled at each piconet node sharing a WAN link. When a packet is received by a node over a GRE tunnel, it is decapsulated and passed to the node's routing engine. Since the destination address of the decapsulated packet corresponds to the receiver's LAN address, the packet is forwarded to the LAN.

Figure 9 shows the topology of the testbed we used for emulating an entire end-to-end system; all subsequent results presented in this section are from experiments conducted on this testbed. The membership of the piconet in our experiments varied from two to five notebook computers running Linux (2.2.16 kernel), each with built-in support for GRE tunnels. We selected relatively low-performance systems with an eye toward ultimately supporting even lower performing handheld PDAs. Forwarding was enabled on each piconet node. Our proxy was implemented on a Linux-based desktop PC.

The piconet members were connected to each other via a 10Mb/s Ethernet. WAN links were emulated by connecting a wired serial null modem running PPP to the *NISTnet* [25] network emulator whose transmission link characteristics we could control. As in simulations presented in Section V, the transmission speed of each serial link was set at 115.2kb/s. Each piconet member, with the exception of the dedicated data receiver, had both an emulated WAN interface and an Ethernet interface. The data receiver could only communicate upstream to the Internet using one of the other members as a gateway.

Traffic generation, collection and measurement was performed using NetIQ's *Chariot* network measurement tool version 4.2 [27]. Chariot *end-points* running on the data source and receiver generated various packet flows, emulating reliable data transfers, streams, etc.

B. Experimental Results

1) TCP Throughput: To validate our simulation results in practice, we measured TCP throughput by transferring a 1MB file from a data source to a piconet receiver using two to four identically-configured, aggregated links. Each experiment was repeated 50 times, with the results averaged. To provide a baseline for measured TCP throughput we also performed the experiment with a single channel (i.e., no aggregation) both with and without the proxy in the data path. Performance was measured using both round-robin and random striping policies. Figure 10 plots the measured TCP throughput as the number of links in the aggregate bundle changes, with error bars showing the minimum and maximum measured throughput among the 50 trials.

The average TCP throughput achieved with no proxy was 45kb/s. The TCP throughput with a single link and the proxy in the data path is 38kb/s, not significantly lower than the throughput achieved without a proxy, indicating that the proxy does not introduce a long delay. The TCP throughput measured in the testbed was lower than the simulation results due to PPP overhead and the presence of background traffic. However, the trends with respect to the number of piconet members were similar in both the cases.

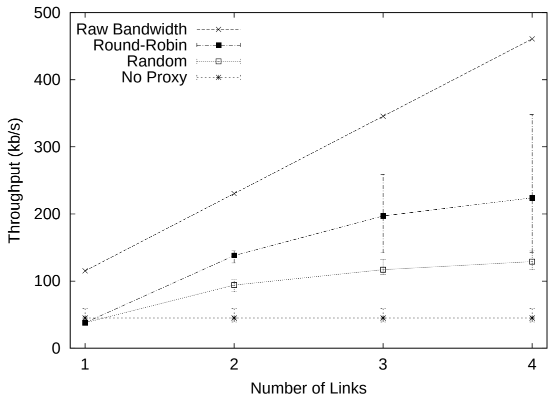

Fig. 10. Effect of piconet size on TCP throughput.

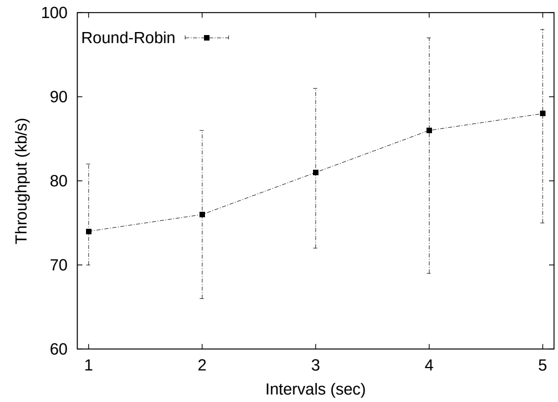

Fig. 12. Effect of *up-time* interval length on TCP throughput.

