Technical Report

Department of Computer Science
and Engineering
University of Minnesota
4-192 EECS Building
200 Union Street SE
Minneapolis, MN 55455-0159 USA

TR 01-049

Localized Approach to Providing Quality-of-Service

Srihari Nelakuditi

October 30, 2001

UNIVERSITY OF MINNESOTA

This is to certify that I have examined this bound copy of a doctoral thesis by

Srihari Nelakuditi

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Zhi-Li Zhang and David H.C. Du

Name of Faculty Advisers

Signature of Faculty Advisers

Date

GRADUATE SCHOOL

Localized Approach to Providing Quality-of-Service

A THESIS
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Srihari Nelakuditi

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Zhi-Li Zhang and David H.C. Du
Faculty Advisers

December 2001

Dedicated to my motherland Medikonduru

Abstract

The exponential growth of Internet brings to focus the need to control such large scale networks. It is a challenge to regulate such a complex network of heterogeneous elements with dynamically changing traffic conditions. To make such a system reliable and manageable, the decision making should be decentralized. It is desirable to find *simple local rules and strategies* that can produce *coherent and purposeful global* behavior. Furthermore, these control mechanisms must be adaptive to effectively respond to continually varying network conditions. Such adaptive, distributed, localized mechanisms would provide a scalable solution for controlling large networks. The need for such schemes arises in a variety of settings. In this thesis, we focus mainly on quality-of-service based routing. However, we also develop scalable solutions in the following contexts: stored video delivery across resource constrained networks; channel assignment and admission control in wireless cellular networks.

In quality-of-service (QoS) based routing, paths for flows are selected based upon the knowledge of resource availability at network nodes and the QoS requirements of flows. Several QoS routing schemes have been proposed that differ in the way they gather information about the network state and select paths based on this information. These schemes can be broadly categorized into *best-path routing* and *proportional routing*. The best-path routing schemes gather *global* network state information and always select the best path for an incoming flow based on this global view. It has been shown that the best-path routing schemes require frequent exchange of network state, imposing both the communication overhead on the network and processing overheads on core routers. On the other hand, the proportional routing schemes proportion incoming flows among a set of candidate paths. We show that it is possible to compute near-optimal proportions using only *locally* collected information. Furthermore, a few good candidate paths can be selected using infrequently exchanged global information and thus with minimal communication overhead. In this thesis, we study these schemes extensively and demonstrate that proportional routing schemes can achieve higher throughput with lower overhead than best-path routing schemes.

Delivery of a video for full quality playback requires a certain amount of network bandwidth and client buffer resources. But when these resources are limited, a naive transmission of video may cause packet drops at the network and frame drops at the client, resulting in wastage of resources. To avoid this, a server may need to *preemptively discard frames locally* taking advantage of application-specific information. We develop an algo-

rithm to find, given network bandwidth and client buffer constraints, the minimum number of frames that must be discarded in order to meet these constraints. We formulate the problem of optimal *selective frame discard* using the notion of a cost function and present several efficient heuristic algorithms. Furthermore, we extend the selective frame discard algorithms for layered video and develop adaptive *selective layer discard* algorithms. We apply these algorithms for the case of stored video delivery over bandwidth-varying channel and show that they help provide smoother quality video playback.

Distributed dynamic channel assignment algorithms run at each base station in a wireless cellular network attempt to reduce the network-wide call blocking and call dropping probabilities while making assignment decisions based on its neighborhood information only. They may also reassign channels being used by calls in progress to make room for a new request. We propose two new channel selection strategies based on *local packing* for compact packing of channels and show that these schemes either carry more traffic or reduce the number of relocations needed. Further, call admission control schemes in mobile cellular networks reserve some guard channels for handoffs and thus block new calls in order to ensure continuation of service to already admitted calls. Improper selection of guard channels can result in either forced termination of on-going calls or low spectrum utilization. We present a reassignment based call admission control scheme that dynamically adjusts the number of guard channels based on *reassignment frequency in the neighborhood*. This scheme attempts to maximize revenue by balancing the penalty for reassignments against the reward for serviced calls while ensuring continuity of service to all admitted calls.

Acknowledgments

I thank my advisors Zhi-Li Zhang and David Du for their guidance and support throughout my graduate study. I am grateful to David Du for accepting me as a member of his research group and initiating me into the area of computer networking. I would like to especially thank Zhi-Li Zhang for being patient with me during my long stay at the university. He has been both a friend and a mentor to me and I consider myself extremely fortunate to have him as my advisor.

I would also like to thank members of my thesis committee Anand Tripathi and David Lilja for their valuable feedback on my work. The courses they have taught on simulation and performance analysis techniques aided me well in my research. I also wish to thank Rose Tsang for supporting me and giving me opportunity to work with her at Sandia National Laboratories in Livermore. The collaboration with her gave a definite direction and shape to this thesis.

Many people have helped me at various stages of my dissertation. In particular, I would like to thank Raja Harinath who was always available for discussions and gave invaluable suggestions whenever I was stuck with a problem. I also thank my colleagues Yen-Jen, Weihsiu, Rahul, Tai-Sheng, Vatsan, Sanjai, Hung, Sejun, Ewa, Baek Young, Yingfei, Zhenhai, James and Jaideep for providing a friendly and conducive environment for research.

I have made many wonderful friends at the university. My roommates Dsu and Majhi broadened my horizons and helped me appear a little less uncivilized. Kota and Koka have been like family members to me. Sanjai and Vatsan sheltered me during the toughest of times. Archana and Sanjai accompanied me during many of my late night travails. Madhu and Kotam offered me spiritual guidance and perspective on life.

Quite a few friends (Sandeep, Guru, Mandu, Bibhu, Karuna, Sathyan, Sowmi, Venu, Davis, Jayesh, Anand, Mukund, Upen, Harita, Sujal, Swethal, Prem, Raghu, Rami, Inturi and many others) cheered me towards graduation. Special thanks to Phani for encouraging me to apply and join this university. Finally, I like to thank all those who played volleyball, football, soccer, badminton, broom ball etc. and made my stay at the university enjoyable and memorable.

Contents

Chapter 1

Overview

Large scale networks call for scalable solutions to make them reliable and manageable. The central theme of this dissertation is how to design *localized mechanisms* that produce desired *global behavior*. Such mechanisms that use simple local rules and strategies can provide scalable solutions for controlling large networks. The need for such schemes arises in a variety of settings. In this thesis, we focus mainly on quality-of-service (QoS) based routing. However, we also develop scalable solutions in the following contexts: stored video delivery across resource constrained networks; channel assignment and admission control in wireless cellular networks.

1.1 Localized Adaptive Proportioning Approach to QoS Routing

In QoS routing, some knowledge regarding the global network state is crucial in performing judicious path selection. This knowledge, for example, can be obtained through periodic information exchange among routers in a network. Under this approach, which we refer to as the *best-path* routing approach, each router constructs a global view of the network (QoS) state by piecing together the QoS state information obtained from other routers, and selects the "best path" for a flow based on this global view of the network state. Best-path routing schemes work well when each source node has a reasonably *accurate* view of the network QoS state. However, as the network resource availability changes with each flow arrival and departure, maintaining an accurate network QoS state is impractical, due to the prohibitive communication and processing overheads entailed by frequent QoS state information exchange. In the presence of *inaccurate* information regarding the global network

QoS state, best-path routing schemes suffer degraded performance as well as potential instability.

As a viable alternative to the best-path routing approach, we propose a novel *localized proportional routing* approach to QoS routing. Under this approach, instead of (periodically) exchanging information with other routers to obtain a global view of the network QoS state, a source router attempts to *infer* the network QoS state from *locally collected flow statistics* such as flow arrival/departure rates and flow blocking probabilities, and performs adaptive proportioning of flows among a set of *candidate* paths based on this local information. Under pure localized approach, the candidate path set remains static while their proportions are adjusted dynamically. A network node under localized approach can judge the quality of paths only by routing some traffic along them. So it is not possible to update the candidate path set based on local information alone. On the other hand, due to changing network conditions, a few good paths cannot be selected statically. Hence we propose a path selection procedure that dynamically selects a few good candidates based on infrequently exchanged global information. The inaccuracy in candidate path selection is cushioned by adaptively proportioning traffic among candidates. In this thesis, we first address the issue of *finding near-optimal proportions* for a given a set of candidate paths based on locally collected flow statistics. We then look into the *selection of few good candidate paths* based on infrequently exchanged global information.

The above discussion assumes that each router in the network is aware of the topology and the state of the whole network. This is referred to as *flat* routing where each router participates in link state updates and maintains detailed information about the entire network. This introduces significant burden on every router and as the size of the network grows, the overhead at each router increases tremendously. To provide a scalable solution, *hierarchical routing* is suggested as an alternative to flat routing. Under hierarchical routing, a network is divided into multiple areas. The routing within the area is flat with each router having detailed information about routers and links in that area. But the routers have only sketchy *aggregate* information about other areas. To route traffic destined for other areas, a source router may select a partial higher level path, based on the aggregate information, that gets expanded, based on the detailed information, at the ingress border router of each area along the path. Such a hierarchical routing reduces the overhead at each router by limiting the scope of link state updates and maintaining only summary information about other areas. The hierarchical routing approach while reduces the burden on a router, introduces inaccuracy in the information available for routing. Hence the performance of a hierarchi-

cal routing scheme depends heavily on how information about an area is aggregated and how it is utilized in routing across areas. This thesis extends the proportional routing approach to hierarchical routing. It addresses the issue of how to perform *topology and state aggregation under multipath routing* and *path selection based on aggregate information*.

1.2 Stored Video Delivery across Resource Constrained Networks

Video delivery from a server to a client across a network is an important component of many multimedia applications. Stored video typically has high bandwidth requirements and exhibits significant rate variability. In a network where resources such as the network bandwidth and buffering capacity are constrained, it is a major challenge to design an efficient stored video delivery system that can achieve high resource utilization while maximizing users' perceived quality-of-service (QoS). Video smoothing techniques have been proposed for reducing the network bandwidth requirement of bursty video streams by taking advantage of client buffering capabilities. Similar techniques have also been developed when network bandwidth is constrained instead of the client buffer. In reality, however, both network bandwidth and client buffering capacity are likely to be limited. Under such circumstances, there may *not* be a *feasible* server transmission schedule that can deliver full quality video. A naive approach at the server may attempt to transmit each frame with no awareness of the resource constraints. As a result the network may drop packets causing frame losses. In addition, the client may be forced to drop frames that arrive too late for playback. This results in wastage of network bandwidth and client buffer resources.

There are several approaches to adapting the quality of the video playback to the amount of resources available. These approaches can be categorized into *adaptive quantization*, *frame rate adaptation* or *layered encoding*. Adaptive quantization aims at reducing the transmission rate by adjusting the video frame quality. While this approach is viable for live video, it may not be appropriate for stored video due to re-quantization. Frame rate of a video can be adjusted by selectively playing frames and thus effectively reducing the bandwidth required for its playback. However, due to variable bit nature of the video, it is not obvious which frames should be chosen for playback. Layered encoding separates the video signal into components of differing visual importance: the base layer contains coarser details while upper layers contain increasingly finer details of the video. A prefix of these layers could be chosen such that the resource constraints are satisfied. It is not a trivial task to select layers such that better but consistent quality playback is ensured when

the network conditions are constantly varying. In this thesis, we first address the issue of how to *selectively discard frames at the server* such that the perceived quality of service at the client is maximized while maintaining efficient utilization of the network resources. We then address the issue of how to *capture the user's perception* of video quality under layered encoding and how to *provide smoother quality* layered video stream.

1.3 Channel Assignment and Admission Control in Cellular Networks

A mobile cellular system consists of cells, each of which has a base station that serves mobile hosts in that cell. A radio channel used in one cell can be used in another if they are separated by *co-channel reuse distance*. Concurrent use of a channel in cells within this distance causes *co-channel interference*. Frequency reuse without causing interference is the core concept of cellular systems. To establish a communication session, a mobile host requests the base station in its cell for a channel. The base station assigns an unused channel if available. Otherwise the call is blocked. If a mobile host moves from one cell to another during a session, the responsibility of continuation of service is handed over to the new base station. This is known as *handoff*. If the new base station cannot find an idle channel, the on-going session must be terminated, i.e., the call is dropped. The *reduction of call blocking and call dropping probabilities* is the main goal of channel assignment schemes.

Distributed dynamic channel assignment algorithms run at each base station attempt to reduce the network-wide call blocking and call dropping probabilities while making assignment decisions based on its *neighborhood information only*. It is observed that a certain pattern of channel usage among cells in a neighborhood can maximizing spectral reuse. The challenge is how to assign channels dynamically such that they form a compact pattern using only local information. This is known as *local packing* To aid in compact packing of channels, aggressive algorithms *reassign* channels being used by calls in progress to make room for a new request. While reassignments maximize channel reuse, they may inconvenience users and also incur communication and processing overheads. Hence reassignments must be used prudently to be beneficial. Another issue is that of call admission control. Since forced termination of an existing call is less desirable than blocking a new call, call admission control schemes are needed to reduce the call dropping probability, possibly at the expense of increased call blocking probability. Most call admission control schemes are based on the *guard channel concept*. This approach offers a generic means

of improving the probability of successful handoffs by simply reserving some channels in each cell exclusively for handoffs. However, improper selection of guard channels can result in either forced termination of on-going calls or low spectrum utilization. This thesis addresses the following issues: how to *select channels locally* such that they form a *compact pattern globally*; how to *adjust guard channels dynamically* such that continuation of service is ensured *without wasting the spectral resources*.

1.4 Thesis Contributions

The central theme of this thesis is how to effect purposeful global behavior using simple local strategies. We have studied this problem under various settings and made the following contributions.

- We studied the feasibility of localized proportional routing by developing theoretical models and comparing them against optimal proportional routing. We investigated the impact of granularity on the performance of localized proportional routing and shown that path-level information yields nearly the same performance as link-level information. We proposed two simple heuristic strategies for localized proportioning: *equalization of blocking probabilities* (ebp) and *equalization of blocking rates* (ebr). We have shown that *ebp* strategy converges to near-optimal proportions and results in lower blocking than *ebr* strategy [74, 75, 76, 77].

  We introduced a notion of *width* of a set of candidate paths and developed a candidate path selection scheme *widest disjoint paths* (wdp) that selects widest paths that are disjoint *w.r.t* to bottleneck links. We have demonstrated that *wdp* based selection of candidates and *ebp* based proportioning among candidates yields almost optimal performance using only a few paths. We have also shown that proportional routing approach with *wdp* and *ebp* achieves higher throughput with lower overhead than best-path routing approach [78, 79].

  We proposed an aggregation method that summarizes the state of multiple paths between two routers using a single metric. We designed two hierarchical multipath routing schemes that are based on this metric. We evaluated their performance and show that the proposed hierarchical multipath routing schemes perform as well as flat multipath routing schemes. Furthermore, we demonstrated that these schemes with only aggregate information outperform even flat best-path routing schemes having detailed information about the network [80].

- We developed an algorithm to find, given network bandwidth and client buffer constraints, the minimum number of frames that must be discarded in order to meet these constraints. We formulated the problem of optimal selective frame discard using the notion of a cost function and developed several efficient heuristic algorithms for both JPEG and MPEG video [117, 118].

  We defined smoothness criteria for a layered video playback and designed metrics for measuring it. We developed off-line algorithms for layer selection that maximize smoothness for the case where the network bandwidth is varying but known a priori. We also developed an adaptive algorithm for providing smoothed layered video delivery that doesn't assume knowledge about future bandwidth availability [72, 73].

- We have proposed two new channel selection strategies, *residual* and *coaxial*, based on local packing for compact packing of channels. These schemes were compared with other selection strategies, and shown that they either carry more traffic or reduce the number of channel relocations needed [69].

  We showed that the number of guard channels can be dynamically determined based on reassignment frequency in the neighborhood of a cell. A revenue based call admission control scheme is presented which attempts to maximize income by balancing the penalty for reassignments against the reward for serviced calls [70, 71].

1.5 Thesis Outline

This thesis is divided into three parts each corresponding to the context in which we studied the problem of identifying local objectives that have good global effect. The first part is dedicated to quality of service based routing which is the main focus of this thesis. In the second part, we describe selective frame discard and selective layer discard schemes for delivering stored video across resource constrained networks. The third part presents channel assignment and call admission control in wireless cellular networks.

Part I

Localized Adaptive Proportioning Approach to QoS Routing

Chapter 2

Introduction

Routing in the current Internet focuses primarily on connectivity and typically supports only the "best-effort" datagram service. The routing protocols deployed such as OSPF [65] use the *shortest path only* routing paradigm, where routing is optimized for a single metric such as hop count or administrative weight. While these protocols are well-suited for traditional data applications such as ftp and telnet, they are not adequate for many emerging applications such as IP telephony, video-on-demand and teleconferencing, which require stringent delay and bandwidth guarantees. The "shortest paths" chosen for the "best-effort" service may not have sufficient resources to provide the requisite service for these applications. Furthermore, with the explosive growth of the Internet traffic, the *shortest path only* routing paradigm of the current Internet also leads to unbalanced traffic distribution — links on frequently used shortest paths become increasingly congested, while links not on shortest paths are underloaded.

In order to address these issues, multi-path traffic engineering techniques have been proposed [6], of which Quality-of-Service (QoS) based routing [9, 17] is an important new mechanism. In QoS routing, paths for flows are selected based upon knowledge of the resource availability (referred to as *QoS state*) at network nodes (i.e., routers) and the QoS requirements of the flows. It is expected that QoS routing will select, among the many possible choices, a path that has sufficient resources to accommodate the QoS requirement of a given flow. QoS routing can significantly improve the network throughput because of its awareness of the network QoS state.

In QoS routing, some knowledge regarding the (global) network QoS state is crucial in per-

forming judicious path selection. This knowledge, for example, can be obtained through (periodic) information exchange among routers in a network. Under this approach, which we refer to as the *global best-path* routing approach, each router constructs a global view of the network QoS state by piecing together the QoS state information obtained from other routers, and selects the "best" path for a flow based on this global view of the network state. Examples of the global best-path routing approach are various QoS routing schemes [2, 56, 110, 17, 115] based on QoS extension to the OSPF routing protocol as well as the ATM PNNI [5] routing protocol. Global best-path routing schemes work well when each source node has a reasonably *accurate* view of the network QoS state. However, as the network resource availability changes with each flow arrival and departure, maintaining an accurate network QoS state is impractical, due to the prohibitive communication and processing overheads entailed by frequent QoS state information exchange. In the presence of *inaccurate* information regarding the global network QoS state, best-path routing schemes may suffer degraded performance as well as potential instability [103, 74, 75].

As a viable alternative to the global best-path routing approach, we propose a novel *localized proportional routing* approach to QoS routing. Under this localized proportional routing approach, instead of (periodically) exchanging information with other routers to obtain a global view of the network QoS state, a source router attempts to *infer* the network QoS state from *locally collected flow statistics* such as flow arrival/departure rates and flow blocking probabilities, and performs adaptive proportioning of flows among a set of *candidate* paths based on this local information. As a result, the localized proportional routing approach avoids the drawbacks of the conventional best-path routing approach such as degraded performance in the presence of inaccurate routing information. Furthermore, it has several important advantages: *minimal* communication overhead, *no* processing overhead at *core* routers, and *easy* deployability.

We investigate an important and fundamental issue in the design of localized QoS routing schemes — the *granularity* of locally collected QoS state information and its impact on the convergence process of these schemes and their performance. We consider flow statistics collected at two different granularity levels: *link* level and *path* level. At the (*finer*) link level, a source node collects *both* the blocking statistics (i.e., whether a flow is blocked or not) of flows routed along a path from the source node to a destination node, *and*, in the case of a blocked flow, the identity of the link where the flow is blocked. The latter information can be gathered, for example, by attaching the identity of the link in the flow setup failure notification sent back to the source node. At the (*coarser*) path-level, a source node collects

only the flow blocking statistics for each path between the source node and a destination node. Clearly, the path-level flow statistics are much easier to collect and maintain, but they also convey much less precise information regarding the (global) network QoS state.

We propose theoretical models based on the link-level and path-level flow statistics to study the impact of granularity. These models are developed based on the notion of *virtual capacity* of a link or a path as *perceived* by a source node. The virtual capacity of a link or a path is computed as a function of the amount of offered load and the corresponding observed blocking probability on that link or path. Through numerical investigation, we study the convergence process of virtual capacity based theoretical models and show that it is possible to design localized proportional routing schemes that converge to a stable point. We find that though granularity of information does have impact on the rate of convergence and the equilibrium blocking probability, the performance penalty due to coarser path-level information is not significant. Based on these theoretical results, we proceed to develop practical proportioning strategies that are simple and easy to implement. We propose two such strategies that require only path-level information: *equalization of blocking probabilities* (ebp) and *equalization of blocking rates* (ebr). We compare their performance with optimal proportional routing and show that *ebp* strategy yields near-optimal proportions.

