UNIVERSITY OF CALIFORNIA, SAN DIEGO

Multi-Access Services in Heterogeneous Wireless Networks

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy
in Electrical Engineering
(Communication Theory and Systems)

by

Kameswari Chebrolu

Committee in charge:

Professor Ramesh R. Rao, Chair
Professor Geoffrey M. Voelker
Professor Rene L. Cruz
Professor George Varghese
Professor Laurence B. Milstein

2004

The dissertation of Kameswari Chebrolu is approved, and it is acceptable in quality and form for publication on microfilm:

Chair

University of California, San Diego

2004

To my father

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

During the course of my PhD, I had the good fortunate to interact with a great set of people, who opened my mind to new possibilities and made this whole process very rewarding. First and foremost, I would like to thank my adviser, Prof. Ramesh Rao for his constant support, encouragement and for giving me the freedom to pursue ideas and research of my own liking. Despite his very busy schedule, he was always accessible, ready to help with any problem. I would also like to thank Prof. Geoffrey Voelker for the useful discussions, helpful suggestions and constant encouragement. My meetings with him always gave me a boost of that additional energy needed to get research work going at full pace. Prof. Rene Cruz, Prof. George Varghese and Prof. Laurence Milstein are part of my dissertation committee. I thank them for their time and feedback at various stages of my dissertation.

Rajesh Mishra and Per Johansson joined our lab as Ericsson researchers and got with them the industry experience. I have enjoyed working with them and gained perspective from the industry standpoint on many discussions we had on various topics. Rajesh was particularly helpful in overlooking some of the implementation aspects of my work.

During the course of my stay at San Diego, I consider myself lucky to have met with some real interesting and warm people. Joseph and Sudhakar were a constant source of support and helped me through thick and thin. Peter, Bogdan, Uma, Santosh, Gayathri, Sangeetha were among the first people I met in San Diego. I will always remember the good times I had with them. My wonderful lab mates Vikram, Pavan, Ashay, Shweta, Patrick and Prasad made the work place quite fun through lively and interesting discussions.

I would not have come this far if not for the constant support and encouragement from my parents. I am grateful that they believed in me to let me run after my dreams. I am particularly thankful to my father for setting such high standards and for being patient with me through the years. I would also like to

thank my brothers for all the encouragement and help over the years.

Finally, to Bhaskar, I owe the largest debt. I attribute to him, much of what I have learned about conducting research: organization, time management and discipline. Thank you for making my PhD experience a pleasant and memorable experience. On the personal front, I doubt if I can even express in words the difference you made in my life the past two years.

Chapter 5 is in part a reprint of the material in the following papers: K. Chebrolu and R. R. Rao, *Bandwidth Aggregation for Real-Time Applications in Heterogeneous Wireless Networks* (submitted for publication) and K. Chebrolu and R. R. Rao, *Selective Frame Discard for Interactive Video*, Proceedings of IEEE ICC 2004, Paris, France. Chapter 6 is in part a reprint of the material in the paper: K. Chebrolu, B. Raman and R. R. Rao, *A Network Layer Approach to Enable TCP over Multiple Interfaces*, Journal of Wireless Networks (WINET) (accepted).

VITA

1998	B.E., Electronics and Communications Engineering, Andhra University, Visakhapatnam, India
2001	M.S., Electrical and Computer Engineering, University of California, San Diego
2004	Ph.D., Electrical and Computer Engineering, University of California, San Diego

PUBLICATIONS

K. Chebrolu, B. Raman and R. R. Rao, *A Network Layer Approach to Enable TCP over Multiple Interfaces*, Journal of Wireless Networks (WINET) (Accepted).

K. Chebrolu and R. R. Rao, *Bandwidth Aggregation for Real-Time Applications in Heterogeneous Wireless Networks*, (Submitted for publication).

K. Chebrolu and R. R. Rao, *Selective Frame Discard for Interactive Video*, Proceedings of IEEE ICC 2004, Paris, France.

K. Chebrolu and R. R. Rao, *Communication using Multiple Wireless Interfaces*, Proceedings of IEEE WCNC 2002, Orlando FL.

H. Jin, K. Chebrolu, A. Pande, X. Chen, J. Razavilar and B. Subbiah, *Seamless Hand-off between Heterogeneous Wireless Networks*, Proceedings of 3G wireless 2002, San Francisco CA.

P. Nuggehalli, V. Srinivasan, K. Chebrolu and R. R. Rao, *Energy Aware Sampling Schemes*, Proceedings of IEEE WCNC 2000, Chicago IL.

FIELDS OF STUDY

Major Field: Electrical and Computer Engineering

Studies in Networks.
Professor Ramesh R. Rao

Studies in Information Theory and Coding.
Professors Jack K. Wolf and Kenneth Zeger

Studies in Digital Communications.
Professor Laurence B. Milstein

ABSTRACT OF THE DISSERTATION

Multi-Access Services in Heterogeneous Wireless Networks

by

Kameswari Chebrolu
Doctor of Philosophy in Electrical Engineering
(Communication Theory and Systems)
University of California, San Diego, 2004
Professor Ramesh R. Rao, Chair

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. We term the services enabled by such simultaneous use of multiple interfaces as *Multi-Access Services*. These services constitute Bandwidth Aggregation, Mobility/Reliability Support, Resource Sharing and Data-Control Plane Separation.

As a first step towards realizing in practice the above mentioned services, we develop a network layer architecture that introduces minimal changes to the infrastructure. We also identify and implement on an experimental testbed, the various functional components that make up this architecture.

While the architecture can support many different services, we explore in depth one such service provided by the architecture: Bandwidth Aggregation (BAG) in the context of *Video* and *TCP* applications.

For video applications, an important aspect of the architecture when providing BAG services is the scheduling algorithm that partitions the traffic onto different interfaces. 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, implementation and simulations that EDPF performs close to an idealized Aggregated Single Link (ASL) discipline and outperforms by a large margin other scheduling approaches based on weighted round robin. Apart from the scheduling algorithm, we also consider a content adaptation algorithm, *Min Cost Drop (MC-DROP)* to selectively drop video frames when adequate bandwidth cannot be reserved on the interfaces.

While BAG services can improve the throughput of TCP applications, it introduces challenges in the form of packet reordering. So, we propose *Packet Pair based Earliest-Delivery-Path-First for TCP (PET)* scheduling algorithm that minimizes reordering by sending packet pairs on the path that introduces the least amount of delay. A *Buffer Management Policy (BMP)* is also introduced at the client to hide any residual reordering from the TCP receiver. We show through simulations that PET in combination with BMP achieves good bandwidth aggregation under a variety of network conditions.

Chapter 1

Introduction and Motivation

The past decade has seen an explosive growth of the Internet. Commensurate with this growth, a variety of wireless technologies are being deployed to provide Internet access. Examples include GPRS, EDGE, CDMA2000, UMTS, Iridium, Bluetooth, IEEE 802.11, HiperLan etc. Heterogeneity has become the norm of today's wireless world.

Traditional wireless research has looked into improving the performance of applications using these technologies in a wireless setting as characterized by packet errors, mobility, asymmetric bandwidths, etc [10, 9, 29, 14]. With the incidence of technologies with different characteristics, seamless migration of connections [37] (vertical handoff) from one interface to another, content adaptation [21] to suit the characteristics of the interface in use have also been addressed. However, the basis of most of the research has been confined to single interface use at any given time to meet all the connectivity requirements of the end user applications.

When technologies with widely different characteristics (bandwidth, QoS support, pricing) co-exist, and coverage areas overlap, 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. In this work, we remove such an assumption and show how simultaneous use of multiple interfaces can help solve some of the limitations of wireless media and enable other new and interesting possibilities.

We term the services enabled by such simultaneous use of multiple interfaces as *Multi-Access Services*.

1.1 Multi-Access Services

There are many new classes of services arising from simultaneous use of multiple interfaces, both in scenarios involving a single host as well as more than one host. We now discuss some of the important ones below.

1.1.1 Bandwidth Aggregation (BAG)

The first multi-access service we consider is that of bandwidth aggregation. While we have come a long way in terms of peak data rates in mobile wireless 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 kbps [12] – 40kbps in 1xRTT, 80kbps in EDGE, 250kbps in UMTS. Even an 802.11 interface that can provide speeds upto 11Mbps, often is constrained in bandwidth in a loaded network because most hot spots these days connect to the Internet via "broadband" (DSL/Cable) which constitutes a bottleneck.

It is unlikely that the wireless bandwidth (especially wide-area networks) can keep pace with that available through wired means. Supporting real-time streaming and interactive applications (video telephony/conferencing) with stringent QoS requirements, large file transfers, even intense web sessions is a difficult task and may not be possible if confined to a single interface. Gleaning bandwidth available from all possible sources may be the only option to increase one's bandwidth and improve quality of or support demanding applications.

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

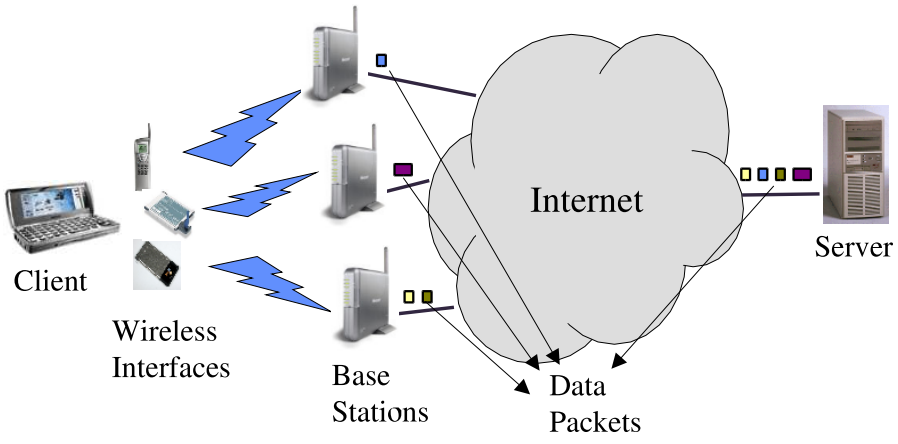


Figure 1.1: Bandwidth Aggregation (BAG): An Example

so as to avail all available bandwidth. Fig. 1.1 depicts bandwidth aggregation across three interfaces.

1.1.2 Mobility Support

The next multi-access service we consider is that of mobility. Mobility is an integral part of the wireless environment. With mobility arises the need for handoff of a connection across cells (Base Stations). This potentially causes disruption in the connection during the handoff. The delay associated with the handoff can be significantly reduced when an alternate communication path is always kept alive via a backup interface. In this case, while the original interface performs handoff related processing, the connection can be shifted to the already active backup interface. At the end of the handover, the connection can be shifted back to the original interface. Through this approach, the only disruption the connection experiences is the time it takes to shift the connection from one interface to another which is usually much smaller than the total time needed to perform handoff on an interface.

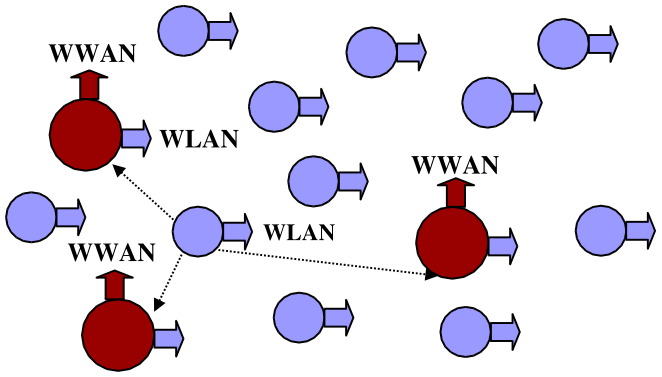


Figure 1.2: Resource Sharing: A Multi-Access Service

1.1.3 Reliability Support

A third multi-access service is the following. In environments that involve high loss rates and blackouts, providing reliability guarantees is in general difficult. However for applications that need high reliability, reliability can be provided by duplicating and/or encoding some or all packets and sending them on the multiple paths corresponding to different interfaces.

1.1.4 Resource Sharing

While the above multi-access scenarios involve a single client host, the idea can be extended to broader scenarios involving more than a single host.

Consider an ad hoc network formed by a group of devices in close proximity, using their local area interfaces (LAN) – such as 802.11 or Bluetooth. Often their means of Internet connectivity is through an access point (AP) with a high-speed wired connectivity. In places where there is no such wired public access points, the nodes can still connect to Internet if a subset of nodes have wide area wireless interfaces (WAN) such as UMTS, CDMA2000.

The above possibility is depicted in Fig. 1.2. Here, the wide area resources can be shared effectively across the nodes to access Internet. Each node uses other nodes as gateway routers to route its traffic over their wide area interfaces. This

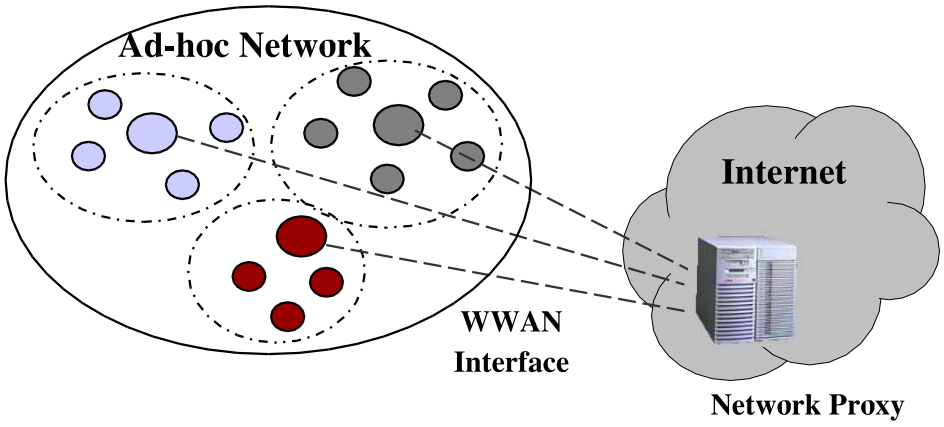


Figure 1.3: Data-Control Plane Separation: a Multi-Access Service

can be done on a per connection basis, where a node routes/gets all traffic of a new connection on the least loaded gateway. In case of demanding applications, it can also be done on a per packet basis, by striping traffic of a node on more than one gateway.

1.1.5 Data-Control Plane Separation

Another multi-access scenario involving more than one node, similar to the one above, is the following. In an ad hoc/sensor network formed by nodes using their local area interfaces, the wide area interfaces can also be used for *out of band* control communication to aid distributed ad hoc protocols such as routing (while the data is still sent on the ad-hoc LAN).

For instance, to assist distributed routing, a multi-homed node can collect local neighborhood information on its LAN interface and pass the information on its WAN interface to an infrastructure proxy on the Internet. The proxy can then calculate routes from the information collected from all such multi-homed nodes in the ad hoc network and pass the routing information back to the multi-homed nodes for local distribution. This brings about a clean separation between control and data plane, whereby the local network can now be mostly used for

data communication. Such an approach can significantly reduce the complexity of an ad-hoc routing protocol. This is depicted in Fig. 1.3.

1.2 Dissertation Goals and Assumptions

To realize in practice the multi-access services listed above, we need an architecture to support multiple communication paths. In this work, we begin by providing a general framework in the form of such an architecture.

Our main goal with respect to the architecture is that it enable diverse multi access services with *minimal changes* to the infrastructure for ease of deployment. This includes being transparent to application and transport protocols, backward compatibility with existing infrastructure. The existing infrastructure here includes the various application servers (e.g. video/HTTP servers) as well as the different wireless network components such as base-stations. It is important for reasons of deployment, to not require changes in the various application servers. A given solution should be usable for a wide range of already deployed servers. Similarly, our goal is also to not require changes to the already deployed wireless network components. Our only requirement is the overlap of areas of coverage, which is already present in many instances.

While there are many services enabled by simultaneous interface use, in this dissertation, we focus our attention on just the first multi-access service: **B**andwidth **Ag**gregation (BAG). In BAG, the application data is spread or striped across the various interfaces to achieve an effective higher bandwidth.

We explore in depth this particular service in the context of two broad classes of applications: (a) Video applications (both streaming and interactive video), and (b) TCP applications. We design and evaluate the performance of various algorithms for bandwidth aggregation for these scenarios.

We have some specific goals with respect to BAG services. First, the scheduling algorithm that stripes data onto the multiple interfaces needs to ensure

that all available bandwidth of the interfaces is properly utilized while minimizing the reordering of packets sent on the paths. Second, irrespective of the care taken when scheduling, some amount of reordering is inevitable due to fluctuating path characteristics. We need to ensure that these out of order packets are delivered in order. Third, in the presence of losses, it is often not possible to distinguish if a packet were lost or merely reordered. So, some amount of delay is visible at the application layer. We seek to minimize this delay to the best possible extent.

In the design of the various algorithms, we make some assumptions about the underlying infrastructure. The first assumption is with respect to the next generation Radio Access Networks. We assume that the Base Stations (BSs) are IP-based, and an extension of the Internet. This is a reasonable assumption as we expect future wireless networks to be more integrated with Internet technologies and Internet-based data services.

Our second assumption in the design of our algorithms is that the last-hop wireless link is the bottleneck. This is reasonable to assume given the current trend of growth in wireless as well as wired network speeds.

In addition to the above, we make specific assumptions in the context of bandwidth aggregation for the two broad classes of applications. For the first class of real-time applications, it is very difficult to support these 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 these efforts, our assumption is that the BSs provide QoS in the form of a *dedicated* wireless channel, with capacity equal to the rate negotiated by the mobile client. Once the QoS is negotiated, the channel is retained for the whole session (no release/grant happens).

We make the above assumption only for real-time applications, and relax this for our consideration of TCP applications. Here we no longer assume the presence of dedicated channels. This is in line with the fact that most TCP applications today are best-effort. The lack of dedicated channels does introduce some

challenges in the design of our bandwidth aggregation algorithms for TCP applications. This is because of the possible presence of cross-traffic at the bottleneck wireless link (i.e., at the wireless base-station). In this context, we assume that the BSs use *Weighted Fair Queuing (WFQ)* [16] packet scheduling, where all flows through the BSs are given the same weight. We believe this to be a good choice as it permits equal sharing of the scarce wireless link capacity among all the flows.

We now detail the various challenges that need to be addressed to achieve effective bandwidth aggregation.

1.3 Challenges in Providing Bandwidth Aggregation Services

Bandwidth aggregation requires that the application data be striped across various network paths, to reach the mobile host via the different interfaces. The first challenge is that of placement of this *functionality of striping*. This arises because of our goal of requiring minimal changes to the infrastructure. We can neither impose changes at the server, nor at the base-stations. The former means modifying all server software (hence deployment problems) and the latter is not technically feasible given we might employ different technologies spanning different administrative domains.

The second main challenge to bandwidth aggregation is the following. While the use of multiple interfaces can increase available raw bandwidth, the use of multiple paths introduces new problems. Each path along the different interfaces may have varying characteristics in terms of bandwidth and latency, and especially so in the presence of heterogeneous technologies. This can mean potential *packet reordering* when application data is sent striped across these various interfaces.

The above two observations – requirement of no changes at the server or at the base-stations, and packet reordering – are depicted in Fig. 1.4. These result in different challenges for the two classes of applications under consideration.

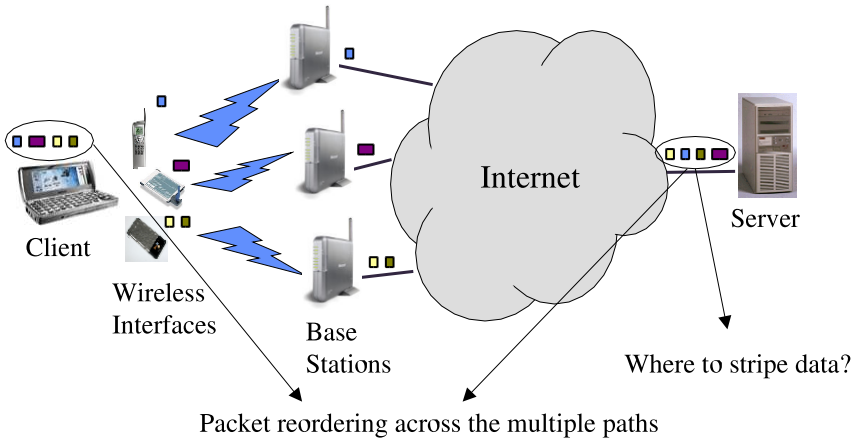


Figure 1.4: Challenges in Bandwidth Aggregation: Placement of Striping Functionality, and Packet Reordering

First, with respect to real-time applications, they have very stringent Quality of Service (QoS) requirements. For example, interactive applications like video telephony, video conferencing need one way latency under 150ms for excellent quality of service and under 400ms for acceptable quality. For these applications, out of order packets may have to be buffered, and this can result in additional delay. This additional delay is often equivalent to packet loss due to missed playback deadline. Streaming applications that employ smoothing buffers can tolerate this reordering to an extent. However, for interactive applications, proper care needs to be taken when striping data onto the multiple paths to minimize the excess delay caused by reordering,

Next, with respect to TCP applications, since we mandate that no changes should be made at the server, it is not possible that the application splits its data across multiple TCP connections, each established on a different interface. (We term such a multiple-TCP solution as MTCP – we shall revisit this later). Hence we have to deal with packet reordering which may occur within a single TCP connection. Now, packet reordering can significantly degrade TCP performance due to several reasons:

- For every reordered packet, a TCP receiver generates a duplicate ACK (DUP-

ACK). On receiving more than 3 DUP-ACKs, the TCP sender considers the packet lost and enters fast retransmit and resends the packet that was only delayed (on one of the interfaces) – this wastes scarce bandwidth.

