Bandwidth Aggregation for Real-Time Applications in Heterogeneous Wireless Networks

Kameswari Chebrolu and Ramesh R. Rao

Abstract—A variety of wireless interfaces are available for today's mobile user to access Internet content. When coverage areas of these different technologies overlap, a terminal equipped with multiple interfaces can use them simultaneously to improve the performance of its applications. In this paper, we motivate the advantages that can be had through simultaneous use of multiple interfaces and present a network layer architecture that enables diverse multiaccess services. In particular, we explore in depth one such service provided by the architecture: Bandwidth Aggregation (BAG) for real-time applications. An important aspect of the architecture when providing BAG services for real-time applications is the scheduling algorithm that partitions the traffic onto different interfaces such that the QoS requirements of the application are met. We propose one such algorithm Earliest Delivery Path First (EDPF), that ensures packets meet their playback deadlines by scheduling packets based on the estimated delivery time of the packets. We show through analysis that EDPF performs close to an idealized Aggregated Single Link (ASL) discipline, where the multiple interfaces are replaced by a single interface with same aggregated bandwidth. A prototype implementation and extensive simulations carried using video and delay traces show the performance improvement BAG with EDPF scheduling offers over using just the Highest Bandwidth Interface (HBI) and other scheduling approaches based on weighted round robin.

Index Terms—Network architecture and design, video, scheduling, algorithm/protocol design and analysis, implementation, simulation.

1 Introduction

The explosive growth of the Internet has been a major driving force in the proliferation of a variety of wireless technologies. Examples include 802.11, Bluetooth, GPRS, CDMA2000, UMTS, etc. Several research challenges [1], [2], [3], [4] related to the use of a single wireless technology at the mobile client have been explored so far. With the incidence of a variety of wireless technologies, seamless migration of connections [5] (vertical handoff) from one interface to another, content adaptation [6] to suit the characteristics of the interface have also been addressed. However, the basis of most of the research in this domain has been confined to single interface use at any given time to meet all the connectivity requirements of the applications.

Existing wireless technologies differ widely in terms of services Offered—bandwidth, coverage, QoS support, pricing, etc. Restricting usage to one single interface at a time limits the flexibility available to the end user in making the best use of all available resources on his interfaces. The use of multiple interfaces simultaneously opens new way of addressing some of the limitations of wireless media and can enable other new and interesting possibilities:

- Bandwidth Aggregation. Bandwidth offered by the multiple interfaces can be aggregated to improve quality or support demanding applications that need high bandwidth.
- Mobility Support. The delay associated with handoff can be significantly reduced when an alternate communication path is always kept alive.
- Reliability. For applications requiring strict reliability guarantees, some or all packets can be duplicated/encoded and sent on the multiple paths.
- Resource Sharing. While the above scenarios involve a single client host, the idea can be extended to broader scenarios. For instance, in an ad hoc network of nodes connected via their local interfaces (LAN—802.11 or Bluetooth), a subset of nodes may have wide area (WAN) connections. These WAN bandwidth resources can be shared effectively across the nodes to access Internet.
- Data-Control Plane Separation. Similarly, the WAN interfaces in an ad hoc/sensor network can also be used for *out of band* control communication (via an infrastructure proxy) to aid distributed ad hoc protocols such as routing. The LAN interface can thus mostly be used for data, thereby achieving a clean separation between control and data planes.

We term the services enabled by such simultaneous use of multiple interfaces as *MultiAccess Services*. To realize, in practice, the services listed above, we need an architecture to support multiple communication paths. In this paper, we begin by providing a general framework in the form of such an architecture. In particular, we focus our attention on one of the services provided by the architecture: **B**andwidth **Ag**gregation (BAG) for real-time applications.

The architecture can be addressed at different layers of the protocol stack. We choose a network layer approach as opposed to transport/application layer solution to introduce minimal changes in the existing infrastructure thus

• *The authors are with the Department of Electrical and Computer Engineering, University of California, San Diego, 9500 Gilman Drive, La Jolla, CA 92093-0436. E-mail: chebrolu@cwc.ucsd.edu, rrao@ucsd.edu.*

Manuscript received 10 Apr. 2004; revised 3 Oct. 2004; accepted 29 Nov. 2004; published online 15 Feb. 2006.
For information on obtaining reprints of this article, please send e-mail to: tmc@computer.org, and reference IEEECS Log Number TMC-0131-0404.

providing application transparency. Our network layer architecture consists of an infrastructure proxy. A proxy may provide services to a set of mobile clients equipped with multiple interfaces, and multiple proxies may be provisioned for reliability and scalability. Some of the features of the network proxy are similar in spirit to that provided by Mobile IP [7]. The client uses a fixed IP address acquired from the proxy in establishing connections with the remote host. The proxy captures the packets destined for the client and uses IP-within-IP encapsulation to tunnel them to the client. However, unlike Mobile IP, the proxy can manage multiple care-of-addresses and perform intelligent processing and scheduling of packets.

One of the services provided by the architecture is that of aggregating bandwidth available on multiple interfaces to increase application throughput. We explore in depth this particular service in the context of real-time applications. While the use of multiple interfaces can increase one's bandwidth, the use of multiple paths, each with varying characteristics introduces new problems in the form of excess delay due to potential packet reordering. Streaming applications that employ smoothing buffers can tolerate this reordering to an extent. However, for interactive applications, if care were not taken to minimize the delay resulting from reordering, such delay is often equivalent to a packet loss. In the context of our architecture, we look at this issue in the form of the scheduling algorithm at the network proxy (or mobile client in the uplink direction) that partitions the data stream onto the multiple paths corresponding to the different network interfaces. We propose the Earliest Delivery Path First (EDPF) algorithm that has the explicit objective of reducing delay due to reordering. It estimates the delivery time of the packets on each Internet path (corresponding to each interface), and schedules each packet on the path that delivers it the earliest. This approach effectively minimizes reordering and thereby the delay and jitter experienced by the application.

To understand the behavior of EDPF, we perform both *analysis* and *simulation/implementation*. The ideal scheduling algorithm would aggregate bandwidth such that the performance is similar to the case where a single link with the same aggregate bandwidth is used—we call this the Aggregated Single Link (ASL) algorithm. We analyze the performance difference between EDPF and the idealized ASL algorithm in terms of several metrics: the number of bits serviced, delay experienced by the packets, the jitter under buffering, and the maximum buffer requirement for in-order delivery. In addition to the analysis, we study the performance of EDPF through *a prototype implementation* and *trace-based simulations* for both real-time streaming and interactive applications. Our results show that EDPF mimics ASL closely and outperforms round-robin-based approaches [8] by a large margin.

While we have introduced BAG in the context of wireless interfaces, wired (e.g., dialup) links can also be included in bandwidth aggregation. Further, the scheduling algorithm EDPF can be used to provide QoS in many systems that use multiple paths. Examples of such systems include high-end storage (host connected to RAID server via multiple channels), Ethernet/ppp link aggregator systems [9], [10].

The rest of the paper is organized as follows: In Section 2, we describe our architecture. The scheduling algorithm EDPF along with several properties is presented in Section 3. Section 4 presents the results of our experiments using a prototype implementation and trace driven simulations. We discuss the deployment complexity and some of the assumptions we make in our work in Section 5. We present related work in Section 6 and, finally, conclude in Section 7.

2 Architecture and Services

In this section, we first motivate our choice of a network layer architecture that enables multiaccess services and then proceed to discuss the functional components that make up our architecture. We also elaborate on one of the services provided by the architecture—BAG, which is the focus point of this paper. Additional details of the architecture can be found in [11].

2.1 Why a Network Layer Architecture?

The architecture can potentially be addressed at different layers of the protocol stack. Link layer solutions are infeasible in this setup, as the networks span different domains, controlled by different service providers. An application-level solution is a possible design alternative. Making applications aware of the presence of multiple interfaces can lead to application specific optimization and can be very efficient. However, given the diversity of applications, this approach would mean modifying/rewriting the various applications while ensuring compatibility with existing infrastructure, making wide spread deployment a difficult job. Further, the applications need to keep track of the state of different interfaces, which increases their complexity. And, when multiple applications share common client resources (interfaces), they have to be designed with care such that they are friendly to each other.

Transport layer solutions (e.g., for use with TCP-based applications) share some of the same features as application layer solutions. While they can be efficient, they still need all server software to be upgraded to use the new transport protocols and careful design to ensure fair access to client resources.

With IP as a unifying standard, a network layer proxy based approach has the advantages of being transparent to applications and transport protocols and does not need any changes to existing server software. Our choice of a network layer solution mainly stems from its ease of deployment. Legacy applications, in particular, can benefit with this approach as they have no other design alternative. Another advantage with a network layer setting is a centralized approach to end user flow management that can potentially prevent any unfair interaction.

While the network layer approach overcomes most limitations of the other approaches, it may not always be very efficient as it operates further down the stack. However, we believe that with careful design, most inefficiencies can be minimized. Our design choice, though an alternate to application/transport layer solutions, as such, does not preclude further optimization at the higher layers. In the sense that our architecture can complement these approaches in terms of mobility support when handling multiple interfaces. In the absence of this solution, higher layer approaches may have to handle mobility themselves or rely on multiple Mobile IP initiations (one for each interface, which to our knowledge is not supported by Mobile IP).

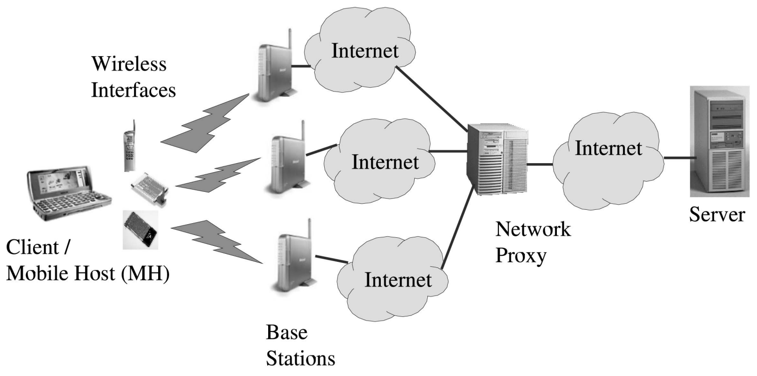

Fig. 1. Architecture to support multiple communication paths.

We now proceed to discuss the main details of our architecture.