The localized proportioning approach described above splits the traffic bound to a destination adaptively among a set of *candidate* paths. Two key questions that arise in proportional routing are how many candidate paths are needed and how to find these paths. Clearly, the number and the quality of the paths chosen as candidates dictate the performance of a proportional routing scheme. There are several reasons why it is desirable to minimize the number of paths used for routing. First, there is a significant overhead associated with establishing, maintaining and tearing down of paths. Second, the complexity of the scheme that distributes traffic among multiple paths increases considerably as the number of paths increases. Third, there could be a limit on the number of explicitly routed paths such as label switched paths in MPLS [92] that can be setup between a pair of nodes. Therefore it is desirable to use *as few paths as possible* while at the same time *minimize the blocking probability* in the network. Furthermore, it is not possible to provide each node with accurate information about the network state due to prohibitive communication and processing overheads. Hence it is important to devise candidate path selection schemes that *work well even with infrequent link state updates*. We propose such a scheme *widest disjoint paths* (wdp) that selects widest paths that are disjoint *w.r.t.* bottleneck links. It uses *infrequently* exchanged *global* information for selecting a few good paths based on their long term

available bandwidths. The traffic is proportioned among the candidate paths using *local* information to cushion the short term variations in their available bandwidths. This *hybrid* approach to QoS routing adapts at different time scales to the changing network conditions.

The above discussion assumes that each router in the network is aware of the topology and the state of the whole network. This is referred to as *flat* routing and under flat routing, each router participates in link state updates and maintains detailed information about the entire network. This introduces significant burden on every router and as the size of the network grows, the overhead at each router increases tremendously. To provide a scalable solution, *hierarchical routing* is suggested as an alternative to flat routing. Under hierarchical routing, a network is divided into multiple areas. The routing within the area is flat with each router having detailed information about routers and links in that area. But the routers have only sketchy *aggregate* information about other areas. To route traffic destined for other areas, a source router may select a partial higher level path, based on the aggregate information, that gets expanded, based on the detailed information, at the ingress border router of each area along the path. Such a hierarchical routing reduces the overhead at each router by limiting the scope of link state updates and maintaining only summary information about other areas. The hierarchical routing approach while reduces the burden on a router, introduces inaccuracy in the information available for routing. Hence the performance of a hierarchical routing scheme depends heavily on how information about an area is aggregated and how it is utilized in routing across areas.

We propose an aggregation method that summarizes the state of multiple paths between two routers using a single metric. This metric in essence captures the traffic carrying capacity of multiple paths between a pair of routers. We propose two inter-area routing schemes based on this aggregate metric: *hierarchical widest disjoint paths* (hwdp) and *hierarchical widest border routers* (hwbr). The *hwdp* scheme is a hierarchical source routing scheme where a source router selects a set of higher level skeletal paths to the destination as candidates and proportions flows among them. The *hwbr* scheme is a hierarchical next-hop routing scheme that selects only the next-hop border routers which in turn select higher level paths to the destination. Both these schemes use *wdp* scheme mentioned earlier, for intra-area routing to expand the skeletal higher level paths to actual physical paths. They essentially differ in the way the network outside an area is aggregated by a border router and propagated to the interior routers. We evaluate the performance of these hierarchical proportional routing schemes and show that these schemes with only aggregate information outperform even flat global best-path routing schemes having detailed information about the network. We argue

based on our results that our *hwbr* scheme due to its low overhead and high throughput, is a suitable choice for hierarchical routing across large networks.

The rest of this part of thesis is organized as follows. In the next chapter, we define the problem setting we consider in this thesis. Chapter 4 presents the related work. In Chapter 5 we study the theoretical models for proportional routing and propose some practical schemes in Chapter 6. The candidate path selection scheme is described in Chapter 7 and the hierarchical proportional routing schemes are presented in Chapter 8. The conclusions and future work are discussed in Chapter 9.

Chapter 3

Problem Setting

Network support for traffic with Quality of Service (QoS) guarantees usually requires a connection establishment procedure which will select a path, perform admission control and reserve resources along the selected path. The path selection procedure in turn requires link state updates to gather information about each link in the network. In the following, we state the assumptions made in this thesis about the QoS requirements, routing procedures and the update policies. Similarly the performance metrics used for evaluating the efficacy of various QoS routing are also discussed.

3.1 Bandwidth Guarantees

Different applications have different quality of service requirements. Some require throughput guarantees, some end-to-end delay guarantees while others require loss rate guarantees. It is the job of the network to map these application requirements to network resources such that the requested QoS can be guaranteed. Resource provisioning that ensures both guaranteed service to applications and efficient utilization of resources is a very complex task. In this thesis, we assume that application requirements are either specified or characterized by a single network resource, bandwidth. This could be the *effective bandwidth* that captures the traffic characteristics of the application. This is not too limiting an assumption since delay constraints can either be handled by translating them to corresponding bandwidth guarantees or by bounding the number of hops during path selection. Furthermore, though there may be need for supporting applications with multiple QoS parameters, it is likely that initial deployments focus simply on bandwidth based guarantees to reduce the oper-

ational complexity. In other words, in this thesis we assume that an application requests for a specific bandwidth and the network admits an application only if it can reserve the requested bandwidth to satisfy the application requirements.

3.2 Explicit Routing

The current Internet employs hop-by-hop routing that selects shortest paths periodically and maintains the path information in the routing table so that routing results in a simple table look up. Under QoS routing, different flows from various source nodes to the same destination may request different QoS. Hence it is not possible to maintain a single routing table to route flows with divergent requirements. Explicit routing is suggested as an alternative to support QoS routing. Under explicit routing, a source node selects, for each flow, an explicit path to the destination. This requires an ability to setup explicit routes in the network. This can be implemented in the current Internet using a form of loose source routing. But the overhead of carrying the complete explicit route with each packet is prohibitive. However, Multi-Protocol Label Switching (MPLS) an emerging Internet Engineering Task Force (IETF) standard [92] provides such a capability. MPLS replaces the standard destination based hop-by-hop forwarding paradigm with a label swapping forwarding paradigm. It makes explicit routing practical by allowing the explicit route to be carried only at the time label switched path is set up. This thesis assumes that such an explicit routing is supported by the network.

Under source directed QoS routing, upon arrival of a flow, the source node first selects a path that is likely to satisfy the requested QoS. This path selection is performed by the source node based on its own local view of the network state that is gathered through link state updates and previous routing attempts. The source node may prune (seemingly) *infeasible* links, with fewer resources than required by the flow, in order to select a *feasible path*. When no feasible path is found, the flow is blocked due to *routing failure*. This routing failure could be due to either lack of network resources or staleness of link state. Once a path is selected for the flow, the source then sends a setup request to reserve resources at each link along the selected path. Each node along the path performs admission control to see whether sufficient resources are available on the link to support this flow. If the link can accommodate the flow, resources are reserved on that link for the flow and the setup message is forwarded to the next link along the path. If all the links along the path have sufficient resources, the setup request is accepted and the flow is admitted. The resources

reserved for a flow remain with it for the entire duration of the flow. Moreover, the route is *pinned*, i.e., all the packets of the flow follow the same path for the duration of the flow, even if a "better" path is found during that time. This helps reduce the variance in the delay experienced by the packets of a flow.

When a link does not have adequate amount of available resources, the setup request is rejected and the flow is blocked. This is categorized as *setup failure*. A setup failure is the result of the discrepancy between the actual link state and the view of the source node. The main focus of this thesis is how to reduce the inaccuracy in a source node's view without incurring excessive update overhead and how to select "good" paths in the presence of inevitable inaccuracy. Thus a flow may be blocked due to either *routing failure* or *setup failure*. The main objective of any QoS routing scheme is to minimize the overall blocking probability. When a setup request for a flow results in failure, we can make an attempt to reroute the flow by finding an alternate path. This is known as *crankback*. While it is possible that crankbacks can decrease the probability of blocking a flow, they increase the signaling overhead. Moreover, a failed setup request could be an indication of the overall load in the network. In such a case, alternate routing may consume resources along the primary paths of other source-destination pairs increasing the blocking probability for future flows between those pairs. Thus crankbacks can potentially increase the overall blocking probability in an effort to accommodate an individual flow. In this work, we assume *no crankback*, i.e., if a setup request is rejected the flow is blocked and no attempt is made to reroute the flow.

3.3 Link State Updates

Current Internet routing protocols, such as OSPF, distribute connectivity information throughout the network so that "shortest" paths may be selected. Path selection under QoS routing requires resource availability information apart from the connectivity information about the network. It is suggested that current routing protocols such as OSPF be extended [29, 115] to carry *QoS state* information also in their link state updates. The QoS state of a link may be captured by its instantaneous available bandwidth, i.e., the amount of bandwidth available at the time of the update and/or average available bandwidth, i.e., the amount of bandwidth available on the average since the last update. Maintaining accurate network connectivity information usually requires an insignificant amount of routing updates since network connectivity changes are infrequent. However, maintaining accurate QoS state in-

formation requires a much greater amount of link state updates because network resource availability changes with each flow arrival and departure. Since our focus is on studying the impact of QoS state update frequency on the performance of path selection schemes, we make the following simplifying assumptions. We assume that topology does not change and links do not fail during a simulation run. Also, we ignore the link propagation delay since it does not have a bearing on the outcome of this study.

3.4 Performance Metrics

The goal of QoS routing is efficient utilization of resources while ensuring quality of service to each admitted flow. In our *flow based model with bandwidth guarantees*, a flow is admitted only if the requested amount of bandwidth is available along the path through which it is routed. The quality of service for an admitted flow is guaranteed by reserving the requested amount of bandwidth for the entire duration of the flow and by reclaiming it only after the flow is departed. Under this bandwidth reservation model, a flow may be rejected at the setup time but once admitted all the packets of the flow are guaranteed to receive the requested service. Hence we only need to perform flow level simulation to study the performance of various QoS routing schemes. The objective of a QoS routing scheme is then to maximize the number of admitted flows into the network or in other words minimize the blocking probability experienced by a flow arriving in the network. Thus, a measure of performance of a QoS routing scheme is the *overall flow blocking probability*. It is computed as the ratio of the number of flows blocked and the total number flows that have arrived at the network.

There are several overheads associated with a QoS routing scheme. Measuring the blocking performance alone without the corresponding overheads involved in the implementation is not sufficient to evaluate the overall performance of a routing scheme. We categorize the overheads incurred by a QoS routing scheme into *update overhead, path computation overhead* and *path management overhead*. There is a fundamental tradeoff between the amount of overhead due to link state updates and the blocking performance of path selection schemes. The higher the frequency of updates is, the lesser the inaccuracy at a source node is and the better the blocking performance of a path selection scheme is. Ideally, we would like to have lower blocking with minimal overhead. But these are conflicting objectives and the job of a QoS routing scheme is to limit the update overhead while dealing with the corresponding inaccuracy by selecting paths intelligently. In this thesis, we assume simple

periodic link state updates and we measure its overhead by the *frequency of updates*.

Another overhead associated with QoS routing is the time spent in finding a suitable path. A path may be computed on-demand upon each flow arrival or a path may be chosen from a set of precomputed paths. Since identifying a suitable path may involve searching the entire graph, the cost of computation is not insignificant. The complexity of the algorithm used for finding a path is referred to as *path computation overhead*. Finally, it is desirable to minimize the number of paths used for routing. There is a significant overhead associated with establishing, maintaining and tearing down of paths. Moreover, there could be a limit on the number of explicitly routed paths such as label switched paths in MPLS [92] that can be setup between a pair of nodes. Hence the number of paths used for routing, averaged across all source-destination pairs, is taken as a measure of the *path management overhead*. Essentially the overall objective of QoS routing is to minimize the above mentioned overheads while minimizing the overall flow blocking probability.

Chapter 4

Related Work

The problem of QoS routing has been addressed in many contexts and there have been several proposals for providing QoS routing. These proposals differ in where the path is chosen (source or hop-by-hop), how the network state information is gathered (global updates or local observations), what type of information is exchanged (instantaneous available bandwidth or long-term average bandwidth), which path is selected (widest or shortest), etc. A survey of various QoS routing schemes can be found in [9]. We can broadly categorize them into *global QoS routing* schemes that are based on global link state updates and *localized QoS routing* schemes that are based on local path state observations. The following sections discuss these QoS routing approaches in detail.

4.1 Global QoS Routing

The majority of QoS routing schemes [2, 17, 29, 56, 105, 110, 115] proposed so far require periodic exchange of *link QoS state* information among network nodes to obtain a *global view of the network QoS state*. Based on this *current* global view of the network state, a source node dynamically determines the "best" feasible path for a flow originating from it to a destination node. We refer to this approach to QoS routing as the *global* QoS routing approach or *global best-path routing* approach. The proposed global QoS routing schemes primarily differ in their path selection criteria and the network state update triggering mechanisms.

Path selection algorithms have to deal with the fundamental trade-off in minimizing the resource usage and balancing network load. The resource usage by a flow can be minimized

by selecting the shortest path which may be heavily loaded. The network load can be balanced by choosing the least loaded path which may be longer and hence consumes more resources. Several path selection algorithms have been proposed that weigh limiting hop count and balancing network load differently. They include widest-shortest path (*wsp*) [2], shortest-widest path (*swp*) [110], and shortest-distance path (*sdp*) [56]. They all attempt to select a *feasible* path. A path is considered *feasible* if its *bottleneck bandwidth* (smallest available bandwidth along the path) is greater than or equal to the requested bandwidth. The above mentioned schemes differ in the selection of a feasible path when there are many such choices as explained below.

Widest-shortest path A path with fewest number of hops among all feasible paths. If there are several shortest feasible paths, the one with the largest bottleneck bandwidth is chosen. If more than one shortest path with same bandwidth exist, one of the paths is randomly selected.

Shortest-widest path A path with largest bottleneck bandwidth among all feasible paths. If more than one widest feasible path exist, the one with the minimum hop count is chosen. If there are several such paths with the same hop count, one of them is randomly selected.

Shortest-distance path A feasible path with the shortest distance. The distance function for a path r is defined by

$$dist(r) = \sum_{j=1}^{k} \frac{1}{C_j} \tag{4.1}$$

where C_j is the bandwidth available on a link j along path r.

The widest-shortest path favors shorter paths thus giving higher priority to limiting resource usage, while the shortest-widest path favors wider paths thus giving higher priority to distributing network load. The shortest-distance path attempts to strike a balance by using the distance function. Among these, *wsp* is the most popular and well studied algorithm for selecting the "best" feasible path and hence we use it as a representative of global best-path routing schemes in the following discussion.

Each network node under global QoS routing approach generates link state updates informing all other nodes about the current state of the links attached to it. Various update policies

are possible that differ in when an update is triggered and what information is contained in it. Most of the schemes proposed so far [2, 56, 110] exchange information about currently available bandwidth while some [30, 74] exchange information regarding the probability that a requested bandwidth is available. The mechanisms used to trigger updates can be based on *threshold*, *class* or *timer*. In case of threshold based policy, an update is triggered whenever the relative change from the previously advertised to the current link state exceeds a certain threshold. Class based policies partition the available bandwidth into multiple classes and trigger an update whenever the current link state value crosses a class boundary. The partitioning into classes could be either fixed-size or exponentially distributed. Timer based triggers may be used to generate updates periodically. They, referred to as *clamp-down* timers, are often used in conjunction with one of the above triggers to enforce a minimum spacing between two consecutive updates. While the periodic updates are simple to implement, more complex change based triggers attempt to avoid unnecessary updates by generating an update only when the link state changes *significantly* from previously advertised state. Another trade-off in these schemes is between the frequency of updates and the accuracy of network state information available at each node for path selection.

Current Internet routing protocols, such as OSPF, distribute connectivity information throughout the network so that "shortest" paths may be selected. Maintaining accurate network connectivity information usually requires an insignificant amount of routing updates since network connectivity changes are infrequent. However, maintaining *accurate* network QoS state requires *frequent* information exchanges among the network nodes because network resource availability changes with each flow arrival and departure. The prohibitive communication and processing overheads entailed by such frequent QoS state updates preclude the possibility of *always* providing each node with an accurate view of the current network QoS state. Consequently, *the network QoS state information acquired at a source node can quickly become out-of-date when the QoS state update interval is large relative to the flow dynamics*. Under these circumstances, exchanging QoS state information among network nodes is superfluous. Furthermore, path selection based on a *deterministic* algorithm such as Dijkstra's shortest path algorithm, where *stale QoS state information is treated as accurate*, does not seem to be judicious. The *best* path selection based on *inaccurate* information could cause instability: after one QoS state update, many source nodes choose paths with shared links because of their perceived available bandwidth, therefore causing over-utilization of these links. After the next QoS state update, the source nodes would

avoid the paths with these shared links, resulting in their under-utilization. This oscillating behavior can have severe impact on the system performance, when the QoS state update interval is large. Due to these drawbacks, it has been shown that when the QoS update interval is large relative to the flow dynamics, the performance of global QoS routing schemes degrades drastically [74, 75].

Several solutions have been proposed to deal with inaccuracy that is inevitable in the information available to path selection process. One approach [2] categorizes the inaccuracy into *systematic* and *random* based on the type of update triggers used. When a change based trigger is employed it is possible to infer the range for actual link metric value given its last advertised value. This type of systematic inaccuracy could be suitably accounted for by a path selection algorithm and thereby choose a path that is most likely to have the required resources. In [30] a path selection algorithm is proposed to find the *most reliable path* assuming that information regarding the probability $p_l(x)$ that a link l can accommodate a flow which requires x units of bandwidth is known to the source node. This is further experimented in their paper [3] on *safety-based routing*, where safety, i.e., $p_l(x)$, of a link l for a bandwidth x is inferred from its last advertised available bandwidth value assuming that a change based update triggering policy is used. However when large *clamp-down* timers are used, it is almost impossible to estimate the amount of inaccuracy. Some randomization algorithms are proposed [3] to cope with this kind of random inaccuracy where advertised bandwidth values are used only as clues in selecting paths. We have proposed a probabilistic approach [74] where network nodes exchange link availability probability information similar to $p_l(x)$. But instead of choosing the *best* or *safest* path our algorithm distributes load across the network in accordance with availability of links while favoring shorter paths.

While all the above remedial schemes reduce the impact of inaccuracy on the performance of path selection, they either work well only in some cases or introduce additional overhead at core routers. For easy deployability and scalability of QoS routing, we need to devise schemes that perform well without introducing more complexity at core routers and more communication overhead on network than the current routing protocols. As an alternative to global QoS routing, *localized* QoS routing approach is proposed, where *no global QoS state information exchange among network nodes is needed*. Instead, source nodes infer the network QoS state based on flow blocking statistics collected *locally*, and perform flow routing using this *localized* view of the network QoS state. Some of the *localized* QoS routing schemes are described in the following sections.

4.2 Localized QoS Routing

A source node under QoS routing, upon receiving a request with specific QoS requirements, selects a suitable path to the destination node based on its view of resource availability. It then sends a connection request to reserve resources at each node along the path. It is possible that sufficient resources are not available along the chosen path either because of stale routing information at the source node or because of changes in the network state while the connection is being established. In such a case, the request is rejected and the flow is blocked. Localized QoS routing approach attempts to infer the network state from these flow blocking statistics and performs path selection based on this local information. Several localized dynamic routing schemes have been proposed in the context of telephone networks. Here we consider two such schemes based on sticky routing and learning automata that make use of the feedback information regarding flow admission or rejection for routing future flows.

4.2.1 Sticky Random Routing

The *dynamic alternative routing* (dar) is a well known routing scheme [25] where a source always tries the direct one-link path to the destination first and in case of a crankback chooses a two-link path using *sticky random routing* (srr). Since in our setting we do not consider re-routing, the *srr* scheme (equivalent to *dar* with a dummy direct link) is used for comparison. The *srr* scheme remembers a path known as *preferred* path for each destination. A flow to a destination is always routed through its corresponding preferred path. If the connection setup is successful, the preferred path remains same. But in case of a failure, the flow is blocked and a new preferred path is chosen randomly from set of feasible paths to that destination excluding the current preferred path. The *srr* scheme essentially sticks to a path as long as it can accommodate offered traffic.

The analysis of *dar* presented in [25] shows that *dar* equalizes the blocking rates over two-link paths for each source destination pair. It claims that overflow streams, i.e., flows directed to two-link paths, under *dar* can be modeled as if they arise from proportional routing, with proportions depending on the blocking rates of links. But it also cautions that the approximation procedure used in the analysis could break down if the overflow is large and needs to be spread over a number of alternatives. This is precisely the case with networks like Internet that may have more than one minhop path and many alternative paths between each source-destination pair.