- The TCP sender also assumes loss as indicative of network congestion and reduces its sending rate by cutting down the congestion window by half.
- Depending on the particular TCP implementation, reordering can also generate bursts of packets. If the TCP sender is not allowed to send packets in response to DUP-ACKs, when a new ACK covering new data arrives, it produces a burst[1].
- Reordering can also affect the calculation of round-trip time (RTT) estimation and hence retransmission timeout (RTO) as for every packet that is needlessly retransmitted, the RTT sample is ambiguous and cannot be used.

Minimizing reordering requires monitoring of path characteristics which is not always easy, given the dynamic nature of Internet paths. And further, bandwidth aggregation for TCP can be especially challenging since we assume no dedicated channel (only best-effort) at the bottleneck wireless base-stations. Further, we also work under the premise that there is no special support from the base-stations in terms of providing any network-related information or such.

1.4 Dissertation Methodology and Contributions

In this dissertation, we design, implement and evaluate solutions to the above posed challenges and defined goals. We begin with an understanding of the components needed to enable the diverse multi-access services and design an architecture based on it (Contribution - I). We then focus on one of the multi-access services: Bandwidth Aggregation (BAG) that increases end user bandwidth by aggregating bandwidth from all active interfaces. We look at this service in the

[1]If the TCP implementation uses max-burst factor as outlined in [19], burst sizes can be reduced.

context of real-time applications and design appropriate algorithms and protocols that satisfy their QoS needs (Contribution - II). Apart from real-time applications, we also evaluate BAG services in the context of the most predominant transport protocol, the Transmission Control Protocol (TCP). We identify a set of design criteria that will help improve overall TCP performance in the presence of reordering. We then propose appropriate algorithms based on these design principles (Contribution - III).

We now describe our dissertation contributions in each of the three parts mentioned above.

1.4.1 An Architecture for Multi-Access Services

We seek an architecture to support the various multi-access services described earlier. The first design question to be answered is: at which layer of the protocol stack should the architecture be addressed? We choose a network layer architecture as opposed to transport/application layer solutions in line with our design goals. Application/Transport layer solutions are cumbersome to implement and involve many changes to the infrastructure – all application and/or server software has to be changed. On the other hand, network layer solutions, have the advantage of being totally transparent to applications and involve only minimal changes, thus making their deployment easy. Legacy applications in particular can benefit from this approach.

Our network layer architecture consists of an infrastructure proxy. A proxy provides multi-access services to a set of mobile clients equipped with multiple interfaces, and multiple proxies may be provisioned for reliability and scalability. This is depicted in Fig. 1.5. Some of the features of the network proxy are similar in spirit to that provided by Mobile IP [31]. The client acquires a fixed IP address from the proxy and uses it in establishing connections with the remote server. The proxy (like the Home-Agent in Mobile-IP) captures packets destined for the client. The proxy is aware of the multiple interfaces of the client,

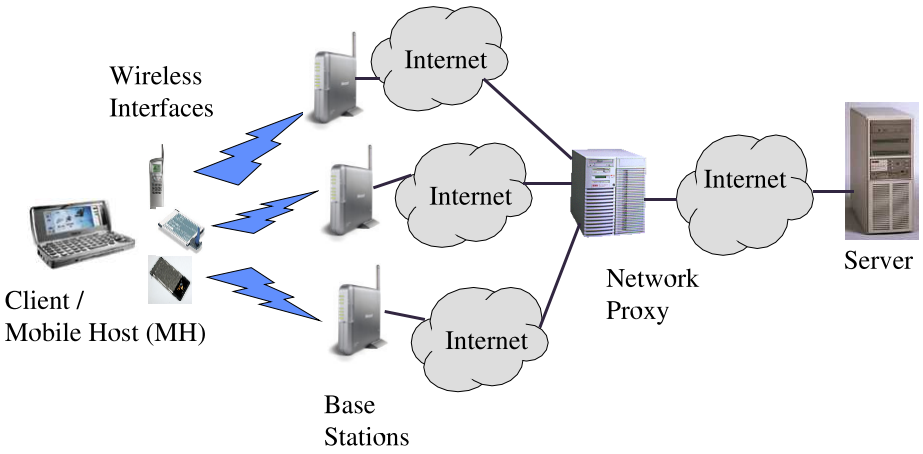


Figure 1.5: A Network Layer Architecture for Supporting Multi-Access Services

and tunnels the captured packets to the client using IP-in-IP encapsulation. Unlike Mobile IP, the proxy can manage multiple care-of-addresses and perform intelligent scheduling of (tunneled) packets depending on the service provided.

We have identified the various functional components needed to provide the different multi-access services. These functional components, depicted in Fig. 1.6, reside at the mobile client and/or at the network proxy. The main components include (a) *Access Discovery* and *Access Selection* to discover and bring up necessary interfaces at the mobile client, (b) *Profile Server* to specify how to handle different application flows of the client, (c) *Mobility Manager/Server* to handle client mobility, (d) *Performance Monitoring Unit* to monitor the path characteristics of the different interfaces, and finally (e) the *Traffic Manager* to perform intelligent processing of client traffic.

We have implemented most of these components as Linux loadable kernel modules. Of these components, the *Traffic Manager* is the one where the important algorithmic functionalities of data striping reside. The design of the algorithms in this components is central to achieving effective bandwidth aggregation for video as well as TCP applications.

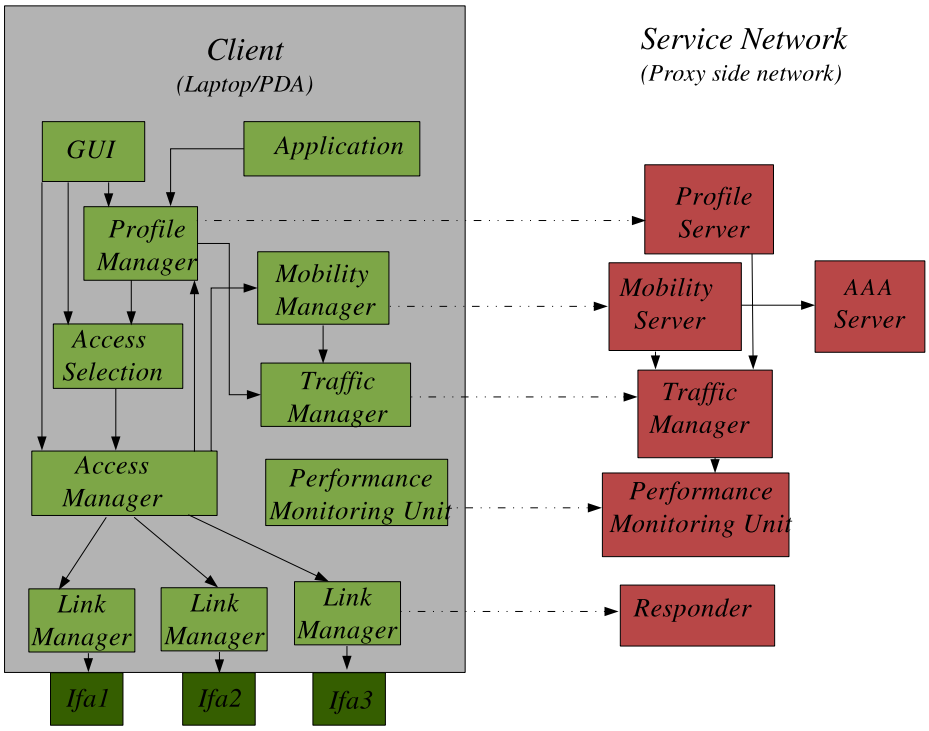


Figure 1.6: Functional Components of the Architecture

1.4.2 Bandwidth Aggregation for Video Applications

One of the services provided by the architecture is that of aggregating bandwidth available on the multiple interfaces to increase application throughput. We have explored in depth this particular service in the context of real-time applications. A key ingredient that dictates the effectiveness of this service is the scheduling algorithm that resides within the *Traffic Manager* 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.

In this context, we propose the *Earliest Delivery Path First (EDPF)* scheduling 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 (a) a *theoretical analysis*, (b) a *simulation* based study, as well as (c) an *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 *trace-driven simulations* as well as *a prototype implementation*. We do this for both real-time streaming and interactive applications. Our results show that EDPF mimics ASL closely and outperforms round-robin based approaches [7] by a large margin.

Apart from the scheduling algorithm, we also consider a content adaptation algorithm at the *Traffic manager* to selectively drop video frames when

adequate bandwidth cannot be reserved on the interfaces. We call this frame discard policy - *Min Cost Drop (MC-Drop).* In MC-Drop, when a frame arrives at the HA: (1) It is determined whether it should be forwarded (MC-Drop). (2) If so onto what interface (EDPF). We rely on a crucial aspect of video stream - Group of Pattern (GOP) correlation to decide whether to send or drop a frame. The decision to drop a frame is based on the impact the frame drop has on meeting future frame deadlines and hence on overall quality of the video. Frames with high priority are less likely to be dropped. Performance evaluations of MC-DROP through simulations show that there is significant improvement in performance when using MC-DROP, as compared to the scenario where no form of frame discard is employed. Especially when the reserved bandwidth is small, MC-Drop outperforms by a large margin other considered approaches.

1.4.3 Bandwidth Aggregation for TCP Applications

Apart from Real-Time applications, we also experiment with BAG services for TCP applications. Unlike for real-time applications, where we considered dedicated wireless channels, we now introduce cross traffic and deal with best effort channels. Accordingly, we propose a new scheduling algorithm based on EDPF - *PET (Packet-Pair based Earliest-Delivery-Path-First algorithm for TCP applications).* As with EDPF, PET minimizes reordering by estimating the delivery time of packets on each Internet path and scheduling packets on the path that delivers it the earliest. However, it obtains bandwidth estimates by sending packets in pairs [26] as far as possible and using their inter-arrival spacing for calculating the estimate.

Given the dynamic nature of Internet paths, some amount of reordering is inevitable and this reordering can have quite a detrimental effect on TCP. To get around this, we propose a client-side *Buffer Management Policy (BMP)* that tries to hide any residual reordering from TCP, so that unnecessary retransmissions are avoided. BMP buffers out of order packets at the network layer and tries to pass

them to TCP in order. It also attempts to detect losses and react to them in a timely fashion.

We study the performance of the proposed approaches through simulations under a variety of network conditions. PET in conjunction with BMP outperforms by a large margin naive schemes like weighted round robin (WRR) that don't attempt to minimize reordering. Also the performance of PET-BMP is close to an application layer bandwidth aggregation scheme MTCP, where multiple TCP connections are opened, one on each interface. Our network layer approach is effective in addressing the challenge of reordering, and is thus performance-effective in addition to being flexible.

1.5 Dissertation Outline

The rest of the dissertation is organized as follows. The next chapter (Chapter 2) discusses related work. We outline past research in areas related to architectures for providing multi-access services and bandwidth aggregation. In Chapter 3, we present the design of our architecture that enables multi-access services. We discuss the solution components that make up our architecture along with the implementation details.

The topic of the subsequent three chapters is the specific multi-access service: Bandwidth Aggregation (BAG). In Chapter 4, we look at the challenges faced in providing such a service. We explore BAG services in the context of real-time video applications in Chapter 5. We consider both streaming and interactive video applications. We study the performance of our proposed scheduling algorithm through analysis, implementation as well as extensive simulations. Next, Chapter 6 looks into BAG services in the context of TCP applications. We identify a set of design criteria for improving the performance of TCP and evaluate the usefulness of our proposed scheduling and buffer management algorithms through simulations.

Finally, we present concluding discussions and avenues for further interesting exploration in Chapter 7.

Chapter 2

Related Work

Research related to our work falls into two broad categories:

- Architecture for Multi-Access services, and
- Bandwidth Aggregation.

Section 2.1 presents related work in the first category. While, in Section 2.2, we present various solutions to bandwidth aggregation addressed at different layers of the protocol stack.

2.1 Architecture for Multi-Access Services

Mobile IP [31] is a network layer protocol standardized by IETF to support mobility in IP networks. When a client changes its physical point of attachment (care-of address) due to mobility, unless this change of IP address is hidden from the transport software, the end-to-end connections can get disrupted. To overcome this problem, the mobile host (MH) maintains a permanent IP address (home address) independent of its physical point of attachment (care-of address) by registering the care-of address with an agent at its home network (home agent). The home agent (HA) intercepts the packets destined for the mobile and tunnels them to the mobile's current location. One of the features of Mobile IP is *Simultaneous Binding*, where a mobile node can register more than one care-of address

at the HA. In this case, the HA duplicates the datagrams and tunnels them to each care-of address. This feature was introduced to improve wireless connectivity during handoff.

Our architecture relies on a similar principle of tunneling as in Mobile IP, albeit in a different context - to hide the presence of multiple interfaces and present an abstraction of a single interface to the transport software. This mechanism is needed in our architecture even when the client is stationary – for simultaneous use of interfaces in addition to mobility support. Apart from this similarity, our architecture differs from Mobile IP on many grounds including mobility management. Our architecture can support diverse multi-access services and perform intelligent processing and scheduling of packets across multiple interfaces. Our architecture can also support mobility on any number of interfaces the client is equipped with, unlike Mobile IP which can handle mobility on only one interface at a time.

Contemporary to our initial work [15], where we introduced the concept of network layer bandwidth aggregation for real-time applications, the authors in [32], propose a network layer architecture similar to ours based on the same concept of tunneling. Their architecture however looks only at the specific multi-access service Bandwidth Aggregation in the context of TCP applications. This work deals with the architecture at a very high level and does not identify or address the various functional components needed to enable multi-access services. It also does not look into implementation details of the components.

2.2 Bandwidth Aggregation (BAG) Services

Bandwidth Aggregation has been addressed at different layers of the protocol stack: link, network, transport and application layer. We present related work at each of these layers. We also present some miscellaneous work that covers aspects of some of the problems we addressed: selective frame discard and packet reordering.

2.2.1 Link Layer Solutions

Bandwidth aggregation across multiple channels has its origins as a link layer solution in the context of analog dial-ups, ISDN, and ATM. Bonding [17] is a standard inverse multiplexing protocol that uses special hardware at the sender and receiver to combine together 56kbps and 64kbps circuit-switched lines. IMA [4] is a similar approach used in ATM networks that uses special hardware to aggregate bandwidth across multiple point-to-point links. Multilink PPP [35] is another approach used in a wireless setting to bundle multiple data channels into a single logical channel. It uses a special header and a fragmentation protocol for reassembly and in order delivery of fragments. All the above approaches require identical, stable link characteristics, special hardware and/or access to the endpoints of the links or specialized headers. This makes it difficult or infeasible to use them in the present scenario, where the RANs in question belong to different domains controlled by different providers.

The Stripe protocol [7] 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. Moreover the algorithm is not particularly robust against high packet loss rates or fluctuating channel capacities because of the need to synchronize both ends of the multiple paths. For real-time interactive applications, the additional delay introduced by the algorithm often results in packet loss due to missed playback deadlines. The inability of the algorithm to cope with non FIFO delivery and fluctuating channel capacities makes it unsuitable for TCP applications. In later chapters, we do compare our proposed solutions to SRR and variants of it.

2.2.2 Network Layer Solutions

As was mentioned earlier, the closest that comes to our solution is a network layer solution for bandwidth aggregations proposed in [32]. This work has looked at TCP applications with Weighted Round Robin (WRR) scheduling at the network proxy. No buffer management policy to hide reordering was considered. As will be demonstrated later in the sections, WRR performs poorly if the scheduling does not track fluctuating channel capacities. And the resulting reordering if not hidden from TCP can have quite a detrimental effect nullifying any benefits that can be had through bandwidth aggregation. To overcome reordering, some suggestions to tune TCP parameters like permit large window sizes, set high values for retransmission timeouts and avoid fast retransmissions on duplicate ACKs were also suggested. However, their work did not consider losses, and these suggestions, when implemented, will perform very poorly in presence of losses.

2.2.3 Transport Layer Solutions

The Reliable Multiplexing Transport Protocol (RMTP) [30] is a reliable rate-based transport protocol that multiplexes application data onto different channels. It performs bandwidth estimation over the multiple channels and uses this information at the transport layer to perform flow and congestion control. Reliability is provided by way of selective acknowledgments. Parallel TCP (pTCP) [24] is another transport layer approach that opens multiple TCP connections one for each interface in use. pTCP manages the send buffer across the different connections and performs congestion window based scheduling of data onto the multiple connections. Further it handles congestion and blackout by data reallocation and redundant striping. Another reliable transport protocol proposed for use in message-based signaling in IP networks is the Stream Control Transmission Protocol (SCTP) [38]. SCTP supports multi-streaming and multi-homing. However, it does not ensure in-order delivery across data streams which then has to be handled at the application layer.

The focus of this paper is on an architecture that introduces minimum changes to the infrastructure while enabling many diverse services. Transport layer solutions, while can be efficient, involve many changes to the existing infrastructure – all server software has to be changed and don't quite fit within our desired goals. Further the proposed solutions, cannot support real-time applications which may not employ TCP as the transport protocol because of delay constraints. With respect to TCP applications, these solutions can be TCP unfriendly when the multiple TCP connections they open share a common bottleneck. Additionally, unless mobility support is integrated within, these approaches may have to rely on a solution similar to ours for mobility support.

2.2.4 Application Layer Solutions

Some application layer approaches to bandwidth aggregation using multiple TCP connections have been proposed albeit in a different context, all the TCP connections are over the same path. In [36], a new application XFTP is proposed that opens multiple TCP connections to simulate a large virtual TCP receive window to overcome TCP's limited throughputs over satellite circuits. Similarly, in [34], the authors develop a Parallel Sockets (PSockets) library, that stripes application data over several open sockets, to achieve low latency and improved bandwidth, without having to manually tune the TCP buffer size. An extension to FTP protocol called GridFTP was proposed in [28] for bulk data transfers, where multiple TCP connections are opened to get a larger share of the bottleneck bandwidth.

As with transport layer solutions, these solutions involve many changes to the infrastructure. All applications that wish to use multiple interfaces simultaneously have to be rewritten, while ensuring backward compatibility.

2.2.5 Miscellaneous Work

There are two other topics that are of interest to us and which we address in our work: 1) Selective frame discard for video applications when adequate bandwidth cannot be reserved irrespective of bandwidth aggregations, and 2) Hiding residual packet reordering from TCP.

Selective Frame Discard

Frame discard based on time stamps, or priority information of the frames have been considered earlier [22, 23]. Most schemes in this domain either time stamp the frames, whereby intermediate routers drop frames that are unable to meet their deadlines or drop low priority frames in the event of network congestion (increased queue sizes). Our network-layer architecture was designed with the objective of introducing minimal changes to the infrastructure for ease of deployment. Among past work that has considered frame discard algorithms, most schemes either do not fit in with our architecture (they need time stamping, clock synchronization, additional functionality at intermediate routers etc) or do not perform well in our particular setup of multiple interfaces (as will be shown in later chapters).

In [40], the authors propose a selective frame discard algorithm at the video server for stored video applications. The algorithm attempts to discard frames by minimizing a QoS based cost function that captures video quality. However the scheme assumes full knowledge of the video characteristics (frame sizes). While this approach works for stored video applications, it does not fit in our setup when dealing with interactive video as video frames are generated on the fly – no prior knowledge of video characteristics possible.

Hiding Packet Reordering

Some modifications to the TCP sender have been proposed in [11, 39] to cope with reordering that can be caused by one of many reasons - high-speed

switches, satellite links, differentiated services etc including multi-path. In this work, the TCP sender is extended to detect unnecessary retransmissions due to packet reordering through the use of duplicate selective acknowledgment (DSACK). DSACK reports to the sender any duplicate packets received permitting the sender to undo any effects (reduction in congestion window) the spurious retransmission had on congestion control state. This work does not look into bandwidth aggregation, only how to cope with reordering. Thus, our scheduling algorithm PET can also be used in conjunction with this TCP modification, eliminating the need for BMP (Buffer Management Policy). Though this warrants further study, we believe BMP is a better approach as it a pro-active approach in that it prevents the need for retransmissions (hence wasting scarce bandwidth) in the first place as opposed to taking a corrective measure. In fact BMP when integrated into the TCP receiver can be viewed as a receiver side modification to TCP to make it robust against reordering.

2.3 Summary

This chapter outlines the background material in architecture and algorithm support to enable multi-access services. We observe that the current approaches either do not fit well or solve comprehensively our desired goals of efficiently and transparently enabling multi-access services. This sets the stage for our work. In the next chapter, we present our architecture that enables diverse multi-access services. This forms the basis to discuss in detail one such service - Bandwidth Aggregation presented subsequently.