2.2 Architecture

Fig. 1 shows a high-level overview of the architecture. The network proxy provides many different services (Bandwidth Aggregation, mobility support, resource sharing, etc.) to the client (equivalently, the MH), which is connected to the Internet via multiple network interfaces. The MH, when using the services of the network proxy, acquires a fixed IP address from it and uses it to communicate directly with the remote server. The MH also registers the care-of IP addresses of its multiple active interfaces with the proxy. When the application traffic of the MH passes through the domain of the proxy (since the source address used by the MH in the packets corresponds to an address in the proxy network), the proxy intercepts the packets and performs necessary application specific processing. It then tunnels them using IP-within-IP encapsulation to the client's different interfaces. This mechanism is similar to that used in Mobile IP, but has been extended to handle multiple interfaces. Note that this mechanism is needed in our architecture not just for mobility support, but for simultaneous use of interfaces—it is essential even when the client is stationary.

Many Radio Access Networks (RANs) often use private IP address space (NAT functionality), and/or discard packets if the packets contain a source IP address different from those configured for use within the access network (ingress filtering). The mechanism used in our architecture to overcome these problems is the same as that used in Mobile IP. To overcome ingress filtering, we employ reverse tunneling, i.e., outgoing packets from the MH are now tunneled to the network proxy. To overcome NAT usage in access networks, we perform tunneling on top of UDP.

The functional components that make up our architecture, which reside on the MH and on the network are as shown in Fig. 2. We explain briefly the functionality of each component. Additional details can be found in [11].

- *Profile Manager/Server*. The Profile Manager and Profile Server have the main role of handling application requirements and conveying the same to the necessary modules. For each application the MH starts, the Profile Manager generates a profile based on user input and application needs. The

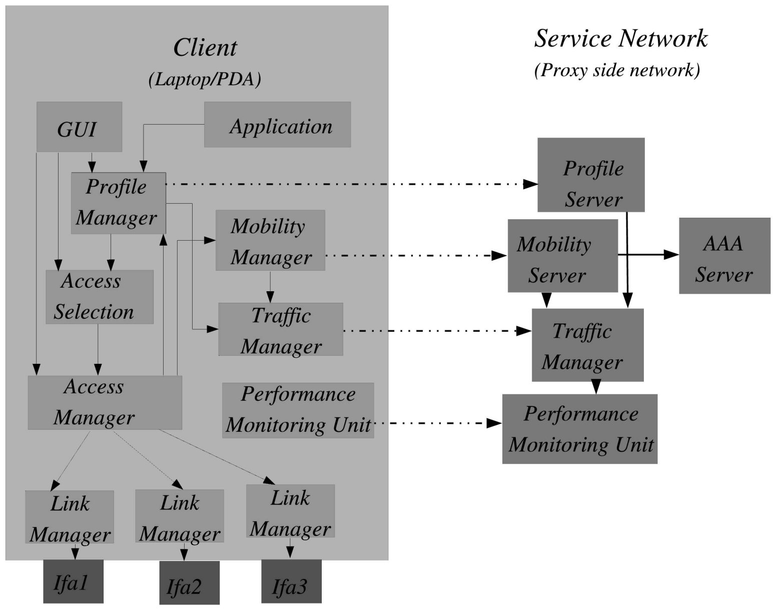

Fig. 2. Functional components of the architecture.

profile includes the interfaces to use, the type of scheduling to perform across the chosen interfaces (which algorithm), granularity of scheduling (per packet or per flow), and any additional functionality needed (duplication of packets for reliability, content adaptation, etc.). The Profile Manager conveys this profile information to the Profile Server to facilitate it in handling the application traffic that passes through the proxy.

- *Access Manager and Access Selection Unit*. Based on the profile generated, the task of the Access Selection unit is to bring up the necessary interfaces in conjunction with Access Manager. The Access Manager manages its tasks by talking with one or more Link Managers, which handle individual interfaces.
- *Mobility Manager/Server*. The newly acquired care-of IP addresses are registered by the Mobility Manager on the client side with Mobility Server on the proxy network. The mobility units also handle client mobility by tearing down old tunnels and establishing new ones with the newly acquired care-of addresses.
- *Traffic Manager*. The Traffic Manager is the component which constitutes the core of multiaccess services. This unit resides both on the client and the proxy network and hosts the various scheduling algorithms needed to provide different services. The Profile Manager/Server informs the Traffic Manager on the exact handling of a client flow. Typically, each data packet flows through the traffic manager. For each packet, the Traffic Manger determines its flow-id, accesses the correct profile and performs appropriate processing.
- *Performance Monitoring Unit*. The component which provides performance input for the Traffic Manager is the Performance Monitoring Unit. The Performance Monitoring Units residing on both the client and the proxy network, through collaboration monitor the characteristics of the underlying paths between the network proxy and the client.

2.2.1 BAG Services

One of the services provided by the architecture toward increasing application throughput is that of Bandwidth Aggregation (BAG). Consider a user equipped with two interfaces, each of which provides on average 100kbps and 50kbps bandwidth. By simultaneous use of both interfaces, the user can increase his total bandwidth to 150kbps. Bandwidth Aggregation, attempts to increase user bandwidth by striping data onto the multiple interfaces so as to avail all available bandwidth.

While we have come a long way in terms of peak data rates in mobile networks, 9.6kbps (GSM-TDMA) in 2G to 2Mbps (UMTS) in 3G, the typical rates one can expect to see in a loaded network are still very small [12]—40kbps in 1xRTT, 80kbps in EDGE, 250kbps in UMTS. Supporting real-time applications with stringent QoS requirements, large file transfers, intense Web sessions is a difficult task and may not even be possible if confined to a single interface. Using bandwidth available from all possible sources may be the only option to increase one's bandwidth and support demanding applications. In this paper, we focus our attention on two such demanding applications—real-time streaming and interactive video. In concurrent work [13], we have considered BAG services for TCP applications. In the context of the overall architecture, a crucial aspect that dictates real-time video performance is the scheduling algorithm that resides in Traffic Manager which splits traffic across the different paths. We now turn to a discussion of the design of this algorithm.

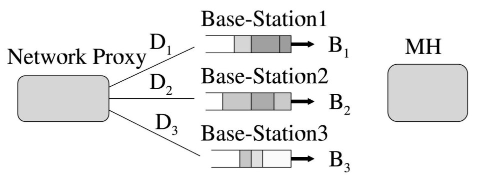

Fig. 3. A simplified view of the network between Proxy and MH.

3 The Scheduling Algorithm

For real-time applications, the scheduling algorithm not only has to effectively aggregate bandwidth of the interfaces, but also minimize delay experienced by packets due to potential reordering caused by varying characteristics (delay, bandwidth, loss) of the multiple paths. We first present a scheduling algorithm under ideal conditions, that achieves our desired objectives (Section 3.1), along with some useful properties (Section 3.2). In subsequent sections, we explain how the algorithm fits in practical scenarios.

3.1 The Earliest Delivery Path First (EDPF) Scheduling Algorithm

The overall idea behind EDPF is to 1) take into consideration the overall path characteristics between the proxy and the MH—delay, as well as the wireless bandwidth, and 2) schedule packets on the path which will deliver the packet at the earliest to the MH. In explicit terms, EDPF can be described as follows;

The network between the proxy and the MH can be simplified as shown in Fig. 3. Each path l (between the proxy and the MH) can be associated with three quantities: 1) D_l, the one-way wireline delay associated with the path (between the proxy and Base Station—BS), 2) B_l, the bandwidth negotiated at the BS,[1] and 3) a variable A_l, which is the time the wireless channel becomes available for the next transmission at the BS. If we denote by a_i, the arrival instance of the ith packet (at the proxy) and by L_i, the size of the packet, this packet when scheduled on path l would arrive at the MH at d_i^l.

$$d_i^l = MAX(a_i + D_l, A_l) + L_i/B_l. \tag{1}$$

The first component computes the time at which transmission can begin at the BS, and the second component

1. The client negotiates certain bandwidth from the access network at the beginning of connection, which the access network guarantees for the duration of connection. Real-time applications cannot be supported without such QoS guarantees.

computes the packet transmission time (we ignore the wireless propagation delay). EDPF schedules the packet on the path p, where

$$p = \{l : d_i^l \leq d_i^m, 1 \leq m \leq N\},$$

N being the number of interfaces. That is, the path with the earliest delivery time. EDPF then updates A_p to d_i^p, i.e., the next transmission can begin only at the end of the current packet reception. EDPF tracks the queues at each of the base-stations through the A_l variable. By tracking the queues at the base-stations and taking it into account while scheduling packets, EDPF ensures that it uses all the available path bandwidths, while achieving minimal packet reordering. The explanation so far focused only on downlink transmission where the MH acts as a sink. The same algorithm can also be used for the uplink case where the roles of MH and proxy are interchanged.

3.2 Properties of EDPF

We now analyze some of the properties of EDPF. Our goal is to bound the performance behavior of EDPF, as well as to compare it with the idealized ASL case. When the arrival rate of traffic is less than the total aggregated bandwidth, there is enough spare bandwidth available to mask the inefficiency of scheduling. It is only when the BSs have queue build-up (which is when delays are higher), scheduling becomes more important. In such a scenario, we can simplify the analysis by noting that we expect the A_l factor to dominate in (1). Accordingly, we assume that the wireline delay D_l experienced by the packets is 0.[2] This helps us to highlight the relevant properties of EDPF without introducing additional complications. Further, this makes comparison against ASL more meaningful; with asymmetric delay across the different paths, it is not straightforward to determine the wireline delay for the ASL case to make a meaningful comparison.

In the analysis below, we carry over the notations N, B_l, A_l, a_i, and L_i from above. In addition, we use the following notation: We define the links corresponding to the highest and lowest bandwidth as $hb = argmax_l\{B_l\}$ and $lb = argmin_l\{B_l\}$, respectively. We define $B_{max} = B_{hb}$ and $B_{min} = B_{lb}$. Each link l has a weight, $w_l = B_l/B_{min}$. We let L_{max} be the maximum packet size.

Let $T_l(t) = max\{t, A_l\}$. $T_l(t)$ is in essence the time at which a packet arriving at time t can begin transmission on link l. Note that when packet i is scheduled on link l, if d_i is its delivery time at the client, $T_l(a_i^+) = d_i$, where a_i^+ refers to the time instant just after a_i (arrival time of packet i at proxy). When buffering is used with EDPF, we distinguish between the delivery time to the client (d_i), and the receive time at the application, denoted r_i. Thus, $r_i \geq d_i$. We set the initial value of $A_l = 0$, and let the first packet arrive at time 0 ($a_1 = 0$).