4.2.2 Learning Automata based Routing

An application of automata to the routing problem is given by Narendra and Mars [67]. The incoming flows are offered to a path r according to a probability distribution p_r, which is updated using feedback information regarding flow admission or rejection. These schemes reward a path on which a flow is successful and punish a path on which a flow fails. If a route i is chosen at time n and the flow is successful, then updating is

$$p_i(n+1) = p_i(n) + a(1 - p_i(n))$$

$$p_j(n+1) = (1-a)p_j(n) \quad j \neq i$$

while if the flow fails

$$p_i(n+1) = (1-\epsilon)p_i(n))$$

$$p_j(n+1) = \frac{\epsilon}{r-1} + (1-\epsilon)p_j(n) \quad j \neq i$$

where a and ϵ are adjustable parameters, $0 < a < 1$, $0 < \epsilon < 1$ with ϵ small compared with a, and a is itself usually small, so that the updating is gradual. Under certain assumptions [67, 68, 106] show that $L_{R-\epsilon P}$ automata tends to approximately equalize blocking probabilities, b_r, while L_{R-P} automata for which $\epsilon = a$ in the above equalizes blocking rates ($p_r b_r$). One problem with this scheme is that probability associated with a path is changed per every flow arrival. In addition, no account is taken of the length of a path and also the selection of candidate paths. In the later chapters we compare the performance of these schemes with our schemes and show that our schemes yield much lower blocking probability.

4.3 Hybrid QoS Routing

As a viable alternative to the best-path routing approach, we proposed [75, 76] a novel *proportional routing* approach to QoS routing. Under this proportional routing approach, a source router uses *locally collected flow statistics* such as flow arrival/departure rates and flow blocking probabilities, and performs adaptive proportioning of flows among a set of

candidate paths based on this local information. This localized proportional routing approach is somewhat similar in spirit to the dynamic routing schemes in telephone networks [46] described above. However, the actual mechanisms for adaptive proportioning are quite different. Moreover, we enhance the performance of localized proportional routing by selecting a few good candidate paths.

Under pure localized approach, the candidate path set remains static while their proportions are adjusted dynamically. A network node under localized approach can judge the quality of paths only by routing some traffic along them. So it is not possible to update the candidate path set based on local information alone. On the other hand, due to changing network conditions, a few good candidate paths cannot be selected statically. Hence we propose [78, 79] a *hybrid* approach to proportional routing, where a few good candidate paths are selected dynamically based on *infrequently* exchanged global information. The resulting inaccuracy is cushioned by adaptively proportioning traffic among multiple "good" candidate paths instead of routing all the traffic along the "best" path. In the following chapters, we first address the question of how to proportion traffic adaptively among a set of candidate paths using only local information. We then turn to the issue of how to select a few good candidate paths based on infrequently exchanged global information.

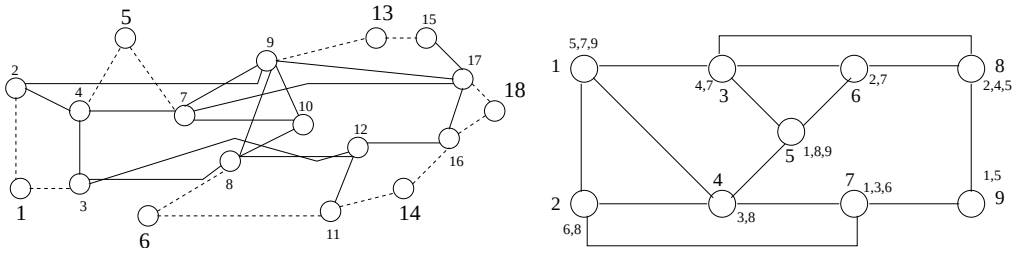
Chapter 5

Localized Proportional Routing: Theoretical Models

In this chapter we study several issues in the design of *localized* QoS routing schemes, which make *local* routing decisions based on *locally collected* QoS state information. In particular, we investigate the granularity of local QoS state information and its impact on the design of localized QoS routing schemes from a theoretical perspective. We develop two theoretical models for studying localized proportional routing: one using the locally collected link-level QoS state information, and the other using locally collected path-level QoS state information. We compare the performance of these localized proportional routing models with that of a global optimal proportional model that has knowledge of the global network QoS state. We first describe the global optimal proportioning scheme and then localized adaptive proportioning schemes.

5.1 Global Optimal Proportional Routing

As a basis for comparing the performance of localized proportional QoS routing models, in this section we present the global optimal proportional routing model, which has been studied extensively in the literature (see [93] and references therein). In this model, we assume that each source node knows the *complete topology information of the network (including the capacity of each link)* as well as the *offered traffic load between every source-destination pair*. With the global knowledge of the network topology and offered traffic loads, the *optimal proportions*, for distributing flows among the paths between each source-



(a) *bigisp* (b) *minisp*

Figure 5.1: The topologies used in our study

destination pair, can be computed as described below.

Consider an arbitrary network topology with N nodes and L links (see Figure 5.1 for example topologies). For $l = 1, 2, \ldots, L$, the capacity of link l is $c_l > 0$, which is assumed to be fixed and known. The links are unidirectional, i.e., carry traffic in one direction only. Let $\sigma = (s, d)$ denote a source-destination pair in the network. Let λ_σ denote the average arrival rate of flows arriving at source node s destined for node d. The average holding time of the flows is μ_σ. Recall that each flow is assumed to request one unit of bandwidth, and that the flow arrivals are Poisson, and flow holding times are exponentially distributed. Thus the offered load between the source-destination pair σ is $\nu_\sigma = \lambda_\sigma/\mu_\sigma$. Let R_σ denote the set of (explicit-routed) paths for routing flows between the source-destination pair σ. The global optimal proportional problem can be formulated [41, 42, 45] as the problem of finding the optimal proportions $\{\alpha_r^*, r \in R_\sigma\}$ for each source-destination pair σ and its route set R_σ, where $\sum_{r \in R_\sigma} \alpha_r^* = 1$, such that the overall flow blocking probability in the network is minimized. Or equivalently, finding the optimal proportions $\{\alpha_r^*, r \in R_\sigma\}$ such that the total carried traffic in the network,

$$W = \sum_{\sigma} \sum_{r \in R_\sigma} \alpha_r \nu_\sigma (1 - b_r) \tag{5.1}$$

is maximized.

In (5.1), b_r denotes the blocking probability along path r. Under the *link independence* assumption [41, 42, 45], b_r can be expressed as follows:

$$b_r = 1 - \prod_{l \in r} (1 - b_l) \tag{5.2}$$

N :	$\{1, 2, ..n\}$, vector of n nodes in the topology of the given network.
L :	$\{1, 2, ..l\}$, vector of l links in the topology of the given network.
C :	$\{c_1, c_2, ..c_l\}$, vector of capacities of the links, $c_i > 0$.
σ :	a pair (s, d), where $s, d \in N$.
r :	a path, i.e a set of links $\in L$, from source to destination.
R_σ :	a set of paths between a source-destination pair σ.
ν_l :	load on the link $l \in L$.
ν_σ :	load on the source-destination pair σ.
ν :	$n \times n$ traffic matrix, i.e, $[\nu_\sigma]_{\forall \sigma \in N \times N}$.
b_l :	blocking probability of link $l \in L$.
b_r :	blocking probability of path r.
α_r :	load proportion along $r \in R_\sigma$ for a particular σ.

Figure 5.2: Notation

where $l \in r$ means that link l is part of route r, and b_l is the blocking probability of link l. The blocking probability b_l of link l is in turn given by the Erlang loss formula,

$$b_l = E(\nu_l, c_l) = \frac{\frac{\nu_l{}^{c_l}}{c_l!}}{\sum_{n=0}^{c_l} \frac{\nu_l^n}{n!}}. \tag{5.3}$$

Here the load offered on link l, ν_l, is the sum of all the *reduced* loads (i.e., after *independent load thinning*) from any source-destination pair σ which has a route passing through link l. Namely,

$$\nu_l = \sum_{\sigma} \sum_{r \in R_\sigma : l \in r} \alpha_r^* \nu_\sigma \prod_{m \in\ r - \{l\}} (1 - b_m) \tag{5.4}$$

The global optimal proportional routing problem (5.1) is a *constrained nonlinear optimization problem* and can be solved using an iterative procedure based on the *Sequential Quadratic Programming (SQP) method* [82, 15]. Each stage of the iterative procedure has two steps. First, given a set of flow proportions α_r, the fixed-point equations (5.3) and (5.4) involving b_l's and ν_l's are solved. Using these values, W given by equation 5.1 is recomputed. Then this algorithm essentially searches for a new set of improved flow proportions based on the revenue W. The complete process needs to be repeated for various initial conditions to search for the potential global optimal solution. The optimization procedure (referred to as *optpropo*) used in our study is given in Figure 5.3. Figure 5.2 summarizes the notation used in this procedure (and the rest of the thesis).

The global optimal proportional routing model presented above assumes that the bandwidth requests of flows are homogeneous (i.e., one unit of bandwidth). In the case that flows have heterogeneous bandwidth requests, the Erlang loss formula can be extended [40, 90] and

1. PROCEDURE optpropo(N, L, C, R, ν)
2. Start with some arbitrary proportions $\alpha_r^{(0)}, \forall r \in R_\sigma, \forall \sigma = (s,d), s,d \in N$.
3. Use *Sequential Quadratic Programming (SQP) method* for solving (in i^{th} step):
4. maximize $W = \sum_\sigma \sum_{r \in R_\sigma} \alpha_r^{(i)} \nu_\sigma (1 - b_r^{(i)})$
5. under the constraints $\sum_{r \in R_\sigma} \alpha_r^{(i)} = 1$.
6. where $b_r^{(i)} = 1 - \prod_{l \in r} (1 - b_l^{(i)})$
7. Use Gauss-Newton method to solve the following nonlinear equations for $b_l^{(i)}$
8. $b_l^{(i)} = E(\nu_l^{(i)}, c_l)$
9. $\nu_l^{(i)} = \sum_\sigma \sum_{r \in R_\sigma : l \in r} \alpha_r^{(i)} \nu_\sigma \prod_{m \in r - \{l\}} (1 - b_m^{(i)})$
10. END PROCEDURE

Figure 5.3: The *optpropo* procedure

the global optimal proportional routing problem can be analogously formulated and solved using multi-rate loss models [62, 93]. The computational complexity of the exact solution is generally prohibitive for large networks. To address this problem, several approximation algorithms have also been proposed [62, 64]. In particular, when the capacity of a link is large, the blocking probability of a flow of type i can be approximated as follows [91]. Suppose that type i flow requests for d_i units of bandwidth and the load of type i flows on link l is ν_l^i. The blocking probability for type i flows on link l is given by $b_l^i = \frac{d_i}{\delta} E(\frac{\sum \nu_l^i d_i}{\delta}, \frac{c_l}{\delta})$, where δ is an "equivalent rate" given by $\delta = \frac{\sum \nu_l^i d_i^2}{\sum \nu_l^i d_i}$. In other words, the ratio of blocking probabilities of flow types i and j would be same as the ratio of their bandwidth requests, i.e., $\frac{b_i}{b_j} \approx \frac{d_i}{d_j}$. We later argue that in this realistic scenario of link capacity being much larger than an individual flow's bandwidth request, it is not necessary to distinguish between different types of flows for the purpose of proportional routing.

5.2 Localized Proportional Routing

We now turn our attention to the problem of modeling *localized* proportional routing. Unlike in the global case, in localized proportional routing we assume that each source node has only a *local* (and thus *partial*) view of the network state. For example, a source node may only have knowledge of the offered traffic loads between the source-destination pairs originating from itself. Also, it may have partial network topology information only (in particular, the link capacity information may not be available to a source node). As mentioned in the introduction, in this chapter we will focus on local QoS state information gathered at two different granularity levels: the *link* level and the *path* level. At the (finer) link level, each source node can collect the following information locally: 1) the offered traffic load of flows from the source to a destination; 2) the number of flows routed along a path from

the source to a destination that are blocked; and 3) in the case where a flow is blocked, the identity of the link at which the flow is blocked. The third type of information can be made available to a source node by simply piggybacking the identity of the link at which a flow is blocked in the flow setup failure notification sent back to the source node. At the (coarser) path level we assume that each source node only collects the first and second types of the local information listed above. In other words, when a flow is blocked, the source node does not have the knowledge of the link identity at which the flow is blocked. As a result, the path-level local information provides a source node with a much "vaguer" view of the global network QoS state.

5.2.1 Virtual Capacity Model

Given only locally collected flow statistics, determining "optimal" proportions for distributing flows among multiple paths between a source-destination pair becomes a difficult problem. In particular, since each source node does *not* know the capacity of a link and the total offered load on the link, the Erlang loss formula cannot be directly used to derive flow blocking probability at a link. To address this problem, we introduce the notion of *virtual capacity* of a link (or a path) *perceived by a source node*. For a link l, let $\nu_{s,l}$ be the load placed by a source node s, and $b_{s,l}$ be the blocking probability observed by node s. Intuitively, the virtual capacity, $vc_{s,l}$, of link l perceived by the source node s is the (perceived) amount of bandwidth consumed by the flows routed from source s along link l, given the observed blocking probability $b_{s,l}$. Formally, $vc_{s,l}$ is defined via the inverse of the Erlang loss formula as follows:

$$vc_{s,l} = E^{-1}(b_{s,l}, \nu_{s,l}), \tag{5.5}$$

where $E^{-1}(b, \nu) := \min\{c : E(\nu_{s,l}, c) \leq b_{s,l}\}$, the inverse function of the Erlang loss formula with respect to the capacity. The virtual capacity of a *path* can be defined analogously by replacing the link l with a path r, $\nu_{l,s}$ and $b_{l,s}$ with $\nu_{r,s}$ and $b_{r,s}$, the load offered and blocking probability observed by node s along path r.

Note that $E^{-1}(b_{s,l}, \nu_{s,l})$ defined above is an integer-valued function. This means that if we vary either the blocking probability $b_{s,l}$ or the offered load $\nu_{s,l}$ slightly, $E^{-1}(b_{s,l}, \nu_{s,l})$ is likely to yield exactly the same virtual capacity value. This is an undesirable property that could cause some potential problem in the convergence process of localized proportional

routing schemes we study later. To circumvent this problem, we resort to the *continuous* version of the Erlang loss formula defined in [20]:

$$\tilde{E}(\nu, c) = \frac{1}{\int_0^\infty (1 + \frac{x}{\nu})^c e^{-x} dx} \tag{5.6}$$

It can be shown that the continuous version of the Erlang loss formula $\tilde{E}(\nu, c)$ defined above is analytic in c, and coincides with the discrete version $E(\nu, c)$ when c is an integer. Therefore, its inverse function $\tilde{E}^{-1}(b, \nu)$ with respect to c is well-defined. (It is easy to see that the inverse of the Erlang loss formula with respect to the offered load is also well-defined. This inverse will also be used in our localized proportional QoS routing schemes, as will be seen later.) In general, computing $\tilde{E}(\nu, c)$ and its inverses using (5.6) directly are quite difficult and time-consuming. Approximation methods for computing $\tilde{E}(\nu, c)$ and its inverse have been proposed in [20, 36]. In the rest of the chapter, the continuous version of the Erlang loss formula will be used, and for simplicity of notation, we will drop the superscript $\tilde{}$.

The notion of virtual capacity defined above has several interesting and important properties that are key to our study of localized (adaptive) proportional QoS routing. First of all, it is clear that the virtual capacity of a link or path can be computed solely based on local information (e.g., load offered and blocking probability observed by a source node). Second, the notion of virtual capacity provides a *quantitative* measure of capacity share of a link or path grabbed by the flows originated from a source node. The larger the load a source node offers on a link or a path, the more capacity share the node grabs. To see this, consider a link of capacity c_l, which is shared by traffic originated from source nodes $s_1, \ldots, s_n$. Suppose the offered load from each node s_i is ν_{l,s_i}. Then the blocking probability b_l on link l, as is observed by each node s_i, is given by

$$b_l = E(\sum_{i=1}^{n} \nu_{l,s_i}, c_l) \tag{5.7}$$

Therefore, the virtual capacity perceived by node s_i is

$$vc_{l,s_i} = E^{-1}(b_l, \nu_{l,s_i}) \tag{5.8}$$

Clearly, the large ν_{l,s_i} is, the bigger vc_{l,s_i} is, as the observed blocking probability by each node s_i, $b_{l,s_i} = b_l$, is determined by the *total* offered load and the capacity of the link, not

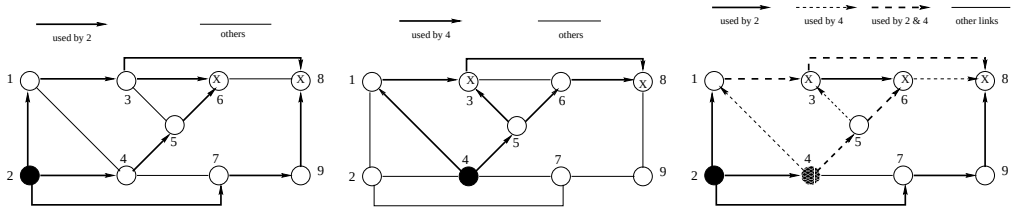
the individual load offered by each source. Third, the virtual capacity perceived by a node is a function of *both* its offered load *and* the observed blocking probability, which changes as the *overall* load on a link or a path varies. Consequently, a node can adjust its offered load to effect a change in the observed blocking probability, or as a response to the change in the observed blocking probability. The notion of virtual capacity therefore provides a theoretical basis for the analysis of how flow proportions should be adjusted based on locally collected statistics. We can extend this notion of virtual capacity for heterogeneous traffic also and compute class based proportions by maintaining statistics separately for each traffic class.

Based on the notion of virtual capacity, in the next section we develop two theoretical models for localized proportional routing: *virtual link based minimization* (*vlm*) and *virtual path based minimization* (*vpm*), that compute flow proportions using, respectively, link-level and path-level flow statistics collected locally at source nodes. In both models, each source collects local QoS state information, and based on this local QoS state information, periodically recomputes flow proportions assigned to the paths from the source to a destination. This distributed dynamic adaptation procedure can be viewed as an iterative process where in each iteration each source independently attempts to minimize the observed blocking probability by adjusting the amount of traffic routed through each path. These models differ in the type and granularity of the local QoS state information collected, and therefore in the manner that the flow proportions are derived.

5.2.2 Virtual Link based Minimization

In the virtual link based minimization model, a source collects link-level flow blocking statistics with the assistance from the connection admission control (CAC) module. We assume that whenever a flow setup request fails at a link, the identity of that link is also recorded and piggybacked to the source. The CAC module at the source node informs the QoS routing module of the flow setup failure and the identity of the link where the flow is blocked. Such link-level flow blocking information can be gathered by a source with very little overhead on the network.

With the locally collected link-level flow statistics, a source knows exactly the offered traffic load on a link contributed by flows originating from that source. Unlike the global routing model, the source, however, does not have any information regarding the traffic loads offered by the other sources on the link. It neither has any knowledge of the capacity



(a) view of source 2 (b) view of source 4 (c) merged view

Figure 5.4: Virtual network views of sources 2 and 4

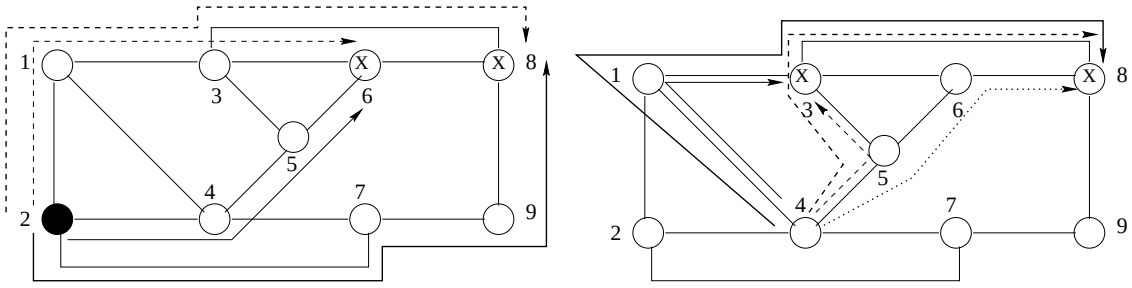
of the link. The source can only infer the state of the link from the flow blocking probability at the link it observes. Using the notion of virtual capacity of a link, the source can infer its share of the bandwidth at each link, and piece together a partial *virtual view* of the network from its own perspective.