Chapter 3

An Architecture for Multi-Access Services

To enable the various multi-access services discussed in Chapter 1, we need a supporting architecture. The design of such an architecture is the topic of this chapter. We first motivate our choice of a network layer architecture in Sec. 3.1. We then proceed to discuss the design of our architecture in Sec. 3.2. Subsequently, in Section 3.3, we provide the description of the functional components that make up our architecture. In Section 3.4, we touch upon some of the implementation details. Finally, in Sec. 3.5, we illustrate with examples, the interaction of the various architectural components to enable the different multi-access services.

3.1 Why a Network Layer Architecture?

The architecture to enable multi-access services can potentially be addressed at different layers of the protocol stack. We discussed some link layer solutions in Sec. 2.2, as proposed in [17, 35, 4]. These require identical, stable link characteristics, special hardware and/or access to the endpoints of the links or specialized headers. This makes them difficult or infeasible to use them in the scenario of our interest, where the wireless networks differ in technology, span different Internet domains, and are 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 optimizations 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 application software infrastructure. This would make widespread deployment a difficult job. Further, the applications need to keep track of the state of different interfaces, which increases their complexity. And when multiple application flows at the mobile share common client resources (interfaces), they have to be designed carefully to avoid negative interaction.

Transport layer solutions as in [30, 24] that open multiple connections and perform flow and congestion control 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. They may also require cooperation during standardization to prevent negative interaction among the different flows using the different transport protocols when sharing common client interfaces.

With IP as a unifying standard, a network layer approach has the advantages of being transparent to applications and transport protocols. It does not need any changes to existing server software. Our choice of a network layer solution mainly stems from its ease of deployment, and meets out goal of requiring minimal changes to existing infrastructure. 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 negative interaction among flows.

While the network layer approach overcomes most limitations of the other approaches, it poses challenges in terms of efficiency, since it operates further down the stack. However, as we illustrate in this dissertation, with careful design, most inefficiencies can be minimized.

Another point worth noting is that our design choice as such does not preclude further optimization at the higher layers. Also, if application-specific

information is available, it can often be used at the network layer for optimizations. We in fact illustrate this in the case of MPEG video applications, where we use application-specific packet-priority information for selectively discarding packets at the network layer (Chapter 5). Further, our architecture can lend support to higher layer approaches in terms of mobility support. 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).

Given our overall choice of a network layer approach, we now proceed to discuss the details of the design of our architecture.

3.2 Design of the Architecture

One of the main goals of our architecture is to achieve application/transport layer transparency for ease of deployment. Many of the existing transport layer protocols do not support multi-homing and use just one IP address of the client to identify a connection, irrespective of the number of interfaces and hence IP addresses the client possesses. For example, TCP connections are indexed by a four tuple: the source (server) IP address, source port number, destination (client) IP address and destination port number. However, when the client wishes to use multiple interfaces simultaneously, the packets of the connection have to be addressed with different destination IP addresses (corresponding to each interface) to reach the client on the multiple interfaces.

Since the server software does not support this feature of multiple IP addresses, a way to handle this would be to use the services of an intermediate entity. We term this intermediate entity in the infrastructure that will provide this service as the *network proxy*. A static IP address provided by the proxy can be used by the client to establish connections with the server. The client traffic which then passes through the domain of the proxy can be intercepted by the proxy and

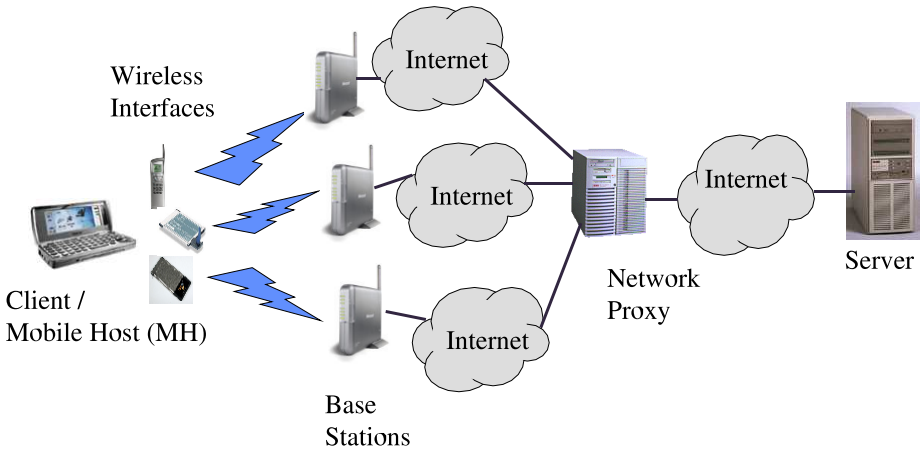


Figure 3.1: Architecture to Support Multiple Communication Paths

delivered to the client on the multiple interfaces.

Mobile IP [31] is an IETF standard for mobility support in IP networks. Mobile IP relies on a similar mechanism as described above, although in a different context – to handle client mobility. The problem in this case is not with multiple IP addresses of the client but the requirement that the client maintain a fixed IP address even when it is mobile and acquires new care-of IP addresses. Failure to maintain the fixed IP address results in disruption of the end-to-end connection. So, the solution in Mobile IP, is for the client to use a fixed IP address (Home Address) to establish connections with the server and register the care-of IP addresses it acquires during its movement at a node in the network (Home Agent). The Home Agent (HA) then intercepts the packets destined for the client and routes them to the client using a mechanism called *tunneling*. In tunneling, the packets received at the HA are encapsulated by a new IP header that contains the client's care-of-address as the destination address. Each time the client acquires a new care-of address, it registers the new address at the HA, so that the HA can tunnel packets to the correct destination.

Our architecture is based on the same mechanism of tunneling but has been extended to handle multiple interfaces. Fig. 3.1 shows a high level overview

of our network-proxy based architecture. The network proxy can provide many different multi-access services (bandwidth aggregation, mobility support, resource sharing etc) to a set of clients (equivalently, the MHs), which have multiple network interfaces. Multiple proxies can be provisioned for reliability and scalability.

The MH, when using the services of the network proxy, acquires a fixed IP address from it and uses it to establish connections 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, the proxy intercepts the packets and performs necessary multi-access-service-specific processing. It then tunnels them using IP-within-IP encapsulation to the client's different 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.

It is not always the case that the client needs the services of the proxy in managing its applications on multiple interfaces. We distinguish between two modes of operation for any application: (1) *Client Controlled (CC)* mode, where the client manages an application flow on its own, without any support from the network proxy. (2) *Network Controlled (NC)* mode, where the proxy helps the client in handling that particular application.

We also distinguish between two types of scheduling for an application, *per-flow* scheduling, and *per-packet* scheduling. In per-flow scheduling, each flow is essentially an end-to-end connection (TCP/UDP etc), where the flow is bound to the interface that best satisfies its requirements. All packets of the flow for the entire duration of the connection are sent/received only on that interface. In contrast, in per-packet scheduling, the packets of a single flow can be split across the various interfaces.

In the context of the above two kinds of scheduling, the CC mode has some limitations. First, CC mode cannot support per packet scheduling, only per-flow scheduling. Further, only flows that client initiates can use CC mode

and not flows that originate on the network side. However, CC mode is simple to implement and can handle some scenarios very well – like per flow bandwidth aggregation and resource sharing.

With the above overall architecture in mind, we now look at the various functional components needed to enable various multi-access services.

3.3 Functional Components

The functional components that make up our architecture can be broken up into two main classes: those which reside on the MH, and those on the service network. The various components are shown in Fig. 3.2. In this section, we explain the functionality of each component. In the subsequent section, we touch upon some of the implementation details as well as the communication between the components.

3.3.1 Access Manager and Access Selection Unit

The first two components that come into play in the presence of multiple interfaces are the *Access Manager* and the *Access Selection* unit. The role of the Access Manager is to continuously monitor the interfaces of the client for connectivity and bring up/down the interfaces as dictated by the *Access Selection* unit. The Access Manager also passes the collected state information of the interfaces to the Access Selection unit to help it choose appropriate interfaces and to the *Mobility Manager* to establish new tunnels and initiate mobility related handoff processing.

The Access Manager manges its tasks by talking with one or more *Link Managers*, which handle individual interfaces. The link manager gathers connectivity information as provided by the interfaces. In the event, the interface does not provide this information, the Link Manager can send probe messages using UDP every few seconds to a *Responder* node in the network. If a reply is ob-

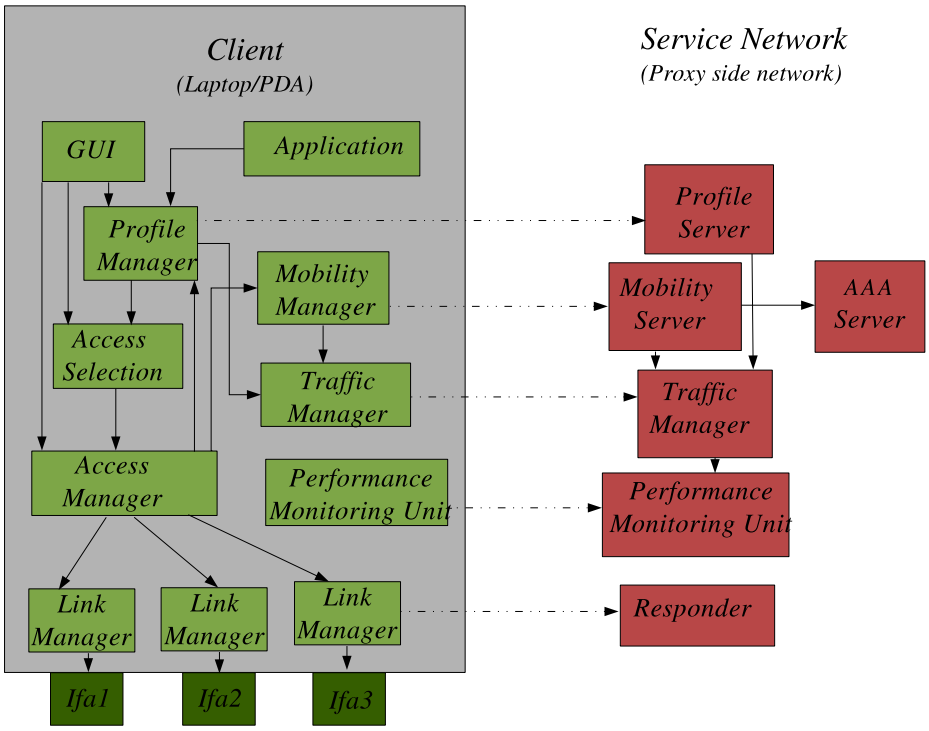


Figure 3.2: Functional Components of the Architecture

tained from the responder node, the connection is assumed up. Apart from this, the Link Manager also manages locally detectable state - radio link present, cable unplugged etc. The Link Manager activates/deactivates interfaces (add/remove routes), depending on the signaling it receives from the Access Manager.

The component which interacts closely with the Access Manager is the Access Selection unit. The Access Selection module is in charge of selecting the right interfaces based on input from the *Profile Manager*, and end-user through the *GUI*. It also uses other information it collects/stores such as the battery state, cost of access for that wireless interface etc. At startup, the Access Selection activates interfaces according to a pre-configured set of preferences. At run time, it can use an algorithm to pick the right interfaces based on some suitably defined metrics. In our implementation, the unit brings up interfaces (through appropriate commands to the Access Manager) based on input received from the GUI or based on some pre-defined preferences. The user can normally override any previous decisions taken by this unit. Apart from activating right interfaces, the Access Selection unit also informs the user through the GUI if it were not able to honor his request along with an appropriate reason.

3.3.2 Mobility Manager/Server

We have two components, one on the client side *(Mobility Manager)*, one on the proxy side *(Mobility Server)* which are responsible for establishing tunnels and handling client mobility. The mechanism to handle mobility is similar to that used in Mobile IP, except that we now extend it to handle multiple interfaces. An IP address obtained from the proxy can now be bound to multiple care-of addresses. Note that in our architecture, there is no concept of Home Network as defined in Mobile IP, as the client cannot do without the services of the proxy when using multiple interfaces, unless it operates in CC mode.

At startup, the Mobility Manager after receiving the interface state information (care-of address) from the Access Manager, sends registration request

for each active interface to the Mobility Server. The Mobility Server, after authenticating the request via a *AAA server*[1], allocates an IP address and establishes tunnels between the proxy and client with the respective care-of addresses as the destination. This binding of an IP address with different care-of addresses has a lifetime, as in Mobile IP; at the end of the lifetime, the particular association becomes invalid. The Mobility Server then passes on a registration reply that includes the static IP address to be used by the client in talking with application servers. The Mobility Manager at client, on receipt of this information, sets up the static IP address as the virtual IP address of the client so that all connections originated from the client will use this address as the source address.

Many Radio Access Networks (RANs) use a private IP address space, and drop packets with source IP not in their domain. Hence we establish bi-directional tunnels i.e. the Mobility Manager also tunnels packets up to the proxy. After establishing needed tunnels, the tunnel information is conveyed by the Mobility Manager/Server to the Traffic Manager so that it can schedule packets to the appropriate tunnels.

Handoff on an interface is detected by a link trigger or agent advertisements – the actual mechanism is dependent on the wireless network and is independent of our functional architecture. During such handoff, the Mobility Manager sends binding update information to the Mobility Server, which after authentication, reestablishes the tunnel for that particular interface with the new care-of address. Similarly any time an interface is activated/deactivated, a registration/deregistration request is sent to the Mobility Server to establish/tear-down the tunnel corresponding to that interface.

3.3.3 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. The

[1]Security is a very important concern to address in this architecture. Though we provide a means of authentication through a AAA server, this mechanism needs further study.

application requirements translate to a profile that specifies how exactly to treat its flows. The profile includes (a) the mode of operation (Client or Network Controlled), (b) the interfaces to use, (c) the type of scheduling to perform across the chosen interfaces (which algorithm), (d) granularity of scheduling (per packet or per flow), and (e) any additional functionality needed (duplication of packets for reliability, content adaptation etc). The Profile Manager that resides in the client, charts an application profile each time a new application is started based on end-user input or some predefined set of preferences. It then instructs the Access Selection to bring up any additional interfaces needed. On successful completion of binding the new interface IP address to the static IP address, the Profile Manager passes the application profile locally and via the Profile Server to the Traffic Managers on both ends, so that the Traffic Manger can handle the flows accordingly. The profile of an application can be updated when interface/path conditions change as conveyed by the Access Manager, Performance Monitoring Units on both ends. The Profile Manager/Server units periodically talk with each other to keep up to date the profiles of the client applications.

3.3.4 Traffic Manager

The *Traffic Manager* is the component which constitutes the core of multi-access 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. The processing could involve decapsulation (if packet reached end of the tunnel), content adaptation (e.g. video frame discards), duplication (for reliability), and/or more importantly, scheduling onto the "appropriate" tunnel interfaces. The *scheduling algorithm* to choose the tunnel interface can be different for different services, and is an important topic of

study in this dissertation.

3.3.5 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. This monitoring can be done in a passive or an active mode. In passive mode, the units monitor the incoming and outgoing traffic. In active mode, they can send periodic probes. The collected information is periodically exchanged between both ends so that the Traffic Manager can perform scheduling based on the underlying conditions of the paths. The Performance Monitoring Unit on the proxy side, can gather information from other sources as well – e.g. through service agreements between proxy network and the Radio Access Networks (RANs). This helps the proxy network to better manage the traffic of the many clients it serves by performing appropriate load-balancing across RANs preventing unnecessary congestion.

3.4 Implementation Details

We have implemented the various functional components needed by our architecture on a Linux platform – as different loadable kernel modules. We now highlight some of the special features of the implementation. Note that network side components need not reside on the same machine and can be distributed. But for convenience, we have implemented all except the AAA server on the same machine.

An important aspect of the implementation is gaining control of the packets as they traverse the kernel protocol stack. In our architecture, the modules (Traffic Manager or the Performance Monitoring Unit) trap the data packets through the *Netfilter* utility provided on Linux platforms. This utility provides

hooks at various points of the protocol stack. In this case, the modules register to listen at the pre-routing and post-routing hooks (depending on whether the packet is traveling up or down the protocol stack). When the netfilter hook is called from the core kernel code, control of the packet is passed to the module registered at that hook to manipulate the packet as needed. Note that the control of the packet can pass from one module to another (e.g. Performance Monitoring Unit to the Traffic Manager) before control is finally handed back to the kernel.

Another aspect of our implementation that comes into play in the CC mode of operation, is the need to bind a flow to a specific interface. We implement this mode only for TCP connections, where we intercept the *connect()* call the end-user application uses for establishing TCP connections. Our routine residing in the connect() procedure, passes control to the Traffic Manager module, which then determines the appropriate interface to bind the flow to, based on the performance statistics observed on the interfaces. Once a flow is bound to a specific interface, all packets of the flow get forwarded by the kernel on the same interface.

Communication between the components in our architecture is handled by using a client/server relation over a TCP connection. Normally both components if not residing in the same terminal have to authenticate themselves before establishing the connection. Figure 3.2 shows this relation, where the arrow points from the client to the server. The actual message exchange over the TCP connection can take one of three forms: ASCII, Network Layer Management (NLM), or Diameter Messages – as we describe below.

ASCII Messages

An ASCII message consists of a sequence of bytes terminated by a linefeed (\n) and is used for communication between components located on the same terminal. An ASCII message consists of number of fields separated by spaces. The first field is a single character, indicating message type (opcode). For example, the character “S” is used for conveying state information, the character “P” for

conveying profile information etc. The other fields are either numeric (e.g. number of packet transmitted), a state (+ for up, - for down, or ? for unknown) or a text (e.g. interface name - eth0).

Network Layer Management (NLM) Messages

The NLM messages are exchanged between the client and the proxy network. They consist of a fixed header (common to all messages) and an optional payload. The fixed header is 7 bytes long, consists of 2 byte message ID, 1 byte opcode for identifying the message type, 2 byte client ID, 2 byte payload length. Most messages have additional parameters carried in the payload.

Diameter Messages

The Diameter messages are mainly used for communication between the mobility Server and AAA server for authentication purposes. They follow the RFC standard as defined in [13].

An exception to the above communication exchange is the Performance Monitoring Unit, which does not communicate with other units. It rather, writes all statistics collected to a file using the /proc file system to be viewed by the user or polled periodically by other units (Traffic Manager/Profile handling units) that read the necessary statistics as needed for their use. In our implementation, the Performance Monitoring Unit after obtaining the control of the packet, records the flow id, packet arrival time and size. The information recorded is used to generate various statistics – average throughput of a flow, average jitter, available bandwidth of an interface, etc which then get written to a file.

3.5 Example Usages of the Architecture

The interaction of the architectural components and the overall architecture are better illustrated through example scenarios involving each of the different

multi-access services. Most of the services are similar in spirit to BAG. So we explain the example scenario involving BAG in detail and briefly explain how the other services work.

Bandwidth Aggregation

To illustrate how bandwidth aggregation may be achieved in practice, consider a demanding video application. When this application starts, it communicates its bandwidth and QoS requirements to the Profile Manager. The Profile Manager then generates an appropriate profile for this application and invokes the Access Selection unit to bring up the necessary interfaces. The Access Selection unit manages this tasks in conjunction with the Access Manager. If the interfaces are QoS enabled, it also negotiates the required bandwidth on these interfaces.

If any new interfaces are activated in this process, the Access Manager triggers the Mobility Manager to perform the needed registrations for tunnel establishment. The Mobility Manager manages this task in conjunction with the Mobility server after appropriate authentication from the AAA server. The Profile Manager, after ensuring that the interfaces can indeed support this particular application, passes the profile of the application locally to the client Traffic Manager. It also passes the profile remotely to the Profile Server, which in turn instructs the Traffic Manager on the proxy side to do the needed scheduling (we are assuming the traffic flow is towards the client).

The application traffic, when it passes through the Traffic Manager of the proxy, is striped by a scheduling algorithm that suits the specific video applications onto the tunnel interfaces. At the client, the receiving Traffic Manager, decapsulates the packets and passes them to the application (potentially after removing any reordering). During the duration of the connection, the Performance Monitoring units on both sides, collect statistics for use by the Profile Handlers and Traffic Managers.

Enhanced Mobility Support

Mobility support is enhanced in the above architecture as follows. Whenever a need for handoff is detected by a Link Manager on a specific interface, the information is conveyed through the Access Manager to the Mobility Manager which performs necessary processing. This includes the discovery of a new care-of IP address, registration and reestablishment of the tunnel with the new care-of address, etc. The Access Manager also informs the impending handoff to the Profile Manger, which then transfers the applications using that interface onto other active interfaces by updating their profiles and conveying the same to the Profile Server. The Traffic Manager on the proxy side, on instructions from the Profile Server would now stop scheduling the application traffic onto the interface undergoing handoff and tunnel it through the other active interfaces.

Reliability Support

Providing reliability support to applications that need high reliability guarantees is similar in operation to Bandwidth Aggregation, except that the Traffic Manger now duplicates or encodes the received packets of this application and passes them on the multiple tunnels.