We first present a useful lemma that is used to derive some of the properties of EDPF.

Lemma 1. *At any time t, if $T_n(t) \leq T_m(t)$, then $T_m(t) - T_n(t) \leq L_{max}/B_n$.*

2. The analysis applies even when this delay is finite but identical across interfaces.

Proof. We prove the above lemma by induction on the packet number i as follows: We will show that in the interval $[0, a_2]$, the lemma holds. Assuming that it holds in $[0, a_i]$, we will then show that it holds in the interval $(a_i, a_{i+1}]$. (Recall that $a_1 = 0$.)

Basis. The first packet is scheduled on the link with the highest bandwidth, i.e., hb, to deliver it the earliest. A_{hb} would now take on the value L_1/B_{max} and $A_{m \neq hb} = 0$. Consequently,

$$T_{hb}(0^+) - T_m(0^+) = L_1/B_{max} \leq L_{max}/B_m.$$

The lemma holds at time 0^+. For any $0 < t \leq a_2$ since $T_m(t) = max\{t, A_m\}$, the difference between T_ms decreases linearly with t.

Inductive step. Assume that the lemma holds for packets $1, 2, \ldots, i-1$, i.e., it holds in the interval $[0, a_i]$. Let l be the link chosen for transmission of packet i. Then, according to EDPF,

$$d_i = T_l(a_i) + L_i/B_l \leq T_m(a_i) + L_i/B_m, 1 \leq m \leq N.$$

At time a_i^+, $T_l(a_i^+)$ takes on the value of d_i and the other Ts do not change. Hence, we have:

$$T_l(a_i^+) \leq T_m(a_i^+) + L_i/B_m. \tag{2}$$

We now consider the following two cases:

Case 1. $T_l(a_i^+) > T_m(a_i^+)$. According to (2), $T_l(a_i^+) - T_m(a_i^+) \leq L_i/B_m$.

Case 2. $T_l(a_i^+) \leq T_m(a_i^+)$. Since the lemma holds at time a_i, we have $T_m(a_i) - T_l(a_i) \leq L_{max}/B_l$. Since $T_l(a_i) < T_l(a_i^+) \leq T_m(a_i^+) = T_m(a_i)$, from above inequality we get, $T_m(a_i^+) - T_l(a_i^+) \leq L_{max}/B_l$.

Thus, the lemma holds at time a_i^+ in both cases. As in the basis, at any time ($a_i < t \leq a_{i+1}$), the difference between Ts decreased linearly with t and, hence, the lemma follows. □

When packets are of constant size, it is easy to see that with EDPF, they will arrive in order at the client. Consider two packets $\{i, j : j > i\}$. Packet j may arrive before i only if it were scheduled on a different link. If packet sizes are the same and the link on which j was transmitted delivers packets the earliest, EDPF when scheduling i would have picked that link for its transmission. Thus, packets will always arrive in order. Note that this property does not hold for other scheduling schemes based on Weighted Round Robin (WRR) or variants of it such as Surplus Round Robin (SRR) [8], Longest Queue First.

When packets are of variable size, it is important that the scheduling algorithm distribute the bits across the links properly. Given P packets of variable size for transmission, we can say the algorithm achieves good bandwidth aggregation if the maximum difference between the normalized bits allocated to any two pairs of links m, n is at most a constant. The constant should not be a function of P. The following theorem upper-bounds this constant by L_{max} for EDPF. In case of WRR, this quantity is a function of P and can grow without bound. To understand why, consider the case of two links with equal weights, where packet sizes alternate between maximum and minimum size. For SRR it is $2L_{max}$ (proof not presented).

Theorem 1. *For EDPF, given P packets to transmit, the maximum difference between the normalized bits allocated to any two pairs of links m, n is upper bounded by L_{max}.*

$$max_{m,n}\left|\frac{Sent_m}{w_m} - \frac{Sent_n}{w_n}\right| \leq L_{max}.$$

Proof. Let t be the time instance at which one of the links first becomes idle, i.e., at t the particular link in question finishes serving its share of the load P. For any link l, $T_l(t)$ would essentially indicate the overall time for which the link was used for transmission. Therefore, $T_l(t) * B_l$ would be the total number of bits sent on the link—$Sent_l$. For any two links m, n,

$$\left|\frac{Sent_m}{w_m} - \frac{Sent_n}{w_n}\right| = \left|\frac{T_m(t) * B_m}{w_m} - \frac{T_n(t) * B_n}{w_n}\right|.$$

Since $B_l/w_l = B_{min}$ and since the difference between the Ts cannot exceed L_{max}/B_{min} from Lemma 1, the right-hand side is at most L_{max}. This proves the theorem. □

The behavior of a system with multiple links differs from its single link counterpart ASL on several grounds. First, packets no longer arrive in order due to multiple paths. Second, work can accumulate as packets may be serviced at a rate less than in ASL. This accumulation can result in packets experiencing excess delay on average. The low service rate also increases the jitter experienced by the packets. In the rest of this section, we compare EDPF with ASL by providing upper-bounds on the above mentioned differences—work, delay, jitter, and buffering required. For better readability, we just state the theorems here and discuss the results at the end of the section. The interested reader can find the proofs in Appendix A.1.

Theorem 2. *For any time t, the difference between the total number of bits W serviced by ASL and EDPF is upper bounded as*

$$W_{ASL}(0,t) - W_{EDPF}(0,t) \leq L_{max}\left(\sum_{l=1}^{N} w_l - 1\right).$$

Proof. See Appendix A.1. □

Theorem 3. *The difference in delay experienced by a packet i in ASL and EDPF is upper bounded as*

$$d_i^{EDPF} - d_i^{ASL} \leq \frac{L_{max}(\sum_{l=1}^{N} w_l - 1)}{\sum_{l=1}^{N} B_l} + \frac{(N-1)L_i}{\sum_{l=1}^{N} B_l}.$$

Proof. See Appendix A.1. □

Jitter is defined as the difference in delay experienced by two consecutive packets, i.e., $J_i = (r_i - r_{i-1}) - (a_i - a_{i-1})$. It is easy to see that if the packets are not buffered ($r_i = d_i$), $J_i \leq L_i/B_{min}$. The worst case jitter happens when both the packets are transmitted on the link corresponding to lb.

Theorem 4. *When buffering is employed, the jitter experienced by a packet i is upper bounded by L_i/B_{max}.*

Proof. See Appendix A.1. □

Theorem 5. *The buffer size needed (at the client) to deliver the packets in order (to the application) is at most $(N-1)L_{max}$.*

Proof. See Appendix A.1. □

3.3 Discussion

An important property a scheduling algorithm should have is that it utilize the bandwidths of the links properly. EDPF ensures that this difference in normalized bits allocated to any two links is a small constant L_{max} (Theorem 1). Further, Theorem 2 shows that the work carried over in EDPF in comparison to ASL is again a constant independent of time. Another property the scheduling algorithm should have is that it minimize reordering and, thus, the delay and jitter experienced by the packets. Here, too, EDPF performs close to ASL. The difference in delay experienced by the packets, between EDPF and ASL, is bounded (Theorem 3). The bound is proportional to the bandwidth asymmetry as well as the number of interfaces. The jitter is bounded by a small constant if buffering is used (Theorem 4), and the amount of buffering required to achieve this is only linear in the number of interfaces, and independent of other factors.

Though looked at in the context of bandwidth aggregation, EDPF can also be used in *Queuing* disciplines to provide QoS. What we have analyzed is the performance of a "single queue—multiple server system" based on EDPF scheduling. We have compared such a system with one that employs a single server but which serves the queue at a rate equal to the sum of the rates of the multiple servers.

4 Experimental Evaluation

In this section, we evaluate the performance of the scheduling algorithm in the context of two important class of real-time applications: streaming and interactive video. We experiment with streaming applications on a prototype implementation of our architecture as a proof of concept for BAG services and to quantify the performance improvement BAG services bring over conventional single interface use. With respect to interactive applications, we consider an appropriate simulation setup in line with efforts in the next-generation networks to support QoS and show the performance advantages EDPF scheduling has over weighted round-robin based approaches.

4.1 Streaming Video Applications

Streaming video involves the transfer of data as a continuous steady stream that allows a client to display video before the entire file has been downloaded. The client normally employs a smoothing buffer to hide the variability in the bit rate of the data stream and to present a good quality video with fewer halts. We show that BAG can help streaming applications by significantly reducing the buffering time needed to ensure continuous playback, thereby enhancing end-user experience.

TABLE 1
Buffering Time (in Seconds) for Continuous Playback

Algo./ Video	Lecture $\langle 58, 690\rangle$	Star Trek $\langle 69, 1200\rangle$	Star Wars $\langle 53, 940\rangle$	Susi & Strolch $\langle 79, 1300\rangle$
EDPF	2.3	3.1	2.9	4.6
HBI	7.9	8	8.3	8.6

4.1.1 Implementation Details

We implemented a prototype of the setup as depicted in Fig. 1 for streaming video. The video server is trace-driven—it uses frame size traces of several video sequences taken from [14]. It reads generation-time/size information from the trace file, generates appropriate sized packets and streams them to the client using a UDP socket. The duration of the video sequences used in this experiment is 30 minutes.

The client machine (MH) connects to the Internet using multiple interfaces. It binds the multiple care-of addresses to a virtual IP address (that of the proxy) and uses the virtual address to talk to the video server (via proxy if interfaces are NAT enabled). We used two 1xRTT cards (CDMA2000) in our experiments. Ideally, we would have liked to use two separate technologies, but other available interfaces were not very conducive. HDR-based 1xEVDO had no Linux drivers and GPRS was unstable (while shorter runs showed good performance improvement, in longer runs, the delay experienced by some packets were in excess of 20 seconds possibly due to a bug in the implementation). The purpose of this experiment is to demonstrate proof of concept of BAG—we believe that similar performance as shown in this paper can be achieved with other stable interfaces.

The functional components that make up our architecture (Fig. 2) have been implemented as Linux loadable kernel modules. The Traffic Manager (TM) is the main components relevant to this experiment. So we elaborate more on this component. For ease of implementation, we integrated some parts of the Performance Monitoring Unit with the TM. The TM resides in kernel space and intercepts all incoming packets before the routing module. At the proxy, the TM encapsulates the captured packets with a header whose destination IP address is determined by the EDPF algorithm implemented within. At the MH, it removes the outer IP header and collects interface statistics. After the appropriate processing, the TM passes control of the packet to the routing module to be handled as usual. The MH's TM module also communicates the parameters needed by EDPF (D_l and B_l) to the proxy using UDP. We use the average values of delays and throughput observed on the interfaces as values for these parameters. Note that reordering is not much of an issue in streaming applications, given the buffering of packets. So, EDPF does not really need an accurate estimation of these parameters.