This notion of virtual network view of a source can be illustrated using Figure 5.4. First suppose that node 2 is the only source in the network and nodes 6 and 8 are its destinations. There are two minhop paths $2 \rightarrow 1 \rightarrow 3 \rightarrow 6$ and $2 \rightarrow 4 \rightarrow 5 \rightarrow 6$ to node 6. Similarly $2 \rightarrow 1 \rightarrow 3 \rightarrow 8$ and $2 \rightarrow 7 \rightarrow 9 \rightarrow 8$ are the minhop paths to node 8. All the links along these paths form the virtual network view of source 2. When node 2 is the only source and links are not shared by other sources, the virtual capacities of these links would be the same as the actual capacities. Thus the network in Figure 5.4(a) with only the thick links correspond to the virtual network of source 2. Similarly, the virtual network view of source 4 is shown in Figure 5.4(b), where nodes 3 and 8 are its destinations. The thick links correspond to two minhop paths to node 3 and three minhop paths to node 8. Once again when node 4 is the only source in the network, the thick links form the virtual network of source 4.

Now consider the case where both nodes 2 and 4 are the sources. The merged virtual view of these sources is shown in Figure 5.4(c). Here the thick links are used by only source 2 and the thin dashed links are used by source 4 only. Both sources use the thick dashed links $1 \rightarrow 3$, $3 \rightarrow 8$, $4 \rightarrow 5$, and $5 \rightarrow 6$ to route their traffic. Under the link-level localized QoS routing model, each source does not have any knowledge about this sharing. This sharing is indirectly reflected in the flow blocking probability on the links observed by each source, which leads each source to derive its share of the link capacity using the link virtual capacity.

1. PROCEDURE VLM(s)
2. For each link $l \in L$
3. Compute virtual capacity $vc_{s,l}^{(n)} = E^{-1}(\nu_{s,l}^{(n)}, b_{s,l}^{(n)})$
4. For each path $r \in R_s$
5. Assign new load $\nu_r^{(n+1)}$ such that
6. $\sum_{r \in R_s} \nu_r^{(n+1)}(1 - b_r)$ is maximum, where
7. $b_r = 1 - \prod_{l \in r}(1 - b_l)$
8. $b_l = E(\nu_{s,l}^{(n+1)}, vc_{s,l}^{(n)})$
9. $\nu_{s,l}^{(n+1)} = \sum_{r \in R_s : l \in r} \nu_r^{(n+1)} \prod_{m \in\, r - \{l\}}(1 - b_m)$
10. $\sum_{r \in R_\sigma} \nu_r^{(n+1)} = \nu_\sigma, \forall \sigma$
11. END PROCEDURE

Figure 5.5: The *vlm* procedure at source node s

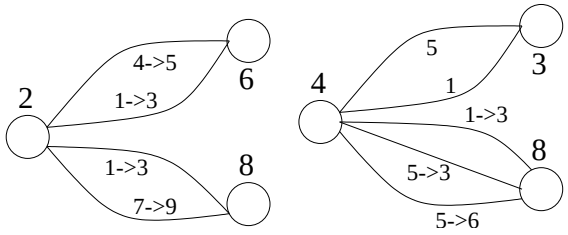


(a) view of source 2

(b) view of source 4

Figure 5.6: Virtual network views of sources 2 and 4

With the virtual network view, each source can employ a localized version of the global optimal proportional routing scheme (*optpropo*) to compute the "optimal" flow proportions for each of its destinations: we replace the actual capacity of a link by its virtual link capacity, and only offered traffic loads from the source are used to compute the optimal flow proportions for the source. The resulting optimization procedure, referred to as the virtual link based minimization (*vlm*) procedure, is shown in Figure 5.5, where s is a source node. This localized flow proportioning scheme is an iterative process where each iteration is performed after an observation interval by each source asynchronously. In the nth iteration, the current virtual capacity $vc_{s,l}^{(n)}$ of each link l with respect to s, is computed, based on the current offered load $\nu_{s,l}^{(n)}$ and the corresponding observed blocking probability $b_{s,l}^{(n)}$ (lines 2-3). The local minimization is then performed on the virtual network thus formed with each link l having the capacity $vc_{s,l}^{(n)}$ (lines 4-10).



(a) view of source 2 (b) view of source 4

Figure 5.7: Virtual network views of sources 2 and 4

5.2.3 Virtual Path based Minimization

In the virtual path based minimization (*vpm*) model each source collects only path-level flow statistics: the number of flows routed along each path between the source to a destination, and the number of flows blocked along the path. Unlike the link-level localized QoS routing model, here we assume that the identity of the link at which a flow is blocked is *not* available to a source.

With only locally collected path-level flow statistics, a source does not have any ways to infer the QoS state of any individual link. A source can only obtain some knowledge about the "quality" of a path based on the traffic offered on the path and the corresponding observed flow blocking probability along the path. Similar to the virtual link based QoS routing model, in the virtual path based routing model we associate a virtual network with each source-destination pair, using the notion of virtual capacity of a path. Consider a source-destination pair (s, d). Suppose there are k (explicit-routed) paths between source s and destination d. Using the notion of virtual capacity of a path, we treat these k paths *as if they were disjoint and each consisted of a single virtual link*. The virtual capacity of a path r is represented by vc_r, which is determined by the offered load from source s to destination d along route r and the observed blocking probability b_r of flows routed along the route. Although the real network topology of these paths may be very complex (e.g., multiple paths have shared links, or share links with other source-destination pairs), the notion of virtual capacity of a path allows us to circumvent these difficulties by essentially capturing the "capacity share" of flows routed along various paths.

The concept of virtual capacity of a path can be illustrated using the figure shown in Figure 5.6. The virtual path view of a source can be understood as follows. Imagine a network where each physical link is split into multiple virtual links, one per each path passing

```
1.  PROCEDURE VPM(σ)
2.    For each path r ∈ R_σ
3.      Compute virtual capacity vc_r^(n) = E^-1(ν_r^(n), b_r^(n))
4.    For each path r ∈ R_σ
5.      Find new load ν_r^(n+1) such that
6.      Σ_{r∈R_σ} ν_r^(n+1) E(vc_r^(n), ν_r^(n+1)) is minimum
7.      Σ_{r∈R_σ} ν_r^(n+1) = ν_σ
8.  END PROCEDURE
```

Figure 5.8: The *vpm* procedure for a pair σ

through that link. In such a network all paths would be mutually disjoint. For example, Figure 5.7(a) represents virtual path view of source 2 corresponding to the physical network in Figure 5.6(a). Similarly, Figure 5.7(b) corresponds to that of Figure 5.6(b). Here the link $1 \rightarrow 3$ is shared by paths $2 \rightarrow 1 \rightarrow 3 \rightarrow 6$, $2 \rightarrow 1 \rightarrow 3 \rightarrow 8$, $4 \rightarrow 1 \rightarrow 3$ and $4 \rightarrow 1 \rightarrow 3 \rightarrow 8$. These paths can be made to appear disjoint by splitting it into 4 virtual links.

Given the path-level virtual network view for a source-destination pair, the "optimal" flow proportions for the paths between the pair can be computed to minimize the overall flow blocking probability experienced by the flows routed along these paths. Formally, consider a source-destination pair σ. Let R_σ denote the set of paths between the source-destination pair σ. For each path $r \in R_\sigma$ let vc_r denotes the virtual capacity of the path (perceived by the source-destination pair σ). The flow proportions for the paths can be computed using an iterative procedure, referred to as the virtual path based minimization (*vpm*) procedure, as is shown in Figure 5.8. In this procedure, the virtual capacity $vc_r^{(n)}$ of each path r is computed using the Erlang inverse formula, given the current offered load $\nu_r^{(n)}$ along the path r and the corresponding observed blocking probability $b_r^{(n)}$. Based on these path virtual capacities, new loads $\{\nu_r^{(n+1)}\}$ are reassigned to paths such that $\sum_{r \in R_\sigma} \nu_r^{(n+1)} E(vc_r^{(n)}, \nu_r^{(n+1)})$ is minimized. This procedure is performed iteratively and independently at each source for all the source-destination pairs originating at the source.

Before we leave this section, it is interesting to contrast the link based and path based localized proportional routing model with the global optimal proportional routing model in the way they handle the sharing of links among paths. While the global model is aware of how the links are shared by all the paths between any source to any destination, the localized link-level model is only aware of sharing of links among the paths from the same source. The localized path-level model is completely oblivious of any link sharing. However, this lack of knowledge about explicit sharing between paths is somewhat compensated by the

notion of virtual capacity, which indirectly accounts for the effect of link sharing. Moreover, the localized models make up for the absence of such knowledge by employing an iterative process to compute flow proportions, in an attempt to approach the optimal flow proportioning. This iterative procedure can be thought of as continually refining the (partial) virtual network view of each source. Each source uses its virtual network view to compute flow proportions for routing flows along various paths, and this in turn improves the virtual network views of all the sources. This iterative procedure eventually converges to an equilibrium state that yields near-optimal flow proportions.

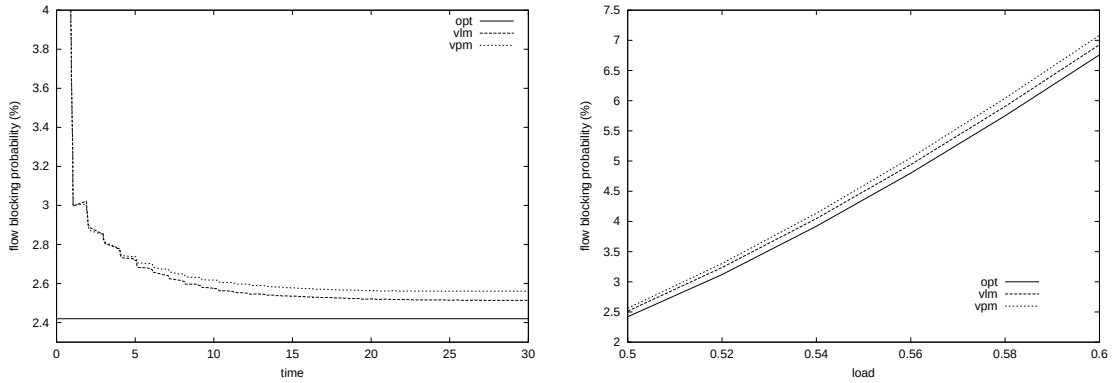
5.2.4 Performance Comparison

In this section, we demonstrate the convergence process of the localized proportional routing models, and compare their stable performance with that of the global optimal proportional routing model through numerical investigation. Before we present the results, we first describe the system setup.

5.2.4.1 System Setup

The topologies used in our study are shown in Figure 5.1. The *bigisp* is the topology of an ISP backbone network used in [2, 56] also. There are two types of links: *solid* and *dotted*. All solid links have same capacity with C_1 units of bandwidth and similarly all the dotted links have C_2 units. For simplicity, all the links are assumed to be bidirectional and of equal capacity in each direction. The nodes labeled with a bigger font are considered to be source (ingress) or destination (egress) nodes. The *minisp* topology is almost like the core of the *bigisp* topology. It has only solid links and for each source a subset of nodes (shown in smaller font) are chosen as destination nodes. By default, results presented in this section correspond to *minisp* topology and we explicitly mention whenever *bigisp* is used.

The flow dynamics of the network are modeled as follows (similar to the model used in [103]). Each flow is assumed to require one unit of bandwidth. Flows arrive at a source node according to a Poisson process with rate λ. The incoming traffic at a source is uniformly split among its destination nodes. The holding time of a flow is exponentially distributed with mean $1/\mu$. Following [103], the offered network load is given by $\rho = \lambda N\bar{h}/\mu(L_1C_1 + L_2C_2)$, where N is the number of source nodes, L_1 and L_2 are the number of solid and dotted links respectively, and $\bar{h}$ is the mean number of hops per flow, averaged across all source-destination pairs. The parameters used in our study are $C_1 = 20$,



(a) load of 0.50 (b) varying loads

Figure 5.9: Performance comparison of the optimal and the localized schemes

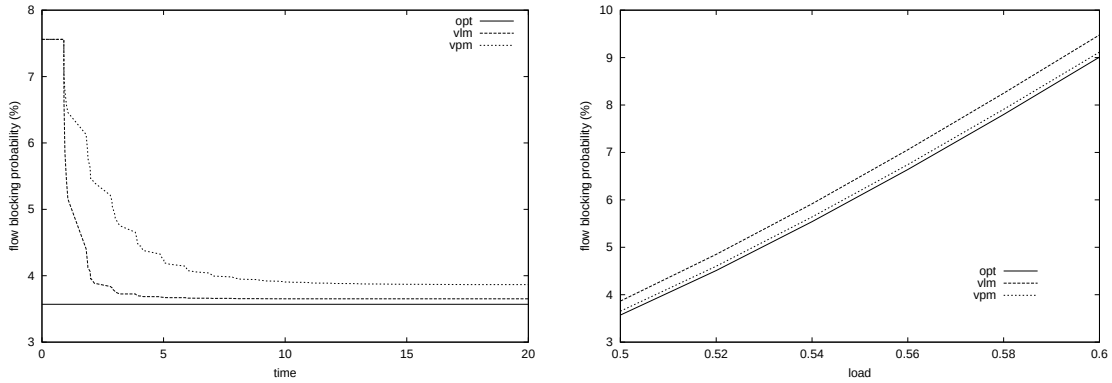
$C_2 = 30$, $\mu = 1$ minute. The topology specific parameters for *minisp* are $N = 9$, $L_1 = 26$, $L_2 = 0$, $\bar{h} = 2.64$. Similarly for *bigisp* these parameters are $N = 6$, $L_1 = 36$, $L_2 = 24$, $\bar{h} = 3.27$. The average arrival rate at a source node λ is set depending upon the desired load.

5.2.4.2 Convergence

Each source under the localized proportional routing schemes observes either link-level or path-level flow blocking probabilities and periodically recomputes flow proportions for routing flows among its paths. These reassignments of flow proportions are done independently and asynchronously by each source in a distributed manner. However with the help of the virtual capacity model, they indirectly cooperate with each other and gradually converge to near-optimal proportions as demonstrated below.

Figure 5.9(a) shows the convergence process of the localized routing schemes. The load ρ is set to 0.50 on *minisp*. Only the minhop paths are chosen as the candidate paths for each source-destination pair. The average period between recomputations is set to 1. The overall blocking probability is plotted as a function of time (i.e., the number of iterations). The performance of the global optimal routing scheme is also shown for reference. Note that the performance of the localized schemes only varies with time.

It can be seen that the overall blocking probability of both the localized routing schemes gradually decreases as the number of iterations increases. Both the schemes eventually



(a) load of 0.50 (b) varying loads

Figure 5.10: Performance under *bigisp* topology

converge and each to a different convergence point. Starting with arbitrary initial proportions, the localized schemes approach close to their respective convergence points within 15 iterations. The final convergence points are 2.56%, 2.51% respectively for the schemes *vpm* and *vlm*. The global optimal scheme yields an overall blocking probability of 2.42%. There is very little difference in the performance of finer-grained link-level scheme and the coarser-grained path-level scheme. More importantly, the performance of both the localized schemes is quite close to that of the global optimal scheme.

5.2.4.3 Blocking Performance

The performance of the localized schemes is studied under various load conditions. The load is varied from 0.50 to 0.60. The overall blocking probability at the convergence point of each experiment is plotted as a function of load in Figure 5.9(b). It can be seen that the performance of the *vlm* scheme is slightly better than that of the *vpm* scheme across all loads, and that their performance is quite close to that of the *opt* scheme.

We have also run experiments with the larger *bigisp* topology. These results are shown in Figure 5.10. As before, both the localized virtual capacity based schemes gradually converge to near-optimal proportions even with *bigisp* topology. The blocking at the final convergence points are 3.653% and 3.866% respectively for *vlm* and *vpm* while the optimal scheme yields an overall blocking of 3.57%. Comparatively, these schemes seem to converge relatively sooner in case of *bigisp* than *minisp*. Also, the performance difference between *vlm* and *vpm* is slightly higher in case of *bigisp* than *minisp*. However, the results

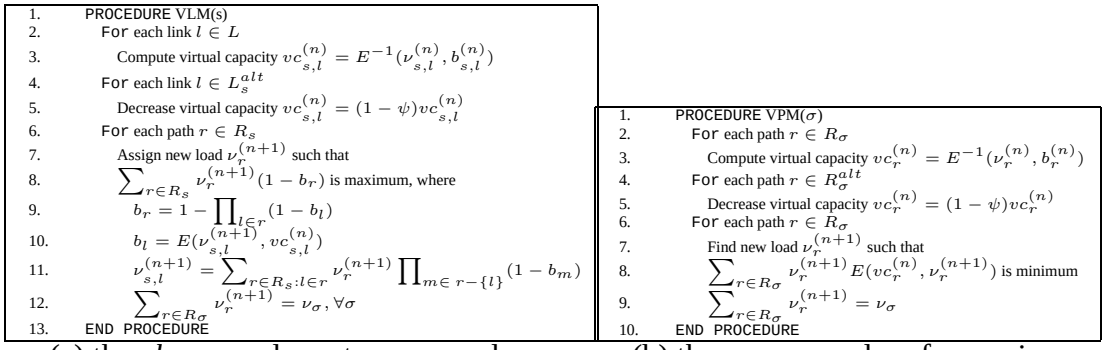
of *minisp* and *bigisp* are not very different and the observations made with *minisp* are applicable to *bigisp* also. In the next section, we illustrate the effectiveness of localized trunk reservation using *minisp* and show some simulation results on *bigisp* in the later sections.

5.3 Alternative Paths and Localized Trunk Reservation

The virtual capacity based local minimization schemes described in the previous section treat all candidate paths equally. Since an admitted flow consumes bandwidth and buffer resources at all the links along a path, clearly path length is also an important factor that must be taken into consideration. There is a fundamental trade-off between minimizing the resource usage by choosing shorter paths and balancing the network load by using lightly loaded longer paths. As a general principle, it is preferable to route a flow along *minhop* (i.e. shortest) paths than paths of longer length (also referred to as *alternative* paths). By preferring minhop paths and discriminating against alternative paths, we not only reduce the overall resource usage but also limit the so-called "knock-on" effect [41, 42], thereby ensuring the stability of the whole system.

The "knock-on" effect refers to the phenomenon where using alternative paths by some sources forces other sources whose minhop paths share links with these alternative paths to also use alternative paths. This cascading effect can cause a drastic reduction of the overall throughput of the network. In order to deal with the "knock-on" effect, trunk reservation [42] is employed where a certain amount of bandwidth on a link is reserved for minhop paths only. With trunk reservation, a flow may be rejected even if sufficient resources are available to accommodate it. A flow along a path longer than its minhop path is admitted only if the available bandwidth even after admitting this flow is greater than the amount of trunk reserved. Trunk reservation provides a simple and yet effective mechanism to control the "knock-on" effect. However, trunk reservation cannot be used directly in localized routing schemes, since it requires global configuration. Furthermore, core routers have to figure out whether a setup request for a flow is sent along its minhop path or not, introducing undesirable burden on them. We propose to address this by having each source router locally discriminate against its own alternative paths *without any explicit global trunk reservation*.

A source node employing a localized scheme can control the amount of alternative routing by adjusting the virtual capacities in its virtual network. This can be thought of as an implicit localized trunk reservation performed by each source independently. The exact method in which alternative paths are discriminated varies between *vlm* and *vpm*. While



(a) the *vlm* procedure at source node s (b) the *vpm* procedure for a pair σ

Figure 5.11: The localized schemes with implicit trunk reservation

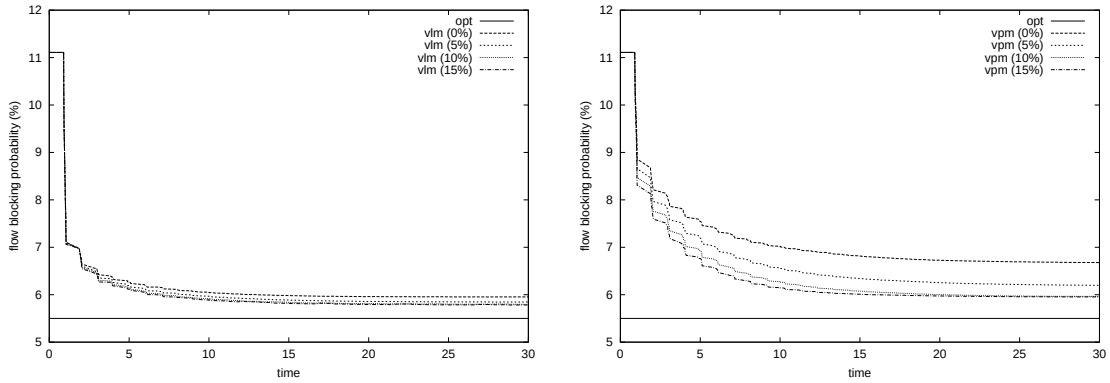
vlm employs link-level discrimination, *vpm* does path-level discrimination. The details are presented below.