Resource Sharing

Resource Sharing involves a group of nodes forming an ad-hoc network, where the wide area wireless resources of a subset of nodes (gateways) are shared by every node. In simplified terms, the whole ad-hoc network can be logically collapsed to a single client with its multiple interfaces representative of the wide-area interfaces. The nodes generating traffic can be viewed as different applications generating traffic in the client. Resource Sharing then boils down to plain Bandwidth Aggregation.

However there is an additional important aspect that needs to be considered here. This is the communication across nodes on the local area network. We

now illustrate this operation. We assume the Client Controlled mode of operation in our illustration – the Network Controlled mode is similar in spirit.

Whenever a node starts a new connection, the Traffic Manager residing within binds that connection to a particular gateway (hence wide area interface) i.e. all packets of this connection are addressed on the local network to the same gateway node, which then routes them on its wide area interface onto the Internet. The decision of which gateway to select, depends on the load levels on the wide-area interfaces. This information is periodically broadcast in the network and recorded by the Performance Monitoring Units of the nodes for use by the Traffic Managers.

A slight variation in this ad-hoc mode of operation as opposed to a CC mode of operation by a single client is that, the source IP address used by the connection corresponds to the local area interface of the node and not the IP address of the gateway wide area interface. The gateways, in this case provide NAT (Network Address Translation) functionality [18]. This mechanism prevents collision of flow ids when different nodes use the same gateway to talk with the same destination server using the same port numbers.

Data Control Plane Separation

The final multi-access service in our series of examples, Data-Control-Plane separation, differs significantly from others which involve some form of load balancing across multiple interfaces. The setup for this service is similar to that for Resource Sharing, in that we have an ad-hoc network with some nodes equipped with an additional wide area interface. However, in this scenario, we use the services of the proxy to assist distributed ad-hoc protocols such as routing. We now illustrate how data control plane separation can be achieved to assist routing.

When a node with dual interfaces wishes to join the ad-hoc network, it acquires the ad-hoc group ID along with necessary security tokens after proper authentication of itself with the group. The Mobility Manager of the node now uses this information to register its wide-area interface with the Mobility Server on

the proxy network. Note that the Mobility Server in this case need not establish any tunnels as there is no need for transparency. The Mobility server just needs to pass the IP address of the wide area interface onto the Traffic Manager for scheduling.

During normal operation, the Traffic Manager of the node gathers routing (control) information of the neighborhood from its routing module. It then passes this information to the Traffic Manager on the proxy side. The Traffic Manager on proxy side, after processing the information received from all dual interface equipped nodes, generates a global picture of the ad-hoc network, calculates appropriate routes and passes this information on to all registered wide area IP addresses.

The Traffic Manager on the client on receipt of this information, hands it over to routing module for appropriate distribution in the local neighborhood. This achieves a clean separation between control and data plane and the local network can now mostly be used for data transfer.

Note that the above mechanism is not specific to any routing protocol, and is only an enabling mechanism for a simpler or more efficient routing protocol. Also, if in the above mechanism, the network proxy were to fail or become unreachable, we can fall-back on a regular ad-hoc routing protocol which operates without data-control-plane separation.

3.6 Summary

In this chapter, we have presented our architecture for enabling multi-access services. Our architecture is based at the network layer and provides many different multi-access services – bandwidth aggregation, reliability support, resource sharing, data-control plane separation to the end MH. Further, it is transparent to applications and involves minimum changes to the infrastructure. The only changes needed are at the MH and the deployment of proxies, no changes are

needed in the radio network or in server software.

The design of our architecture is based on similar mechanisms as used in Mobile IP but have been extended to handle multiple interfaces. We identify the various functional components needed in the proxy network and the MH to enable the different services. We have illustrated the interaction of the various components through example scenarios. We now turn our attention to a detailed design and analysis of algorithms needed to deploy the Bandwidth Aggregation multi-access service on top of this architecture.

Chapter 4

Bandwidth Aggregation

One of the services provided by the architecture is that of bandwidth aggregation (BAG). This service is the focus of this chapter as well as the following two chapters. Bandwidth Aggregation attempts to utilize fully the available bandwidth of the interfaces to increase the throughput of demanding applications. It essentially tries to achieve the following – if we can obtain, say a bandwidth of 200kbps and 100kbps from two interfaces, can we aggregate bandwidth and obtain in total a bandwidth of 300kbps?

In this chapter, we lay out the challenges faced when providing BAG services. Based on these challenges, we summarize the goals in the design of the algorithms needed to provide this service.

Challenges in Performing Bandwidth Aggregation

Fundamental to bandwidth aggregation is the scheduling algorithm that resides in the Traffic Manger. This scheduling algorithm splits the client's traffic onto the different paths corresponding to different interfaces. There are two things the scheduling algorithm needs to consider when striping traffic:

1. In what ratio to split the traffic onto the interfaces, and
2. In which order to send packets on the interfaces

Lack of proper care when striping can result in a lot worse performance, nullifying any advantages BAG can offer. This is best illustrated by a simple example. Consider two interfaces, each offering 40kbps (ifa 1) and 10kbps (ifa 2) bandwidth. Consider the total download time of 10 packets, each of size 1000 bytes using the interfaces. Ignoring delay on interfaces, it is easy to see that a scheduling algorithm that distributes the traffic in proportion to the bandwidth ratios i.e. eight packets on ifa 1 and two packets on ifa 2, will finish the download in 1.6 sec. However an improper scheduling, that mistakes the bandwidth ratios to be say 1:1 and thereby striping 5 packets each on the interfaces will finish the download in 4 sec.

This 4 sec is in fact more than what could have been achieved if all the packets were sent on just the highest bandwidth interface, i.e. it would have taken just 2 sec to download the packets on ifa 1 alone. So, when scheduling traffic, it is very important to take into consideration the underlying characteristics of the interfaces, otherwise we may do more harm through bandwidth aggregation than without.

However, this monitoring of path characteristics is not always easy, given the dynamic nature of the Internet paths between the proxy and the client. A packet on a path can potentially experience queuing delays at intermediate routers, experience different transmission rates on the wireless hop due to channel conditions, may be retransmitted repeatedly if lost, all of these adding elements of randomness in the overall delay experienced.

Another important aspect when scheduling is the order in which to send packets. Consider the same example as earlier, sending first 8 packets on ifa 1 and next 2 packets on ifa 2, does achieve good bandwidth aggregation, but this can cause packet reordering. In this example, packet 9 will arrive much ahead of packets 5,6,7,8. While this may be fine for certain applications, this reordering can have quite a detrimental effect on some applications. For real-time applications, the reordering often means, some packets that should have arrived earlier got

unnecessarily delayed. This delay may often result in the packets being useless due to missed playback deadlines.

For TCP applications, the reordering can significantly lower throughput because the TCP sender may misconstrue the duplicate acknowledgments generated by reordered packets as indicative of packet loss (and hence congestion) and cut down the sending rate. Further reordering often results in retransmission of these reordered packets which weren't actually lost, resulting in wastage of scarce wireless bandwidth. Other problems include, TCP sender generating bursts of packets, ambiguous round-trip time (RTT) samples and hence retransmission timeouts.

With respect to TCP applications, buffering out of order packets and passing them in order to TCP receiver can help. However, in the presence of losses, this is not so straight forward as it is difficult to distinguish if a packet were merely reordered or lost. An indefinite wait for the next expected sequence number comes would eventually trigger a retransmission timeout at the TCP sender for each packet loss resulting in very low throughputs.

Summary

In summary, the scheduling algorithm plays a vital part in providing BAG services. Any inefficiency on its part, can result in a lot worse performance with bandwidth aggregation than without. So, it is very important that the scheduling algorithm carefully monitor the characteristics of the underlying paths so as to make avail of all available bandwidth. And then, while striping, it ensure that reordering is minimized.

Since some amount of reordering is inevitable, care must be taken to hide the reordering from higher layers through a buffer management policy. When buffering out of order packets, it is important to detect losses and react to them in a timely fashion to avoid unnecessary delays.

With the above design factors in mind, we next look at how we achieve

the desired goals in providing BAG services to two class of applications: Video and TCP applications.

Chapter 5

Bandwidth Aggregation for Video Applications

When providing BAG services for real-time video applications, a crucial aspect that dictates video performance is the scheduling algorithm that resides in Traffic Manager which splits traffic across the different paths. We first present the design of this algorithm (Sec. 5.1), along with some useful properties (Sec. 5.2). In subsequent sections, we explain how this algorithm fits in practical scenarios involving streaming and interactive video. In particular, we experiment with streaming video on a prototype implementation and show how BAG services can enhance end user experience over using just a single interface (Sec. 5.3). We also experiment with interactive video on an appropriate simulation setup, and show the performance improvement our scheduling algorithm offers over other scheduling approaches based on weighted round robin (Sec. 5.4). Lastly, we consider different frame discard algorithms to cope with the case of constrained bandwidth in spite of performing bandwidth aggregation (Sec. 5.5).

5.1 The Scheduling Algorithm

For real-time applications, the scheduling algorithm has two objectives - 1) To effectively aggregate bandwidth of the interfaces, 2) Minimize delay expe-

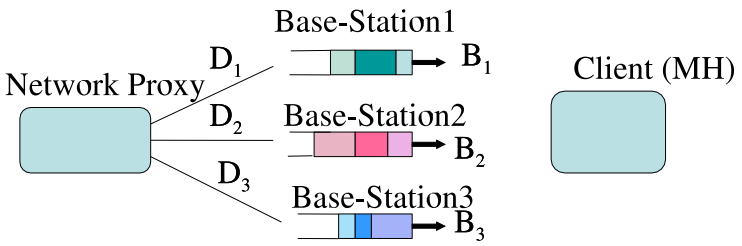


Figure 5.1: A Simplified View of the Network between Proxy and MH

rienced by packets due to potential reordering caused by varying characteristics (delay, bandwidth, loss) of the multiple paths. We now present an idealized form of such a scheduling algorithm called Earliest Delivery Path First Scheduling algorithm that achieves our desired objectives.

5.1.1 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. 5.1. 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 i^{th} 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.

$$d_i^l = MAX(a_i + D_l, A_l) + L_i/B_l \tag{5.1}$$

[1]The MH 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.

The first component computes the time at which transmission can begin at the BS, and the second component 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 MH acts as the server.

While the above discussion has assumed a single data flow into the MH (from a single server), the case of multiple data flows is easily handled. When we have multiple applications running on the MH, the scheduling algorithm at the proxy needs to partition the traffic from multiple input queues (corresponding to each application) onto multiple output paths. We can achieve this by combining a fair queuing algorithm like Weighted Fair Queuing (WFQ) [16] (which partitions traffic from multiple input queues onto a single output link) with EDPF. When a packet arrives at the proxy, it is determined as to which application it belongs, and after calculating its departure time (based on WFQ), it is placed in the application's queue. Each time any of the wireless channels (say i) become idle (since we are scheduling at the proxy, the time instance at which we schedule precedes the time when the channel actually becomes idle, by the wireline delay associated with that path), the packet with the minimum departure time is selected (WFQ) and is scheduled on the channel (say j) that delivers it at the earliest (EDPF). Note that j need not be the same as i. This process of scheduling is repeated till a packet is scheduled on the idle channel i. It is possible that packets may not be scheduled on the idle channel if there are not enough packets on the input queues.

In such a case, we erase (undo) the scheduling done previously as it is possible for packets arriving later to depart before the packets presently in the application queues due to high priority. In the remainder of the paper, we consider only the case of a single application.

5.2 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 Aggregated Single Link (ASL) case, where a single interface with the same aggregated bandwidth is used in place of multiple interfaces. 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 $hb = \{l : B_l \geq B_m, 1 \leq m \leq N\}$ and $lb = \{l : B_l \leq B_m, 1 \leq m \leq N\}$ 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.

For simplicity of analysis, we assume that the wireline delay D_l experienced by the packets is 0. In general, the wireline delay is time varying, however if this quantity is upper bounded by some constant, the results can easily be extended. 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 MH, $T_l(a_i^+) = d_i$, 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 MH (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* $T_m(t) - T_n(t) \leq L_{max}/B_n$.

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_m's 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 $T's$ do not change. Hence we have:

$$T_l(a_i^+) \leq T_m(a_i^+) + L_i/B_m \tag{5.2}$$

We now consider the following two cases,

Case1: $T_l(a_i^+) > T_m(a_i^+)$. According to 5.2, $T_l(a_i^+) - T_m(a_i^+) \leq L_i/B_m$.

Case2: $T_l(a_i^+) \leq T_m(a_i^+)$. 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 T's 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 MH. 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) [7], 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 T's 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. For one, packets no longer arrive in order due to multiple paths. Two, 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.

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

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

Proof. See Appendix A □

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 □

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. □

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

Proof. See Appendix A. □

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.

5.3 Streaming Video

In this section, we present a prototype implementation of our architecture as a proof of concept for BAG services. Specifically, we experiment with streaming applications to quantify the performance improvement BAG services bring over

conventional single interface use. We show that BAG can help streaming applications by significantly reducing the buffering time needed to ensure continuous playback, thereby enhancing end-user experience.

5.3.1 Implementation Details

We implemented a prototype of the setup as depicted in Fig. 3.1 for streaming video. The video server is trace-driven – it uses frame size traces of several video sequences taken from [1]. It reads generation-time/size information from the trace file, generates appropriate sized packets, and streams them to the MH using a UDP socket. The duration of the video sequences used in this experiment is 30 min.

The 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. 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.

When the MH's traffic passes through the proxy, the Traffic Manager within, encapsulates the captured packets with a header whose destination IP address is determined by the EDPF algorithm implemented within. At the MH, the Traffic Manager removes the outer IP header and passes control of the packet to the routing module to be handled as usual. The MH's Performance Monitoring Unit communicates with the Performance Monitoring unit in the proxy to pass on the parameters needed by EDPF (D_l and B_l). 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.

5.3.2 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.

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.

5.3.3 Experimental Results

Table 5.1 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. 5.2. At $L = 0.5sec$, EDPF has a FL of 0.5%, while HBI has 7.3%. At $L = 2sec$, EDPF

Table 5.1: Streaming Video: Buffering Time (in sec) for Continuous Playback

Alg/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

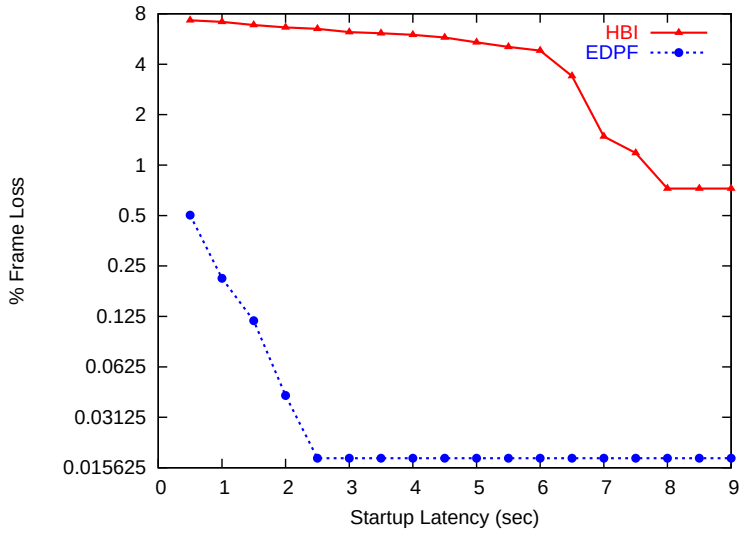


Figure 5.2: Streaming Video (Lecture): % Frame Loss vs Startup Latency

achieves FL of 0.04% while HBI still suffers a high 6.6% frame loss. Streaming applications that support VCR functions require one way delays in the range of 1-2 sec. If less than 1% 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, 8 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.

5.4 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 an important class of real-time applications: interactive multimedia.

Interactive applications like video telephony, video conferencing have very stringent delay requirements - they need one way latency under 150ms for excellent quality of service and under 400ms 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 1sec. 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.

5.4.1 Experimental Methodology

The network topology shown in Fig. 3.1 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 "Office Cam" [1] video, which captures the activity of a person in front of a terminal. The reason for choosing Office Cam of all the available traces is that 1) Interactive applications like video tele-

phony/conference will be similar in nature to office cam. 2) The bandwidth it needs compares to that we can obtain by aggregation in next-generation Radio Access Networks (RANs). We considered both MPEG-4 and H.263 encodings. The duration of the clip in each case is 30 minutes. For MPEG-4, the frame period is 40ms, the mean and peak bit rate are 400kbps and 2Mbps respectively. For H.263, the frame period is variable, a multiple of the reference frame period of 40ms. The mean and peak bit rates are 260kbps and 1.5Mbps respectively. A frame could potentially be sent as multiple packets, if it is too big. The maximum packet size we considered was 1400 bytes.

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 22ms (the same trace file was used on all the paths between proxy and BSs). This makes the one-way wireline delay 37ms, which is reasonable to expect on wireline Internet paths. 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.

Base-Stations & 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 [7] 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 [6].

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 one-way delay experienced by the packets between the server and the client application. (2) 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. (3) 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.

5.4.2 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 asymmetry, number of interfaces, 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.

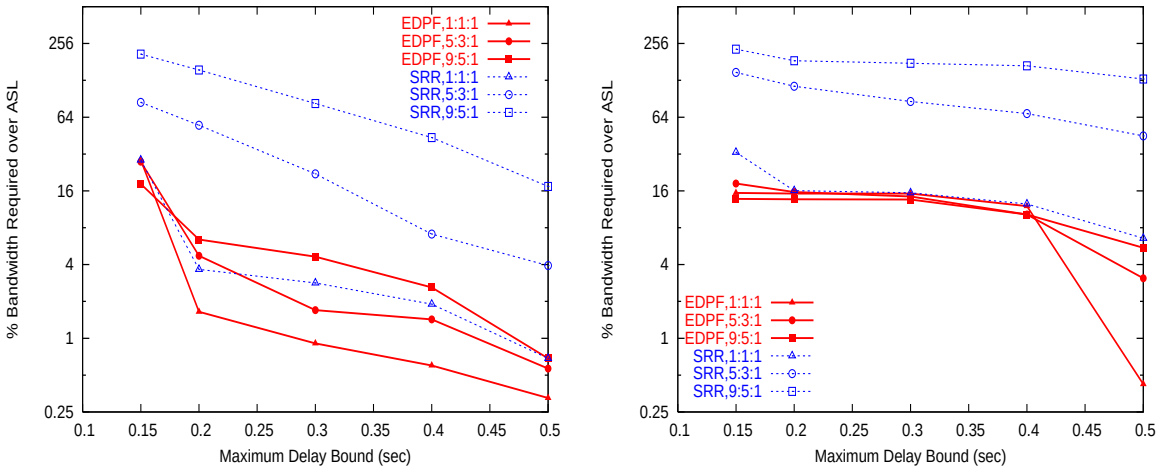
Given a packet stream of P packets, we would like to determine bandwidth B such that the delay experienced by the packets is bounded by DB_{max}. We are also interested in finding the buffer capacity C that is needed to ensure that there is no overflow at the MH.

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.

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. 5.3 shows this percentage for the case of MPEG-4 and H.263 encodings, when the bandwidth is split among 3 interfaces 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.



(MPEG-4) **(H.263)**

Figure 5.3: Interactive Video: % Bandwidth Needed over ASL (0% F_{loss}, Interfaces = 3)

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

Table 5.2: Interactive Video: Average Bandwidth required (in kbps) for ASL, EDPF, and SRR

Alg/DB_{max}(ms)	150	200	300	400	500
ASL	707	645	605	579	569
EDPF	872	696	624	591	574
SRR	1513	1129	805	674	616

Table 5.3: Bandwidth Splits

Ifaces	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

same. Table 5.2 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 5.3.

Application Performance Measures

While the previous sub-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) for MPEG-4 and 450kbps (1.75 times mean) for H.263. A choice of a much lower bandwidth than this results in $> 1\%$ of the packets experiencing delay in excess of 500ms, 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). Note that when we present results for the two encodings - MPEG-4 and H.263, it is not possible to compare them as the mean rate of the streams is different.

However, as will be shown, the performance trend in case of H.263 is not much different from MPEG-4. Hence, we mention the results for H.263 only briefly, unless the explanation provided with respect to MPEG-4 does not hold. We now present the various performance metrics in turn.

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. 5.4. The different plots in each graph are for the different algorithms, and for different values of the bandwidth split. For MPEG-4, 99.8% of the packets have delay less than 200 ms for ASL. In case of EDPF, this value ranges between 99.2% to 99.6% for different splits. For SRR, its between 56.5% and 99.2%. In case of H.263, 99.9% of the packets have delay less than 200 ms for SL. In case of EDF, this value ranges between 96.5% to 99.6% for different splits. For SRR, its between 40.7% and 97.4%.

Another point worth mentioning here is the amount of reordering seen in the experiments. Since buffer size directly correlates to reordering, we present the average and maximum buffer occupancy. When averaged over the different splits (Table. 5.3), EDPF had an average buffer occupancy of 0.32 packets, maximum of 4 packets. SRR on the other hand had an average buffer occupancy of 0.71 packets, maximum of 12 packets.