4.1.2 Metrics of Evaluation

The client application at the MH, buffers incoming packets and begins video display after a fixed delay which we term *Startup Latency*, and denote by L. Once the display begins, the application displays frames consecutively every t seconds (frame period). If at one of these epochs, the client's buffer does not have the complete frame, the frame is considered lost (we discard its dependent frames as well). At the next epoch, the client will attempt to display the next frame.

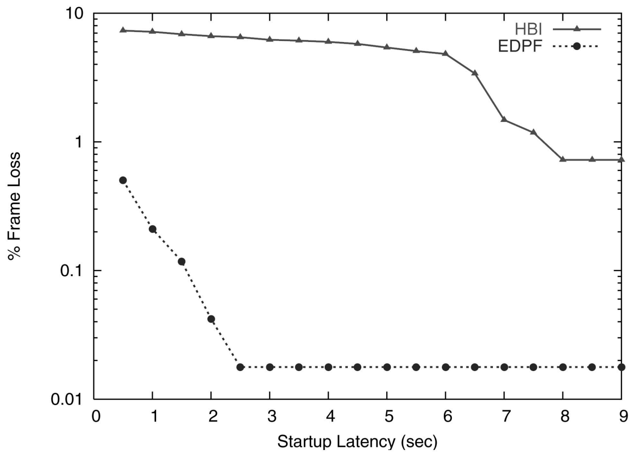

Fig. 4. Lecture video: Percent frame loss versus startup latency.

We use two metrics for comparison: 1) The buffering time (BT) needed to ensure continuous playback of received frames. In other words, with $L = BT$, no received frame misses its playback deadline. And, 2) the Frame Loss ratio (FL) for a given Startup Latency. This ratio includes frames lost en route as well as frames lost due to late arrivals.

4.1.3 Experimental Results

Table 1 shows the first metric—the buffering time needed (in sec) to ensure continuous playback of received frames for various video sequences. The mean and peak bit rates in kbps of the video sequence are also shown. We compare BAG/EDPF with the use of just a single interface—the Highest Bandwidth Interface (HBI). As can be seen, BAG with EDPF achieves a much lower startup latency than HBI. BAG achieves twice the bandwidth of HBI in this experiment (two similar interfaces), and the performance improvement in terms of BT is more than proportionate—in most cases it is over a factor of two lesser.

The variation of FL with L for the "Lecture" video is as shown in Fig. 4. At $L = 0.5sec$, EDPF has a FL of 0.5 percent, while HBI has 7.3 percent. At $L = 2sec$, EDPF achieves FL of 0.04 percent while HBI still suffers a high 6.6 percent frame loss. Streaming applications that support VCR functions require one way delays in the range of 1-2 seconds. If less than 1 percent frame loss is required, BAG can support this, while using just one interface cannot.

Another interesting result we observed is that the packets discarded en-route was much higher for HBI, than in EDPF for all the runs. For example, eight packets were discarded for EDPF as compared to 326 packets fro HBI. We believe this to be caused due to buffer overflow at the wireless base-station. When using multiple interfaces, the load gets uniformly distributed resulting in lesser losses —another advantage of simultaneous interface use.

4.2 Interactive Video

In the previous section, we have demonstrated on an experimental testbed the benefits of BAG services for streaming video applications. We now consider real-time interactive multimedia.

Interactive applications like video telephony, video conferencing have very stringent delay requirements—they need one way latency under 150 ms for excellent quality of service and under 400 ms for acceptable quality. Present mobile systems (GPRS, CDMA2000, HDR) as they stand today are best effort based with one way delays in the range of a few hundred ms to excess of 1 second. It is, in general, very difficult to support interactive applications on systems that provide no QoS guarantees. Efforts are now underway to integrate QoS support in both the core backbone as well as radio access segment of the next-generation systems. In line with efforts in this direction, we consider an appropriate simulation setup and study the performance of interactive video when using BAG services. We now describe the experimental methodology and present experimental results subsequently.

4.2.1 Experimental Methodology

The network topology shown in Fig. 1 captures the vision of next generation networks where the Base Station (BS) is an extension of IP-based Internet. We implement/simulate each of the components that make up the topology. We assume that the radio access network provides QoS support and that the wireless hop is the bottleneck link.

The Server. As in the previous section, we simulate video server behavior using frame size traces. We consider a high quality MPEG4 "Office Cam" [14] video, which captures the activity of a person in front of a terminal. The mean and peak bit rates of this video are 400kbps and 2Mbps, respectively. The reason for choosing this video is 1) Interactive video applications like video telephony/ conference will be similar in nature. 2) The bandwidth it needs compares to that we can obtain by aggregation in next-generation Radio Access Networks (RANs).

The Internet Paths. In the next generation networks, the BS is considered to be an extension of the Internet. Accordingly, we used delay traces collected on different Internet Paths to simulate the delay experienced by the packets up to the BS. The mean value of this delay between server and proxy is 15 ms and between proxy and BSs is 22 ms (the same trace file was used on all the paths between proxy and BSs). The traces were collected by generating packets of appropriate size (derived from the frame size trace) and measuring the round trip time (RTT) on paths between hosts located at the following universities: UCSD, UCB, CMU, and Duke. Note that this 37 ms average delay corresponds to just the delay experienced by the packet on the wireline segment. The total one way delay experienced by the packet often is much higher, given the backlog at the BS (queuing delay) and wireless transmission and propagation delay. And, it is this total delay that has to be under a reasonable value (say, 250 ms) for achieving good quality video.

Base-Stations and the Wireless Channels. Since we assume that the underlying network provides QoS, the BSs are simulated to have a link capacity equal to negotiated rate and no cross traffic. They serve the packets in their queue on a first-come-first-served basis. This is a reasonable assumption because, in systems that provide QoS, once QoS (bandwidth/loss) is negotiated, the channel is retained for the whole session (no release/grant happens). Fluctuating channel conditions and resulting losses are overcome by FEC, limited ARQ, and increasing power of transmission (to maintain loss rate below the negotiated value). In appropriate experiments, we also simulate channel losses—the base-stations introduce errors in the packets and may retransmit the packet based on the retransmission policy in place.

The Network Proxy. The proxy implements two types of scheduling Algorithms—EDPF and Surplus Round Robin (SRR) (for comparison purposes). Surplus Round Robin (SRR) was proposed in [8] as a generic bandwidth aggregation algorithm, it is similar to WRR but adjusted to account for variable sized packets, where the surplus (unused bandwidth) is carried on to the next round. SRR needs the negotiated bandwidth B_l of the interfaces in its calculations. EDPF in addition to B_l also needs wireline delay D_l. In the simulations, we use the average value of the Internet path delay traces for EDPF calculations. In practice, D_l can be estimated by sending signaling packets to the MH during connection setup (clock synchronization is not required since only the relative delay between the different paths matters). This in general suffices because Internet path delays are known to vary only slowly, over several tens of minutes [15].

The Client. The packets arriving at the client are placed in a buffer to overcome any reordering and passed in order to the video application.

Application Performance Metrics. To measure the quality of the video reception, we use the following performance metrics.

1. The backlog in the system between the HA and the client application.
2. The one-way delay experienced by the packets between the server and the client application.
3. F_{loss}—the fraction of frames that were discarded because packets that make up the frame experience delay in excess of maximum delay bound (DB_{max}, a configurable parameter) or were lost en route. Note that when a frame is discarded, we also discard its dependent frames (P/B frames are discarded when the corresponding I frame is lost). This metric mainly captures the effect excessively delayed packets have on the overall quality of the video.
4. Glitch duration (G_d) and Glitch Rate (g). We define G_d as the number of consecutive frames that were discarded. We define g as the number of glitches that occur per ms.

4.2.2 Experimental Results

We first address the issue of how much bandwidth to allocate to support QoS requirements of the application. We then fix the bandwidth at a suitable value and evaluate the performance using a set of metrics. Later, we measure the sensitivity of the scheduling algorithms to bandwidth/delay

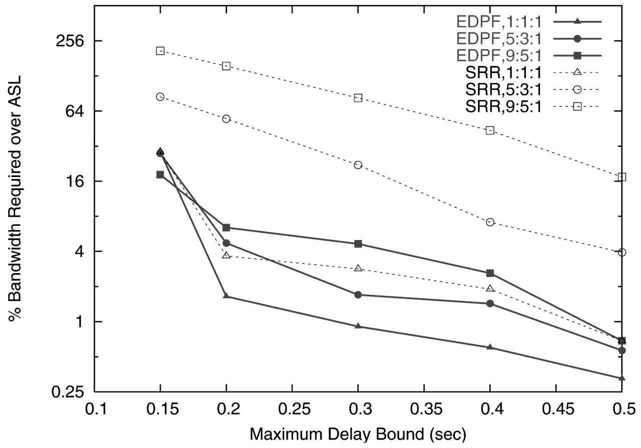

Fig. 5. Bandwidth needed over ASL for 0 percent F_{loss}.

asymmetry, number of interfaces, bandwidth/delay variation, and channel losses.

Bandwidth Allocation. To enable continuous video playback, appropriate bandwidth must be allocated to the video stream. Allocating just the average rate for Variable Bit Rate encodings would not in general satisfy the maximum delay requirements of the video. Peak allocation on the other hand result in very low bandwidth utilization.

Given a packet stream of packets, we would like to determine bandwidth B such that the delay experienced by the packets is bounded by DB_{max}. We are also interested in finding the buffer capacity C that is needed to ensure that there is no overflow at the MH.

When the wireline delay D and the bandwidth *split* (ratio in which the bandwidth is allocated to the various interfaces) are fixed, the bandwidth B that bounds the maximum delay by DB_{max} can be obtained by doing a binary search. Note that, in practice, the bandwidth split cannot be known in advance at the client. Without knowledge of total bandwidth, the client would not know how much bandwidth to negotiate on each interface. However, the server can help the client in the negotiation by providing a range of values corresponding to different splits.

When sufficient bandwidth has been allocated to achieve the delay objective as mentioned above, the maximum buffer capacity needed to avoid overflow at the client is given by $B(DB_{max} - D)$. The proof for this can be found in Appendix A.2. Note that for bandwidths under 1Mbps and DB_{max} under 500 ms, the buffering needed to avoid overflow is quite small (< 62.5kbytes).