5.3.1 Localized Link-level Trunk Reservation

Each source router after determining the corresponding virtual private network adjusts the virtual capacities of its links to account for trunk reservation. From the perspective of a source, a link is categorized into two cases: *alternative-only* or *minhop-also*. A link l is said to be *alternative-only* link w.r.t. a source s, if l lies only along alternative paths from the source s to its destinations. Otherwise if a minhop path from source s to any destination passes through link l, then l is categorized as *minhop-also* w.r.t. source s. The links that are used only by alternative paths for routing traffic from this source are targeted for the adjustment. Their capacities are reduced by an amount ψ where, ψ is the trunk reservation parameter, i.e., $vc_{s,l} = (1-\psi)vc_{s,l}$ if l is *alternative-only* w.r.t. source s. The capacities of other links are left unchanged. The modified *vlm* procedure that incorporates localized trunk reservation is shown in Figure 5.11(a). The virtual capacities for *alternative-only* links are adjusted in lines 4-5. The rest of the procedure remains unchanged.

We now present the rationale behind the way a source under *vlm* categorizes a link as *alternative-only* and applies trunk reservation on it. Consider the three possible cases that a link l can fall into: 1) *minhop-also* w.r.t. to s; 2) *alternative-only* w.r.t. to s and *minhop-also* w.r.t. some other source; 3) *alternative-only* w.r.t. to all sources. We address each of these cases separately as follows

Case 1: No explicit discrimination against alternative paths is required in this case since



(a) *vlm* (b) *vpm*

Figure 5.12: The effectiveness of localized trunk reservation (load = 0.60)

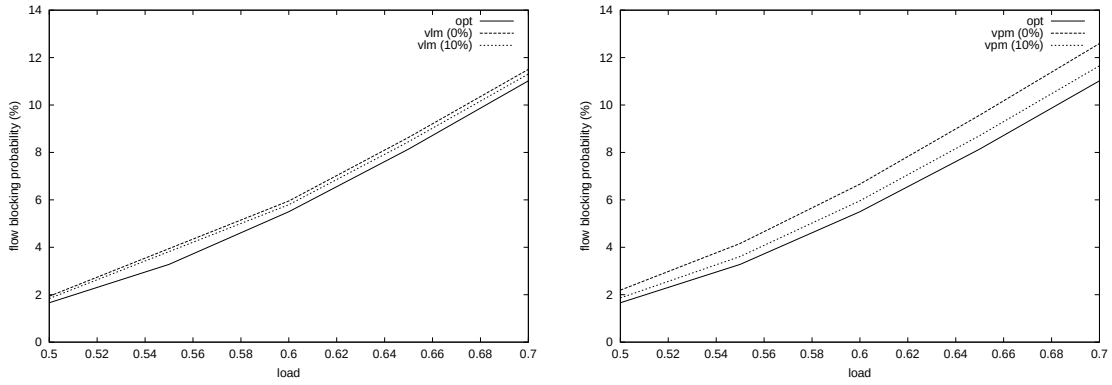
minimization procedure at source s is expected to account for resource usage while dealing with any sharing of this link l by an alternative path. An alternative path would be assigned an higher capacity on link l only if the minhop path is relatively bad due to other links in its path.

Case 2: By locally reducing the target virtual capacity of link l, source s voluntarily backs off from such links and avoids knock-on effect. This allows minhop paths of other sources to gradually occupy more share on link l. This reduces the resource usage and in turn decreases the overall blocking probability.

Case 3: In this case all the sources reduce the target virtual capacity of link l even though it may not be necessary. This could lead to under-utilization of the link. However, the extent of under-utilization of such links can be limited by setting a low value for trunk reservation parameter ψ since it can be shown that knock-on effect can be avoided even with reasonably small ψ values.

5.3.2 Localized Path-level Trunk Reservation

The virtual path based scheme limits the extent of alternative routing by applying path-level discrimination against alternative paths. It locally adjusts the target virtual capacity of alternative paths. Given a trunk reservation parameter ψ, the target virtual capacity of alternative paths is reduced by an amount ψ, i.e., $vc_r = (1-\psi)vc_r$ if r is an alternative path. The virtual capacities of minhop paths are left unchanged. The minimization procedure is



(a) *vlm* (b) *vpm*

Figure 5.13: The effectiveness of localized trunk reservation

then applied locally at the source on the virtual network with these adjusted capacities. The revised *vpm* procedure is shown in Figure 5.11(b). The virtual capacities for alternative paths are adjusted in lines 4-5.

Once again this discrimination against alternative paths helps avoid the knock-on effect. As described in link-level case, when an alternative path shares a link with a minhop path, in path-level case also this implicit trunk reservation by a source helps minhop paths capture more share forcing alternative paths to gradually back off. In other words, the virtual capacity of an alternative path keeps reducing when it shares a bottleneck link with a minhop path. On the other hand when an alternative path doesn't share any bottleneck links with any minhop path, the amount of reduction in its target virtual capacity will be limited by the trunk reservation parameter ψ thus limiting the amount of under-utilization of resources.

5.3.3 Effectiveness of Localized Trunk Reservation

We now study the effectiveness of localized trunk reservation method and the impact of the parameter ψ on the performance. The setup described in Section 5.2.4.1 is used in this study also. However, apart from the minhop paths, paths of length minhop+1 are also chosen as the candidate paths for each source-destination pair.

The Figure 5.12 shows the convergence process where overall blocking probability is plotted as a function of time for a load of 0.60. The performance of local schemes is shown for four different values of the trunk reservation parameter ψ: 0%,5%,10%,15%. The blocking

probability of global optimal scheme is shown for reference. It is quite evident that the performance of localized schemes with ($> 0\%$) trunk reservation is better than without (0%) it. However, as the ψ value is increased the performance gain is reduced. There is almost no difference in performance between ψ values of 10% and 15%. We also ran for several cases with loads ranging from 0.50 to 0.70. In each case the localized schemes were run with ψ values of 0% and 10%. The Figure 5.13 shows the overall blocking probability as a function of load. Once again, across all loads, the performance of schemes with localized trunk reservation is better than without it.

These results show that localized trunk reservation is quite effective. However, the impact of localized trunk reservation on *vlm* is much less significant than on *vpm*. This is expected since *vlm* even without any trunk reservation, using finer-grained link-level information, accounts for sharing of links between minhop and alternative paths from a source to all its destinations. The role of localized trunk reservation in *vlm* is limited to avoiding overloading of minhop paths of a source by traffic on alternative paths of another source. On the other hand, the localized trunk reservation plays a much more critical role in *vpm*. The minhop paths to a destination have to be guarded from alternative paths to the same destination besides from alternative paths to other destinations. This is due to availability of only coarser-grain information and thus lack of knowledge about sharing of links between different paths. However with localized trunk reservation, the *vpm* scheme tides over this shortcoming and performs comparably to the *vlm* scheme. Hereafter, we focus only on the proportional routing schemes such as *vpm* that are based on path-level information which is easier to collect with less overhead than link-level information.

Chapter 6

Localized Proportional Routing: Practical Schemes

The localized schemes described in the previous chapter have been shown to approach the performance of the optimal scheme using only local information. Furthermore, even with coarser-grain path-level blocking information the *vpm* scheme performs as well as the *vlm* scheme that uses finer-grain link-level blocking information. It is easier to collect path-level statistics and simpler to implement path-based schemes. Hence we focus on path-based localized schemes and further investigate the issues involved in implementing them.

6.1 Heuristic Equalization Strategies

The *vpm* scheme first computes the virtual capacity of each candidate path and then performs local minimization. Though the complexity of this minimization procedure is much less than that of optimal scheme, it could still be significant. We are interested in simple schemes that are easy to implement. A simple alternative to *minimization* procedure is *equalization* of either blocking probabilities or blocking rates of candidate paths.

Equalization of Blocking Probabilities

The objective of the equalization of blocking probabilities (*ebp*) strategy is to find a set of proportions $\{\tilde{\alpha}_1, \tilde{\alpha}_2, \ldots, \tilde{\alpha}_k\}$ such that flow blocking probabilities of all the paths are

1. PROCEDURE VCEBP(σ)
2. Set mean blocking rate of minhop paths, $\bar{\beta}^{(n)} = \sum_{r \in R_\sigma^{min}} \alpha_r^{(n)} b_r^{(n)}$
3. Set minimum of minhop path's blocking probability, $b^* = \min_{r \in R_\sigma^{min}} b_r^{(n)}$
4. For each path $r \in R_\sigma^{min}$
5. Compute virtual capacity $vc_r^{(n)} = E^{-1}(\nu_r^{(n)}, b_r^{(n)})$
6. For each path $r \in R_\sigma^{min}$
7. Compute target load ν_r^* such that $\bar{\beta}^{(n)} = E(\nu_r^*, vc_r^{(n)})$
8. For each alternative path $r \in R_\sigma^{alt}$
9. Compute target load ν_r^* such that $(1-\psi)b^* = E(\nu_r^*, vc_r^{(n)})$
10. For each path $r \in R_\sigma$
11. Compute new proportion $\alpha_r^{(n+1)} = \frac{\nu_r^*}{\sum_{r \in R_\sigma} \nu_r^*}$
12. END PROCEDURE

Figure 6.1: The *vcebp* procedure for a source-destination pair σ

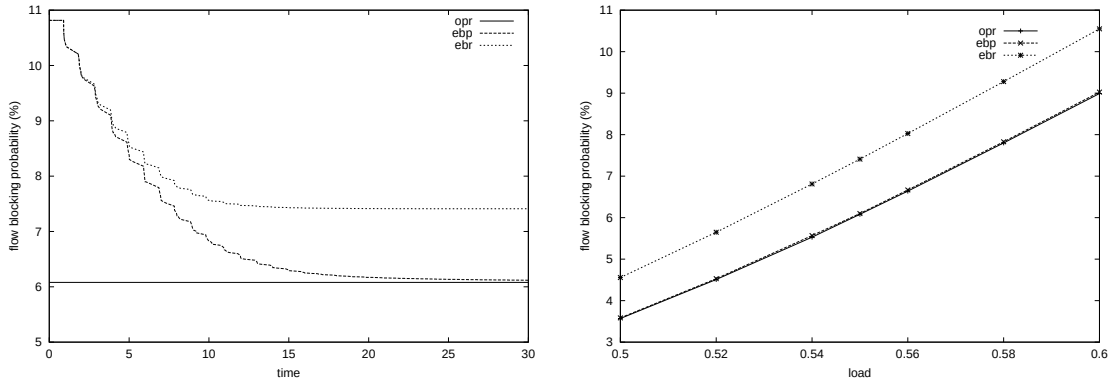
equalized, i.e., $\tilde{b}_1 = \tilde{b}_2 = \cdots = \tilde{b}_k$, where $\tilde{b}_i$ is the flow blocking probability of path r_i, and is given by $E(\tilde{\alpha}_i \nu, c_i)$.

Equalization of Blocking Rates

The objective of the equalization of blocking rates (*ebr*) strategy is to find a set of proportions $\{\hat{\alpha}_1, \hat{\alpha}_2, \ldots, \hat{\alpha}_k\}$ such that *flow blocking rates* of all the paths are equalized, i.e., $\hat{\alpha}_1 \hat{b}_1 = \hat{\alpha}_2 \hat{b}_2 = \cdots = \hat{\alpha}_k \hat{b}_k$, where $\hat{b}_i$ is the flow blocking probability of path r_i, and is given by $E(\hat{\alpha}_i \nu, c_i)$.

The proportions corresponding to the above equalization strategies can be computed using an iterative procedure similar to the *vpm* procedure. The key difference is in the way new proportions for paths are computed based on their current virtual capacities in each iteration. While *vpm* tries to minimize the aggregate blocking probability for flows between a source-destination pair, equalization strategies attempt to equalize the probability or rate of blocking experienced by flows across different candidate paths between the pair.

The virtual capacity based equalization of blocking probabilities (*vcebp*) procedure is shown in Figure 6.1. At any given iteration $n \geq 0$, let $\nu_r^{(n)}$ be the amount of the load currently routed along a path $r \in R_\sigma$, and let $b_r^{(n)}$ be its observed blocking probability on the path. Then the virtual capacity of path r is given by $vc_r = E^{-1}(\nu_r^{(n)}, b_r^{(n)})$ (line 5). For each minhop path, the mean blocking probability of all the minhop paths, $\bar{\beta}^{(n)}$, is used to compute a new target load (lines 6-7). Similarly, for each alternative path, a new target load is computed using the target blocking probability $(1-\psi)b^*$ (lines 8-9). Here b^* is the minimum flow blocking probability of all the minhop paths (line 3) and ψ is a configurable parameter



(a) convergence process

(b) blocking under varying load

Figure 6.2: Performance of *ebp* and *ebr*

to limit the knock-on effect. The basic idea behind this alternative routing method is to ensure that an alternative path is used to route flows between the source-destination pair only if it has a "better quality" (measured in flow blocking probability) than any of the minhop paths. Given these new target loads for all the paths, the new proportion of flows, $\alpha_r^{(n+1)}$, for each path r is obtained in lines 10-11, resulting in a new load $\nu_r^{(n+1)} = \alpha_r^{(n+1)}\nu_\sigma$ on path r. Similar procedure can be used to equalize the blocking rates also by computing mean blocking rate in line 2 and using that as the target rate in line 7 for computing the target loads.

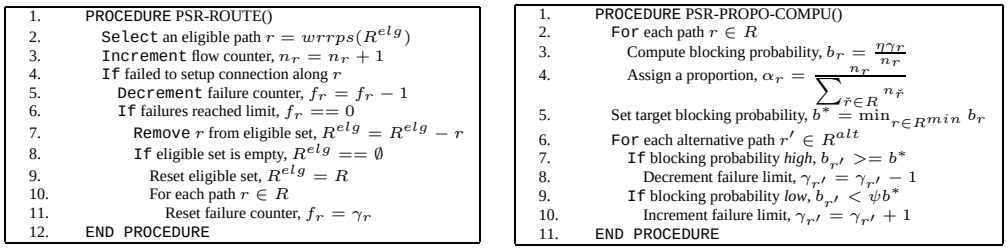
Figure 6.2 compares the performance of *ebp* and *ebr* with *opr*. We consider the ideal case where the observed blocking probabilities on paths are precisely computed numerically. These experiments are conducted on the *bigisp* topology shown in Figure 5.1(a) in the previous chapter. In these experiments only the minhop paths are made candidates and the load is set to 0.55. Figure 6.2(a) shows the convergence of the proposed heuristic schemes. We can see that *ebp* scheme converges to almost optimal proportions. The *ebr* scheme also converges but its blocking performance is worse than *ebp*. Figure 6.2(b) shows the blocking performance of these schemes under various offered loads. Across all loads, there is no discernible difference in the performance between *opr* and *ebp*. On the other hand, *ebr* performs consistently worse than *ebp*. The reason is that *ebr* scheme assigns more load to a higher blocking path than *ebp* scheme in an attempt to equalize the blocking rates of candidate paths. This effect gets amplified when there is lot of sharing of links between paths. However, *ebr* is a reasonable strategy that effectively assigns loads to paths in proportions

that are inversely proportional to their blocking probabilities. In the following we proceed to explore practical implementations of both these strategies.

The localized routing schemes presented so far are based on theoretical virtual capacity model. We have shown that they yield near-optimal performance using only local information. However, there are two difficulties involved in implementing the virtual capacity model. First, computation of virtual capacity and target load using Erlang's Loss Formula can be quite cumbersome. Second, and perhaps more importantly, the accuracy in using Erlang's Loss Formula to compute virtual capacity and new load relies critically on steady-state observation of flow blocking probability. Hence small statistic variations may lead to erroneous flow proportioning, causing undesirable load fluctuations. In order to circumvent these difficulties, we are interested in simple yet robust implementation of these schemes. In the following we discuss two such schemes that approximate *ebr* and *ebp* respectively. We first present the *psr* scheme that we designed initially and show that it is viable alternative to popular global best-path routing scheme *wsp*. We then describe an approximation of *ebp* that is found to perform even better than *psr*.

6.2 Proportional Sticky Routing

The proportional sticky routing (*psr*) scheme attempts to equalize blocking rates of candidate paths. It is called by that name because it essentially does proportional routing while obtaining proportions through a form of sticky routing. The *psr* scheme can be viewed to operate in two stages: 1) proportional flow routing, and 2) computation of flow proportions. The proportional flow routing stage proceeds in *cycles* of variable length. During each cycle incoming flows are routed along paths selected from a set of eligible paths. A path is selected with a frequency determined by a prescribed proportion. A number of cycles form an *observation period*, at the end of which a new flow proportion for each path is computed based on its observed blocking probability. This is the computation of flow proportion stage. The flow proportions for minhop paths of a source-destination pair are determined using the *ebr* strategy, whereas flow proportions for alternative paths are determined using a target blocking probability. In the following we will describe these two stages in more detail.



(a) proportional routing

(b) computation of proportions

Figure 6.3: The *psr* procedure

Proportional flow routing

Given an arbitrary source-destination pair, let R be the set of (*explicit-routed*) paths between the source-destination pair, where $R = R^{min} \cup R^{alt}$. We associate with each path $r \in R$, a *maximum permissible flow blocking* parameter γ_r and a corresponding *flow blocking counter* f_r. For each minhop path $r \in R^{min}$, $\gamma_r = \hat{\gamma}$, where $\hat{\gamma}$ is a configurable system parameter. For each alternative path $r' \in R^{alt}$, the value of $\gamma_{r'}$ is dynamically adjusted between 1 and $\hat{\gamma}$, as will be explained later. As shown in Figure 6.3(a), at the beginning of each cycle, f_r is set to γ_r. Every time a flow routed along path r is blocked, f_r is decremented. When f_r reaches zero, path r is considered *ineligible*. At any time only the set of *eligible* paths, denoted by R^{elg}, is used to route flows. A path from current eligible path set R^{elg} is selected using a weighted-round-robin-like path selector (*wrrps*). The *wrrps* procedure is described below. Once R^{elg} becomes empty, the current cycle is ended and a new cycle is started with $R^{elg} = R$ and $f_r = \gamma_r$.

Weighted Round Robin Procedure for Path Selection

Given a set R^{elg} of eligible paths and their associated proportions $\{\alpha_r, r \in R^{elg}\}$, *wrrps* picks a path $r \in R^{elg}$ based on its weight, $w_r = \frac{\alpha_r}{\sum_{s\in R^{elg}} \alpha_s}$. Instead of using a probabilistic method such as picking a path r with probability w_r, we opt to employ a deterministic algorithm to ensure that flow proportions are preserved within as small a time window as possible. This is implemented by using a deterministic sequence of paths which has the property that the paths are distributed periodically with a frequency which closely approximates the prescribed flow proportions. This sequence is generated by *wrrps* on the fly: for an incoming flow, *wrrps* generates the next path in the sequence and routes the flow along the path.

k	:	total number of paths in set R^{elg}.
r_i	:	path associated with index i.
w_{r_i}	:	weight associated with path r_i.
n_{r_i}	:	number of times path r_i was selected.
l	:	run length of the most recently selected path.
$\mathcal{W}_i$	:	$w_{r_i} + w_{r_{i+1}} + \cdots + w_{r_k}$.
$\mathcal{N}_i$	:	$n_{r_i} + n_{r_{i+1}} + \cdots + n_{r_k}$.

(a) notation

```
1.   PROCEDURE wrrps()
2.     For i = 1, 2, ..., k
3.       If l*W_{i+1} < w_{r_i} and W_{i+1}*n_{r_i} <= w_{r_i}*N_{i+1}
4.         break
5.       Set W_{i+1} = W_{i+1} + w_{r_i} - w_{r_{i+1}}
6.       Set N_{i+1} = N_{i+1} + n_{r_i} - n_{r_{i+1}}
7.       Swap r_i and r_{i+1}
8.       Set l = 0
9.     Set n_{r_0} = n_{r_0} + 1; N_{r_0} = N_{r_0} + 1;
10.    Set l = l + 1
11.    Return r_0
12.  END PROCEDURE
```

(b) path selection

Figure 6.4: The *wrrps* procedure

The *wrrps* procedure is shown in Figure 6.4. It keeps track of the number of times each path was selected (n_{r_i}) and the run length (l) of the most recently selected path. It maintains an ordered list of paths and the first path in the list is selected as long as it satisfies both the following constraints: 1) its weight is more than its run length times the weight of the rest of paths ($l\mathcal{W}_1 < w_{r_0}$); 2) ratio of the number of times it was selected and the number of times all others were selected is less than or equal to the ratio of its weight and weight of the rest of paths ($\mathcal{W}_1 n_{r_0} \leq w_{r_0}\mathcal{N}_1$). Otherwise this path is pushed down the order and the run length is reset to 0. Then it returns the first path in the list. A sample *wrrps* generated sequence where the current eligible set R^{elg} has four paths r_1, r_2, r_3 and r_4 with weights $1/2$,$1/4$,$1/8$, and $1/8$ respectively is: $r_1\ r_2\ r_1\ r_3\ r_1\ r_2\ r_1\ r_4\ r_1\ r_2\ r_1\ \mathbf{r_3}$. This sequence has the property that in every window of size 2 there is an r_1 and an r_2 in every window of size 4. Similarly, one r_3 and one r_4 in all windows of size 8. Assuming that up to the last r_1 are the paths chosen so far, the next path selected on the fly by the *wrr* path selector would be r_3. Note also that every time the eligible path set R^{elg} changes, a new sequence is generated, and flows arriving thereafter are thus routed according to this new sequence.