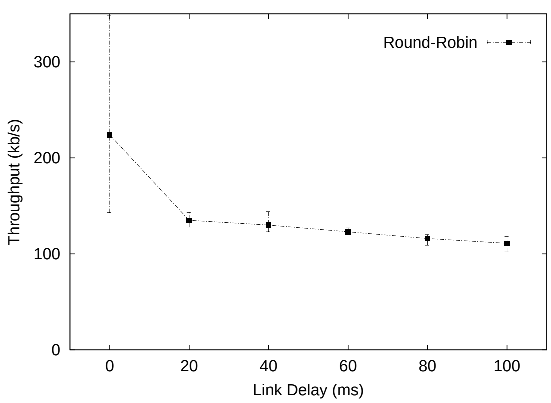

Fig. 11. Effect of link latency variation on TCP throughput.

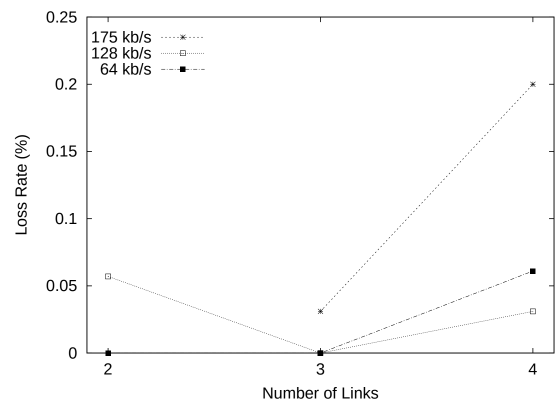

Fig. 13. Effect of piconet size on RTP stream loss rate.

To study the effect of varying transmission link parameters of different WAN channels on TCP throughput, we used the *NISTnet* emulator to add extra transmission delay to one of the WAN channels. Figure 11 shows the change in TCP throughput as the extra delay of one of the four links is varied from 0 to 100ms. As expected, increasing the link delay decreases throughput. There are two reasons for this. First, the increased delay can cause additional packet misordering, introducing reassembly delays. Second, the extra delay results in a larger computed value for RTT, directly decreasing throughput. With unequal link latencies, round-robin is not the optimal scheduling algorithm. Weighted fair queuing techniques such as that proposed in [34] can reduce packet misordering and hence improve TCP throughput.

We now measure the TCP throughput in a highly dynamic piconet where channels are added and removed from the resource pool. The topology is the same as we used in Section V-A. Table X shows the TCP throughput as the duty cycle was changed for two transient links among four total links. We observe that as the duty cycle increases, the average TCP throughput increases in most cases. Although some of the links in the bundle are rather short lived, link aggregation is nonetheless able to use their capacity to improve the transfer rate.

Figure 12 shows the TCP throughput when the length of *up-time* interval is changed for one link while the other link is persistent. The result verifies our earlier observation from simulation (Figure 6) that the interval duration as well as the frequency of change in the number of active channels affect TCP throughput.

2) Streaming Media via UDP: We next conducted experiments to study how high bandwidth streaming is enabled with channel aggregation. In these experiments a server streams a stored media file to the receiver at one of various bit rates (64kb/s, 128kb/s, 175kb/s, and 256kb/s). Chariot generates a traffic pattern intended to resemble the video transmission of Cisco's IP-TV. RTP [32] is used as the stream transport protocol. Each experiment was repeated 25 times, measuring the loss rate and RTP delay jitter observed by the receiver.

Without channel aggregation the receiver can only receive a stream with negligible loss at the 64kb/s rate. Higher bit-rate streams suffered more than 70% loss, and due to this high loss rate, the tests were prematurely terminated by Chariot. Note that the limited available bandwidth precludes use of retrans-

TABLE X
TCP THROUGHPUT (KB/S) WITH TRANSIENT LINKS.

Link up/down time	Duty cycle	One transient link			Two transient links		
		Avg.	Min.	Max.	Avg.	Min.	Max.
0 sec/20 sec	0%	197	142	259	138	127	145
5 sec/15 sec	25%	207	143	280	150	134	171
10 sec/10 sec	50%	188	143	291	161	140	219
15 sec/5 sec	75%	200	144	329	175	142	266
20 sec/0 sec	100%	224	143	348	224	143	348

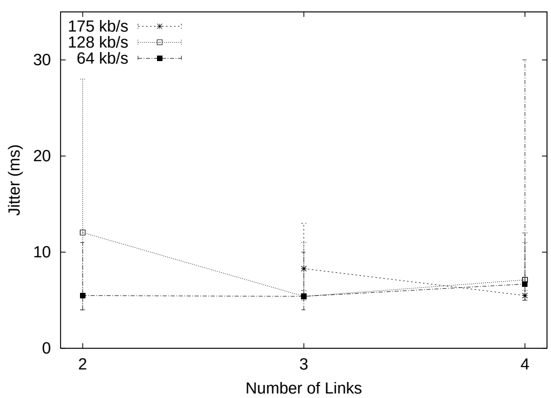

Fig. 14. Effect of piconet size on RTP stream jitter.