We have calculated the bandwidth needed for EDPF, SRR, and ASL for various delay bounds (DB_{max}) and bandwidth splits. Since ASL is the ideal case, we express the bandwidth required in the other two cases as a percentage over that required for ASL. Fig. 5 shows this percentage for the case of three interfaces when the bandwidth is split among them in different ratios. Note that the y-axis is set to log-scale. We see that EDPF performs close to the ideal case ASL, and outperforms SRR by a huge margin in most cases.

We have performed a range of experiments, varying the number of interfaces as well as the bandwidth splits. The nature of the results remains the same. Table 2 summarizes the results for all these runs by averaging the bandwidth needed over these experimental runs—the averaging is done across various bandwidth splits. We considered 20 different splits as summarized in Table 3.

TABLE 2
Average Bandwidth Required (in kbps) for ASL, EDPF, and SRR

Ifas	Split 1	Split 2	Split 3	Split 4	Split 5
2	1:1	3:1	5:1	7:1	9:1
3	1:1:1	3:2:1	5:3:1	7:4:1	9:5:1
4	1:1:1:1	3:1:1:1	5:2:1:1	7:2:2:1	9:3:2:1
5	1:1:1:1:1	3:2:1:1:1	5:2:1:1:1	7:3:2:2:1	9:5:3:2:1

4.2.3 Application Performance Measures

While the previous section looked at the bandwidth required to satisfy a given delay bound, we now look at application behavior for a given bandwidth allocation. For the rest of this section, we fix the aggregate bandwidth at 600kbps (1.5 times mean). A choice of a much lower bandwidth than this results in $>$ 1 percent of the packets experiencing delay in excess of 500 ms, maximum permissible for interactive video. The number of wireless interfaces considered is three for most experiments. The use of two interfaces has less scope for reordering than three interfaces; hence, we present results for three interfaces (the nature of the results remains the same for two interfaces). We now present the various performance metrics in turn.

System Backlog. Since we are providing bandwidth that is much less than the peak rate, some amount of backlog is inevitable. The faster the algorithms service the packets, the faster the backlog clears and the better the algorithms aggregate bandwidths. Fig. 6 shows the backlog in the system as observed between the HA and the client application for the different algorithms when the bandwidth split is 3:2:1. For SL, since the packets are serviced at a constant rate, the backlog captures the variable bit rate part of the video. The maximum and mean backlog for SL are 21,200 bytes and 3,100 bytes, respectively. For EDPF and SRR, since we have multiple interfaces, the backlog is higher. However, the backlog for EDPF is very close to SL with a few additional peaks. The

TABLE 3
Bandwidth Splits on the Interfaces

Ifas	Split 1	Split 2	Split 3	Split 4	Split 5
2	1:1	3:1	5:1	7:1	9:1
3	1:1:1	3:2:1	5:3:1	7:4:1	9:5:1
4	1:1:1:1	3:1:1:1	5:2:1:1	7:2:2:1	9:3:2:1
5	1:1:1:1:1	3:2:1:1:1	5:2:1:1:1	7:3:2:2:1	9:5:3:2:1

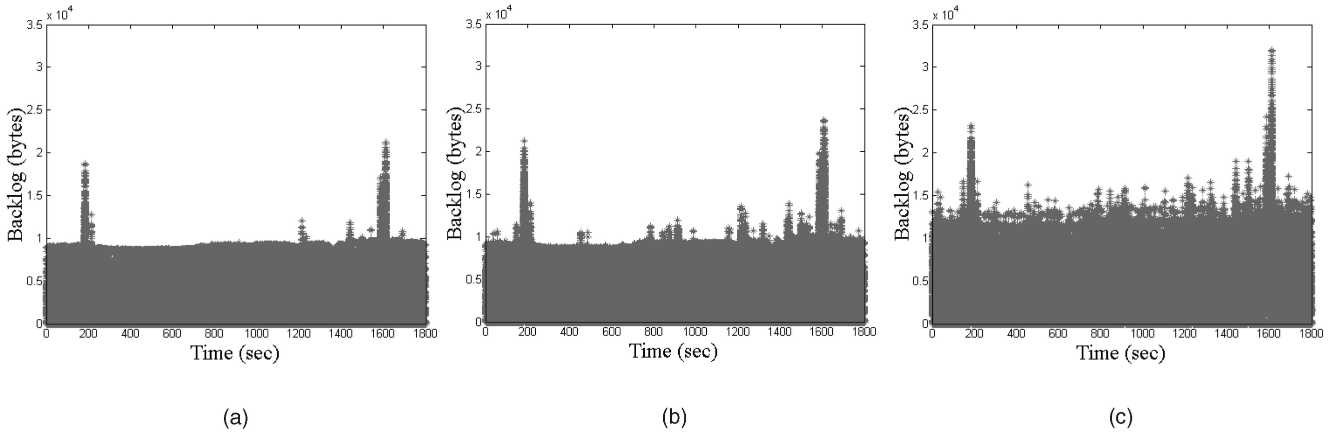

Fig. 6. Backlog in the system between HA and Client Application. (a) SL, (b) EDPF, and (c) SRR.

maximum and mean backlog for EDPF is 23,380 bytes and 3,700 bytes, respectively. SRR, on the other hand, has more peaks and the peaks are rather high. The maximum and mean backlog values for SRR are 32,150 bytes and 4,800 bytes, respectively.

The backlog as measured above between the HA and the client application includes the buffering needed to deliver the packets in order to the application. This is of interest from the application perspective. However, the backlog when measured between the HA and the client receiver can be directly related to the number of bits serviced (work done) by the algorithms—Theorem 2 in Section 3. Since the arrival distribution at the HA is identical across the algorithms, the difference in the number of bits serviced by the algorithms will be the same as the negative of the difference in the backlog induced by the algorithms. We measured this difference in backlog for EDPF and SRR over SL for different splits. We observe that this difference always satisfied the maximum bound for EDPF. However, for SRR, this difference was much higher than even the maximum permissible under EDPF. For example, for a 3:2:1 split, this difference was about 3,596 bytes for EDPF, less than 5,000 bytes as given by the bound. Where as for SRR, this difference was about 9,096 bytes.

Delay Distribution. The Cumulative Distribution Function (CDF) of the delay experienced by the packets (including buffering delay needed to deliver the packets in order) is shown in Fig. 7. The different plots in each graph are for the different algorithms, and for different values of the bandwidth split. For ASL, 99.8 percent of the packets have delay less than 200 ms. In case of EDPF, this value ranges between 99.2 to 99.6 percent for different splits. For SRR, its between 56.5 and 99.2 percent.

We have also verified to see if the delay experienced by the packets (between the HA and client receiver) is indeed under the delay bounds as derived in Section 3, Theorem 3. We observe that, as with the work done, the difference in delay of EDPF scheduling over SL was under the bounds for all splits considered. However, with respect to SRR this difference was large. For example, the difference in delay for EDPF in case of 5:3:1 split was 63 ms, well under the 133 ms delay bound. However, in case of SRR, this difference was 196 ms, well above EDPF.

Another point worth mentioning here is the amount of buffering needed to overcome reordering. For the splits considered, this was under 3,600 bytes for EDPF, where as for SRR it was as high as 11,500. We observed that in most cases, the buffering needed for EDPF exceeded the bound derived in Theorem 5 in Section 3. This was mainly due to the Internet path delay variations. When we fixed these delay values to a constant, we observed that the buffering needed was under the derived values. For example, for the 5:3:1 split, with fixed delays, buffering needed for EDPF was 1,305, well under the 2,000 bytes maximum bound.

Frame Discard Ratio. Fig. 8 shows F_{loss} as a function of different DB_{max} when the number of interfaces is fixed at three. As expected, F_{loss} decreases as DB_{max} increases. When DB_{max} is set at 200 ms, EDPF achieves a F_{loss} less than 0.6 percent while for SRR it can be as high as 20 percent loss (for ASL it is 0.2 percent).

Glitch Statistics. The glitch rate is another useful metric that captures the disruption in the video presentation due to discarded frames. Table 4 shows the glitch statistics when the number of interfaces used is three and for 300 ms delay bound. In terms of the glitch rate too, SRR performs very poorly. Though EDPF has higher average glitch duration than SRR, it should be looked in relation to the glitch rate. For EDPF, glitches happen less often and when they do,

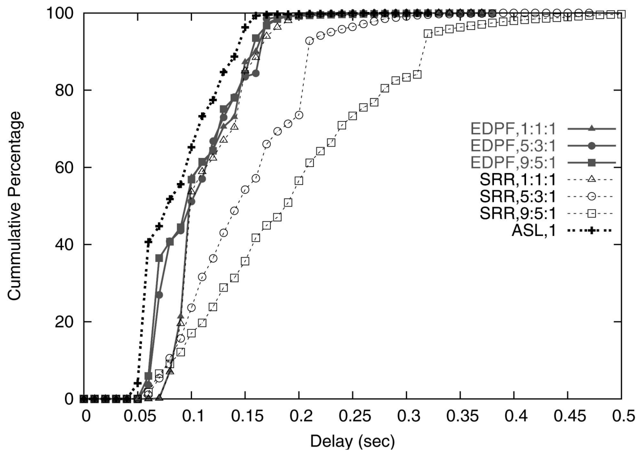

Fig. 7. Cumulative percentage of delay.

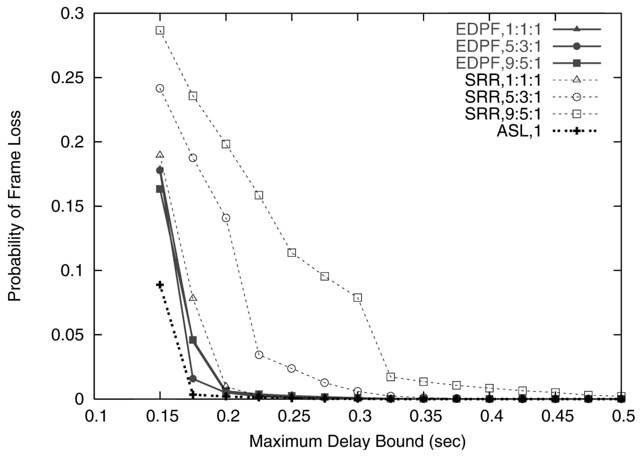

Fig. 8. Probability of frame loss.