Frame Discard Ratio

Fig. 5.5 shows F_{loss} as a function of different DB_{max} when the number of interfaces is fixed at 3. As expected, F_{loss} decreases as DB_{max} increases. In case of MPEG-4, when DB_{max} is set at 200ms, EDPF achieves a F_{loss} less than 0.6% while for SRR it can be as high as 20% loss (for ASL it is 0.2%).

A feature we observe in the plots corresponding to EDPF is that the split that causes the maximum F_{loss} is different for the different DB_{max}. For instance,

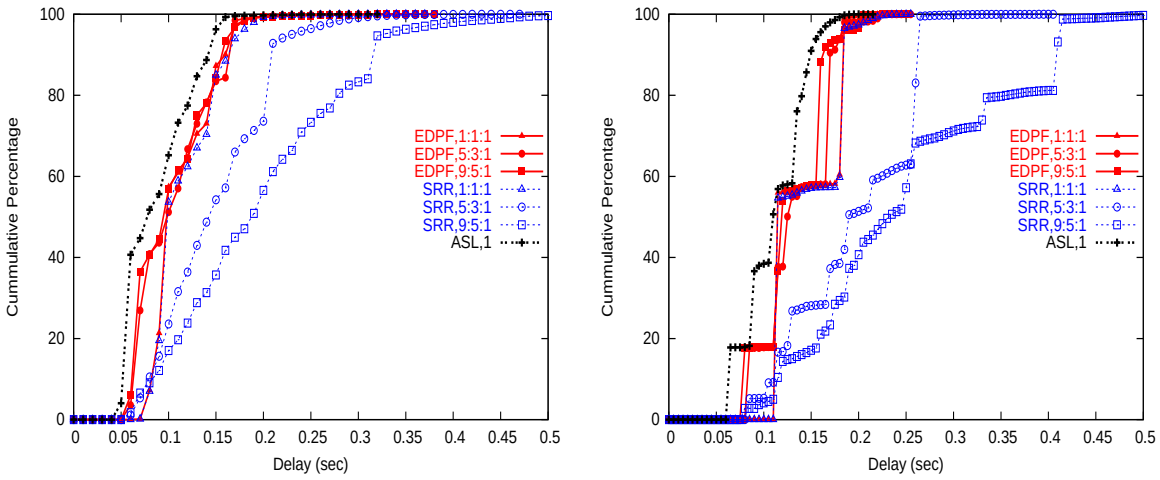


Figure 5.4: Interactive Video: Cumulative Percentage of Delay (Interfaces = 3)

in the MPEG-4 case, 5:3:1 has the highest loss when DB_{max} is set at 150 ms, and at 200 ms it becomes the lowest. The reason for this behavior is as follows. Packets can suffer excess delay at a BS over a large time scale, where the buffers at the BSs grow over time as traffic is injected at a rate that exceeds the service rate, or over a small time scale where a burst of packets belonging to a single large frame arrive. The former case results in lot of successive frames getting discarded, whereas the latter case results in only the frame in question getting lost. Single frame losses happen less often when the asymmetry is large. On the other hand, successive frame losses are less when the asymmetry is small as the load gets more uniformly distributed. When DB_{max} is small, successive frame losses will be almost the same across the splits, so single frame losses will dominate. Thus the lower asymmetry splits will have higher losses. When DB_{max} is increased, the single frame losses go down and the dominant losses will be successive frame losses and hence the higher asymmetry splits have higher losses. For SRR, higher asymmetry inherently introduces more reordering and hence loss.

In case of H.263 at smaller DB_{max}, the frame loss is close to 100% which is

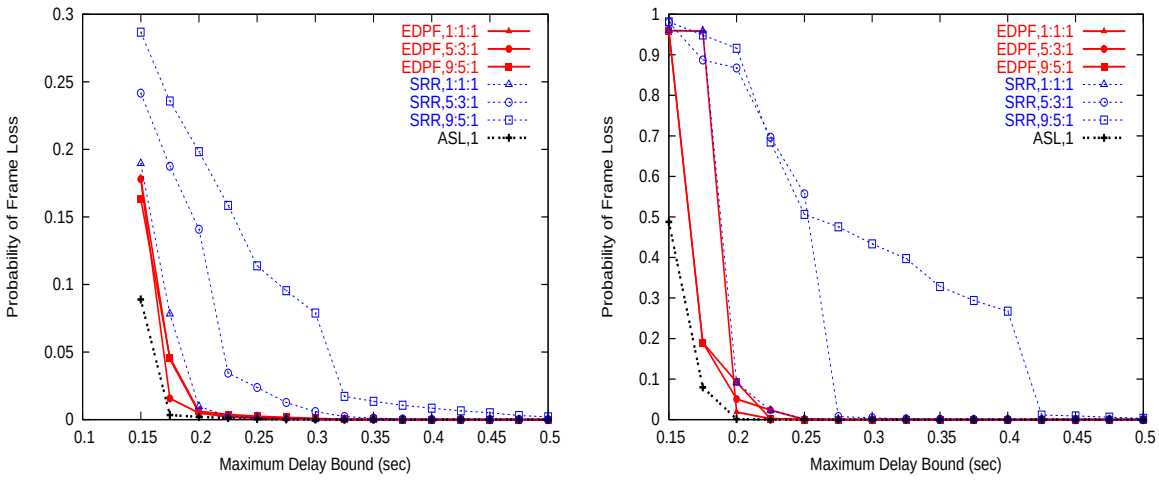


Figure 5.5: Interactive Video: Probability of Frame Loss, (Interfaces = 3)

way above that for MPEG-4 encoding. (Note that the y-axis scale is different in the case of H.263). However, the packet loss (not shown) seen at the same latencies is not very different (around 35%) and follows the same trend with increase in latency as in MPEG-4. This difference is due to the fact that most frames in H.263 encoding are large and span multiple packets. The mean frame size in MPEG-4 is 2,000 bytes while in H.263 it is 6,200 bytes. Even if one of the packets of these large frames is lost, the whole frame cannot be displayed. In case of SRR, at lower DB_{max}, though higher asymmetry introduces more packet loss, the frame loss does not follow the same trend. This is because, frame loss is a function of how packet losses occur - in clusters or more spread out.

Glitch Statistics

The glitch rate is another useful metric that captures the disruption in the video presentation due to discarded frames. Table. 5.4 shows the glitch statistics for MPEG-4, when the number of interfaces used is 3 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, 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.

Table 5.4: Interactive Video: Glitch Statistics (MPEG-4, Interfaces = 3, DB_{max} = 0.3 sec)

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
g (per ms)	0.55	0.55	2.78	7.22	3.89	140	1809
Avg G_d	4	6	3.2	2.77	2.14	1.063	1.089
Max G_d	4	6	8	8	7	6	9

In the remainder of this section, all results presented are for MPEG-4 encoding, the behavior does not vary much with H.263.

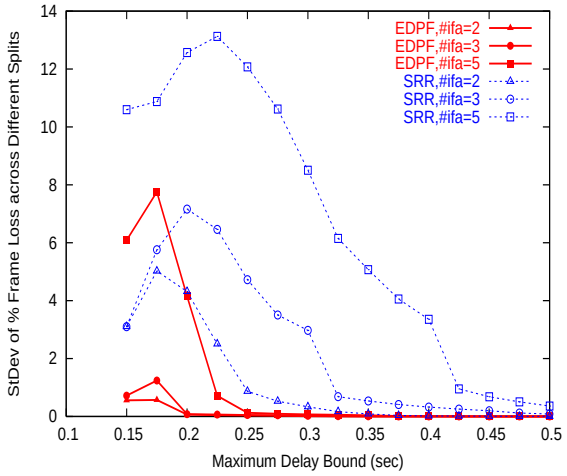


Figure 5.6: Interactive Video: Sensitivity to Bandwidth Asymmetry

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. 5.3) for a given number of interfaces and delay bound. The standard deviation of the obtained values (expressed in %) in shown in Fig. 5.6 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 don't 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.

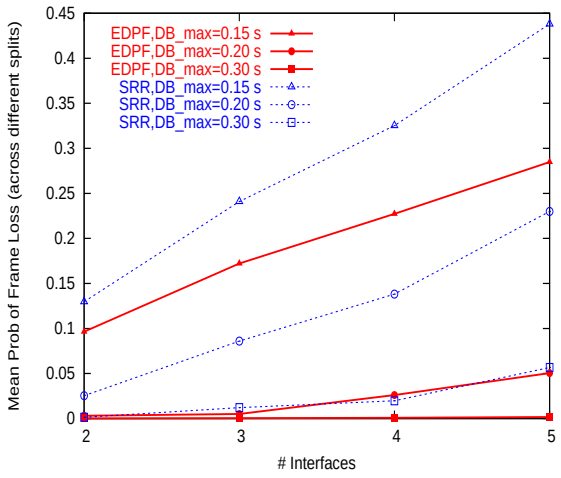


Figure 5.7: Interactive Video: Sensitivity to Number of Interfaces

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 in Fig. 5.7. 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 200ms, when increasing the number of interfaces from 3 to 4, EDPF showed an increase in F_{loss} of only 2.1% while SRR showed an increase of 5.2%.

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% uniformly distributed channel losses, no retransmissions gave us a F_{loss} of 1.9%, while retransmissions brought it down to 0.2%.

Wireline Delay Variations

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 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). The delay variation will likely be masked by this queuing delay. In equation 5.1 of Section 5.1.1, 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 225ms, for a truncated Guassian delay distribution with mean 22ms and no delay variation F_{loss} was 0.26% frame loss and for 10 ms standard deviation in delay, it was 0.28%.

In addition to the "Office Cam" video trace, we have experimented with other video traces from [1] as well as H.263 encoding. We obtained similar re-

sults as shown above. EDPF in all cases, effectively aggregated bandwidth while minimizing delay experienced by the packets.

5.5 Selective Frame Discard

Often in spite of performing bandwidth aggregation, it may not often be possible to reserve enough bandwidth on the interfaces to satisfy user QoS. While considerable research has gone into coming up with feasible video transmission schedules when network bandwidth (or client buffering capacity) is constrained [20]. In our set up, no such feasible transmission schedule that avoids frame loss exists because available bandwidth on the interfaces is too small. One is then left with two choices. The first choice is to choose an available low quality, low rate video - which still may or may not satisfy user's QoS depending on the reserved bandwidth. The second choice is to choose a better quality high rate video but drop frames selectively to minimize their impact on the overall quality of the video. We now experiment with the second choice of selective frame discard. That is, prior to applying EDPF, we first determine if the packet has to be dropped or sent, according to the frame-discard policy in place. If it has to be sent, then on what interface according to EDPF algorithm. We next present various frame discard algorithms that make this decision of video frame discard.

5.5.1 Frame Discard Algorithms

The decision to drop or send a frame depends heavily on the structure of the video sequence. In this work, our focus is on the MPEG standard – the same ideas can be extended to other standards with inter-frame dependencies.

The MPEG standard encodes the information of a scene into multiple Group of Pictures (GOP) consisting of three different types of frames - I, P, and B. I frames are coded autonomously, while P frames are coded in reference to the most recent I or P frame. B frames are coded using the closest previous and future

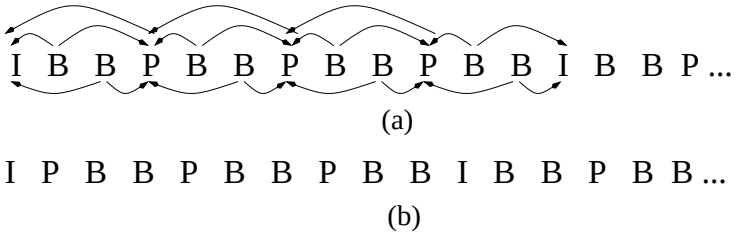


Figure 5.8: Dependency Structure of MPEG (GOP = 12): (a) Playback; (b) Transmission Order

I or *P* frames. Fig 5.8 illustrates the dependency structure for a GOP of size 12. Because of these interdependencies, the loss of an *I* frame results in the loss of the entire GOP and has a much worse impact on the video quality than the loss of say a *B* frame that has no dependent frames. This frame priority information must be taken into account when dropping frames to minimize the effect of losses on the quality of the video. Note that the transmission order of frames differs from the playback order (Fig 5.8). *I* or *P* frames sent earlier are displayed later than *B* frames. This is so as to help decode dependent frames. This gives *I/P* frames more leeway with respect to delay than *B* frames.

When network bandwidth is constrained, a naive approach that attempts to transmit every frame results in high loss rate due to missed playback deadlines. This policy, which we term *NO-Drop*, not only wastes scarce bandwidth by transmitting frames whose deadline cannot be met but prevents future frames from meeting their deadlines. This essentially captures the performance (video quality) one can expect to see in the absence of any frame discard policy.

Selective frame discard attempts to effectively utilize the constrained bandwidth by discarding frames whose loss has minimal effect on the overall video quality. An optimal scheme that discards frames to maximize video quality needs perfect knowledge of the frame size of the entire video sequence (not possible for interactive video) and has a complexity of $O(N2^N)$, where N is the number of video frames [40]. This makes this scheme computationally prohibitive and impractical to implement. However, it is possible to achieve comparable results with several

low complexity $O(N)$ algorithms (O(1) each step). We discuss two such algorithms here.

A straight forward approach is to drop the frames that are unlikely to meet their playback deadline (similar in spirit to that proposed in [22]). Any future dependents of these dropped frames are also discarded when they arrive at the proxy. We term this policy Deadline Drop (*DL-Drop*). This basic idea can be easily integrated into our scheduler EDPF without the need for time stamping and other functionalities. The scheduling algorithm EDPF, estimates the delivery time of a packet to determine the Internet path to use. This estimate can be used to determine if the packet (and hence the frame the packet is a part of[2]) is going to meet its deadline. While *DL-Drop* can achieve much better performance than *NO-Drop*, this approach does not use the priority information of the frame type. If an I frame misses its deadline, the entire GOP is dropped. However, had the preceding B frame (that was sent) dropped to ease network resources, the I frame and its dependents could have been saved. We now present our algorithm Min-Cost Drop (*MC-Drop*) that uses this priority information.

When a packet arrives at the proxy, a decision to send or drop the packet has to be made. If sending the packet results in a future high priority frame missing its deadline, we would ideally like to drop the packet. However, it is not possible to know the future frame sizes and hence their deadlines in advance since we are dealing with interactive video - packets come one at a time at the proxy. An important observation that helps overcome this drawback is that most video streams have a high degree of correlation (> 0.8) of frame sizes across GOPs, for a lag of 1-2 GOPs [5]. This observation, permits us to estimate future frame sizes in the next 1-2 GOPs based on observed frame (packet) sizes in the current/previous GOP. In *MC-Drop*, we maintain a window size of k GOPs. Each time a packet arrives, we estimate the delivery time of all the frames in the window based on the scheduling algorithm EDPF. The present packet is dropped only if by dropping it,

[2]The video server breaks the large frames into smaller sized packets before transmitting them onto the Internet.

it is possible to meet the deadline of a future higher priority frame in the window. In addition, we drop all packets of the frames that miss their deadline and their dependents. A window size of 1-2 GOPs is more than adequate for our purposes because any high priority frame is normally preceded by enough low priority B frames, that dropping these B frames will help meet its deadlines. Note that in *MC-Drop*, it is possible for incorrect estimation to cause some high priority frames to be dropped, but this situation is no worse than *DL-Drop*. Some low priority frames may be unnecessarily dropped in *MC-Drop*, but in general the advantages outweigh the disadvantages as will be shown in Section 5.5.3.

Fundamental to both *DL-Drop* and *MC-Drop*, is the ability to estimate the delivery time of a packet by EDPF. This depends on the estimation of the one-way delay experienced by a packet (on each of the different paths) as it traverses the Internet from the proxy to MH. This delay has two components: (a) Wireline delay - delay experienced by the packet between proxy and Base Station (BS) serving the MH, and (b) Wireless delay - delay experienced by the packet between BS and MH (queuing and transmission delay). Both the delay components can be estimated and can be expected to be stable. This is due to the following reasons. First, we assume that once bandwidth (wireless) is reserved on an interface, it is guaranteed for the entire duration of the session. This is a reasonable assumption since present and future wireless networks do provide QoS support (in the form of bandwidth reservation). So, delay variation if any is in the wired part and not on the wireless part. However, the wire-line link speeds are quite high and the mean wire-line delay and its variation about the mean are normally very small (few ms). Even if large ($\simeq$ 10ms), this variation is usually masked by the backlog at the base-station queue serving the mobile – we assume the wireless hop to be the bottleneck link.

As such, EDPF does not require time synchronization between the proxy and the MH for this estimation, only the relative one-way delays between the multiple paths are required, and not the absolute values. However, in the DL-

Drop/MC-Drop algorithms the one-way delay estimate needs to be correlated with the playback deadline at the MH. This too does not require strict time synchronization between the proxy and the MH, and can be done as follows. Initially, the proxy can record the expected delay of packets sent to the MH. The MH can then report to the proxy as to how much ahead of (or after) its deadline the packet arrived. The proxy can thus correlate the recorded delay estimate for the packet to how much ahead/after its deadline the MH perceived it to be. Since the one-way delay is expected to remain stable, this correlation can be used for future packets as well. The above process can be repeated with multiple packets to improve the estimate.

5.5.2 Experimental Methodology

We now present our methodology for evaluating the frame-discard policies presented above. Our overall approach is one of trace-based simulation. The components that make up the network topology are as shown in Fig 3.1.

To simulate video server behavior, we have used frame size traces of two video clips (class room lectures) representative of interactive applications from [5] [3]. The video clip is encoded into a base layer (I/P frames) with a set target rate of 128Kbps and an enhancement layer (B frames). The server packetizes the video frames into 1000 byte packets and passes them on to the proxy. The proxy implements the EDPF scheduling algorithm along with the frame discard policy. The Base Stations have a link capacity equal to the reserved bandwidth. We do not simulate cross traffic as the channel is considered dedicated. The wired part of the network has high bandwidth (10Mbps) – the wireless links are assumed to be the bottlenecks. The wire-line delay experienced by the packets was introduced from traces collected by measuring round-trip times on different Internet paths [4]. The mean delay experienced by packets between the server and the proxy is 15ms and

[3] These traces are more recent and are temporally encoded for content adaptation and differ from those used in Interactive video experiments.

[4] We collected traces between hosts located at the following universities: UCSD, UCB, Duke, CMU.

that between the proxy and the BSs is 22ms, making the mean one-way *wireline* delay 37ms . Note that this delay does not include queuing and transmission delay on the wireless segment.

As packets arrive at the client, they are placed in a buffer. The video display begins after a fixed delay, which we term *Startup Latency*. Once the display begins, the client displays frames consecutively every frame period (1/30 sec). If at one of these epochs, the client's buffer does not have the completed frame, the frame is considered lost. We set the *Startup Latency* at 200ms in our experiments i.e. the client begins display 200ms after the server transmits the very first packet. This is to ensure that all frames that are displayed have one way latency (total delay between server and MH) less than 200ms as required by interactive applications.

To measure the quality of video reception, we used the following performance metrics - (1) Peak Signal to Noise Ratio PSNR (in dB) of the received video sequence. For frames that could not be displayed, PSNR is assumed to be zero; (2) Glitch Duration G_d - length of consecutive frames that could not be displayed; (3) Glitch Cost G_c that captures the effect a frame loss has on the perceptual quality of video as was proposed in [40]. Every undisplayed frame i is associated with a cost c_i. If frame i belongs to a sequence of consecutive undisplayed frames, $c_i = l_i$, if frame i is the l_i^{th} consecutive undisplayed frame. Otherwise $c_i = 1+1/\sqrt{d_i}$, where d_i is its distance from the previous undisplayed frame. $G_c = \sum_1^N c_i$. This metric captures two important aspects of playback discontinuity – cost due to consecutive discard and that due to spacing between discarded frames, both of which are important measure of perceived quality; (4) Total number of Frames that could not be displayed - F_{loss}.

In addition to *NO-Drop*, *DL-Drop* and *MC-Drop*, we consider two other design alternatives. In the first policy, *ENH-Drop*, we only send the base layer (*I/P* frames) and drop the enhancement layer (*B* frames) totally as the reserved bandwidth is too small to accommodate both [5]. In the second policy, *LR-NoDrop*,

[5]In temporal encoding, a video is encoded into many sub-streams (scalable extensions). It is common place to employ a hierarchical filter to select required number of sub-streams that satisfy a given QoS.

we consider same video clip encoded at a lower target rate. This gives us a measure of the improvement in performance when employing selective frame discard on a high rate, better quality video as opposed to settling for a low rate, low quality video.

5.5.3 Experimental Results

In this section, we present the performance of the different algorithms under different scenarios. Table 5.5 lists the characteristics of the two video clips considered, encoded at a rate of 128 (medium quality) and 64kbps (low quality). The medium quality video is temporal scalable encoded - it has a base and enhancement layer. The base layer target rate (not aggregate) is 128kbps. The low quality is single layer encoded (non-scalable). The aggregate target rate is 64kbps. Refer to [33] for further details. While our focus is on understanding the performance of these algorithms when using multiple interfaces, the ideas apply for single interface use as well. We present results for multiple interfaces and explain in passing the results we obtained for single interface use.