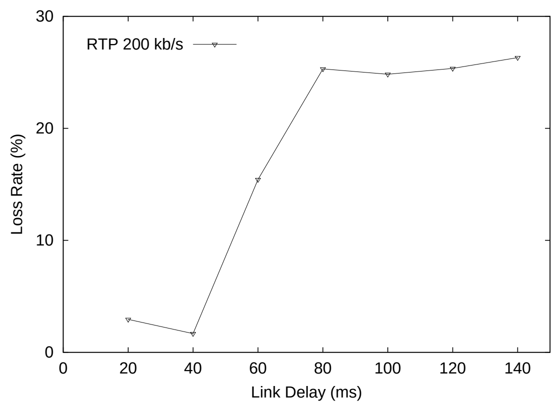

Fig. 15. Effect of link latency variation on RTP stream loss rate.

mission for loss recovery. Techniques such as Forward Error Correction (FEC) can not be used in this setting, especially for low-bandwidth links, as it further increases the bandwidth required. Such a high loss rate can severely degrade the perceived stream reception quality, making it unwatchable. Yet striping over just two links reduced the loss rate dramatically for the 128kb/s stream; every 128kb/s stream test completed with a loss rate of less than 0.1%. The 175kb/s streaming experiment with striping over two links was also terminated before completion due to high loss rate. Striping over four links was capable of sustaining a 256kb/s stream without significant packet loss. Figure 13 shows the streaming data loss rates. Observe that the data loss rate does not exceed 0.2% when dynamic link striping is performed.

Figure 14 shows RTP jitter values. Note that the system generates relatively little jitter. In most cases, the jitter is less than 10ms with the maximum jitter occasionally exceeding 20ms. Such small amounts of jitter can be easily absorbed by the receiver buffer in multimedia applications and will have negligible effect on the viewing experience of the video receiver.

We also studied how streaming is affected by heterogeneous WAN link delays. In this experiment the server streamed a stored media file to the receiver at 200kb/s. Each test was run 25 times and the results averaged. All the four WAN links were active and used for striping. An extra transmission delay of $0 \sim 140$ms was added to one of the links. Figure 15 shows the loss rate as the transmission delay varied. The average loss rate is very low for delays of 20ms and 40ms, but as the delay increased beyond 60ms, the loss rate increased. This result is an artifact of our measurement tool; the packets sent over the link with larger delay arrive "late" at the client and are considered lost. The loss rate is around 25% for 80ms delay and does not increase any further for larger delays. This is because almost all the packets sent over the larger delay link arrive late. Similar to the observation made in Section VI-B.1, the use of a weighted fair queuing algorithm could reduce this loss rate. In an actual noninteractive multimedia application, this kind of loss could also be reduced by merely increasing the receiver buffer size.

The effect of link delay variation on RTP jitter is shown in Figure 16. The jitter increases as the transmission delay of the link increases from 20ms to 40ms. After that point, the jitter decreases with increase in link latency. This corresponds to the increase in loss rate as shown in Figure 15. Once again this is a measurement artifact, as jitter in this region is only computed for received packets, while the late arriving packets are considered lost.

VII. Related Work

A combination of a strong demand for communication bandwidth and high tariffs on WAN links has long made inverse multiplexing a popular technique [12]. In the early 1990s the

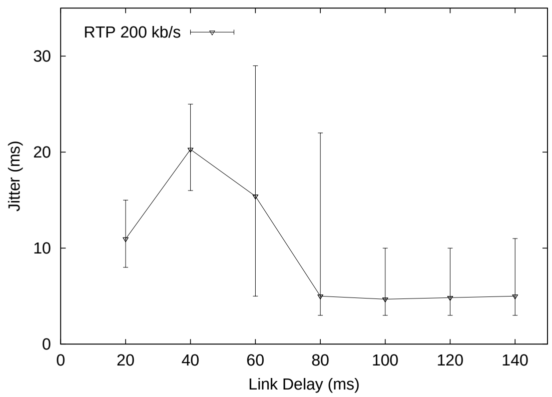

Fig. 16. Effect of link latency variation on RTP stream jitter.