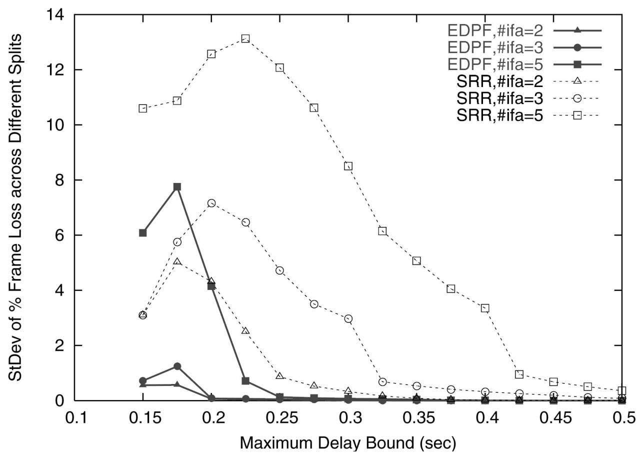

Fig. 9. Sensitivity to bandwidth asymmetry.

they span on average 3-6 frames. While in SRR, glitches happen more often and on average span small intervals 1-3 frames. Usually, the number of occurrences when glitch durations exceeds 3 is about the same for EDPF as in SRR.

4.2.4 Bandwidth Asymmetry and Number of Interfaces

In order to capture the sensitivity of the system performance to bandwidth asymmetry and the number of interfaces, we compute F_{loss} under different splits (see Table 3) for a given number of interfaces and delay bound. The standard deviation of the obtained values (expressed in percent) in shown in Fig. 9 for different number of interfaces. As can be seen in the figure, the standard deviation increases and then falls with DB_{max}, for both EDPF and SRR. When DB_{max} is small, the percentage of lost frames is quite large irrespective of the bandwidth split and, hence, we do not see much variation in loss across splits. But, as DB_{max} is increased, the variation becomes more apparent. For large values of DB_{max}, the frame loss goes down closer to zero and so does the variation. But, overall, compared to SRR, EDPF is more robust to bandwidth asymmetry. This is a desirable feature since it allows the client more freedom to make bandwidth requisitions on the various network interfaces.

To measure the sensitivity of the algorithms to the number of interfaces we measured the mean value of F_{loss} as a function of the number of interfaces. As the number of interfaces increases, so does the scope for reordering and, hence, F_{loss}. However, EDPF is more tolerant of increase in number of interfaces than SRR. For instance, for a DB_{max} of 200 ms, when increasing the number of interfaces from 3 to 4, EDPF showed an increase in F_{loss} of only 2.1 percent while SRR showed an increase of 5.2 percent.

4.2.5 Miscellaneous Issues

Channel Losses. So far, we have not considered channel losses. In this setup, it may not be possible to alter the scheduling to overcome channel losses as the time granularity over which the channel state changes is likely to be finer than the feedback loop between the MH and the proxy. Normally, radio networks that support real-time applications do try to achieve loss rate less than some negotiated value by using efficient FEC, limited ARQ, or through an increase in transmit power. We have run a set of experiments to see the performance of the system under channel losses with limited ARQ. Retransmissions may alter EDPF's estimate of the variable A_l (time when channel becomes available). However, we observed that the effect is very minor, masked by the gains that can be had through retransmissions. For a DB_{max} of 300 ms, 5:3:1 split, 1 percent uniformly distributed channel losses, no retransmissions gave us a F_{loss} of 1.9 percent, while retransmissions brought it down to 0.2 percent.

Wireline Delay Variations and Asymmetry. EDPF uses the estimated delay between proxy and the BSs in determining the delivery time of packets. It may seem that large delay variations may affect EDPF's performance. However, we argue that this is not the case. To perceive

TABLE 4
Glitch Statistics: Startup Latency = 0.3 Seconds and # Interfaces = 3

Algo.	ASL	EDPF 1:1:1	EDPF 5:3:1	EDPF 9:5:1	SRR 1:1:1	SRR 5:3:1	SRR 9:5:1
Glitch Rate (per ms)	0.55	0.55	2.78	7.22	3.89	140	1809
Avg Glitch Dur.	4	6	3.2	2.77	2.14	1.063	1.089
Max Glitch Dur.	4	6	8	8	7	6	9

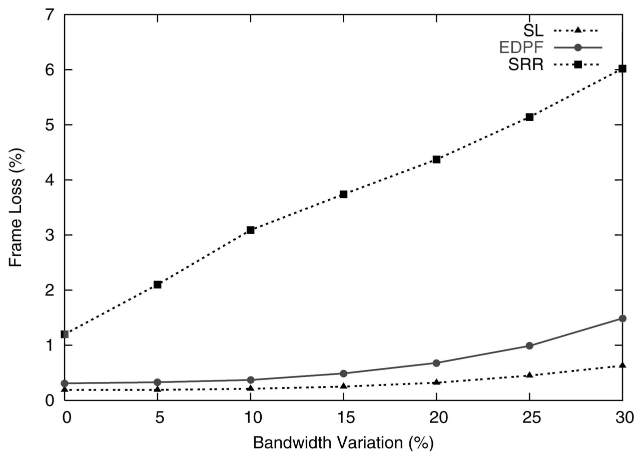

Fig. 10. Sensitivity to bandwidth fluctuations.

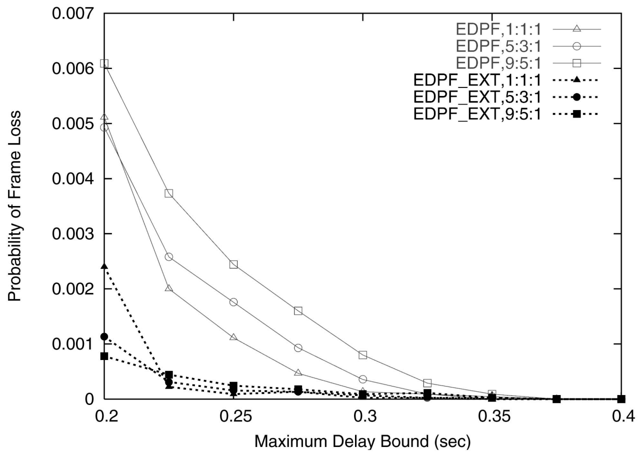

Fig. 11. Improvement with extended EDPF.

good quality video, we would like to achieve $F_{loss} < 1\%$. The bandwidth needed to guarantee such low loss rate should overcome the queuing delay (induced at BS) more than wireline delay. The delay variation will likely be masked by this queuing delay. In (1) of Section 3, A_l dominates $a_i + D_l$ for most packets that experience excess delay. We observe this through experiments as well. At a DB_{max} of 200 ms, for a truncated Guassian delay distribution with mean 22 ms and 10 ms standard deviation, F_{loss} increased by only 0.14 percent over no delay variation. The same reasoning applies to delay asymmetry as well. To make a fair comparison, we consider a 1:1:1 split and compare a wireline delay asymmetry of 44:44:44 to a 22:44:66. We observed only a 0.07 percent increase in F_{loss} at a DB_{max} of 225 ms due to increase in asymmetry.

Bandwidth Fluctuations. Apart from delay variations, we also examined the impact bandwidth fluctuations have on the performance of our system. Given the very stringent delay requirements of interactive video, it is very difficult to support these applications in a setup involving large and rapid bandwidth fluctuations (e.g., resulting from cross traffic). Hence, we do not consider the same. However, small bandwidth fluctuations (e.g., resulting from processing delays at the BSs) about the negotiated bandwidth values can be tolerated to a good extent as we show now.

We introduce bandwidth fluctuation as a percentage over the negotiated bandwidth and measure the effect this fluctuation has on application frame loss F_{loss} and buffering needed to deliver the packets in order. For example, a 10 percent bandwidth variation about 200kbps reserved bandwidth involves choosing a uniform value between 180-220 kbps for each packet transmission on that interface. Fig. 10 shows the variation of F_{loss} as a function of bandwidth variation for a 3:2:1 split of 600kbps and a DB_{max} of 200 ms. In all cases, F_{loss} increases with increase in fluctuations as is expected. For EDPF, this increase in frame loss is hardly perceptible for upto 10 percent variation. From then on it slowly increases, crossing the unacceptable 1 percent frame loss threshold at 25 percent bandwidth variation. For SRR, the increase in F_{loss} with bandwidth variation is rather steep. With respect to buffering required to deliver packets in order, both EDPF and SRR experience increased reordering with increase in fluctuation. At 0 percent variation (no bandwidth fluctuation) this value is about 2,000 bytes for EDPF and 7,164 bytes for SRR. At 25 percent bandwidth fluctuation, this value increases to 6,439 bytes for EDPF and 16,534 for SRR. These values are still too small to effect processing speed or memory requirements at the client.

Extensions to EDPF. It is possible to improve the performance of EDPF further by taking into consideration additional parameters. If the MH provides EDPF with additional information such as maximum tolerable delay, EDPF can drop packets that are unlikely to meet their delay constraints (EDPF already maintains an estimate of it). This saves scarce bandwidth and helps other packets to meet their delay constraints. We have extended EDPF (EDPF_EXT) to support this feature. Fig. 11 shows the relative improvement. Also, if frame priority information can be conveyed in the packets, EDPF can perform appropriate filtering—dropping lower priority frames in presence of congestion.

In addition to the "Office Cam" video trace, we have experimented with other video traces from [14] as well as H.263 encoding. We obtained similar results as shown above. EDPF in all cases, effectively aggregated bandwidth while minimizing delay experienced by the packets.

5 Discussion

In this section, we elaborate on some important points. We first discuss the deployment complexity of our approach. Subsequently, we present two aspects that affect the performance of our network layer approach: delay/bandwidth fluctuations and wireless mobility.

5.1 Deployment Complexity and Overheads

Our network layer architecture has been designed with the goal of introducing minimum changes to the infrastructure. The only changes needed are software changes at the MH and deployment of proxies, no changes are needed in the radio network or server software. The deployment complexity of our architecture is thus minimal. To increase reliability and scalability of the architecture, we envision multiple proxies, each providing service to a subset of MHs.

As far as the complexity of the scheduling algorithm EDPF goes, the per-packet computation complexity is proportional to the number of interfaces, which is likely to be two to three in most cases. In terms of network overhead, the (relative) one-way delay and bandwidth information need to be passed from the client to the network proxy only once during setup for interactive applications and once every few seconds for streaming applications. The buffering required to pass packets in order is also minimal, normally limited to under five packets.