Computation of flow proportions

Flow proportions $\{\alpha_r, r \in R\}$ are recomputed at the end of each observation period (see Figure 6.3(b)). An observation period consists of η cycles, where η is a configurable system parameter used to control the robustness and stability of flow statistics measurement. During each observation period, we keep track of the number of flows routed along each path $r \in R$ using a counter n_r. At the beginning of an observation period, n_r is set to 0. Every time path r is used to route a flow, n_r is incremented. Since an observation period consists of η cycles, and in every cycle, each path r has exactly γ_r flows blocked, the observed flow blocking probability on path r is $b_r = \frac{\eta\gamma_r}{n_r}$. For each minhop path

$r \in R^{min}$, its new proportion α_r is recomputed at the end of an observation period and is given by $\alpha_r = n_r/n_{total}$, where $n_{total} = \sum_{r \in R} n_r$ is the total number of flows routed during an observation period. Recall that for a minhop path $r \in R^{min}$, $\gamma_r = \hat{\gamma}$. Hence $\alpha_r b_r = \frac{n_r}{n_{total}} \frac{\eta\gamma_r}{n_r} = \frac{n_r}{n_{total}} \frac{\eta\hat{\gamma}}{n_r} = \frac{\eta\hat{\gamma}}{n_{total}}$. This shows that the above method of assigning flow proportions for the minhop paths equalizes their flow blocking rates.

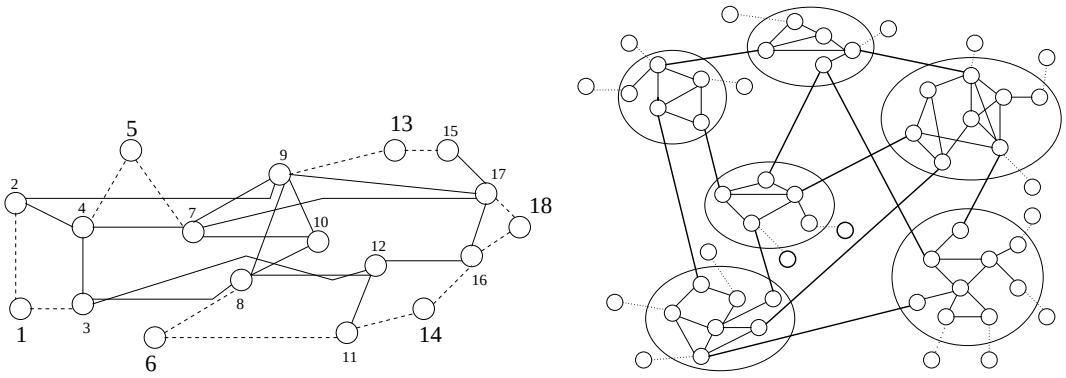
We use the minimum blocking probability among the minhop paths, $b^* = \min_{r \in R^{min}} b_r$, as the reference to control flow proportions for the alternative paths. This is done implicitly by dynamically adjusting the maximum permissible flow blocking parameter $\gamma_{r'}$ for each alternative path $r' \in R^{alt}$. At the end of an observation period, let $b_{r'} = \frac{\eta\gamma_{r'}}{n_{r'}}$ be the observed flow blocking probability for an alternative path r'. If $b_{r'} > b^*$, $\gamma_{r'} := \max\{\gamma_{r'} - 1, 1\}$. If $b_{r'} < \psi b^*$, $\gamma_{r'} := \min\{\gamma_{r'} + 1, \hat{\gamma}\}$. If $\psi b^* \leq b_{r'} \leq b^*$, $\gamma_{r'}$ is not changed. By having $\gamma_{r'} \geq 1$, we ensure that some flows are occasionally routed along alternative path r' to probe its "quality", whereas by keeping $\gamma_{r'}$ always below $\hat{\gamma}$, we guarantee that minhop paths are always preferred to alternative paths in routing flows. The new proportion for each alternative path r' is again given by $\alpha_{r'} = n_{r'}/n_{total}$. Note that since $\gamma_{r'}$ is adjusted for the next observation period, the *actual* number of flows routed along alternative path r' will be also adjusted accordingly.

6.2.1 Performance Evaluation and Analysis

This section evaluates the performance of the proposed localized proportional routing scheme *psr* and compares it with the global best-path routing scheme *widest shortest path* (*wsp*). We start with the description of the simulation environment and then compare the performance of *psr* and *wsp* in terms of the overall blocking probability and routing overhead.

6.2.1.1 Simulation Environment

Figure 6.5 shows the two topologies, *bigisp* and *rand*, used in our study. The *bigisp* topology is same as the one in Figure 5.1(a). However, the simulation setting here is somewhat different from the one described in Section 5.2.4.1. Hence we detail the simulation environment again though there is quite a bit of overlap in the settings. The *rand* topology is a random graph generated by GT-ITM [113] and used in [31]. For simplicity, all the links are assumed to be bidirectional and of equal capacity in each direction. The *rand* topology has three types of links: *thin*, *thick* and *dotted* while *bigisp* topology has only *thin* links. All *thin* links have same capacity with C_1 units of bandwidth and similarly all the *thick* links

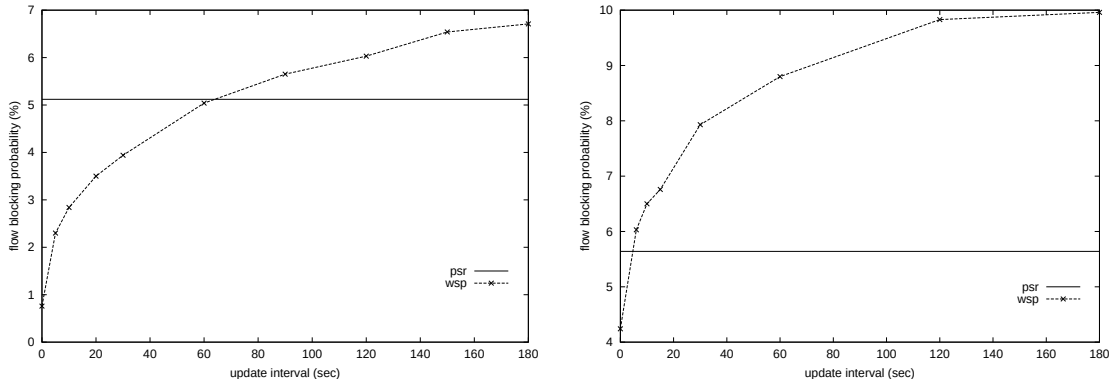


(a) *bigisp* (b) *rand*

Figure 6.5: Topologies used in performance evaluation

have C_2 units. The *dotted* links are the access links and for the purpose of our study their capacity is assumed to be infinite. Flows arriving into the network are assumed to require one unit of bandwidth. Hence a link with capacity C can accommodate at most C flows simultaneously.

The flow dynamics of the network is modeled as follows (similar to the model used in [103]). Flows arrive at a source node according to a Poisson process with rate λ. The destination node of a flow is chosen randomly from the set of all nodes except the source node. In case of *bigisp* all nodes are considered to be capable of being source or destination nodes for flows. But in case of *rand* topology, only the nodes attached to the *dotted* access links are assumed to be end points of flows. The holding time of a flow is exponentially distributed with mean $1/\mu$. Following [103], the offered network load on *bigisp* is given by $\rho = \lambda N\bar{h}/\mu L_1 C_1$, where N is the number of source nodes, L_1 the number of links, and $\bar{h}$ is the mean number of hops per flow, averaged across all source-destination pairs. Similarly the offered load on *rand* is given by $\rho = \lambda N\bar{h}/\mu(L_1C_1 + L_2C_2)$, where L_1 and L_2 are the number of *thin* and *thick* links respectively. The parameters used in simulation are $C_1 = 20$, $C_2 = 40$, $1/\mu = 60$ sec. The topology specific parameters are $N = 18$, $L_1 = 60$, $\bar{h} = 2.36$ for *bigisp* and $N = 56$, $L_1 = 100$, $L_2 = 22$, $\bar{h} = 4.38$ for *rand*. The average arrival rate at a source node λ is set depending upon the desired load.



(a) *bigisp* ($\rho = 0.60$) (b) *rand* ($\rho = 0.40$)

Figure 6.6: Impact of update interval

6.2.1.2 Blocking Probability

The performance of *wsp* and *psr* is compared by measuring the blocking probability under various settings. We first present the impact of update interval on the performance of *wsp* and show how the blocking probability increases rapidly as update interval is increased. We then compare the performance under various loads and conclude that the blocking probability of *psr* is comparable to that of *wsp* even at small update intervals. Finally we measure their performance under non-uniform load conditions and demonstrate that *psr* is better at alleviating the effect of "hot spots".

Varying update interval

Figure 6.6 compares the performance of *wsp* and *psr* for both *bigisp* and *rand* topologies. The values for configurable parameters in *psr* were set to $\eta = 3$, $\hat{\gamma} = 5$, and $\psi = 0.8$. The performance is measured in terms of the overall flow blocking probability, which is defined as the ratio of the total number of blocks to the total number of flow arrivals. The overall blocking probability is plotted as a function of the update interval. The offered load was set to 0.60 in case of *bigisp* and 0.40 in case of *rand*. Note that update interval is used only in *wsp* in conjunction with a threshold based triggering policy, to enforce a minimum spacing between updates. The threshold based policies trigger an update whenever the percent of change in bandwidth is greater than a constant threshold value. In our simulations, this threshold is set to 50%[1]. From the figures, we see that as the update interval of *wsp* in-

[1]Note that blocking performance of threshold based trigger with hold-down timer T would be no better

creases, the blocking probability of *wsp* rapidly approaches that of *psr* and performs worse for larger update intervals. In the case of *bigisp* topology, *psr* performs better than *wsp* when the update interval goes beyond 65 sec. For *rand* topology, this crossover happens at a much smaller update interval of less than 10 sec. This shows that *psr* using only local information performs better than global information exchange based *wsp* even when the update interval is reasonably low.

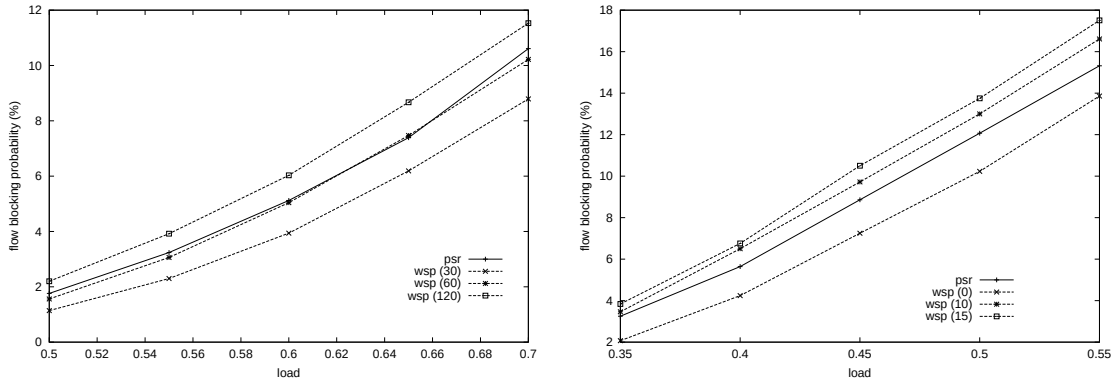
Varying offered load

Figure 6.7 shows the blocking performance of these two schemes as a function of the offered network load. As before, the performance is measured in terms of the overall flow blocking probability. The network load is varied from 0.50 to 0.70 in case *bigisp* and 0.35 to 0.55 in case of *rand*. The performance of *wsp* is plotted for three update intervals of 30, 60 and 120 for the *bigisp* case and similarly for 0, 10, and 15 in case of *rand*. It is clear that the blocking performance of *psr* is quite similar to that of *wsp* with update interval of 60 sec in case of *bigisp* topology. For the case of *rand* the *psr*'s performance is better than *wsp* with update interval of 10 sec. The performance of *wsp* is poor, particularly in case of *rand* where the paths are longer and the number of *minhop* paths are fewer. In such a case, *wsp* scheme due to its preference for shortest paths, seem to select *minhop* paths even when they are congested. On the other hand, *psr* uses alternate paths judiciously and yields much lower blocking probability. These results indicate that even at smaller update intervals *wsp* fares no better than *psr*.

Varying non-uniform traffic

It is likely that a source node receives a larger number of flows to a few specific destinations [11], i.e, a few destinations are "hot". Ideally a source would like to have more up-to-date view of the QoS state of the links along the paths to these "hot" destinations. In the case of *wsp*, this requires more frequent QoS state updates, resulting in increased overhead. But in the case of *psr*, because of its adaptivity and statistics collection mechanism, a source does have more accurate information about the frequently used routes and thus alleviates the effect of "hot spots". We illustrate this by introducing increased levels of traffic between certain pairs of network nodes ("hot pairs"), as was done in [2]. Apart from the normal load that is distributed between all source-destination pairs, an additional load (hot load) is distributed among all the hot pair nodes. The hot pairs chosen for *bigisp*

than simple timer based trigger with update interval of T. The key difference is in the amount of update message overhead.



(a) *bigisp* (b) *rand*

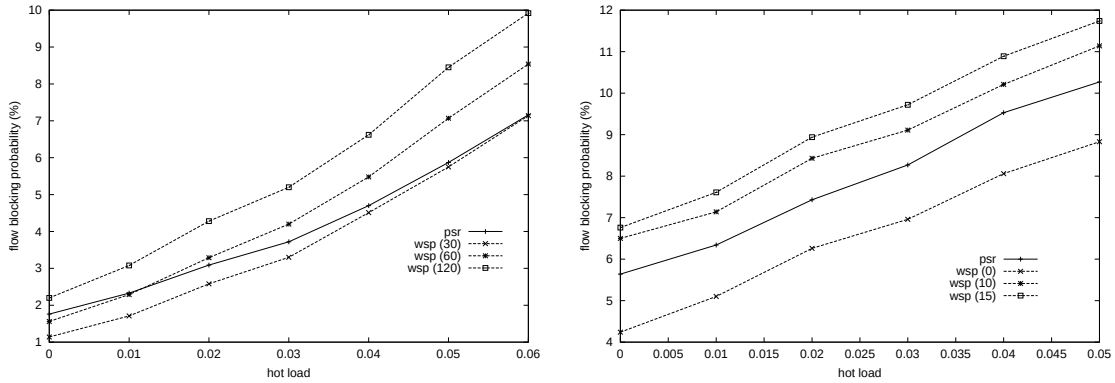
Figure 6.7: Performance under various loads

topology are $(2, 16)$, $(3, 17)$, and $(6, 15)$. Similarly in case of *rand* topology, the thick routers are considered to be the web servers and hence more traffic is assumed to flow from these two nodes to all other nodes. The normal load is fixed at 0.50 for *bigisp* and 0.40 for *rand* and the extra load is varied. Figure 6.8 shows the overall flow blocking probability as a function of the extra load.

First consider the case of *bigisp* topology. When the additional load is less than 0.01, the blocking performance of *psr* is comparable to *wsp* with update interval of 60 sec. But as the traffic between hot pairs increases, *psr* progressively does better in comparison to *wsp*. Particularly when the hot load is 0.06, the performance of *psr* is even better than *wsp* with an update interval of 30 sec. In case of *rand* topology the performance of *psr* is much better than *wsp* with update interval 10 sec irrespective of the amount of hot load. This not only shows the limitation of global QoS routing schemes such as *wsp* but also illustrates the advantage of self-adaptivity in localized QoS routing schemes such as *psr*.

6.2.1.3 Heterogeneous Traffic

The discussion so far is focused on the case where the traffic is homogeneous, i.e., all flows request for one unit of bandwidth and their holding times are derived from the same exponential distribution with a fixed mean value. Here we study the applicability of *psr* in routing heterogeneous traffic where flows could request for varying bandwidths with their holding times derived from different distributions. We demonstrate that *psr* is insensitive to the duration of individual flows and hence we do not need to differentiate flows based on



(a) *bigisp* (b) *rand*

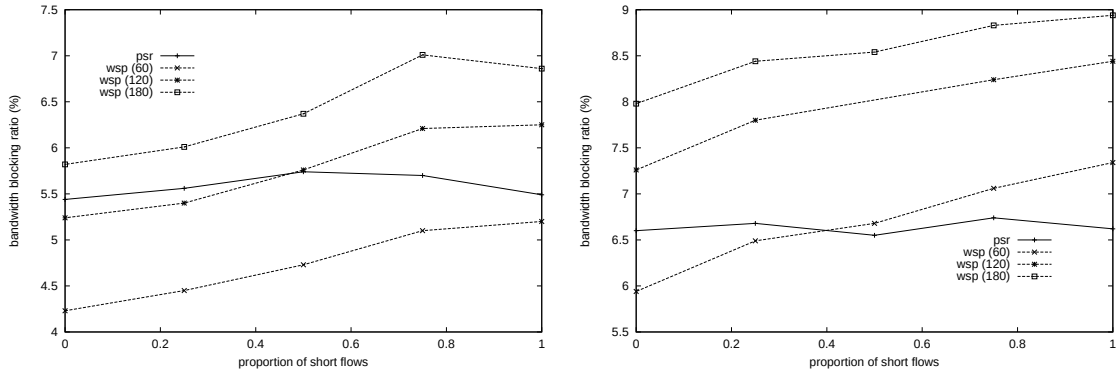
Figure 6.8: Performance under non-uniform load conditions

their holding times. We also show that when the link capacities are considerably higher than the average bandwidth request of flows, it may not be necessary to treat them differently and hence *psr* can be used *as is* to route heterogeneous traffic.

Consider the case of traffic with k types of flows, each flow of type i having a mean holding time $1/\mu_i$ and requesting bandwidth B_i. Let ρ_i be the offered load on the network due to flows of type i, where the total offered load, $\rho = \sum_{i=1}^{k} \rho_i$. The fraction of total traffic that is of type i, $\phi_i = \rho_i/\rho$. The arrival rate of type i flows at a source node, λ_i is given by $\lambda_i = \rho_i \mu_i LC/N\bar{h}B_i$, which is an extension of the formula presented in Section 6.2.1.1. To account for the heterogeneity of traffic, bandwidth blocking ratio is used as the performance metric for comparing different routing schemes. The bandwidth blocking ratio is defined as the ratio of the bandwidth usage corresponding to blocked flows and the total bandwidth usage of all the offered traffic. Suppose b_i is the observed blocking probability for flows of type i, then the bandwidth blocking ratio is given by $\frac{\sum_{i=1}^{k} \frac{b_i \lambda_i B_i}{\mu_i}}{\sum_{i=1}^{k} \frac{\lambda_i B_i}{\mu_i}}$. In the following, we compare the performance of *psr* and *wsp*, measured in terms of bandwidth blocking ratio, under different traffic conditions, varying the fractions ϕ_i to control the traffic mix.

Mixed holding times

First, consider the case of traffic with 2 types of flows, each with different mean holding times but request for the same amount of bandwidth. Figure 6.9 shows the performance of *psr* and *wsp* for this case where $1/\mu_1$ and $1/\mu_2$ are 60 and 120 seconds respectively and $B_1 = B_2 = 1$. The load ρ is set to 0.60 for the uniform load case and it is set to 0.50 for the



(a) *uniform load* (b) *non-uniform load*

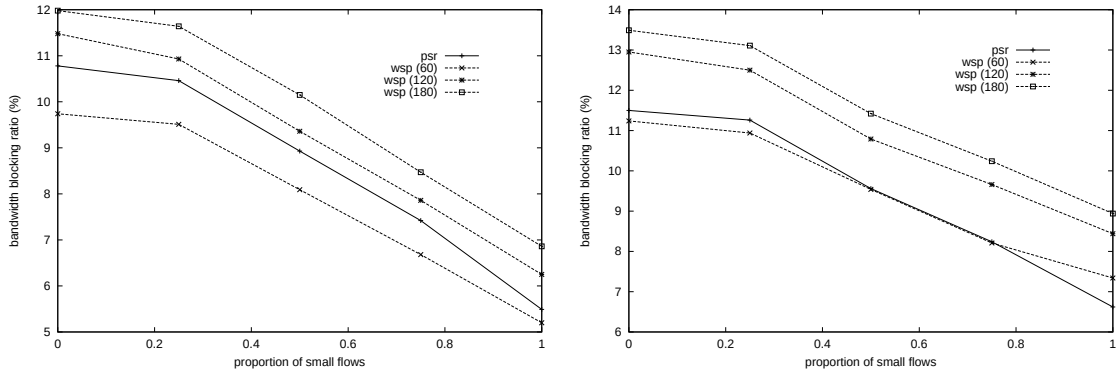
Figure 6.9: Performance for flows with mixed holding times

case of non-uniform load with hot load of 0.05. The bandwidth blocking ratio is plotted as a function of the fraction ϕ_1 corresponding to type 1 flows (*short* flows). It is quite evident that the performance of *wsp* degrades as the proportion of *short* flows increases while that of *psr* stays almost constant. The behavior of *wsp* is as expected since the shorter flows cause more fluctuation in the network QoS state and the information at a source node becomes more inaccurate as the QoS state update interval gets larger relative to flow dynamics. On the contrary, *psr* is insensitive to the duration of flows, which can be explained as follows.