Title (Lecture)	Dur. (min)	Rate(kbps)		PSNR(dB)		GOP Corr. lag=2
		avg	max	avg	stdev	
Medium Quality: Base Layer Target rate 128kbps						
Reisslein	30	222	1361	27.1	2.1	0.90
Gupta	30	205	1255	28.1	2.2	0.82
Low Quality: Base+Enhancement Layer Target rate 64kbps						
Reisslein	30	72	946	23.5	1.7	0.86
Gupta	30	68	1878	23.9	1.6	0.72

Table 5.5: Characteristics of MPEG-4 Temporal Video Traces

Table 5.6 shows the performance of the different frame discard algorithms when the number of interfaces considered is two. The overall reserved bandwidth is 240kbps, split among the two interfaces in the ratio 2:1. As can be seen from the table, sending frames without considering the underlying network resources (*NO-Drop*) results in severe performance degradation. The performance is even

worse than just sending the base layer and dropping the enhancement layer (*ENH-Drop*). While *DL-Drop* achieves significant performance improvement over *NO-Drop*, the occasional I frame drops result in high glitch cost G_c. This situation is avoided mostly by *MC-Drop* and as can be seen, it performs better than *DL-Drop* for all the metrics considered. Since the bandwidth reserved is adequate to get across all frames, the low quality 64kbps encoding *LR-NoDrop* experiences no glitches. However, the PSNR is about 3dB less than *MC-Drop*. The trade off between occasional glitches in high quality video vs no glitch low quality video on overall perceptual quality of video playback is difficult to capture with the metrics considered and needs further study. The performance trend of the different algorithms when using single interface is similar to the case of multiple interfaces except that the avg PSNR is slightly higher and G_c lower. For example, G_c for *DL-Drop* and *MC-Drop* for Reisslein lecture are 8606.8 and 6320.2 respectively.

Algorithm	Loss (%)	G_c (10^3)	G_d (frames)		PSNR (dB)	
			avg	max	avg	stdev
Reisslein: avg PSNR = 27.16						
NO-Drop	64.7	368023	155	26024	9.8	13.2
ENH-Drop	66.7	66.7	2	2	25.5	2.87
LR-NoDrop	0	0	0	0	23.5	1.71
DL-Drop	7.38	10.658	1.7	12	26.1	5.73
MC-Drop	6.6	6.339	1.4	12	26.4	4.92
Gupta: avg PSNR = 28.15						
NO-Drop	35.3	50394	115	8198	18.9	13.9
ENH-Drop	66.7	66.7	2	2	26.8	3.15
LR-NoDrop	0	0	0	0	23.9	1.63
DL-Drop	4.32	4.87	1.7	12	27.5	4.81
MC-Drop	4.15	3.87	1.5	12	27.6	4.56

Table 5.6: Frame Discard: Performance Statistics (Bandwidth = 240kbps, Split = 2:1)

Fig 5.9 presents the impact of different reserved bandwidths on the video quality for Reisslein lecture as captured by G_c. The bandwidth is varied from 200kbps to 300kbps. Note that the y axis is in log scale. Both *DL-Drop* and

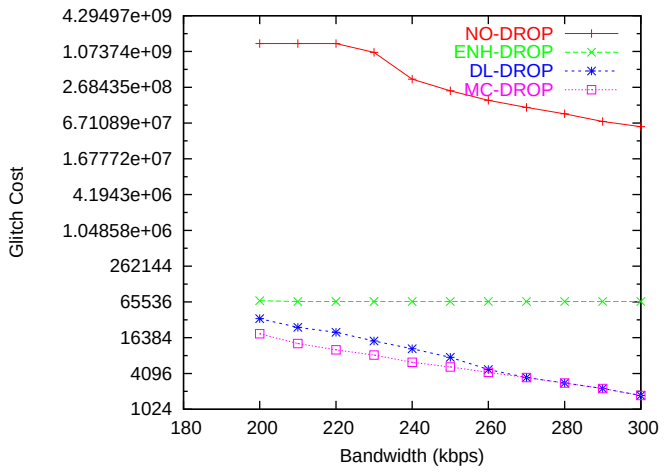


Figure 5.9: Frame Discard: Variation of Glitch Cost with Bandwidth

MC-Drop outperform *NO-Drop* by a large margin. G_c stays constant for *ENH-Drop* as 200kbps is enough to get the base layer across without any losses. As the bandwidth increases, the difference between *DL-Drop* and *MC-Drop* becomes small - 15,247 at 200kbps to 503 at 260kbps and 0 at 270kbps. This is because as bandwidth increases, the high priority frames *I/P* have less trouble meeting their deadline than *B* frames because they have more delay leeway as explained earlier in sec 5.5.1. So, when the reserved bandwidths is small, *MC-Drop* can bring about significant benefits. For the case of single interface, the performance trend is similar except that the difference in G_c between the algorithms is smaller. For example, the difference between *DL-Drop* and *MC-Drop* is 12,753 at 200kbps, 211 at 260kbps and 0 at 270kbps.

In order to capture the sensitivity of the system to the number of interfaces and asymmetry across them, we computed G_c and PSNR for different bandwidth splits across the different interfaces. Table 5.7 shows the performance of *DL-Drop* and *MC-Drop* for Reisslein lecture when the aggregate bandwidth is set at 240kbps [6]. G_c of *NO-Drop* is too high (order of 10^8) for all the cases. PSNR is too low - around 9dB. For *ENH-Drop*, G_c is same as before 66,723. The higher the number of interfaces and asymmetry, the more the reordering and hence delay

[6] A 3:1 split on two interfaces corresponds to a bandwidth of 180kbps and 60kbps.

Algorithm	1:1	3:1	5:1	1:1:1	3:2:1	5:3:1
Glitch Cost						
DL-Drop	6941	8953	14644	11735	13496	18427
MC-Drop	6667	7629	8439	7609	9113	10295
PSNR (dB)						
DL-Drop	26.39	26.19	25.64	25.86	25.56	25.35
MC-Drop	26.41	26.27	26.04	26.11	25.84	25.89

Table 5.7: Frame Discard: PSNR and Glitch Cost for Various Splits (Bandwidth = 240kbps)

and missed deadlines. As can be seen in Table 5.7, G_c which captures frame loss increases as the number of interfaces and asymmetry increases. However, PSNR does not follow the same trend in some cases. For *MC-Drop*, it mostly decreases but not always (e.g 3:2:1 vs 5:3:1) but the drop is normally very minor. This is mainly to do with inter-frame dependencies. It may not always be possible to display a *B* frame even if it arrives on time if its reference *I/P* frames do not arrive before its display time (the *I/P* frames may arrive before their respective display times) [7]. But as can be seen *MC-Drop* consistently performs better than *DL-Drop* for all the cases considered.

5.6 Summary and Discussion

In this chapter, we focus on one of the services provided by the architecture, Bandwidth Aggregation for real-time streaming and interactive video 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. EDPF is a light weight algorithm that incurs minimal overhead. The per-packet computation complexity is proportional to the number of interfaces,

[7] This situation can be avoided by dropping the *B* frame at the proxy if it is estimated that its reference frames will not arrive before its display time. We have not implemented this feature yet.

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.

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.

Another aspect we looked into when providing BAG services is that of frame discard when network bandwidth is limited. Our evaluation using video and delay traces show that our proposed algorithm *MC-Drop* outperforms by large margin a policy that discards no frames (*NO-Drop*). When reserved bandwidths are small, it also performs much better than a policy that discards frames that will miss their playback deadlines (*DL-Drop*).

This chapter is in part a reprint of the material in the following papers: K. Chebrolu and R. R. Rao, *Bandwidth Aggregation for Real-Time Applications in Heterogeneous Wireless Networks* (submitted for publication) and K. Chebrolu and R. R. Rao, *Selective Frame Discard for Interactive Video*, Proceedings of IEEE ICC 2004, Paris, France. The dissertation author was the primary investigator of these papers.

Chapter 6

Bandwidth Aggregation for TCP Applications

In this chapter, we focus our attention on BAG services for another class of applications - applications based on TCP. As with real-time applications, a crucial aspect that dictates TCP performance is the scheduling algorithm that splits the traffic onto the different paths with the objective of minimizing reordering. However, unlike in real-time applications, where we assumed dedicated wireless channels, we have to deal with best effort channels as most TCP applications are best effort based. So, we propose a new scheduling algorithm based on EDPF called PET (Packet Pair based Earliest-Delivery-Path-First Scheduling Algorithm for TCP applications) that estimates available bandwidth on a path based on Packet Pair technique [26]. We also consider a client-side buffer management policy (BMP) that processes the incoming data before passing it on to the TCP layer to hide from TCP any residual reordering that happens [1]. We discuss the design of PET and BMP in Section 6.3. Prior to that, we present our experimental design methodology (Sec. 6.1) to help understand the experiments we performed to establish the design criteria of the algorithms (Sec. 6.2). The results of our experiments to evaluate the effectiveness of PET and BMP are presented in Sec. 6.4.

[1]The above explanation corresponds to down-link traffic. The same holds for up-link traffic, with the roles of PET and BMP reversed at proxy and MH.

6.1 Experimental methodology

Our design and evaluation are based on experimental simulations since this allows us to quickly explore a wide range of possibilities and design choices in a controlled manner. We use the ns-2 network simulator [2] (version 2.1b9a) for our simulations.

We use the generic network topology captured in Fig. 3.1. In our experiments, the main TCP flow is an FTP transfer from the server to the MH. In our studies, we consider a wide variety of scenarios to understand the performance of PET-BMP. We consider both: (a) the presence of cross traffic and losses at the BS, and (b) their absence. While the first is a more realistic setting, the second helps us understand behavior of PET-BMP in response to each parameter better.

For the cross-traffic, we consider a mix of both FTP and web flows that compete with the main flow for the BS's link capacity. Losses are introduced via - 1) congestion at the BSs, where each BS has a maximum queue size and implements a drop-tail queuing policy and 2) channel errors, where the BSs introduce uniformly distributed errors in the packets.

We use Weighted Fair Queuing (WFQ) [16] for packet scheduling at the base stations where all flows through the base station are given the same weight. This permits equal sharing of the scarce wireless link capacity among all the flows. Our WFQ implementation uses a single buffer for storing packets from all the flows.

6.1.1 Parameter Settings

The details of the various parameter settings of our experiments are as follows. We consider either 2 or 3 wireless interfaces (communication paths). We do not consider more than 3 interfaces since such a scenario is unlikely in practice.

The main FTP/TCP flow lasts for 60 seconds, which is duration of the experiment. The server uses the New-Reno variant of TCP, where the maximum

congestion window size is set to 50 packets. The packet size used is 1500 bytes. We also use a max-burst factor, which limits to four the number of packets that can be sent in response to a single ACK. Without this, New-Reno could send a large burst of packets upon exiting Fast Recovery [19]. The TCP sink at the MH does not use delayed Acks.

The number of cross-traffic FTP and web clients considered for the different interfaces vary depending on the experiment. The size of the cross traffic FTP transfers are uniformly distributed between 200 and 2000 kbytes and their start-times are uniformly distributed between 0 and 60 seconds respectively – the total duration of the experiments. The web clients run for the entire 60 sec of the simulation. The details of the CDFs for think/reply/size used in web clients can be found in [3].

We consider a range of values for the link capacities of the various interfaces. For experiments without any cross traffic, we experimented with 3 interfaces with link capacities of 50kbps, 100kbps and 200kbps. These values reflect the bandwidths one can expect to see on WWANs when the wireless channel is dedicated for single use to the MH. In the presence of cross traffic, we increase the link capacities of all interfaces to 1000kbps. Note that even in this case, since we consider different cross traffic patterns at the BSs, the average throughput available on the interfaces can be quite asymmetric.

The server and the proxy are connected by a 10Mbps link with a one-way delay of 15ms. The proxy and Base Stations (BS) are connected by 10Mbps links with one-way delay of 50ms on each. In next generation networks, the BSs are considered to be an extension of the Internet. Accordingly, we set the one way delay from proxy to BSs values typical of present day Internet paths. The results are not particularly sensitive to the exact value of the one-way delay. The bandwidth value of 10Mbps ensures that the wireless interfaces are the bottleneck.

6.1.2 Algorithms Under Comparison

For comparison purposes, we consider three *ideal* systems which place a limit on the best that can be achieved by a network layer approach to bandwidth aggregation. One is an application-layer solution, *MTCP*, where we open multiple TCP connections one on each interface and sum the throughputs achieved on the individual interfaces [2]. The other point of comparison is Aggregated Single-Interface TCP *(ASL)*, where we replace the multiple interfaces with a single interface of the aggregate capacity. The third is a system that employs the idealized scheduling policy Earliest Delivery Path First (EDPF) at the proxy, which has perfect knowledge of the system parameters.

We note that comparison with ASL is meaningful only in the no-cross-traffic case. This is because, if we introduce cross traffic in ASL by summing up the cross traffic at each individual BS, the throughput of the main TCP flow goes down considerably. This is in turn because, the available bandwidth in ASL now gets distributed equally among all the flows. On the contrary, when using multiple interfaces the available bandwidth at a BS gets distributed only among the flows served by it. Also, note that MTCP is in general more aggressive than any single end-to-end TCP connection since it uses multiple congestion windows.

6.2 Design Criteria

To motivate the design of the Earliest Estimated Delivery Path First (PET) and the Buffer Management Policy (BMP), we now present some preliminary results and derive a set of design criteria from them. We first state the criterion, and subsequently explain the reasoning behind it, presenting simulation results as necessary.

Criterion 1: Utilize bandwidth of all interfaces

Our objective is to achieve the maximum possible throughput from the

[2]The transport layer solution as proposed in [24] strives to achieve the same performance as MTCP.

Algorithm	Thr(kbps)	DupAcks	Retrx
MTCP	339.6	0	0
ASL	339.6	0	0
EDPF	339	0	0
WRR	210.6	533	128
WRR-BUFF	338.0	0	0
PET-BMP	336.8	0	0

Table 6.1: TCP Performance: Ideal Situation - No Cross traffic, No losses

server to the MH using TCP over an underlying heterogeneous network, without any modifications to TCP. The maximum throughput is achieved only if we utilize the bandwidth of all the interfaces – hence this criterion.

Criterion 2: Minimize reordering

Let us now look at what happens when one uses all the interfaces. A simple scheduling policy that can be implemented at the proxy is the Weighted Round Robin (WRR), where the number of packets sent on a path (corresponding to an interface) is proportional to the link capacity of the interface. Table 6.1 shows the performance of WRR in comparison with MTCP, ASL and EDPF (focus on the first four rows). As can be seen from the table, the throughput achieved by WRR is much lower than MTCP, ASL or EDPF. This is due to several unnecessary retransmissions. Whenever packets are reordered, the TCP sink generates DUP-ACKs (about 533 packets were reordered in case of WRR). On receipt of more than 3 DUP-ACKs for a packet, the TCP sender considers the packet lost and invokes congestion control by reducing the sending rate (halving the congestion window). On the other hand, as can be seen, EDPF has performed as well as ASL and MTCP. Now, EDPF is an idealized scheduling policy that has perfect knowledge of the system parameters, and is thereby able to *eliminate* reordering altogether. In reality however, one can only estimate these parameters and schedule accordingly. So, eliminating reordering totally may not be feasible, so the best one can do is to *minimize* reordering.

Criterion 3: Hide reordering from TCP

Since reordering is inevitable in practice and can have quite a negative impact on TCP, let us see if its possible to overcome its effects. The main problem with reordering is the generation of DUP-ACKs. Since we do not wish to make any changes to TCP, we can prevent the generation of DUP-ACKs by buffering packets at the client (at the network layer) and passing them in order to TCP. So, in the previous example, suppose we employ such a simple buffering mechanism at the client, the performance of WRR can be significantly improved. As can be seen from Table 6.1, WRR-BUFF (WRR with client buffering) performs similar to EDPF, MTCP, and ASL. Hence this third design criterion: "hide reordering from TCP".

Note that this does not mean that we can relax the second criterion of minimizing reordering assuming that its effects can be masked by buffering. In the previous example experimental setup, the amount of reordering was small. Hence buffering helped in overcoming reordering.

Simply buffering may not help if the amount of reordering is large. To see this, suppose we increase the delay on the third interface (the one with capacity 50kbps) from 50ms to a much higher value of 1s, there is a lot more reordering in WRR. This is because WRR considers only link capacities and not path delays while scheduling. In this setup, EDPF achieves 337.4kbps, WRR 95.6kbps, and WRR-BUFF only 70.4kbps. EDPF achieves good bandwidth aggregation as the scheduling ensures that there is no reordering (unlike WRR, EDPF considers path delays in addition to link capacities). On the other hand, WRR and WRR-BUFF have much lower throughput, even lower than what we could have achieved had we not performed bandwidth aggregation but just used the highest bandwidth interface (200kbps). Further, the performance of WRR with buffering is even worse than without buffering. This is because there are as many as 40 retransmission timeouts in case of WRR-BUFF. Since no DUP-ACKs reached the sender to trigger fast retransmission (due to client buffering), this forced the sender to enter slow

start each time.

Criterion 4: Detect packet losses and react to them in a timely fashion

So far we have not considered losses. The simple buffering policy above works because of this. However in the presence of losses, we could potentially wait indefinitely until the next expected sequence number comes while storing out-of order packets. This would eventually trigger a retransmission timeout at the TCP sender for each packet loss. When losses are present, we need to react to them in a timely fashion, or otherwise risk retransmission timeouts which lower throughput significantly. So in the presence of buffering it is important to detect losses and react to them in a timely fashion.

Criterion 5: Avoid Burstiness of Traffic

Another problem with buffering out of order packets is that sending them all at once to the TCP receiver will generate a burst of ACKs and they in turn generate a burst of packets at the TCP sender. In general bursty traffic is not a good feature and is better avoided as it increases queuing delay, introduces more losses and lowers throughput [27].

Criterion 6: Isolate losses

Since we are using multiple interfaces, the different paths can have different loss rates. Since TCP reacts to losses by reducing the sending rate, it is important to ensure that losses on one path don't affect the achievable throughput on the other paths. We term this as *loss isolation*. Note that MTCP achieves such isolation naturally.

Figure 6.1 shows the instantaneous throughput (averaged over 1 second intervals) of EDPF and MTCP when losses (congestion based) are introduced at the BS with 200kbps capacity by setting the maximum queue size to 30kbytes. As can be seen in the figure, in case of MTCP, the losses on one interface have not affected other interfaces but in case of EDPF, losses on one interface (ifa-0) have lowered the throughput on other interfaces (ifa-1 and ifa-2). This is not desirable. So, the final design criterion is to isolate losses.

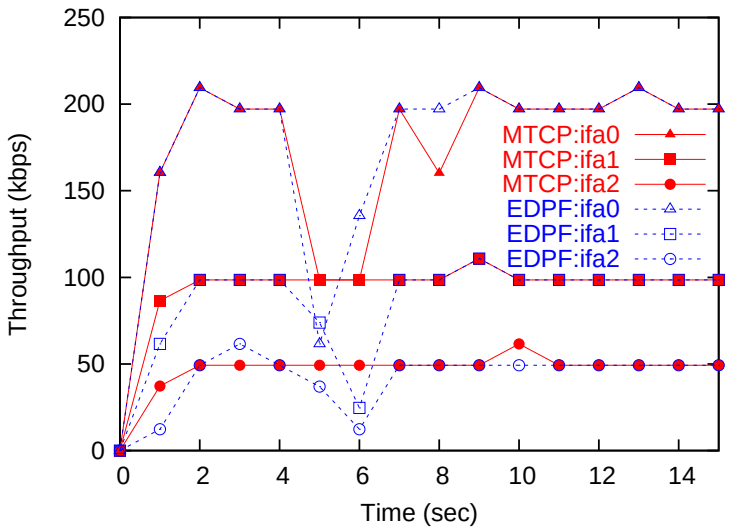


Figure 6.1: TCP: Isolation of Losses

Note that criteria 1, 2 and 6 dictate the design of the scheduling algorithm at the proxy while criteria 3, 4 and 5 dictate the design of buffer management at the client.

6.3 Scheduling and Buffer Management

We now present our design of a network layer solution to bandwidth aggregation for TCP applications based on the design criteria described in the previous section. There are two main parts of our solution: Packet-Pair based Earliest-Delivery-Path-First scheduling algorithm for TCP applications (PET) at the proxy, and the Buffer Management Policy (BMP) at the MH. We look at each in turn.

6.3.1 Packet Pair based Earliest-Delivery-Path-First Scheduling Algorithm for TCP applications (PET)

Consider the EDPF scheduling discipline. It is able to achieve good utilization of bandwidth on all interfaces while minimizing reordering (criteria 1 and 2) because it has perfect knowledge of the system parameters. In reality, we can

only *estimate* these parameters. So, if the design of PET is based on the same concept of EDPF but with perfect knowledge replaced by estimates, we can hope to meet criteria 1 and 2 to some extent. The parameters of concern for PET scheduling at the proxy are: (1) the wireline delay on each of the communication paths from the proxy to the BSs, and (2) the available bandwidth on the wireless link. Note that the variable A_l, the time the wireless channel becomes available for the next transmission at the BS can be estimated from the available bandwidth, which translates to the transmission and queuing delay at the BSs.