Bandwidth on Demand Interoperability Group (BONDING) created a standard inverse multiplexing mechanism to achieve a virtual high capacity WAN link using $n \times 56$ (or 64)kb/s links [6]. Network equipment providers supported inverse multiplexing for various link layer technologies such as frame relay, ISDN, and SMDS. The same technique was later applied and standardized within the context of ATM networks in the Inverse Multiplexing for ATM (IMA) specification [3]. Each of these cases assumed highly reliable, homogeneous links with constant link characteristics such as capacity, delay, and error rates. Moreover, each WAN connection being bundled together originated from and terminated at the same endpoints. striping is achieved by splitting packets into equal-sized fragments and transmitting one fragment on each component link of the aggregate bundle. Extra framing information is added to the fragments to resequence the packets at the reassembler. Additional hardware is required, both for striping and reassembly. The link-layer aggregation is not visible to higher-layer protocols and can be implemented for any link (hop) of the network.

Various striping algorithms have been proposed and implemented to reduce packet reordering, jitter, and load imbalance. Round-robin scheduling is primarily used for striping data over homogeneous links, while variants of weighted fair-queuing algorithms are used in case of heterogeneous links. It has been shown that maximum throughput is achieved by striping data over each channel in proportion to the channel's bandwidth-delay product [1, 30].

More recent research has explored adaptive inverse multiplexing for CDPD wireless networks [34]. In this scheme the packets are split into fragments of size proportional to the observed throughput of component links. Here the goal is to create variable fragments sizes such that each fragment can be transmitted in roughly the same amount of time. The fragment size of each link is dynamically adjusted in proportion to the measured throughput. The fragmented packets are then tunneled over multiple links using Multilink PPP [33]. In this case the endpoints of the WAN connections forming the virtual link are the same.

The bandwidth of mobile users with multiple interfaces is aggregated at the transport layer in pTCP (parallel TCP) [15, 16]. pTCP is a wrapper that interacts with a modified TCP called TCP-virtual (TCP-v). A TCP-v connection is established for each interface, and pTCP manages send buffers across the TCP-v pipes. The striping is performed by pTCP and is based on congestion window size of each TCP-v connection. When congestion occurs on a certain pipe, pTCP performs data reallocation to another pipe with large congestion window. One possible problem of this approach is that the congestion window size may not accurately reflect the bandwidth-delay product.

Coordinating communications from *multiple* mobile computing devices has become a new focus of interest. Network connection sharing has been proposed in [29]. This architecture permits use of a single, idle WAN connection among collaborating mobile devices but it does not address aggregation of multiple links into a high capacity bundle.

Our goal of cooperation and resource aggregation among collaborating devices is similar to the vision of the mobile grouped devices (MOPED) architecture [8, 18]. The goal of MOPED project is to enable group mobility such that a user's set of personal devices appear as a single mobile entity connected to the Internet. The MOPED routing architecture builds a *multipath* layer to encapsulate packets between the home agent and MOPED devices. Unlike our approach of using GRE tunnels, the home agent and the mobile devices in MOPED must implement a new lightweight encapsulation protocol called *multipath routing encapsulation* (MRCAP). MOPED architecture provides a higher-capacity and better-quality connection to the Internet by adapting the MobileIP home agent to support aggregation of multiple links at network and transport layers. It uses transport layer inverse multiplexing for multi-homed devices [21]. Aggregating bandwidth at the transport layer requires different protocols for different applications. Their research presents two new transport protocols, namely: (1) R-MTP (Reliable Multiplexing Transport Protocol) [23] for data, and (2) M-MTP (Multimedia Multiplexing Transport protocol) [22] for multimedia. Additional transport protocols might be needed as the application requirements change. Modifications to both client and server kernels are also required. Our application level approach does not require any kernel changes and allows support for different application profiles.