The behavior of *psr* is not surprising since Erlang formula is known to be applicable even when the flow holding times are not exponentially distributed and blocking probability depends only on the load, i.e., the ratio of arrival rate and service rate. For the above case of two types of flows, the aggregate arrival rate, λ, is given by $\lambda = \lambda_1 + \lambda_2$ and the mean holding time, $1/\mu$, is given by $\frac{1}{\mu} = \frac{1}{\mu_1}\frac{\lambda_1}{\lambda_1+\lambda_2} + \frac{1}{\mu_2}\frac{\lambda_2}{\lambda_1+\lambda_2}$. This heterogeneous traffic can then be treated as equivalent to homogeneous traffic with arrival rate λ, mean holding time $1/\mu$ and the corresponding load $\lambda/\mu = \lambda_1/\mu_1 + \lambda_2/\mu_2$. So for a given load, the blocking probability would be same irrespective of the mean holding times of individual flows. Hence we do not need to differentiate flows based on their holding times.

Varying bandwidth requests

Now, consider the case of traffic with 2 types of flows, each requesting for different amount of bandwidth but having same mean holding time. As above, the load ρ is set to 0.60 for the uniform load case and it is set to 0.50 for the case of non-uniform load with hot load



(a) *uniform load* (b) *non-uniform load*

Figure 6.10: Performance for flows with variable bandwidth requests

of 0.05. Figure 6.10 shows the performance of *psr* and *wsp* for this case where $B_1 = 1$, $B_2 = 2$, and $1/\mu_1 = 1/\mu_2 = 60$ sec. Once again, the bandwidth blocking ratio is plotted as a function of the fraction ϕ_1 corresponding to type 1 flows (*small* flows). As expected, both schemes perform better as the proportion of *small* flows increases. It can also be seen that there is no discernible change in the relative performance of *psr* with respect to *wsp*. Our results show that when the requested bandwidth is significantly smaller than the link capacity, it may not be necessary for *psr* to differentiate between different bandwidth requests. As mentioned before in Section 5.1, when the link capacity is large, $\frac{b_1}{b_2} = \frac{d_1}{d_2}$. This implies that $\frac{\lambda_1 b_1}{\lambda_2 b_2} = \frac{\phi_1}{\phi_2}$, i.e., the blocking rate of flows of a type is proportional to their fraction in the total offered load. Consequently, performance of a equalization based proportional routing scheme would be same with or without categorizing the flows into different classes. Considering that in practice link capacities are much larger than an individual flow's bandwidth request, *psr* can be used *as is* to route heterogeneous traffic also.

6.2.1.4 Routing Overhead

We now take a close look at the amount of overhead involved in these two routing schemes. This overhead can be categorized into path selection overhead and information collection overhead. We discuss these two separately in the following.

The *wsp* scheme selects a path by first pruning the links with insufficient available bandwidth and then performing a variant of Dijkstra's algorithm on the resulting graph to find

the shortest path with maximum bottleneck bandwidth. This takes at least $O(E \log N)$ time where N is the number of nodes and E is the total number of links in the network. Assuming precomputation of a set of paths R to each destination to avoid searching the whole graph for path selection, it still need to traverse all the links of these precomputed paths. This amounts to an overhead of $O(L)$, where L is the total number of links in the set R. On the other hand, the path selection in *psr* is simply an invocation of $wrrps$ whose worst case complexity is $O(|R|)$ which is much less than $O(L)$ for *wsp*.

Now consider the information collection overhead. In *wsp*, each source acquires a network-wide view on the status of links through link state updates. Every router is responsible for maintaining QoS state and generating updates about all the links adjacent to it. These updates are sent either periodically or after a significant change in the resource availability since the last update. They are propagated to all the routers in the network through flooding. As in OSPF [65] each router is responsible for maintaining a consistent QoS state database. This incurs both communication and processing overhead. In contrast, the routers employing *psr* scheme do not exchange any such updates and thus completely do away with this overhead. Only source routers need to keep track of route level statistics and recompute proportions after every observation period. Statistics collection in *psr* involves only increment and decrement operations costing only constant time per flow. The proportion computation procedure in *psr* itself is extremely simple and costs no more than $O(|R|)$.

6.3 Approximation of *ebp*

In the previous section, we presented *psr* scheme that approximates *ebr* strategy and shown that it is a viable alternative to global schemes such as *wsp*. In this section, we describe an approximation of *ebp* strategy that is suitable for practical implementation. We refer to this approximation also as *ebp* scheme. In this section, we first describe the proportion computation procedure in *ebp* that at the end of an observation period computes new proportions based on the offered load and the observed blocking probabilities in that period. We then compare the performance of *ebp* with *psr*, *wsp* and other schemes, and show that *ebp* performs the best among localized proportional routing schemes. Finally, we study the sensitivity and optimality of *ebp* scheme.

6.3.1 Proportion Computation

The *ebp* strategy can be implemented using the following procedure to compute new proportions after an observation period. First, the current average blocking probability $\bar{b} = \sum_{i=1}^{k} \alpha_{r_i} b_{r_i}$ is computed. Then, the proportion of load onto a path r_i is decreased if its current blocking probability b_{r_i} is higher than the average $\bar{b}$ and increased if b_{r_i} is lower than $\bar{b}$. The magnitude of change is determined based on the relative distance of b_{r_i} from $\bar{b}$ and two configurable parameters to ensure that change is gradual. These parameters are *maximum proportional change*, δ and the corresponding *expected proportional change in blocking probability* ϕ. These parameters capture the relative change in blocking probability corresponding to a change in the load, i.e., if the load is changed by a fraction δ, the blocking probability would change by a fraction ϕ. Based on these parameters, if $b_{r_i} > \bar{b}$, the load onto a path r_i is decreased, i.e.,

$$\nu^*_{r_i} = \frac{\nu_i}{1 + min(\delta, \frac{b_{r_i} - \bar{b}}{b_{r_i}} \frac{\delta}{\phi})}$$

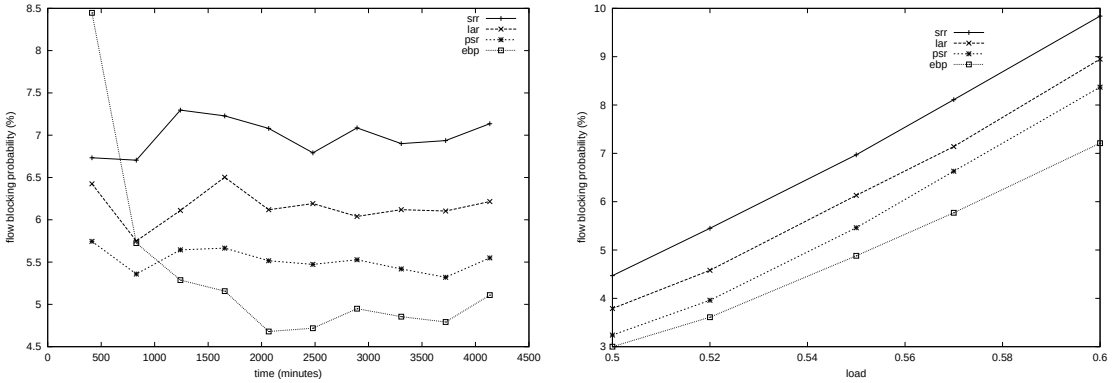
It is increased if $b_{r_i} < \bar{b}$, i.e.,

$$\nu^*_{r_i} = \nu^*_i (1 + min(\delta, \frac{\bar{b} - b_{r_i}}{\bar{b}} \frac{\delta}{\phi}))$$

The corresponding proportions would then be $\alpha^*_i = \frac{\nu^*_{r_i}}{\sum_{j=1}^{k} \nu^*_j}$. The *mean time between proportion computations* is controlled by a configurable parameter θ. The blocking performance of the candidate paths are observed for a period θ and at the end of the period the proportions are recomputed. This period θ should be large enough to allow for a reasonable measurement of the quality of the candidate paths and small enough to ensure adaptivity to changing traffic conditions.

6.3.2 Performance Evaluation

We now evaluate the performance of *ebp* using simulations. The simulation setting used for this study is same as the one presented in Section 5.2.4.1. Only thing we need to mention here is that all the results presented here correspond to simulations on *bigisp* topology. The parameters in the simulation are set as follows by default. Any change from these settings is explicitly mentioned wherever necessary. The values for configurable parameters in *ebp*



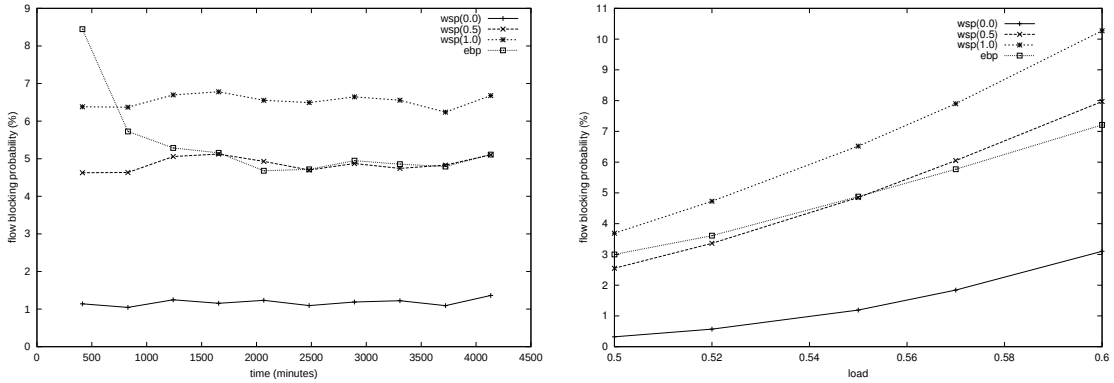
(a) fixed load of 0.55 (b) varying loads

Figure 6.11: Performance of various localized routing schemes

are $\delta = 0.2$, $\phi = 1.0$, $\theta = 60$ minutes (here after written as just *m*). For each pair σ, all the paths between them whose length is at most one hop more than the minimum number of hops (both minhop and minhop+1 paths) are chosen as candidate paths.

We first compare the performance of various localized routing schemes. The *srr* scheme is the sticky random routing scheme described in Section 4.2.1. The *lar* scheme is the learning automata based routing scheme presented in Section 4.2.2. The configurable parameters in *lar* are fixed at $\epsilon = 0.01$ and $a = 0.02$. Both these schemes perform worse when minhop+1 paths are also made candidates and so we use their performance with minhop paths only as candidates for comparison. The parameters in *psr* scheme are set to $\eta = 3$, $\hat{\gamma} = 5$, and $\psi = 0.8$.

Figure 6.11(a) shows the performance of these schemes when the offered load is fixed at 0.55. The overall blocking probability is shown as a function of time. First thing to note is the convergence of *ebp*. Starting with arbitrary proportions, it gradually adapts and reaches a stable state. All the other schemes converge more quickly but block many more flows than *ebp*. Among these schemes *srr* performs worse. This is because sticky routing directs the whole load onto one path till it blocks, while the other path is relatively idle. Though this scheme is supposed [25] to equalize the blocking rates of candidate paths, it does not ensure that the corresponding proportions are maintained in small time intervals. On the other hand, the *psr* scheme attempts to maintain these proportions while using sticky routing principle to obtain the proportions. Consequently *psr* scheme performs better than *srr*. The *ebp* scheme performs even better than *psr*, particularly when the load is high as



(a) fixed load of 0.55

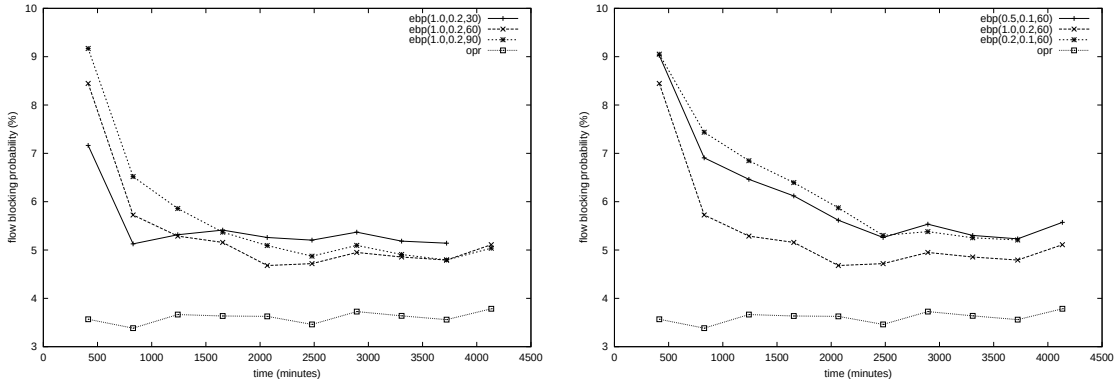
(b) varying loads

Figure 6.12: Performance of *ebp vs wsp*

as shown in Figure 6.11(b). This figure shows the blocking performance of these schemes under various loads. It can be seen that across all loads the performance of *ebp* scheme is better than the rest of the schemes. Hence we choose *ebp* scheme for localized adaptive proportioning of flows and further study its behavior.

We now compare the performance of *ebp* with *wsp*. Once again, the Figure 6.12(a) shows the overall blocking probability as a function of time with the load set to 0.55. The performance of *wsp* is shown for three different update intervals (in minutes): $0.0, 0.5, 1.0$. The update interval of 0.0 implies that an update is generated for every change in a link's available bandwidth. This corresponds to the performance of a server that keeps track of the precise current state of the network and performs path selection and admission control in a centralized manner. As observed earlier, the impact of update interval on *wsp* is drastic as can be seen from the jump in the blocking probability when the update interval is changed from an unrealistic setting of 0 to a more realistic value of 0.5 minutes or 30 seconds. This shows that best-path routing is good when the bandwidth availability information is accurate which requires frequent updates. On the other hand, our *ebp* scheme without any global updates performs as well as *wsp* with update interval of 30 seconds. This is true across various loads as shown in Figure 6.12(b).

We now study the sensitivity of *ebp* to the values chosen for δ, ϕ, and θ. We first look at the impact of observation interval on the performance of *ebp* by varying the interval from 30 to 90 minutes. The values of δ and phi are set to 0.1 and 0.5 and the load is fixed at 0.55. The corresponding results are shown in Figure 6.13(a). With smaller observation



(a) varying θ

(b) varying (δ, ϕ)

Figure 6.13: Sensitivity of *ebp* scheme

interval, *ebp* converges relatively faster. But the overall performance is almost same for all observation interval settings. Figure 6.13(b) shows the results of our simulations when the observation interval θ is fixed at 60 minutes, and δ and phi are varied. Here there is some impact of various settings on the performance of *ebp*. However, the difference in overall blocking probabilities is not significant and we can say that the performance of *ebp* is relatively insensitive to its parameter settings.

One important observation that can be made from Figure 6.13 is the significant gap between the performance of *ebp* and the offline optimal proportioning scheme *opr*. While *ebp* is supposed to be a near-optimal strategy, these results show that it may not work as well in practice as in theory. There are two reasons for this gap. First, due to statistical variations the blocking probability information of a path gathered by a source depends on the observation interval and may not be precise. Second, while in theory it is possible to probe the quality of a path by routing infinitesimally small proportion of traffic, in reality non-negligible proportion of traffic has to be sent along each candidate path to probe its quality. Consequently, bad candidate paths affect the performance of any practical proportional routing scheme. Hence the performance of localized schemes such as *ebp* depends critically on the choice of candidate paths. In the next chapter, we address this problem of how to select a few good candidate paths.

Chapter 7

Candidate Path Selection

7.1 Hybrid Approach to QoS Routing

The localized approach to proportional routing described in the previous chapters is simple and has several important advantages. However it has a limitation that routing is done based solely on the information collected locally. A network node under localized QoS routing approach can judge the quality of paths/links only by routing some traffic along them. It would have no knowledge about the state of the rest of the network. While the proportions for paths are adjusted to reflect the changing qualities of paths, the candidate path set itself remains static. To ensure that the localized scheme adapts to varying network conditions, many feasible paths have to be made candidates. It is not possible to preselect a few good candidate paths statically. Hence it is desirable to *supplement localized proportional routing* with a mechanism that *dynamically selects a few good candidate paths*.

Two key questions that arise in candidate path selection are how many paths are needed and how to find these paths. Clearly, the number and the quality of the paths selected as candidates dictate the performance of a proportional routing scheme. There are several reasons why it is desirable to minimize the number of paths used for routing. First, there is a significant overhead associated with establishing, maintaining and tearing down of paths. Second, the complexity of the scheme that distributes traffic among multiple paths increases considerably as the number of paths increases. Third, there could be a limit on the number of explicitly routed paths such as label switched paths in MPLS [92] that can be setup between a pair of nodes. Therefore it is desirable to use *as few paths as possible* while at the same time *minimize the congestion* in the network.

For judicious selection of paths, some knowledge regarding the (global) network state is crucial. This knowledge about resource availability at network nodes, for example, can be obtained through (periodic) information exchange among routers in a network. Because network resource availability changes with each flow arrival and departure, maintaining *accurate* view of network QoS state requires *frequent* information exchanges among the network nodes and introduces both communication and processing overheads. However, these updates would not cause significant burden on the network as long as their frequency is not more than what is needed to convey connectivity information in traditional routing protocols like OSPF [65]. The QoS state of each link could then be piggybacked along with the conventional link state updates. Hence it is important to devise multipath routing schemes that *work well even when the updates are infrequent*.

We propose such a scheme *widest disjoint paths* (wdp) that uses proportional routing — the traffic is proportioned among a few widest disjoint paths. It uses *infrequently* exchanged *global* information for selecting a few good paths based on their long term available bandwidths. It proportions traffic among the selected paths using *local* information to cushion the short term variations in their available bandwidths. Thus the *hybrid* approach to QoS routing adapts at different time scales to the changing network conditions. The rest of the chapter discusses what type of global information is exchanged and how it is used to select a few good paths. It also describes what information is collected locally and how traffic is proportioned adaptively. We first briefly mention some of the related work.

Related Work

Several multipath routing schemes have been proposed for balancing the load across the network. The Equal Cost Multipath (ECMP) [65] and Optimized Multipath (OMP) [108, 109] schemes perform packet level forwarding decisions. ECMP splits the traffic *equally* among multiple equal cost paths. However, these paths are determined statically and may not reflect the congestion state of the network. Furthermore, it is desirable to apportion the traffic according to the quality of each path. OMP is similar in spirit to our work. It also uses updates to gather link loading information, selects a set of best paths and distributes traffic among them. However, our scheme makes routing decisions at the flow level and consequently the objectives and procedures are different.

Another approach to path selection is to precompute maximally disjoint paths [81] and attempt them in some order. This is static and overly conservative. What matters is not the

sharing itself but *the sharing of bottleneck links*, which change with network conditions. In our scheme we dynamically select paths such that they are disjoint *w.r.t* bottleneck links.

7.2 Widest Disjoint Paths

In this section, we present the candidate path selection procedure used in *wdp*. To help determine whether a path is good and whether to include it in the candidate path set, we define *width* of a path and introduce the notion of *width* of a *set of paths*. The candidate path set R_σ for a pair σ is changed only if it increases the width of the set R_σ or decreases the size of the set R_σ without reducing its width. The widths of paths are computed based on link state updates that carry *average residual bandwidth* information about each link. The traffic is then proportioned among the candidate paths using *ebp*.

A basic question that needs to be addressed by any path selection procedure is what is a "good" path. In general, a path can be categorized as good if its inclusion in the candidate path set decreases the overall blocking probability considerably. It is possible to judge the utility of a path by measuring the performance with and without using the path. However, it is not practical to conduct such inclusion-exclusion experiment for each feasible path. Moreover, each source has to independently perform such trials without being directly aware of the actions of other sources which are only indirectly reflected in the state of the links. Hence each source has to try out paths that are likely to decrease blocking and make such decisions with some local objective that leads the system towards a global optimum.