In the next generation Radio Access Networks, Base Stations (BSs) are considered to be an extension of the IP based Internet. Accordingly, we consider the delay experienced by the packets up to the BS, similar in nature to Internet path delays. This *wireline* delay 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 [6]. Further, any errors in estimation are usually small (as average delays on the backbone are themselves small), and will likely be masked by the transmission and queuing delay at the bottleneck bandwidth [3].

The second parameter, the available bandwidth, is dependent on the amount of cross traffic, fluctuating channel conditions etc. These can definitely change in the middle of a connection. Hence the available bandwidth, unlike delay, needs to be estimated and updated continually throughout the duration of the connection. Our overall approach for estimating this available bandwidth is based on the packet-pair technique [26]. The packet-pair technique estimates the bottleneck capacity of a path from the dispersion (spacing) experienced by two packets which were sent back-to-back. Since the wireless link is often the bottleneck in

[3]Present day wireless technologies such as GPRS, 1xRTT show a high degree of delay variation. These systems are very young and the delay variation is likely due to initial setup problems. Moreover, we believe that the variation is caused on the wireless hop (due to release grant/retransmissions as captured by available bandwidth parameter), than on the path between proxy and BS (as captured by wireline delay parameter).

the network path and since we assume that the BSs implement WFQ, bandwidth estimation based on this technique is feasible.

Using separate signaling packets to probe bandwidth continuously is excess overhead. Further, the probing packets will compete with the main flow for available bandwidth. Hence we rely on the incoming TCP packets themselves for bandwidth estimation by treating every incoming TCP packet as part of a pair and sending packets in pairs on any path.

Since PET at the proxy needs the inter-arrival time between packet pairs to update its bandwidth estimate, we introduce an additional mechanism in the form of a feedback loop between the MH and the proxy to get this information. We achieve this by means of *Signaling-Information* packets (SIG-INFOs) sent from the MH to the proxy for each TCP packet received from the sender (via the proxy). The MH reports the packet arrival times in the SIG-INFOs (again, clock synchronization not necessary since the proxy only needs the inter-arrival times).

Once PET has the delay and bandwidth estimates, it can use EDPF with idealized delay and bandwidth values replaced by the estimates. In essence, the working of PET is as follows. PET treats every incoming packet as part of a pair. To begin with, PET has no bandwidth estimates to perform scheduling. So, there is an *initial phase* where it sends packet-pairs on the various paths in a round-robin fashion, until it gets a bandwidth estimate of the bottleneck in the path through SIG-INFOs. Then, it uses these bandwidth estimates to perform EDPF based scheduling to determine the path (interface) on which to send the first packet of a pair. The second packet of a pair is always sent on the same path as the first packet. Retransmitted packets are not part of any pair as the bandwidth estimate can be ambiguous. As PET clocks out more packets, it gets fresher bandwidth estimates, which it uses to schedule incoming packets with the goal of minimizing reordering.

Some additional details on how the PET scheduling mechanism works are as follows.

- While TCP is in slow start, every TCP ACK generates two packets that arrive back-to-back at the proxy, which helps bandwidth estimation. But once in congestion avoidance phase, packets may not arrive back-to-back at the proxy. However, these packets can still form a pair for bandwidth estimation since during this phase, normally the TCP pipe is not empty [25] and thereby both packets will be buffered at BS (before the bottleneck wireless link) and still give a valid estimate.

- It is possible for bandwidth estimates to be incorrect due to transient changes in cross traffic, or during multiple losses per congestion window where the TCP pipe gets cleared. In this case, there will be more reordering which is normally masked by the BMP at the MH. As new samples arrive, the history clears and the estimate converges to the correct value[4].

- As long as there is backlog, PET/EDPF ensure that bandwidths on all interfaces are utilized effectively. However there is a danger of getting stuck to a single interface – this can happen when the available bandwidth of one interface is estimated to be much higher than another. If losses at this stage slow down the TCP sending rate, to avoid reordering, we may never end up using the low bandwidth interface. This prevents us from getting any future bandwidth estimate updates on it. In the future even if more bandwidth is available on it, we may never use it. To alleviate this, it is important to send TCP packet-pairs on an interface periodically (even if PET chooses another interface) to estimate its bandwidth.

PET thus attempts to achieve design criteria 1 and 2. Design criterion 6, isolation of losses, as we argue now, is not always possible to achieve in a purely network layer solution. This depends heavily on the loss pattern. The reason for this is as follows. When a single loss occurs, if W is the window size just before the

[4]In our bandwidth estimate update mechanism, we use a large weight (0.8) for the current estimate, and a corresponding small weight (0.2) for the history as its important to react rapidly to current conditions, thereby minimizing reordering.

loss detection at the TCP sender, the sender does not send any packets for the first W/2 DUP-ACKs [25]. Normally, this should not clear the TCP pipe (backlog) on all the interfaces, and when TCP resumes after fast-retransmit, the pipes slowly fill up. In this case, the losses on one interface do not affect the others. However, if many packets are lost within a window, by the time W/2 DUP-ACKs arrive, some of the pipes would have cleared. The scheduler at the proxy cannot help in this situation by clever scheduling of packets because of the way TCP reacts to losses.

6.3.2 Buffer Management Policy (BMP)

Due to the use of packet-pairs, and also due to errors in bandwidth estimation, PET scheduling would result in some amount of reordering. In accordance with design criterion 3, we use a client-side buffer to hide this reordering. The main challenge in the design of the Buffer Management Policy (BMP) is the detection of losses when they happen (design criterion 4). We discuss this now.

Since we buffer packets, it is important to know if a packet is lost or merely reordered. A mechanism to do this is as follows. Suppose we are expecting (an in-order) sequence number N. We start a timer associated with it – when the timer expires we conclude that N could not have been reordered, and hence was lost. We then send the buffered packets to TCP so that DUP-ACKs can be sent and fast-retransmit triggered. We call this *timer-based* loss detection.

Timer-based loss detection requires adaptation of the timer value, which can potentially be done based on the amount of reordering seen. However, a simpler mechanism to detect losses exists if we assume that packets always arrive in order on an interface (which is usually the case). Suppose we receive sequence numbers greater than N on all of the interfaces, we can conclude that N is lost. We call this *comparison-based* loss detection. Even if this mechanism is used, a timer based mechanism cannot be dispensed with totally. This is because, if an interface (say ifa-2) is not used for a long time due to low bandwidth, we could wait indefinitely to conclude that N was lost (for a comparison-based loss detection, some sequence

number above N must be received on ifa-2 as well, to conclude loss). This would eventually trigger a TCP timeout, which is undesirable. Similar problems would arise if a loss happens towards the end of a connection, when there are no more new packets to be sent on all the interfaces. Hence, a timer-based mechanism is required, but can act as a backup for comparison-based loss detection. In such a case, since the timer kicks in only rarely, its value is not so crucial, and can be set at a conservative value. (In our experiments, we set it to 0.5sec.).

Design criterion 5 (avoid burstiness) can be achieved in two possible ways. One is to separate the generated ACKs by an interval at the client-side network layer, before sending them out on to the network (ACK pacing [8]). The same effect can also be achieved by separating packets by an interval when sending them to the TCP layer from the client-side buffer. We implemented ACK pacing in our experimental setup.

So in essence, PET attempts to satisfy criteria 1 and 2 by sending packets in pairs to obtain bandwidth estimates which it uses in turn to schedule packets to minimizing reordering. Criterion 6, isolation of losses is difficult to achieve using PET because of default TCP response to losses. BMP on the other hand buffers out-of-order packets and sends them in order to hide the effects of reordering on TCP (criterion 3). It also attempts to react to losses in a timely fashion based on comparison and timer based loss detection (criterion 4). ACK pacing [8] can be used to avoid burstiness (criterion 5).

6.4 Experimental Results

The above design of PET-BMP needs detailed evaluation. This section presents the results of our experiments to demonstrate the effectiveness of PET with BMP in achieving our design criteria.

Let us first look at how well PET-BMP performs in the ideal setup described in Section 6.2, with no cross traffic and no losses. Table 6.1 compares the

performance of PET-BMP with other algorithms. PET-BMP achieves a throughput of 336.8kbps (last row in Table 6.1) – very close to that of EDPF. The slight decrease in throughput is mainly due to two reasons: (1) The initial phase, where until an estimate is available, it sends packets in a round-robin fashion. (2) The use of packet pairs which introduce some small amount of reordering.

Now let us relax the idealistic assumptions in the experimental setting and introduce cross traffic and losses. We first look at each effect separately (Sec. 6.4.1 and Sec. 6.4.2). Later we consider both cross traffic and losses in the same experiment (Sec. 6.4.3).

6.4.1 Cross Traffic and No Losses

In this experiment, we introduce cross traffic at the BSs and ensure that no losses happen by giving adequate queue sizes at the base station. Note that the link capacities here are 1000kbps on each interface. The number of flows that constitute cross traffic during the course of the simulation is 3 ftp and 16 web flows at BS0, 5 ftp and 24 web flows at BS1 and 6ftp and 20 web flows at BS2. These number of flows for the cross-traffic are merely to illustrate the behaviour – we consider various other settings in Sec. 6.4.3.

Figure 6.2 shows the variation in the instantaneous TCP throughput. We compare WRR and PET scheduling, both with BMP implemented at the client. We compare these with the MTCP application-level solution for bandwidth aggregation. We see that PET-BMP follows MTCP very closely, whereas WRR-BMP lags behind by a big margin. The average throughput obtained by the main TCP flow in comparison to MTCP, PET-BMP, and WRR-BMP are 967.6, 960, and 589 kbps respectively. This illustrates that PET-BMP is able to meet the goal of effective bandwidth aggregation in this setting.

Let us now consider losses but no cross traffic.

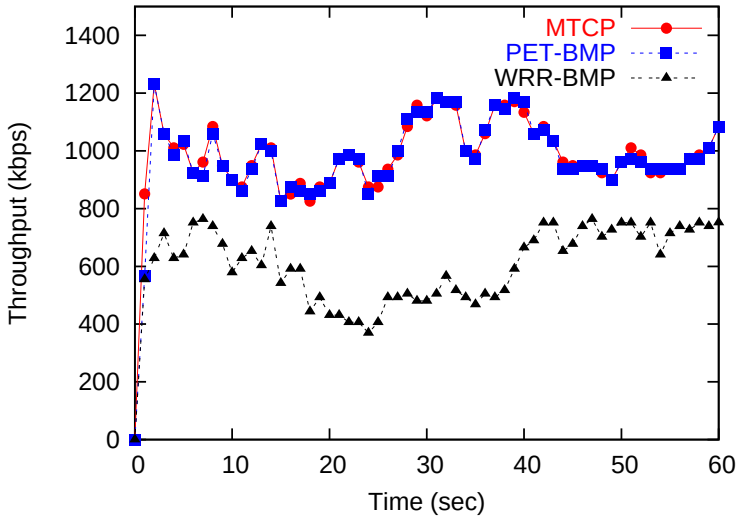


Figure 6.2: TCP Performance: Cross Traffic, No Losses

6.4.2 No Cross Traffic and Losses

As mentioned in Sec. 6.1, we now use 50, 100, and 200 kbps for the link capacities. Instead of introducing random drops, we control the packets that were dropped, so as to explain the behavior of PET-BMP better. We drop a total of 10 packets (this suffices to illustrate the comparative behaviour of PET-BMP). We ensure that there is only one drop per congestion window for the first 5 packets dropped. Later we drop 2 packets per congestion window and then 3.

The throughput achieved by PET-BMP in this case was 330.2kbps, a decrease of 6.6kbps from the no loss case (refer to Table 6.1). The number of retransmissions were 15 - five more than what was needed to recover from losses. Note that such unnecessary retransmissions in case of PET-BMP happen only in response to losses unlike in WRR, where they happen on a regular basis due to reordering. Comparing with ASL, the throughput achieved by ASL for same drop pattern was 338.2, 1.4kbps lesser than the no loss case. For ASL, the number of retransmissions were 10, equal to the number of dropped packets.

The reason for more retransmissions in case of PET-BMP is the following. When a packet is detected lost by the TCP sender, on receipt of 3 duplicate

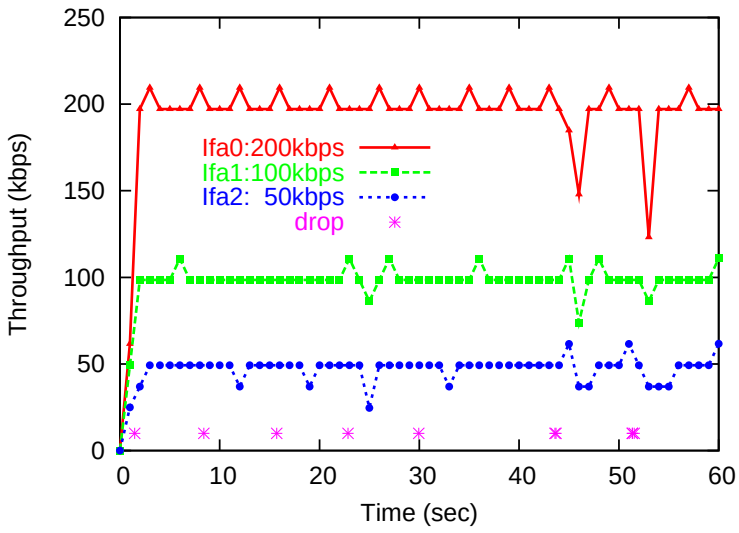


Figure 6.3: TCP Performance: No Cross Traffic, Losses

ACKs, it retransmits the packet and enters fast recovery. It is possible that the retransmitted packet may arrive before other outstanding packets when fast recovery was entered. In this case, when the ACK generated by the retransmitted packet arrives, the TCP sender considers the packet immediately following the acknowledged packet as lost and retransmits it. However as we can see, the drop in throughput is small which shows that PET-BMP is able to react to the losses and recover from them in a timely fashion. If we had depended on a timeout in the BMP to react to the losses, the decrease in throughput would have been much higher.

The drop in throughput of PET-BMP in comparison with ASL is due to an important factor - lack of adequate loss isolation. Fig 6.3 shows the throughput achieved by PET-BMP on the 3 interfaces (ifa-0, ifa-1 and ifa-2) along with the time of dropped packets. As we can see in the figure, the first drop does not affect any interfaces. The next two drops affect ifa-2 but not the other two. The 4th drop affects ifa-2 and ifa-1, but not ifa-0. However, when more than one drop happen per congestion window (the drops after 50 sec), all interfaces suffer. This demonstrates that isolation of losses may be possible with PET-BMP when losses

are spread out, but is difficult to achieve when losses are clustered.

6.4.3 Cross Traffic and Losses

We finally perform an experiment where we consider both cross traffic as well as losses. For this experiment, we randomly generate 10 different cross traffic scenarios. For each scenario, we randomly choose a value between 2 and 8 for the number of FTP flows and a value between 16 and 32 for the number of web flows at a BS (normally, due to randomness, cross traffic profiles on the interfaces are different, introducing asymmetry). This range of possible cross-traffic covers a range of scenarios of the available bandwidth for the main flow on each of the wireless bottleneck links. Note that not all flows are simultaneously active at any given time.

In each "scenario", the start times and file sizes for the cross traffic varies dependent on a random number "seed". So, for each cross traffic scenario, we conduct 10 different runs with different seeds and average the throughput seen by the main TCP flow across the seeds. This run across different seeds ensures averaging across various start/finish times of the cross-traffic.

We consider two types of losses - congestion and channel errors. We present results when considering just congestion losses and also when channel losses are introduced on top of congestion losses. For the second case, we use the same traffic pattern as was used for the case of congestion losses. For introducing congestion losses, we set the maximum queue size at the BSs to 200 kbytes. The distribution used for introducing channel losses is uniform, with a loss rate of 1%.

We first present results for the case of 2 interfaces and then increase the number of interfaces to 3 to show the effect of increased reordering (more interfaces, more scope for reordering) on the performance of PET-BMP.

Fig. 6.4 compares the throughput achieved by the different algorithms when considering just congestion losses for the case of 2 interfaces. Fig. 6.5 shows the throughput when channel losses are introduced on top of congestion losses. In

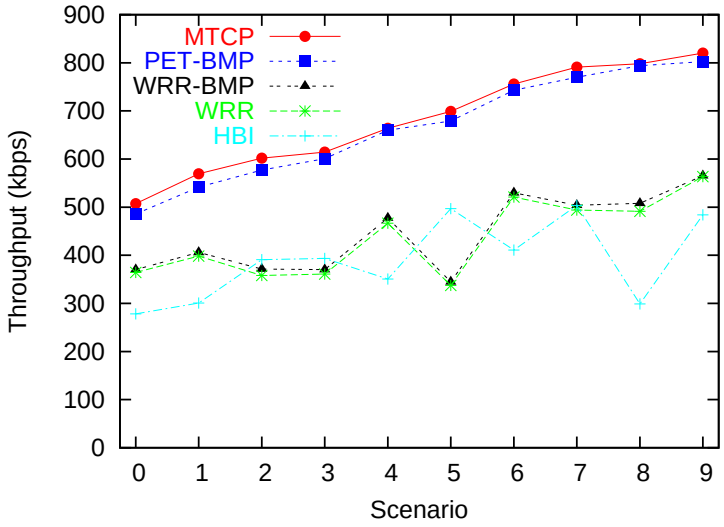


Figure 6.4: TCP Performance: Congestion Losses, No Channel Losses (Interfaces = 2)

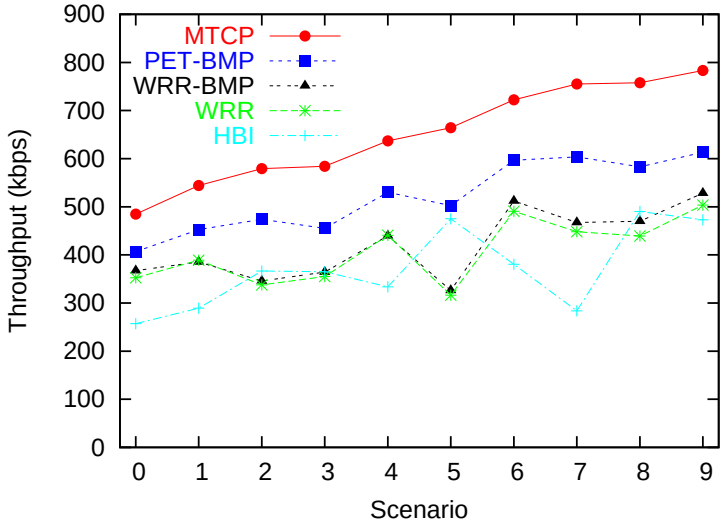


Figure 6.5: TCP Performance: Congestion Losses, Channel Losses (Interfaces = 2)

these figures, we also show the throughput of MTCP as seen on just the highest bandwidth interface (HBI)– this is what would have been TCP's throughput had we done no bandwidth aggregation and simply used just the highest bandwidth interface. For ease of visualization, we sort the 10 scenarios in the order of increasing bandwidth achieved by MTCP.

When considering congestion losses alone, we find that PET-BMP performs very closely with MTCP (the difference ranges between 4-27 kbps). This is true across the wide range of cross-traffic scenarios we have considered. In contrast, WRR lags behind PET-BMP and MTCP considerably. The use of BMP alone with WRR brings in some benefits, but not a whole lot. Compared to the case of using just the Highest Bandwidth Interface, PET-BMP clearly illustrates the advantages of bandwidth aggregation. WRR-BMP performs better than HBI in some cases, while in others it performs worse. This shows that if care is not taken when scheduling to minimize reordering, effects of bandwidth aggregation could be nullified.

When channel losses are considered in addition to congestion losses, PET-BMP performs better than WRR-BMP, and can still bring in considerable benefits over using just the highest bandwidth interface (HBI). However, it falls behind MTCP by a larger margin (78-174 kbps) than with just congestion losses. This is mainly due to the aggressive behavior of MTCP during losses (multiple congestion windows) and also due to the inability of PET to achieve loss isolation.