5.2 Delay/Bandwidth Fluctuations and Estimation

Two important parameters that affect the performance of our scheduling algorithm are 1) bandwidth of the wireless last hop and 2) delay on the paths from proxy to BSs. Let us examine each in turn.

One of the assumptions we make in this work is that the underlying infrastructure provides some form of QoS support in the form of bandwidth guarantees. Without such guarantees, its almost impossible to support applications with very stringent delay requirements like interactive video. Normally, when the infrastructure provides QoS, the client negotiates a certain bandwidth on each of its interfaces. It then passes this information to the proxy for use as an estimate of bandwidth in EDPF scheduling. The BSs try to guarantee this negotiated bandwidth for the duration of the connection. So, large bandwidth variations (e.g., resulting from cross traffic) are ruled out of consideration in this paper because of the very stringent delay requirements of the real-time applications. In [13], we have extended EDPF to handle bandwidth fluctuations for best-effort-based TCP applications. On the other hand, small or intermittent bandwidth fluctuations (e.g., resulting from random processing delays at BSs or retransmission) can be tolerated to some extent without much performance degradation as was shown in Section 4.2.5.

Obtaining delay estimates for the path between proxy and the BSs during the course of the connection without support from the BSs is, in general, a difficult task. This is because it is difficult to figure out the contribution of queuing delay to the overall end to end delay observed on the path. As mentioned earlier, we do not view this as a serious limitation because of the following reasons. For one, delay estimates during connection setup (where there is no queuing) or estimates from the recent past (few tens of seconds to a few minutes) will most likely be sufficient for the duration of the connection. This is because Internet path delays are known to vary only slowly, over several tens of minutes [15]. Further, any errors in estimation or fluctuations which usually are small (as average delays on the backbone are themselves small) will likely be masked by the transmission and queuing delay at the bottleneck bandwidth and do not degrade performance much (Section 4.2.5). In (1) of Section 3, A_l dominates $a_i + D_l$ for most packets. We observed this in our experiments as well (Section 4.2.3)—the average buffer size at the BSs was about 4,500 bytes.

5.3 Mobility and Blackouts

Mobility and blackouts an integral part of a wireless environment and should be addressed in the design of the network architecture. Our architecture handles mobility similar to Mobile IP [7]; however, unlike it, it can support more than one wireless interface. Stalls during handoff can be handled by not sending packets on the interface which is performing handoff related processing. Blackouts on an interface can be similarly handled.

Even though the architecture can support mobility, the performance of real-time applications in mobile environments can suffer due to stalls during handoffs and/or the inability to negotiate adequate bandwidth at the new BS. No amount of clever scheduling onto the interfaces will help in this regard as the available bandwidth is not sufficient to support the video application. Content adaptation is the only viable option. Our network-layer architecture lends itself well to this approach because of the feedback mechanism in place between client and the proxy.[3] For example, the client can notify the proxy of an impending handoff on an interface, whereby the proxy will stop using that interface and in addition drop all low priority frames (e.g., B frames in MPEG4) because of the decrease in available bandwidth. While the above scheme simply drops all low priority frames, in [16], we have considered an intelligent frame discard algorithm for interactive video in a multiple interface setting when adequate bandwidth that provides high quality video cannot be reserved. The algorithm relies on an important aspect of video stream—Group of Pattern (GOP) frame size correlation to predict future frame sizes (and, hence, playback deadlines) in a small window. The decision to drop a frame is based on the impact the present frame drop has on meeting future high priority frame deadlines and, hence, on overall quality of the video.

6 Related Work

Bandwidth aggregation across multiple channels has its origins as a link layer solution in the context of analog dial-ups, ISDN, and ATM [9], [17], [18]. Link Layer solutions are infeasible in our present scenario, where the RANs in question belong to different domains controlled by different service providers.

The Stripe protocol [8] is a generic load-sharing protocol that can be used over any logical First-In-First-Out (FIFO) channels, it was implemented in some routers in the context of Multilink PPP. It is based on Surplus Round Robin (SRR) and provides FIFO delivery of these packets to higher layers with minimum overhead in the form of packet processing (looking up the packet sequence number). The design goals of stripe are different from those considered in this paper, it achieves its objective at the expense of introducing additional delay. For real-time interactive applications, this approach will not work well as was shown in the previous sections.

Contemporary to our initial work [19] that explored some of the ideas presented in this paper, some transport and network layer solutions have been proposed to achieve bandwidth aggregation in a similar setting. A network layer solution based on tunneling was proposed in [20] and performance of TCP has been evaluated. Though similar in spirit to our architecture, this work does

3. Content adaptation at video server would also work if such support is provided by the server and if the feedback delay is small.

not look into real-time application support or address in depth the architecture components that enable diverse services. The Reliable Multiplexing Transport Protocol (RMTP) [21] is a reliable rate-based transport protocol that multiplexes application data onto different channels. Parallel TCP (pTCP) [22] is another transport layer approach that opens multiple TCP connections one for each interface in use. The focus of this paper is on supporting real-time applications which may not employ TCP as the transport protocol because of their delay constraints. Further, our main goal is to introduce minimal changes to the infrastructure while enabling diverse functionalities, which these approaches cannot achieve.

7 Conclusions

In this paper, we motivate the advantages of simultaneous use of multiple interfaces and propose a network-layer architecture that enables such use. Our network layer architecture provides many different services—bandwidth aggregation, reliability support, resource sharing, and data-control plane separation to the end MH. Further, it is transparent to applications and involves minimum changes to the infrastructure.

One of the services provided by the architecture is BAG (bandwidth aggregation) for real-time applications. Implementation/simulations show that BAG services can bring in significant performance improvements over conventional single interface use. The scheduling algorithm that BAG employs (EDPF) mimics closely the idealized Aggregated Single Link (ASL) case and outperforms by large margin approaches based on weighted round robin.

Though introduced in the context of wireless interfaces, BAG and EDPF are applicable in broader contexts. Any system with multiple paths can use the EDPF scheduling algorithm to provide QoS support.

Appendix A

Details of Proofs

A.1 Properties of EDPF

Details of proof for Theorem 2. W_{ASL} takes on a maximum value when the link becomes idle. Let t be such a time. Since ASL is idle, all packets serviced must have arrived before t. We now have the following two cases.

Case 1. One or more of the links in EDPF are idle at t.

The deficit over ASL, EDPF has to serve after t is maximum when: 1) All links except one are busy serving the deficit. 2) The idle link corresponding to lb. Using Lemma 1, this difference in time $T_l(t) - T_{lb}(t)$ for which any link $l \neq lb$ is busy is bounded by L_{max}/B_{min}. The overall deficit in bits is thus bounded by:

$$L_{max} \sum_{l \neq lb} B_l/B_{min} = L_{max}\left(\sum_{l=1}^{N} w_l - 1\right).$$

Case 2. All the links are busy at t.

Let $\tau < t$, be the earliest time instant at which all links in EDPF got busy. Between $[\tau, t]$,

$$W_{ASL}(\tau, t) \leq W_{EDPF}(\tau, t) = \sum_{l=1}^{N} B_l(t - \tau).$$

Thus, the difference at t cannot exceed that at τ, i.e.,

$$W_{ASL}(0, t) - W_{EDPF}(0, t) \leq W_{ASL}(0, \tau) - W_{EDPF}(0, \tau).$$

And, Case 1 bounds the right hand side by $L_{max}(\sum_{l=1}^{N} w_l - 1)$. □

Details of proof for Theorem 3. In case of EDPF, the following two cases arise.

Case 1. When packet i arrives, it finds one or more of the links in EDPF idle. If it were scheduled on the idle link, its delivery time will not exceed $a_i + L_i/B_{min}$. Since EDPF schedules the packet on the link which delivers its the earliest, the departure time of this packet when scheduled on other links would also not exceed this amount, i.e., $d_i^{EDPF} \leq a_i + L_i/B_{min}$. In case of ASL,

$$d_i^{ASL} \geq a_i + L_i / \sum_{l=1}^{N} B_l.$$

Thus,

$$d_i^{EDPF} - d_i^{ASL} \leq \frac{L_i(\sum_{l=1}^{N} w_l - 1)}{\sum_{l=1}^{N} B_l}.$$

Case 2. When packet i arrives it finds all the links busy, let $j < i$ be the latest packet whose arrival busies all the links. Let l_j be the link on which j was scheduled and l_i be the link on which i was scheduled. We now consider the worst case delay that can be experienced by packet i. This happens if

- When j arrives, the number of bits P that still need to be serviced is maximum possible. This essentially increases the time before the system can serve packets j to i. This event happens when $l_j = lb$ and for $l \neq lb$,

  $$T_l(a_j) - T_{l_j}(a_j) = L_{max}/B_{min} \text{ (from Lemma 1)}.$$

  Hence,

  $$P = \sum_{l=1}^{N} T_l(a_j) - T_{l_j}(a_j) \leq L_{max}\left(\sum_{l=1}^{N} w_l - 1\right).$$

- All packets between i and j (inclusive) are delivered ahead of i, i.e., $d_i \geq d_k$ for $j \leq k < i$. So, we have,

  $$d_i = T_{l_i}(a_i+) = max\{T_l(a_i+), \text{ for } 1 \leq l \leq N\}.$$

  If we denote by $\delta_{l_i,l}$ the time spent by link $l \neq l_i$ in the interval $[a_j, d_i]$ either idle (or serving packets $k > i$). We have $\delta_{l_i,l} = T_{l_i}(a_i^+) - T_l(a_i^+)$. The packet i is delayed further if $\delta_{l_i,l}$ is maximum possible, this essentially pushes further the delivery time of packet i, as some of the work (serving packets j to i) that needs to be done on links $l \neq l_i$ got pushed onto link l_i. If we denote by F, the overall idle time in bits in the interval $[a_j, d_i]$, we have $F = \sum_{l \neq l_i} \delta_{l_i,l} * B_l$.

From Lemma 1 (Case 1), we have $\delta_{l_i,l} \leq L_i/B_l$. Thus, $F \leq (N-1)L_i$.