VIII. Conclusion

We have designed, implemented and evaluated a deployable bandwidth aggregation system providing high-speed Internet access to a collaborating community of wireless end-systems. We have demonstrated that the system not only improves access service quality, but enables otherwise unachievable services such as the delivery of high-bandwidth streaming media. Further, we have shown that network and application-aware allocation and assignment policies do indeed improve system performance.

Though not described in this paper, we performed various experiments with bandwidth-adaptive multimedia applications over aggregated connections. Ideally, such applications would measure available bandwidth and smoothly increase or decrease audio or video quality to optimize perceived reception quality. We typically observed an application decreasing its streaming rate to a predefined fraction of its maximum rate; often this rate was well below the available bandwidth of the aggregated connection. The application would subsequently maintain that low rate, remaining unresponsive to any increase in available bandwidth, no matter how large it is. Since the widely-used applications we tested were proprietary, we were unable to modify their adaptation algorithms.

We have also identified a significant number of technical problems that appear to be fertile areas for future research. Some of these include WAN cost sharing, accounting, and security. Currently, we are evaluating the proper design of an efficient monitoring system that will allow the proxy to be rapidly informed of piconet membership changes, time-varying WAN channel communication characteristics (e.g., loss rate), network cross traffic, and piconet member computing workloads. Yet it may be that the primary barrier to adoption of this technology will be the unanswered social question: "Will potential system users choose to share their private WAN connections?" Lessons learned from the distributed computing research community's investigations of Networks of Workstations (NOWs) clearly included observations of reluctant participation in desktop computer sharing, largely due to the perception of security and privacy risks, as well as 'ownership' rights. Yet the success of peer-to-peer file sharing leads us to believe that device owners may be willing to share communication and computational resources as readily as they do information, particularly if they directly and immediately benefit from resource sharing.

References

[1] H. Adiseshu, G. Parulkar, and G. Varghese, "A reliable and scalable striping protocol," in *Proceedings of the ACM Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM'96)*, Stanford, CA, Aug. 1996, pp. 131–141.

[2] J. Apostolopoulos, "Error-resilient video compression via multiple state streams," in *Proceedings of International Workshop on Very Low Bitrate Coding (VLBV'99)*, Kyoto, Japan, Oct. 1999, pp. 168–171.

[3] ATM Forum, "Inverse multiplexing for ATM specification, version 1.0," July 1997.

[4] J. C. R. Bennett, C. Partridge, and N. Shectman, "Packer reordering is not pathological network behavior," *IEEE/ACM Trans. Networking*, vol. 7, no. 6, pp. 789–798, Dec. 1999.

[5] E. Blanton and M. Allman, "On making TCP more robust to packet reordering," *ACM SIGCOMM Computer Communication Review*, vol. 32, no. 1, pp. 20–30, Jan. 2002.

[6] BONDING Consortium, "Interoperability requirements for $n \times 56/64$ kb/s calls, version 1.0," Sept. 1992.

[7] L. F. Cabrera and D. D. E. Long, "Swift: Using distributed disk striping to provide high i/o data rates," *Computing Systems*, vol. 4, no. 4, pp. 405–436, Dec. 1991.

[8] C. Carter and R. Kravets, "User device cooperating to support resource aggregation," in *Proceedings of the 4th IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 2002)*, Callicoon, NY, June 2002, pp. 59–69.

[9] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson, "RAID: High-performance, reliable secondary storage," *ACM Computing Surveys*, vol. 26, no. 2, pp. 145–185, June 1994.

[10] R. L. Cruz, "A calculus for network delay, Part I: Network elements in isolation," *IEEE Trans. Inform. Theory*, vol. 37, no. 1, pp. 114–131, Jan. 1991.

[11] J. Duncanson, "Inverse multiplexing," *IEEE Commun. Mag.*, vol. 32, no. 4, pp. 32–41, Apr. 1994.

[12] P. H. Fredette, "The past, present, and future of inverse multiplexing," *IEEE Commun. Mag.*, vol. 32, no. 4, pp. 42–46, Apr. 1994.

[13] E. Guttman, C. Perkins, J. Veizades, and M. Day, "Service location protocol, version 2," Internet Engineering Task Force, RFC 2608, June 1999.

[14] S. Hanks, T. Li, D. Farinacci, and P. Traina, "Generic routing encapsulation over IPv4 networks," Internet Engineering Task Force, RFC 1702, Oct. 1994.