Though PET-BMP performs sub-optimally in the presence of channel errors, we argue that this is not much of a problem due to the following reason. The scheduling algorithm PET can achieve loss isolation (similar throughputs as MTCP) as long as loss rates are low and losses are more spread out. In the above experiments, while congestion losses were under 0.7%, additional of channel losses increased this percentage to 1.4%. This degradation in throughput of PET-BMP compared to MTCP is mainly due to this increase in loss rate. It does not matter to PET (or to TCP) whether losses are due to congestion or channel errors as long

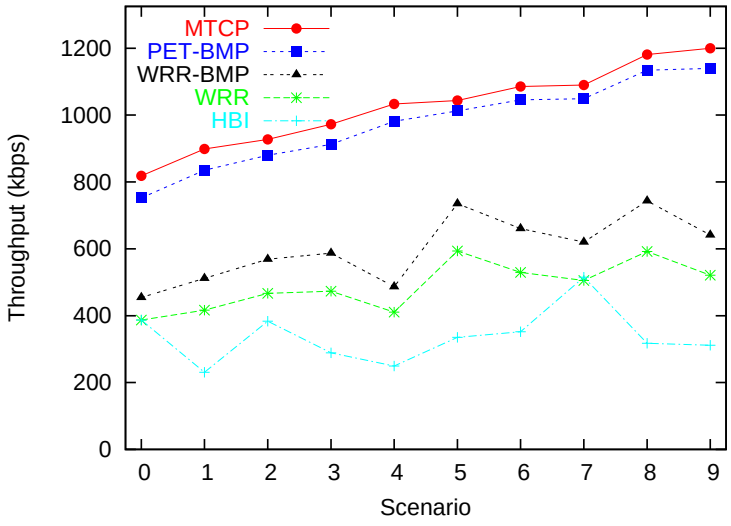


Figure 6.6: TCP Performance: Congestion Losses, No Channel Losses (Interfaces = 3)

as these rates are not too high. Ideally TCP should react to only congestion losses not channel errors. Considerable research [10] has gone into undoing the effect channel errors have on the congestion window of the TCP sender. With some of these mechanisms in place (e.g. through use of ELN/ECN bit), we can expect PET-TCP to perform similar to MTCP, as was seen in the no-channel-loss case in Fig. 6.4. As far as congestion losses are concerned, the loss rates should not reach high values because of TCP default behavior of cutting down its sending rate in response to congestion. The performance of PET-BMP under channel errors with corrective mechanisms in place is a topic of future study.

Fig. 6.6 shows the performance of the different algorithms for the case of 3 interfaces with just congestion losses. As the number of interfaces increase, so does the scope for reordering. As can be seen, the difference between PET-BMP and MTCP is wider than in the case of 2 interfaces. The difference ranges between 30-60 kbps now.

Another point worth mentioning here is the amount of reordering observed. We have calculated the average and maximum buffer occupancy for the

duration of the experiment for various scenarios. In case of PET-BMP, the average buffer size was 7.3 packets and maximum buffer size 32 packets. In case of WRR-BMP, average buffer size was 16.1 packets and maximum buffer size 24 packets. Though the average buffer size in case of PET-BMP is lower than that of WRR-BMP, we cannot quite compare the results directly because the traffic pattern as governed by TCP (which depends on the bandwidth aggregated by the scheduling algorithms, losses etc) is not the same across the two cases. The maximum buffer size (and average buffer size to some extent) is rather high because in some cases, BMP has to timeout to identify a loss from reordering, thereby buffering a whole lot of packets.

6.4.4 Summary of Results

The above experimental results indicate that PET-BMP can bring in significant benefits through bandwidth aggregation over using just a single interface. It performs as well as the application level MTCP solution and outperforms by a large margin approaches based on using Weighted-Round-Robin in most scenarios. It achieves this through meeting the design criteria in Sec. 6.2 – all except "isolation of losses". While, it can still perform close to MTCP, when the loss rates are low, higher loss rates degrade its performance due to its inability to perform "loss isolation" and due to inherent aggressiveness of MTCP. However, when wireless losses are minimized using other mechanisms (e.g. [10]), the performance of PET-BMP can become comparable to MTCP.

For the range of scenarios we have considered, the estimation techniques used by PET are effective, and we have not found the need for any parameter tuning – PET is simple enough to be robust in this regard. Given the ease of deployment of PET-BMP and the performance gains for effective bandwidth aggregation, we believe that it has wide applicability.

6.5 Discussion

In this section, we elaborate on the validity of some of our assumptions, and other issues with our network layer approach.

6.5.1 Validity of Assumptions

Two assumptions that feature in the above work are (a) WFQ implementation at the BSs, and (b) proper estimation of delay on the paths from proxy to BSs.

Unlike First-Come-First-Serve implementation, WFQ implementation or other variants of it, divide the link capacity equally among all the flows and thereby help bandwidth estimation techniques in getting a reliable estimate. Though the scheduling policy employed at a BS is not in our control, we believe that WFQ is a good design choice for a variety of reasons and should be adopted at the edge Radio Access Networks (RANs). For one, it ensures fair allocation of the already scarce wireless capacity to the different flows. It reduces the complexity of bandwidth monitoring tools employed by end users, or by the network operators to monitor link utilization. Different protocols can benefit from good bandwidth estimation to improve their performance. For example, bandwidth estimates can help (regular, single interface) TCP tune its optimal window size. Since the number of flows at the edge is anyway small, the scalability of WFQ is not much of an issue.

We now turn to the issue of delay estimates. 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 don't 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 [6]. Further, any errors in estimation 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. In Equation 5.1 of Section 5.1.1, A_l dominates $a_i + D_l$ for most packets. We observed this in our experiments as well.

Another choice we made when running the experiments is to disable the use of delayed ACKs. This ensured that during slow start packets always come in pairs at the proxy. If this option is enabled, we still get back to back packets but with less frequency and that number can be greater than 2. Our scheme can be extended to work in this case too but the number of samples we collect for bandwidth estimation can go down. This design possibility is worth further study.

6.5.2 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. As was explained in Chapter 3, the deployment complexity of our architecture is minimal.

As far as the complexity of algorithms go, we note that the implementation of BMP at the MH or PET at proxy is unlikely to be a source of major overhead in terms of memory[5] or CPU requirements. Further, although we have presented BMP as a network layer approach, there is no reason why it cannot be integrated into the TCP receiver. This does not need many changes to the infrastructure, only MHs who want to take advantage of bandwidth aggregation only need apply this patch. This can further reduce some of the state that needs to be maintained at the network layer, which TCP receiver already does.

There is a source of network overhead in PET-BMP – the need to send

[5]Maximum buffer size in BMP is at most the size of TCP congestion window, usually under 128 kbytes

a SIG-INFO to proxy for every packet received at the MH. Though this doubles the load in the reverse direction, the additional bandwidth needed is very small as the size of these packets is very small. Even if the wireless links are asymmetric in nature (uplink has much lower bandwidth than downlink), given that we have choice of more than one interface, there will normally be enough left-over bandwidth to send these packets. In the event this were not the case because of heavy uplink traffic, it is possible to minimize the overhead by performing bandwidth estimation at the client and passing information to the proxy only in the event of a considerable increase or decrease in bandwidth. This possibility needs further evaluation.

6.5.3 Miscellaneous issues

An important aspect when performing bandwidth aggregation is to ensure how friendly a TCP flow that uses bandwidth aggregation is to others that don't. Since, we did not make any changes of TCP, it reacts to losses the same way as the other flows and hence bandwidth aggregation does not interfere with other flows. On the contrary, approaches based on opening multiple TCP sockets as in [24, 34] may be too aggressive in face of losses.

An important observation to make is that our network layer solution preserves the semantics of the IP service model, and does not violate the end-to-end design principle. Our solution delays or drops TCP packets, both of which IP is allowed to do in its service model.

This Chapter is in part a reprint of the material in the paper: K. Chebrolu, B. Raman and R. R. Rao, *A Network Layer Approach to Enable TCP over Multiple Interfaces*, Journal of Wireless Networks (WINET) (Accepted). The dissertation author was the primary investigator of this paper.

Chapter 7

Conclusions and Future Directions

In this dissertation, we have dealt with the simultaneous use of multiple interfaces. Such use opens new ways of solving some of the limitations of the wireless media and enables other new and interesting possibilities. We term the services enabled by such simultaneous use of multiple interfaces as multi-access services. Examples of such services include: bandwidth aggregation, mobility/reliability support, resource sharing and data-control plane separation. In this chapter, we summarize the contributions of our work in realizing these services as well as identify directions for future research.

7.1 Summary of Contributions

Traditional wireless research has mostly focused on the use of a single interface at any given time to meet the connectivity requirements of the end user applications. With simultaneous use of multiple interfaces arises the need to define a new architecture that enables simultaneous multi-path communication. In this work, we begin with a general framework in the form of such an architecture.

One of the main goals in the design of the architecture is that it introduce minimal changes to the infrastructure. Accordingly, we design an architecture that

operates at the network layer based on the principle of tunneling. Our architecture consists of an infrastructure proxy that provides services to a set of mobile clients equipped with multiple interfaces. The only changes needed in our architecture are software changes at the MH and the deployment of the proxy. Apart from the design of the architecture, we also identify the various functional components that are needed for providing different multi-access services. We implement most of these components on a testbed as Linux loadable kernel modules as proof of concept for the different services.

While the architecture can support many different services, we explore in depth one such service provided by the architecture: Bandwidth Aggregation (BAG). We look at this service in the context of *Real-time Video* and *TCP* applications.

Aggregating bandwidth across multiple interfaces can be used to improve the raw throughput of the clients' applications. However, it introduces challenges in the form of packet reordering. In case of interactive video applications, the excess delay resulting from packet reordering is often equivalent to packet loss. With respect to TCP applications, the duplicate ACKs generated on packet reordering are misinterpreted by the TCP sender as indicative of packet loss and congestion control is invoked. This can significantly lower TCP throughput and counter any gains that can be had through bandwidth aggregation.

An important aspect of the architecture when providing BAG services for video applications is the scheduling algorithm that partitions the traffic onto different interfaces such that the QoS requirements of the application are met. In this context, we propose the Earliest Delivery Path First (EDPF) scheduling algorithm that has the explicit purpose of minimizing delay experienced by the packets. EDPF schedules packets on the path which it estimates introduces the least possible delay. 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 of BAG for streaming video applications carried on our testbed show the performance improvement BAG with EDPF scheduling can offer over using just the Highest Bandwidth Interface (HBI). For example, for a buffer time of 2 sec, EDPF achieves a frame loss of 0.04% while HBI still suffers a high 6.6% frame loss. Streaming applications that support VCR functions require one way delays in the range of 1-2 sec. This shows that if less than 1% frame loss is required for acceptability (which is often typical), BAG can support these applications, while using just a single interface cannot.

Apart from streaming video applications, we also conducted extensive simulations using video and delay traces for interactive applications. Even here, EDPF scheduling performs close to ASL and outperforms by a large margin other approaches based on weighted round robin based. For example, for the MPEG-4 Lecture video trace considered, 99.8% of the packets experienced delay less than 200ms for ASL. In case of EDPF, this value was between 99.2% to 99.6%, where as for SRR, it could vary as low as 56.5% to 99.2%.

Given the scarcity of bandwidths in wireless environments, it may not always be possible to aggregate adequate bandwidth to support the QoS requirements of demanding video applications. In these circumstances, we propose a content adaption algorithm in the form of selective frame discard to provide some base quality of video. Our algorithm MC-DROP, drops frames based on the impact the frame drop has on meeting future frame deadlines and hence on overall quality of the video. Our trace driven experiments show that attempting to transmit every frame results in severe performance degradation, e.g. resulting PSNR can get as low as 9.8 dB from the original 27.16 dB. Our proposed algorithm MC-DROP, can bring the quality all the way up to 26.4 dB.

In addition to video application, we also experiment with BAG for TCP applications. Since most TCP applications are best-effort based, we extend the EDPF scheduling algorithm to perform bandwidth estimation based on packet pair technique. The new scheduling algorithm Packet Pair based Earliest-Delivery-

Path-First Scheduling Algorithm for TCP applications (PET) attempts to minimize reordering by sending packet pairs on the path that introduces the least amount of (estimated) delay. Irrespective of the care taken in scheduling, some amount of reordering is inevitable, so we propose a buffer management policy (BMP) at the client to hide any residual reordering from TCP. Simulations carried on an NS-2 platform show that PET in combination with BMP achieves good bandwidth aggregation for TCP applications under a variety of network conditions.

7.2 Directions for Future Work

In the context of enabling multi-access services, there are several avenues for future work. We briefly describe these below.

7.2.1 Multiple Application Interaction

In this work, we have mostly focused on a single application that stripes data onto multiple interfaces. The client, however can have multiple applications running that need to share the available bandwidth of the interfaces among themselves so that each gets a fair share of the bandwidth. This can potentially be achieved by combining a fair queuing algorithm such as WFQ [16] with our scheduling algorithm EDPF. The actual interaction in such a scenario needs careful theoretical and experimental evaluation.

7.2.2 Different Radio Access Networks

The underlying characteristics of the different Radio Access Networks (RANs) can be very different, while some are best-effort based, others can provide QoS guarantees. In this work, we considered only similar type interfaces for supporting an application – all QoS based for interactive video and all best-effort based for TCP. To support applications (especially video) over interfaces that are a mix of QoS enabled and best-effort based, we would need to come up with new schedul-

ing algorithms. In addition, a study of frame-drop policies akin to MC-DROP in such a scenario would also be required. It would be interesting to evaluate the performance of the applications over such scenarios.

7.2.3 Other Multi-Access Services

In this work, we have mostly focused on the BAG multi-access service. The other multi-access services such as resource sharing and data-control plane separation need careful design of necessary scheduling and buffer management algorithms. They would also require distributed protocols to convey necessary information (performance statistics on the different interfaces, routing etc) across hosts since we are dealing with more than a single host. The design and implementation of the various architectural components also need further study.

We believe that the architecture developed in this dissertation, and our design methodology form a good basis to experiment with some of these new ideas.

Appendix A

Appendix: 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

Case1: 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: $L_{max} \sum_{l \neq lb} B_l/B_{min} = L_{max}(\sum_{l=1}^{N} w_l - 1)$.

Case2: 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,

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

Case2: 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 $l \neq lb$, $T_l(a_j) - T_{l_j}(a_j) = L_{max}/B_{min}$ (from lemma 1). Hence $P = \sum_{l=1}^{N} T_l(a_j) - T_{l_j}(a_j) \leq L_{max}(\sum_{l=1}^{N} w_l - 1)$.

- 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, $d_i = T_{l_i}(a_i+) = max\{T_l(a_i+)$, 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 (case1), 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,

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

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

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 4 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 (case1), 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 $l(\neq hb)$ and the i^{th} 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 $r_{i-1} \geq d_j$. Therefor, $J_i \leq d_i - r_{i-1} \leq d_i - d_j = L_i/B_{max}$.

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 (case1), 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 proved.

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

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. If we denote by r_i, the display time of packet i, at this instant the buffer will contain packets $j \geq i$ since the previous packets have already been removed for display. Let d^i_{min} be the minimum of the delivery times (at client) of all the packets in the buffer at time r_i. Since packets can arrive out of order due to multiple paths, the packet that corresponds to d^i_{min} may not equal i. Let $\tau_i = r_i - d^i_{min}$. This quantity cannot exceed $(L - D)$, as otherwise packet i would miss its playback deadline. During the interval τ_i, the

buffer can fill at most at a rate of B, so the size of the buffer cannot exceed $\tau_i * B$. So the maximum buffer capacity is given by $C = B(L - D)$.

Bibliography

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

[2] Network simulator. ns2. `http://www.isi.edu/nsnam/ns`.

[3] Web traffic generator. `http://www.isi.edu/nsnam/ns/ns-contributed.html`.

[4] Inverse multiplexing for ATM (IMA) specification, version 1.1, ATM forum doc. AF-PHY-0086.001, 1999.

[5] Location of MPEG-4 temporal video traces. `http://trace.eas.asu.edu/cgi-bin/main.cgi`, 2003.

[6] A. Acharya and J. Saltz. A study of internet round-trip delay. Technical Report CS-TR 3736, Univ. of Maryland, College Park, 1996.

[7] H. Adiseshu, G. Parulkar, and G. Varghese. A reliable and scalable striping protocol. *ACM Computer Communication Review*, 26(4):131–141, Oct 1996.

[8] J. Aweya, M. Ouellette, and D. Y. Montuno. A self-regulating TCP acknowledgment (ACK) pacing scheme. *International Journal of Network Management*, 12(3):145–163, May/June 2002.

[9] H. Balakrishnan, V.Padmanabhan, and R.Katz. The effects of asymmetry on TCP performance. *Mobile Networks and Applications*, 4(3):219–241, Oct 1999.

[10] H. Balakrishnan, V.Padmanabhan, S.Sheshan, and R.Katz. A comparision of mechanisms for improving TCP performance over wireless links. *IEEE/ACM Transactions on Networking*, 5(6):756–769, Dec 1997.

[11] E. Blanton and M. Allman. On making tcp more robust to packet reordering. *Computer Communication Review*, 32(1):20–30, Jan 2002.

[12] D. Brudnicki. Third generation wireless technology. `http://www.seasim.org/archive/sim102001.pdf`.

[13] P. Calhoun, J. Loughney, E. Guttman, G. Zorn, and J. Arkko. Diameter base protocol. RFC 3588, September 2003.

[14] A. Campbell, J. Gomez, S. Kim, Z. Turanyi, C-Y Wan, and A. Valko. Comparison of IP micromobility protocols. *IEEE Wireless Communications*, 9(1):72–82, Feb 2002.

[15] K. Chebrolu and R. R. Rao. Communication using multiple wireless interfaces. In *Proc. IEEE WCNC'02*, Orlando, March 2002.

[16] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queuing algorithm. In *Proc. ACM SIGCOMM'89*, pages 1–12, Austin, Texas, September 1989.

[17] J. Duncanson. Inverse multiplexing. *IEEE Communications Magazine*, 32(4):34–41, Apr 1994.

[18] K. Egevang and P. Francis. The IP network address translator (NAT). RFC 1631, May 1994.

[19] K. Fall and S. Floyd. Simulation-based comparisons of Tahoe, Reno and SACK TCP. *Computer Communication Review*, 26(3):5–21, Jul 1996.

[20] W. C. Feng and J. Rexford. A comparison of bandwidth smoothing techniques for the transmission of prerecorded compressed video. In *Proc. INFOCOM'97*, pages 58–66, Kobe, Japan, April 1997.

[21] B. Girod, M. Kalman, Y.J. Liang, and R. Zhang. Advances in channel-adaptive video streaming. *Wireless Communications and Mobile Computing*, 2(6):549–552, Sep 2002.

[22] A. Gurtov and R. Ludwig. Lifetime packet discard for efficient real-time transport over cellular links. *ACM Mobile Computing Communications Review*, 7(4):32–45, Oct 2003.

[23] M. Hemy, U. Hengartner, P. Steenkiste, and T. Gross. MPEG system streams in best-effort networks. In *Proc. Packet Video'99*, New York, April 1999.

[24] H. Hsieh and R. Sivakumar. A transport layer approach for achieving aggregate bandwidths on multi-homed mobile hosts. In *Proc. ACM MOBICOM'02*, Atlanta, September 2002.

[25] V. Jacobson. Modified TCP congestion avoidance algorithm. end2end-interest mailing list, `ftp://ftp.ee.lbl.gov/email/vanj.90apr30.txt`, April 1990.

[26] S. Keshav. A control-theoretic approach to flow control. *Computer Communication Review*, 21(4):3–15, 1991.

[27] L. Kleinrock, editor. *Queueing Theory*. Wiley, New York, 1975.

[28] J. Lee, D. Gunter, B. Tierney, B. Allcock, J. Bester, J. Bresnahan, and S. Tuecke. Applied techniques for high bandwidth data transfers across wide area networks. In *Proc. CHEP'01*, Beijing, China, September 2001.

[29] S. Lu, V. Bharghavan, and R. Srikant. Fair scheduling in wireless packet networks. *IEEE/ACM Transactions on Networking*, 7(4):473–489, Aug 1999.

[30] L. Magalhaes and R. Kravets. Transport level mechanisms for bandwidth aggregation on mobile hosts. In *Proc. IEEE ICNP'01*, Riverside, November 2001.

[31] C. E. Perkins. Mobile IP. *IEEE Communications Magazine*, 35(5):84–99, May 1997.

[32] 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 *Proc. IEEE INFOCOM'02*, pages 773–781, New York, June 2002.

[33] M. Reisslein, J. Lassetter, S. Ratnam, O. Lotfallah, F. H. P. Fitzek, and S. Panchanathan. Mpeg-4: Single layer, temporal and spatial scalability with PSNR values. `http://trace.eas.asu.edu/pub.html`, 2002.

[34] H. Sivakumar, S.Bailey, and R.L.Grossman. Psockets: The case for application-level network striping for data intensive applications using high speed wide area networks. In *Proc. IEEE Supercomputing'00*, Dallas, November 2000.

[35] K. Sklower, B. Lloyd, G. McGregor, D. Carr, and T. Coradetti. The PPP multilink protocol (MP). RFC 1990, Aug 1996.

[36] S.Ostermann, M. Allman, and H. Kruse. An application-level solution to TCP's satellite inefficiencies. In *Proc. WOSBIS'96*, Rye, November 1996.

[37] M. Stemm and R. Katz. Vertical handoffs in wireless overlay networks. *Mobile Networks and Applications*, 3(4):335–350, 1998.

[38] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, and V. Paxson. Stream control transmission protocol. RFC 2960, October 2000.

[39] M. Zhang, B. Karp, S. Floyd, and L. Peterson. Improving TCP's performance under reordering with DSACK. Technical Report TR-02-006, International Computer Science Institute, Berkeley, Jul 2002.

[40] Z.-L. Zhang, S. Nelakuditi, R. Aggarwal, and R. Tsang. Efficient selective frame discard algorithms for stored video delivery across resource constrained networks. In *Proc. INFOCOM'99*, New York, March 1999.