During the interval $[a_j, d_i]$, the system was busy serving load P, packets from j to i and either staying idle or serving packets $k > i$. Hence, we have,

$$(d_i^{EDPF} - a_j)\sum B_l = \sum_{k=j}^{i} L_k + P + F,$$

$$\begin{aligned} d_i^{EDPF} \leq & a_j + \frac{\sum_{k=j}^{i} L_k}{\sum B_l} \\ & + \frac{L_{max}(\sum w_l - 1)}{\sum B_l} + \frac{(N-1)L_i}{\sum B_l}. \end{aligned}$$

In case of ASL, $d_i^{ASL} \geq a_j + \frac{\sum_{k=j}^{i} L_k}{\sum B_l}$. Thus, the theorem follows. □

Details of proof for Theorem 4. The jitter experienced by a packet i is given by $J_i = (r_i - r_{i-1}) - (a_i - a_{i-1})$. If the packet i is buffered, we will have $r_i = r_{i-1}$ and the jitter will be non positive as $a_i \geq a_{i-1}$. So, in the proof below, we only look at the case where i is not buffered, i.e., $r_i = d_i$. Note that $i-1$ could still be buffered. Also, note that J_i is maximum when r_{i-1} is minimum and $a_i = a_{i-1}$.

We consider the following four different cases based on whether packets $i-1$ and i are transmitted on link hb.

Case 1. Both packets $(i-1)$ and i are transmitted on hb. If $r_i = d_i = a_i + L_i/B_{max}$ i.e., packet i begins transmission immediately on arrival. Then, $J_i < L_i/B_{max}$ as $r_{i-1} - a_{i-1} > 0$. Otherwise, we have $d_i = d_{i-1} + L_i/B_{max}$. Since $a_i - a_{i-1} \geq 0$ and $r_{i-1} \geq d_{i-1}$, we have $J_i \leq d_i - r_{i-1} \leq d_i - d_{i-1} = L_i/B_{max}$.

Case 2. Packet $(i-1)$ is transmitted on hb and packet i is transmitted on some other link $(l \neq hb)$. Since we assume packet i is not buffered, $d_i \geq d_{i-1}$. We have $a_i < d_{i-1}$ as otherwise packet i would have been transmitted on hb. Therefore, $d_{i-1} = T_{hb}(a_i^+)$ and $d_i = T_l(a_i^+)$. From Lemma 1 (Case 1), we have

$$d_i - d_{i-1} = T_l(a_i^+) - T_{hb}(a_i^+) \leq L_i/B_{max}.$$

Since

$$r_{i-1} \geq d_{i-1}, J_i \leq d_i - r_{i-1} \leq d_i - d_{i-1} \leq L_i/B_{max}.$$

Case 3. The $(i-1)$th packet is transmitted on link $l(/ = hb)$ and the ith packet is transmitted on hb. Let $j < i-1$ be the packet that was transmitted latest on link hb. If $d_i = a_i + L_i/B_{max}$, as mentioned in case 1, $J_i < L_i/B_{max}$. Otherwise, if $d_i > a_i + L_i/B_{max}$, we have $d_i = d_j + L_i/B_{max}$. Packet $i-1$ can be passed up only after j, hence $r_{i-1} \geq d_j$. Therefore,

$$J_i \leq d_i - r_{i-1} \leq d_i - d_j = L_i/B_{max}.$$

Case 4. Packet $(i-1)$ is transmitted on link $l(\neq hb)$ and the packet i is transmitted on link $k(\neq hb)$. Again let $j < i-1$ be the packet that was transmitted latest on link hb. Since packet i is not transmitted on hb, $a_i < d_j$. From Lemma 1 (Case 1), we have $d_j = T_{hb}(a_i^+)$ and $d_i = T_k(a_i^+)$ and, hence, $d_i - d_j \leq L_i/B_{max}$. As before, $r_{i-1} \geq d_j$ and, hence, $J_i \leq d_i - r_{i-1} \leq d_i - d_j = L_i/B_{max}$.

Since, in all the four cases the bound holds, the theorem is proven. □

Details of proof for Theorem 5. At any time t, let $T_{max}(t) = max\{T_l(t)\}$. After t, any packet transmitted on a link $l \neq max$, if it is delivered before $T_{max}(t)$ needs to be buffered. Let $\delta_{max,l} = T_{max}(t) - T_l(t)$. Thus, all packets transmitted on link l after t whose summation of packet lengths is less than $\delta_{max,l} * B_l$ will need to be buffered. From Lemma 1, $\delta_{max,l} \leq L_{max}/B_l$. Thus, the total buffer size would be

$$\begin{aligned} &\sum_{l \neq max} \delta_{max,l} * B_l \leq \\ &\sum_{l \neq max} L_{max} = (N-1) * L_{max}. \end{aligned}$$

□

A.2 Interactive Video—Buffering Required to Avoid Overflow

The buffer size at the client increases whenever a packet arrives and decreases whenever a packet has to be displayed. The maximum time a packet can spend in the buffer is $DB_{max} - D$—if it spends anytime more than this, the packet will surely miss its playback deadline. During this interval, the buffer can fill at most at a rate of B, so the size of the buffer C cannot exceed $B(DB_{max} - D)$.

ACKNOWLEDGMENTS

This work was supported by Ericsson Adaptive Systems Group at UCSD and US National Science Foundation under grant 0331690. The authors would like to thank Professor Bhaskaran Raman, Professor Geoffrey Voelker, Professor Alex Snoeren, Rajesh Mishra, and the anonymous reviewers for their helpful suggestions.

REFERENCES

[1] H. Balakrishnan, V. Padmanabhan, S. Sheshan, and R. Katz, "A Comparision of Mechanisms for Improving TCP Performance over Wireless Links," *IEEE/ACM Trans. Networking,* vol. 5, no. 6, pp. 756-769, Dec. 1997.

[2] H. Balakrishnan, V. Padmanabhan, and R. Katz, "The Effects of Asymmetry on TCP Performance," *Mobile Networks and Applications,* vol. 4, no. 3, pp. 219-241, Oct. 1999.

[3] S. Lu, V. Bharghavan, and R. Srikant, "Fair Scheduling in Wireless Packet Networks," *IEEE/ACM Trans. Networking,* vol. 7, no. 4, pp. 473-489, Aug. 1999.

[4] A. Campbell, J. Gomez, S. Kim, Z. Turanyi, C.-Y. Wan, and A. Valko, "Comparison of IP Micromobility Protocols," *IEEE Wireless Comm.,* vol. 9, no. 1, pp. 72-82, Feb. 2002.

[5] M. Stemm and R. Katz, "Vertical Handoffs in Wireless Overlay Networks," *Mobile Networks and Applications,* vol. 3, no. 4, pp. 335-350, 1998.

[6] B. Girod, M. Kalman, Y. Liang, and R. Zhang, "Advances in Channel-Adaptive Video Streaming," *Wireless Comm. and Mobile Computing,* vol. 2, no. 6, pp. 549-552, Sept. 2002.

[7] C.E. Perkins, "Mobile IP," *IEEE Comm. Magazine,* vol. 35, no. 5, pp. 84-99, May 1997.

[8] H. Adiseshu, G. Parulkar, and G. Varghese, "A Reliable and Scalable Striping Protocol," *ACM Computer Comm. Rev.,* vol. 26, no. 4, pp. 131-141, Oct. 1996.

[9] J. Duncanson, "Inverse Multiplexing," *IEEE Comm. Magazine,* vol. 32, no. 4, pp. 34-41, Apr. 1994.

[10] Hewlett-Packard's Auto Port Aggregation, http://www.hp.com, 2006.

[11] K. Chebrolu, R. Mishra, P. Johansson, and R.R. Rao, "A Network Layer Architecure to Enable Multiaccess Services," unpublished.

[12] D. Brudnicki, "Third Generation Wireless Technology," http://www.seasim.org/archive/sim102001.pdf, 2001.

[13] K. Chebrolu, B. Raman, and R.R. Rao, "A Network Layer Approach to Enable TCP over Multiple Interfaces," *J. Wireless Networks (WINET),* Sept. 2005.

[14] Location of MPEG-4, H.263 video traces, http://www-tkn.ee.tu-berlin.de/research/trace/trace.html, 2006.

[15] A. Acharya and J. Saltz, "A Study of Internet Round-Trip Delay," Technical Report CS-TR 3736, Univ. of Maryland, College Park, 1996.

[16] K. Chebrolu and R.R. Rao, "Selective Frame Discard for Interactive Video," *Proc. IEEE Int'l Conf. Comm.,* June 2004.

[17] Inverse Multiplexing for ATM (IMA) Specification, Version 1.1, ATM Forum Doc. AF-PHY-0086. 001, The ATM Forum Technical Committee Std., 1999.

[18] K. Sklower, B. Lloyd, G. McGregor, D. Carr, and T. Coradetti, "The PPP Multilink Protocol (MP)," RFC 1990, Aug. 1996.

[19] K. Chebrolu and R.R. Rao, "Communication Using Multiple Wireless Interfaces," *Proc. IEEE Wireless Comm. and Networking Conf.,* Mar. 2002.

[20] D.S. Phatak and T. Goff, "A Novel Mechanism for Data Streaming across Multiple IP Links for Improving Throughput and Reliability in Mobile Environments," *Proc. IEEE INFOCOM '02 Conf.,* pp. 773-781, June 2002.

[21] L. Magalhaes and R. Kravets, "Transport Level Mechanisms for Bandwidth Aggregation on Mobile Hosts," *Proc. IEEE Int'l Conf. Network Protocol,* Nov. 2001.

[22] H. Hsieh and R. Sivakumar, "A Transport Layer Approach for Achieving Aggregate Bandwidths on Multihomed Mobile Hosts," *Proc. ACM MOBICOM '02 Conf.,* Sept. 2002.

Kameswari Chebrolu received the BE degree in electronics and communication engineering from Andhra University, and the MS and PhD degrees in electrical and computer engineering from the University of California at San Diego. She is a postdoctoral researcher in the San Diego Division of the California Institute of Telecommunications and Information Technology (Cal-(IT)^2). Her research interests are in the areas of wireless network architecture, protocol design, and analysis.

Ramesh R. Rao received the BE degree from the University of Madras and the MS and PhD degrees from the University of Maryland at College Park. He is a member of the faculty of Irwin and Joan Jacobs School of Engineering, where he has been a member of the faculty since 1984. Professor Rao is the former director of UCSD's Center for Wireless Communications (CWC), and the current director of the San Diego Division of the California Institute of Telecommunications and Information Technology (Cal-(IT)^2). As director of the San Diego Division of Cal-(IT)^2, he leads several interdisciplinary, collaborative projects. His research interests include architectures, protocols, and performance analysis of computer and communication networks, and he has published extensively on these topics. Most recently, Dr. Rao was honored by being appointed the first holder of the Qualcomm Endowed Chair in Telecommunications and Information Technologies.