[15] H.-Y. Hsieh and R. Sivakumar, "pTCP: An end-to-end transport layer protocol for striped connections," in *Proceedings of the 10th IEEE International Conference on Network Protocols (ICNP 2002)*, Paris, France, Nov. 2002, pp. 24–33.

[16] ——, "A transport layer approach for achieving aggregate bandwidth on mutli-homed mobile hosts," in *Proceedings of the ACM International Conference on Mobile Computing and Networking (MobiCom 2002)*, Atlanta, GA, Sept. 2002, pp. 83–94.

[17] ISO/IEC moving picture experts group (MPEG). [Online]. Available: http://mpeg.telecomitalialab.com

[18] R. Kravets, C. Carter, and L. Magalhaes, "A cooperative approach to user mobility," *ACM SIGCOMM Computer Communication Review*, vol. 31, no. 5, pp. 57–69, Oct. 2001.

[19] C. M. Krishna and K. G. Shin, *Real-Time Systems*. McGraw Hill, 1997.

[20] J. W. S. Liu, K.-J. Lin, W. K. Shih, R. Bettati, and J. Chung, "Imprecise computations," *Proc. IEEE*, vol. 82, no. 1, pp. 1–12, Jan. 1991.

[21] L. Magalhaes and R. Kravets, "End-to-end inverse multiplexing for mobile hosts," *Journal of the Brazilian Computer Society*.

[22] ——, "MMTP: Multimedia multiplexing transport protocol," in *Proceedings of the first ACM Workshop on Data Communications in Latin America and the Caribbean (SIGCOMM-LA 2001)*, San Jose, Costa Rica, Apr. 2001, pp. 220–243.

[23] ——, "Transport level mechanisms for bandwidth aggregation on mobile hosts," in *Proceedings of the 9th IEEE International Conference on Network Protocols (ICNP 2001)*, Riverside, CA, Nov. 2001, pp. 165–171.

[24] S. McCanne and M. Vetterli, "Joint source/channel coding for multicast packet video," in *Proceedings of IEEE International Conference on Image Processing (ICIP'95)*, Washington, DC, Oct. 1995, pp. 25–28.

[25] National institute of standards and technology's NIST net. [Online]. Available: http://snad.ncsl.nist.gov/itg/nistnet

[26] Netfilter. [Online]. Available: http://www.netfilter.org

[27] NetIQ corporation's Chariot. [Online]. Available: http://www.netiq.com/products/chr/default.asp

[28] *ns-2*, the network simulator. [Online]. Available: http://www.isi.edu/nsnam/ns

[29] M. Papadopouli and H. Schulzrinne, "Connection sharing in an ad hoc wireless network among collaborative hosts," in *Proceedings of the 9th International Workshop on Network and Operating Systems Support Digital Audio and Video (NOSSDAV'99)*, Florham Park, NJ, June 1999, pp. 169–185.

[30] D. S. Phatak and T. Goff, "A novel mechanism for data streaming across multiple IP links for improving throughput and reliability in mobile environments," in *Proceedings of the 21st Annual IEEE International Conference on Computer Communications (INFOCOM 2002)*, New York, NY, June 2002, pp. 773–781.

[31] K. Salem and H. Garcia-Molina, "Disk striping," in *Proceeding of the IEEE 2nd International Conference on Data Engineering (ICDE'86)*, Los Angeles, CA, Feb. 1986, pp. 336–342.

[32] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A transport protocol for real-time applications," Internet Engineering Task Force, RFC 1889, Jan. 1996.

[33] K. Sklower, B. Lloyd, G. McGregor, D. Carr, and T. Coradetti, "The PPP multilink protocol (MP)," Internet Engineering Task Force, RFC 1990, Aug. 1996.

[34] A. C. Snoeren, "Adaptive inverse multiplexing for wide area wireless networks," in *Proceedings of the IEEE Conference on Global Communications (GLOBECOM'99)*, Rio de Janeiro, Brazil, Dec. 1999, pp. 1665–1672.

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

[36] M. Vishwanath and P. Chou, "An efficient algorithm for hierarchical compression of video," in *Proceedings of IEEE International Conference on Image Processing (ICIP'94)*, Austin, TX, Nov. 1994, pp. 275–279.

[37] Y. Wang, M. Orchard, V. Vaishampayan, and A. R. Reibman, "Multiple description coding using pairwise correlating transforms," *IEEE Trans. Image Processing*, vol. 10, no. 3, pp. 351–366, Mar. 2001.

[38] R. Woodburn and D. Mills, "A scheme for an internet encapsulation protocol: Version 1," Internet Engineering Task Force, RFC 1241, July 1991.

Appendix: Selecting a Channel Group to Maximize TCP Throughput

Suppose a proxy finds two or more sets of channels (i.e., channel groups) on which to assign packets from a newly-arriving flow. How should the proxy choose the preferred set of channels for the assignment?

We begin by assuming that our objective is to maximize the throughput of a bulk data transfer via TCP. We will assume that the available channels (and their underlying WAN link transmission characteristics) will persist over the lifetime of the transfer, and that each channel carries no additional traffic. Finding an analytical expression for the throughput of even a single, isolated TCP connection over an inverse multiplexed channel appears to be a formidable problem. Instead, we will select the channel group whose throughput *bound* is highest, optimistically assuming that such a channel group is most likely to realize the highest *actual* throughput.

In the simplest case, suppose that the proxy must choose between either of two individual links $L_i,\ i = 1, 2$ with packet loss probabilities p_i, round-trip times R_i, identical packet lengths B, and link-transmission rates $S_i = S$. The throughput T_i of a TCP connection using only link i is known to be bounded by

$$T_i \leq \frac{C_i B}{R_i \sqrt{p_i}}, \tag{1}$$

where C_i is a constant directly proportional to a link's congestion window size and whose value is ordinarily calculated to be ≈ 1.3. If the two links of identical transmission rate are taken to have identical window sizes, then the proxy would select link 1 over link 2 if

$$\frac{1}{R_1 \sqrt{p_1}} > \frac{1}{R_2 \sqrt{p_2}}. \tag{2}$$

What if the two links also had unequal transmission rates S_1 and S_2? If we assume that each link's window size is linearly proportional to its transmission rate, then the proxy would select link 1 if

$$\frac{S_1}{S_2} \frac{R_2}{R_1} > \sqrt{\frac{p_1}{p_2}}. \tag{3}$$

Note that this analysis extends immediately to the selection of any one link to be used from among a set of n links.

We next consider the case where a proxy must decide between two sets of links, where each set may contain more than one link. In such cases, the proxy would be performing inverse multiplexing of packets across the links in a set. We first recognize that the throughput of n parallel links $T(\vec{S}, \vec{p}, \vec{R})$ of identical transmission rate is bounded by the sum of the throughputs of each of the n links operating independently, or

$$T(\vec{S}, \vec{p}, \vec{R}) \leq \sum_{i=1}^{N} \frac{C_i B}{R_i \sqrt{p_i}}. \tag{4}$$

In general, however, the throughput of a link group will be smaller than the sum of each link's maximum throughput. This reduction is due in part to the increased round trip latency of each component link due to remultiplexing delays (as packets from different links wait to be recombined into a single stream) and longer packet transmission delays on the component links. It is possible to informally develop a reduced upper bound on throughput by calculating these additional delays and rewriting each link's round trip latency R_i in Eq. (4). Evaluating this equation then permits a proxy to compare multiplexed systems with varying numbers of component links to decide which system promises to maximize throughput.

As a simple example, suppose a proxy seeks to choose between a single link with transmission speed S with loss probability p_h and a multiplexed set of n links each with speed S/n. Relative to the single higher speed link, packets of length B bits traversing any one of the n links in the multiplexed system would suffer an addition $\frac{(n-1)B}{S}$ seconds of transmission delay, and an average $\frac{(n-1)}{2}\frac{B}{S}$ seconds of remultiplexing delay. If we assume that for each of the two systems the round trip latency R due to other factors (e.g., propagation delay, queuing delays) is otherwise identical, then a proxy would choose the single higher speed link if the RHS of Eq. (1) exceeded that of Eq. (4), or

$$\frac{1}{R\sqrt{p_h}} > \sum_{i=1}^{n} \frac{1}{(R + \frac{3(n-1)}{2}\frac{B}{S})\sqrt{p_i}}. \tag{5}$$