When identifying a set of candidate paths, another issue that requires attention is the sharing of links between paths. A set of paths that are good *individually* may not perform as well as expected *collectively*. This is due to the sharing of *bottleneck* links. When two candidate paths of a pair share a bottleneck link, it may be possible to remove one of the paths and shift all its load to the other path without increasing the blocking probability. Thus by ensuring that candidate paths of a pair do not share bottleneck links, we can reduce the number of candidate paths without increasing the blocking probability. A simple guideline to enforce this could be that the candidate paths of a pair be mutually disjoint, i.e., they do not share *any* links. This is overly restrictive, since even with shared links, some paths can cause reduction in blocking if those links are not congested. What matters is not the sharing itself but *the sharing of bottleneck links*. While the sharing of links among the paths is *static* information independent of traffic, identifying bottleneck links is *dynamic* since the congestion in the network depends on the offered traffic and routing patterns.

Therefore it is essential that candidate paths be *mutually disjoint w.r.t bottleneck links*.

To judge the quality of a path, we define *width* of a path as the the residual bandwidth on its bottleneck link. Let $\hat{c}_l$ be the maximum capacity of link l and ν_l be the average load on it. The difference $c_l = \hat{c}_l - \nu_l$ is the average residual bandwidth on link l. Then the *width* w_r of a path r is given by $w_r = \min_{l \in r} c_l$. The larger its width is, the better the path is, and the higher its potential is to decrease blocking. Similarly we define *distance* [56] of a path r as $\sum_{l \in r} \frac{1}{c_l}$. The shorter the distance is, the better the path is. The widths and distances of paths can be computed given the residual bandwidth information about each link in the network. This information can be obtained through periodic link state updates. To discount short term fluctuations, the *average residual bandwidth* information is exchanged. Let τ be the update interval and u_l^t be the utilization of link l during the period $(t - \tau, t)$. Then the average residual bandwidth at time t, $c_l^t = (1 - u_l^t)\hat{c}_l$. Hereafter without the superscript, c_l refers to the most recently updated value of the average residual bandwidth of link l.

To aid in path selection, we also introduce the notion of *width* for a *set of paths* R, which is computed as follows. We first pick the path r^* with the largest width w_{r^*}. If there are multiple such paths, we choose the one with the shortest distance d_{r^*}. We then decrease the residual bandwidth on all its links by an amount w_{r^*}. This effectively makes the residual bandwidth on its bottleneck link to be 0. We remove the path r^* from the set R and then select a path with the next largest width based on the just updated residual bandwidths. Note that this change in residual bandwidths of links is local and only for the purpose computing the width of R. This process is repeated till the set R becomes empty. The sum of all the widths of paths computed thus is defined as the *width of* R. Note that when two paths share a bottleneck link, the width of two paths together is same as the width of a single path. The width of a path set computed thus, essentially accounts for the sharing of links between paths.

The procedure to compute the width of a path set R is shown in Figure 7.1. In each iteration, a subset of paths R^* with the largest width w^* are identified (lines 4-5). From these widest paths, a path r^* with the shortest distance d^* is selected (lines 6-7). The width w^* of path r^* is added to the total width W (line 8). The residual capacities of all the links along the path r^* is reduced by an amount w^* (lines 9-10). This in turn affects the widths of other paths in R. The path r^* is removed from the set (line 11) and this process is repeated till the set R becomes empty (line 3). The resulting W is considered to be the width of R. The narrowest path, i.e., the last path removed from the set R is referred to as NARROWEST(R).

```
1.   PROCEDURE WIDTH(R)
2.     W = 0
3.     While R ≠ ∅
4.       w* = max_{r∈R} w_r
5.       R* = {r : r ∈ R, w_r = w*}
6.       d* = min_{r∈R*} d_r
7.       r* = {r : r ∈ R*, d_r = d*}
8.       W = W + w*
9.       For each l in r*
10.        c_l = c_l − w*
11.      R = R \ r*
12.    Return W
13.  END PROCEDURE
```

Figure 7.1: The procedure to compute width for a path set R

Based on this notion of width of a path set, we propose a path selection procedure that *adds* a new candidate path only if its inclusion *increases the width*. It *deletes* an existing candidate path if its exclusion *does not decrease* the total width. In other words, each modification to the candidate path set either *improves the width* or *reduces the number* of candidate paths. The selection procedure is shown in Figure 7.2. First, the load contributed by each existing candidate path is deducted from the corresponding links (lines 2-4). After this adjustment, the residual bandwidth c_l on each link l reflects the load offered on l by all source destination pairs other than σ. Given these adjusted residual bandwidths, the candidate path set R_σ is modified as follows.

The benefit of inclusion of a feasible path r is determined based on the number of existing candidate paths (lines 6-8). If this number is below the specified limit η, the resulting width W_r is the width of $R_\sigma \cup r$. Otherwise, it is the width of $R_\sigma \cup r \setminus$ NARROWEST($R_\sigma \cup r$), i.e., the width after excluding the narrowest path among $R_\sigma \cup r$. Let W^+ be the largest width that can be obtained by adding a feasible path (line 9). This width W^+ is compared with width of the current set of candidate paths. A feasible path is made a candidate if its inclusion in set R_σ increases the width by a fraction ψ (line 10). Here $\psi > 0$ is a configurable parameter to ensure that each addition improves the width by a significant amount. It is possible that many feasible paths may cause the width to be increased to W^+. Among such paths, the path r^+ with the shortest distance is chosen for inclusion (lines 11-13). Let r^- be the narrowest path in the set $R_\sigma \cup r$ (line 14). The path r^- is replaced with r^+ if either the number of paths already reached the limit or the path r^- does not contribute to the width (lines 15-16). Otherwise the path r^+ is simply added to the set of candidate paths (lines 17-18). When no new path is added, an existing candidate path is deleted from the set if it does not change the width (lines 20-22). In all other cases, the candidate path

```
1.  PROCEDURE SELECT(σ)
2.    For each path r in R_σ
3.      For each link l in r
4.        c_l = c_l + (1 − b_r)ν_r
5.    If |R_σ| < η
6.      W_r = WIDTH(R_σ ∪ r), ∀r ∈ R̂_σ \ R_σ
7.    Else
8.      W_r = WIDTH(R_σ ∪ r \ NARROWEST(R_σ ∪ r)), ∀r ∈ R̂_σ \ R_σ
9.    W^+ = max_{r∈R̂_σ\R_σ} W_r
10.   If (W^+ > (1 + ψ) WIDTH(R_σ))
11.     R^+ = {r : r ∈ R̂_σ \ R_σ, W_r = W^+}
12.     d^+ = min_{r∈R^+} d_r
13.     r^+ = {r : r ∈ R^+, d_r = d^+}
14.     r^− = NARROWEST(R_σ ∪ r)
15.     If (|R_σ| = η or WIDTH(R_σ ∪ r^+ \ r^−) = W^+)
16.       R_σ = R_σ ∪ r^+ \ r^−
17.     Else
18.       R_σ = R_σ ∪ r^+
19.   Else
20.     r^− = NARROWEST(R_σ)
21.     If WIDTH(R_σ \ r^−) = WIDTH(R_σ)
22.       R_σ = R_σ \ r^−
23. END PROCEDURE
```

Figure 7.2: The candidate path set selection procedure for pair σ

set remains unaffected. It is obvious that this procedure always either increases the width or decreases the number of candidate paths.

It should be noted that though *wdp* uses link state updates it does not suffer from the *synchronization* problem unlike global QoS routing schemes such as *wsp*. There are several reasons contributing to the stability of *wdp*: 1) The information exchanged about a link is its *average* not *instantaneous* residual bandwidth and hence less variable; 2) The traffic is proportioned among few "good" paths instead of loading the "best" path based on inaccurate information; 3) Each pair uses only a few candidate paths and makes only incremental changes to the candidate path set; 4) The new candidate paths are selected for a pair only after deducting the load contributed by the current candidate paths from their links. Due to such adjustment even with link state updates, the view of the network for each node would be different; 5) When network is in a stable state of convergence, the information carried in link state updates would not become outdated and consequently each node would have reasonably accurate view of the network. Essentially the nature of information exchanged and the manner in which it is utilized work in a mutually beneficial fashion and lead the system towards a stable optimal state.

We now illustrate how *wdp* scheme selects candidate paths using a simple example. Consider a topology shown in Figure 7.3. Suppose that source s has to recompute candidate

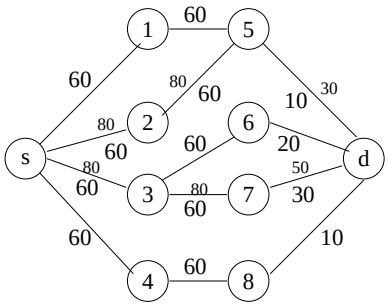


Figure 7.3: Illustration of *wdp* procedure

paths to destination d. There are five possible paths between s and d. Let us assume that s is currently using paths via $2 \rightarrow 5$ and $3 \rightarrow 7$, and proportioning traffic equally between them. Further, assume that the average amount of load successfully routed between s and d is 40. Let the average available bandwidths of links, received by source s through global link state updates, be as shown in bigger font.

Before recomputing candidate paths, source s has to perform local adjustment to discount the bandwidth usage by itself. The source s is currently contributing a load of 20 each to paths $2 \rightarrow 5$ and $3 \rightarrow 7$. So the average available bandwidths of links along these paths are correspondingly increased by 20. The new values after local adjustment are shown in smaller font. Essentially the source s views the available bandwidth on link $5 \rightarrow d$ as 30 instead of 10 while other sources view it differently.

Now if the maximum number of candidate paths allowed, η, is only two, the candidate path set remains same. This is because the current candidate paths are wider than other paths and replacing any of these paths does not increase the total width. If η is more than two, the path via $3 \rightarrow 6$ is added to the candidate set. Only 4 paths with combined width of 110 would be made candidates even if there is no constraint on the number of candidate paths. The path via $1 \rightarrow 5$ would never be added since it would not increase the total width. Note that though paths $s \rightarrow 3 \rightarrow 6 \rightarrow d$ and $s \rightarrow 3 \rightarrow 7 \rightarrow d$ share a link $s \rightarrow 3$, both are preferred as candidates, since the common link is not the bottleneck. On the other hand, $s \rightarrow 1 \rightarrow 5 \rightarrow d$ is included and $s \rightarrow 2 \rightarrow 5 \rightarrow d$ is excluded since they share a bottleneck link $5 \rightarrow d$. Thus *wdp* selects widest paths that are mutually disjoint w.r.t. bottleneck links.

η :	maximum number of paths allowed between a pair
ψ :	width increase threshold for changing the path set
τ :	mean time between link state updates
θ :	mean time between computation of proportions
ξ :	mean time between computation of candidate paths

Figure 7.4: Configurable parameters in *wdp*

7.3 Performance Analysis

In this section, we evaluate the performance of the proposed hybrid scheme *wdp*. We start with the description of the simulation environment. First, we compare the performance of *wdp* with that of the optimal scheme *opr* and show that *wdp* converges to near-optimal proportions. Furthermore, we demonstrate that the performance of *wdp* is relatively insensitive to the values chosen for the configurable parameters. We then contrast the performance of *wdp* with global QoS routing scheme *wsp* in terms of the overall blocking probability and routing overhead.

7.3.1 Simulation Environment

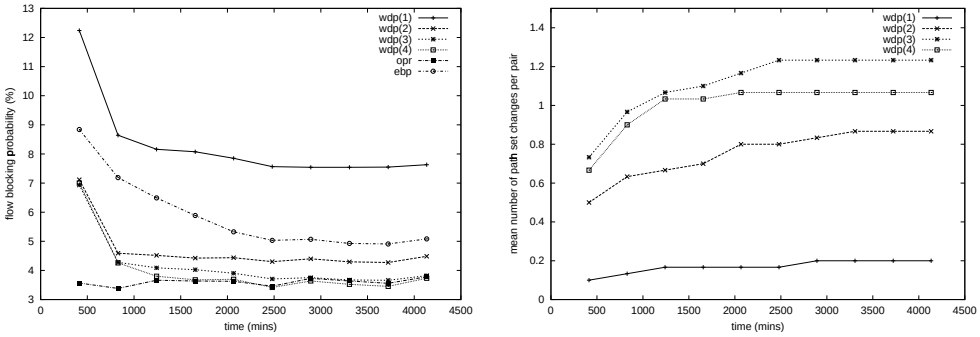
The simulation setting used for this study is same as the one presented in Section 5.2.4.1. Once again, the results here correspond to simulations on *bigisp* topology. The values for configurable parameters in *wdp* are set to $\psi = 0.2$, $\tau = 30$ *m*, $\theta = 60$ *m*, $\xi = 180$ *m*. The description of these parameters is given in Figure 7.4. For each pair σ, all the paths between them whose length is at most one hop more than the minimum number of hops are included in the feasible path set $\hat{R}_\sigma$. The amount of offered load on the network ρ is set to 0.55.

7.3.2 Performance of *wdp*

In this section, we compare the performance of *wdp* and *opr* to show that *wdp* converges to near-optimal proportions using only a few paths for routing traffic. We also demonstrate that *wdp* is relatively insensitive to the settings for the configurable parameters.

7.3.2.1 Convergence

Figure 7.5 illustrates the convergence process of *wdp*. The results are shown for different values of $\eta = 1 \cdots 4$. Figure 7.5(a) compares the performance of *wdp*, *opr* and *ebp*. The performance is measured in terms of the overall flow blocking probability, which is defined



(a) blocking probability (b) number of changes to candidate paths

Figure 7.5: Convergence process of *wdp*

as the ratio of the total number of blocks to the total number of flow arrivals. The overall blocking probability is plotted as a function of time. In the case of *opr*, the algorithm is run offline to find the optimal proportions given the set of feasible paths and the offered load between each pair of nodes. The resulting proportions are then used in simulation for statically proportioning the traffic among the set of feasible paths. The *ebp* scheme refers to the localized scheme used in isolation for adaptively proportioning across all the feasible paths. As noted earlier all paths of length either minhop or minhop+1 are chosen as the set of feasible paths in our study.

There are several conclusions that can be drawn from Figure 7.5(a). First, the *wdp* scheme converges for all values of η. Given that the time between changes to candidate path sets, ξ, is 180 *m*, it reaches steady state within (on average) 5 path recomputations per pair. Second, there is a marked reduction in the blocking probability when the number of paths allowed, η, is changed from 1 to 2. It is evident that there is quite a significant gain in using multi-path routing instead of single path routing. When the limit η is increased from 2 to 3 the improvement in blocking is somewhat less but significant. Note that in our topology there are at most two paths between a pair that do not share any links. But there could be more than two paths that are mutually disjoint w.r.t bottleneck links. The performance difference between η values of 2 and 3 is an indication that we only need to ensure that candidate paths do not share congested links. However using more than 3 paths per pair helps very little in decreasing the blocking probability. Third, the *ebp* scheme also converges, albeit slowly. Though it performs much better than *wdp* with single path, it is worse than *wdp* with $\eta = 2$. But when *ebp* is used in conjunction with path selection under *wdp* it converges quickly to

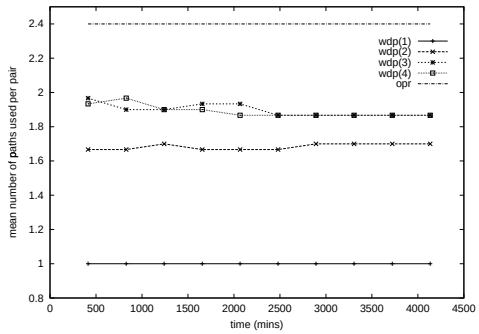
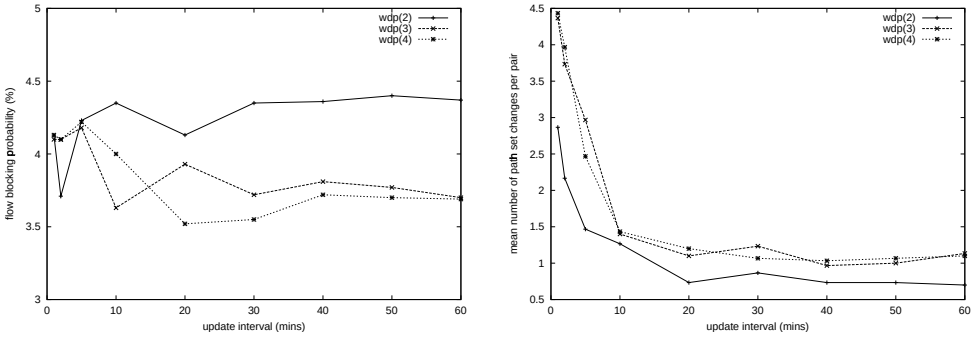


Figure 7.6: Number of paths used for routing

lower blocking probability using only a few paths. Finally, using at most 3 paths per pair, the *wdp* scheme approaches the performance of optimal proportional routing scheme.

Figure 7.5(b) establishes the convergence of *wdp*. It shows the average number of changes to the candidate path set as a function of time. Here the change refers to either addition, deletion or replacement operation on the candidate path set R_σ of any pair σ. Note that the cumulative number of changes are plotted as a function of time and hence a plateau implies that there is no change to any of the path sets. It can be seen that the path sets change incrementally initially and after a while they stabilize. Thereafter each pair sticks to the set of chosen paths. It should be noted that starting with at most 3 minhop paths as candidates and making as few as 1.2 changes to the set of candidate paths, the *wdp* scheme achieves almost optimal performance.

We now compare the average number of paths used by a source-destination pair for routing as shown in Figure 7.3.2.1. Note that in *wdp* scheme η specifies only the maximum allowed number of paths per pair. The actual number of paths selected for routing depends on their widths. The average number of paths used by *wdp* for η of 2 and 3 are 1.7 and 1.9 respectively. The number of paths used stays same even for higher values of η. The *ebp* scheme uses all the given feasible paths for routing. It can measure the quality of a path only by routing some traffic along that path. The average number of feasible paths chosen are 5.6. In case of *opr* we count only those paths that are assigned a proportion of at least 0.10 by the optimal offline algorithm. The average number of such paths under *opr* scheme are 2.4. These results support our claim that *ebp* based proportioning over widest disjoint paths performs almost like optimal proportioning scheme while using fewer paths.



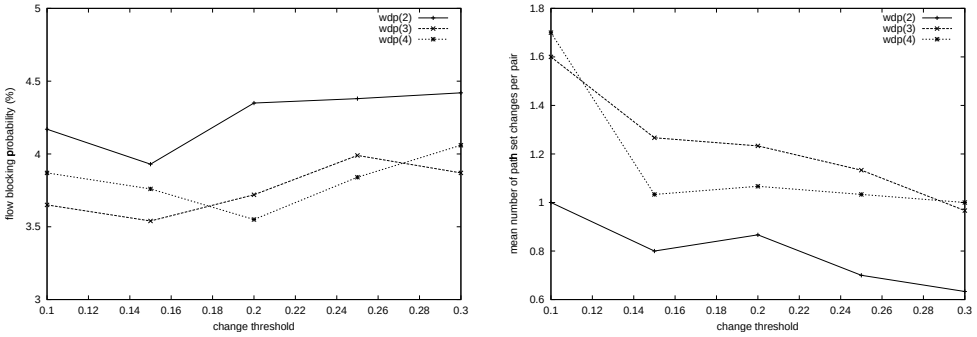
(a) blocking probability (b) number of path set changes

Figure 7.7: Sensitivity of *wdp* to update interval τ

7.3.2.2 Sensitivity

The *wdp* scheme requires periodic updates to obtain global link state information and to perform path selection. To study the impact of update interval on the performance of *wdp*, we conducted several simulations with different update intervals ranging from 1 *m* to 60 *m*. The Figure 7.7(a) shows the flow blocking probability as a function of update interval. At smaller update intervals there is some variation in the blocking probability, but much less variation at larger update intervals. It is also clear that increasing the update interval does not cause any significant change in the blocking probability. To study the effect of update interval on the stability of *wdp*, we plotted the average number of path set changes as a function of update interval in Figure 7.7(b). It shows that the candidate path set of a pair changes often when the updates are frequent. When the update interval is small, the average residual bandwidths of links resemble their instantaneous values, thus highly varying. Due to such variations, paths may appear wider or narrower than they actually are, resulting in unnecessary changes to candidate paths. However, this does not have a significant impact on the blocking performance due to adaptive proportional routing among the selected paths. For the purpose of reducing overhead and increasing stability, we suggest that the update interval τ be reasonably large, while ensuring that it is much smaller than the path recomputation interval ξ.

We also studied the sensitivity of *wdp* to the width increase threshold parameter for changing the path set ψ. The simulations were run for five different values of ψ from 0.10 to 0.30. Figure 7.8(a) shows the flow blocking probability as as a function of ψ. It can be seen that the blocking performance of *wdp* is relatively insensitive to the value of ψ. Once again



(a) blocking probability (b) number of path set changes

Figure 7.8: Sensitivity of *wdp* to change threshold ψ

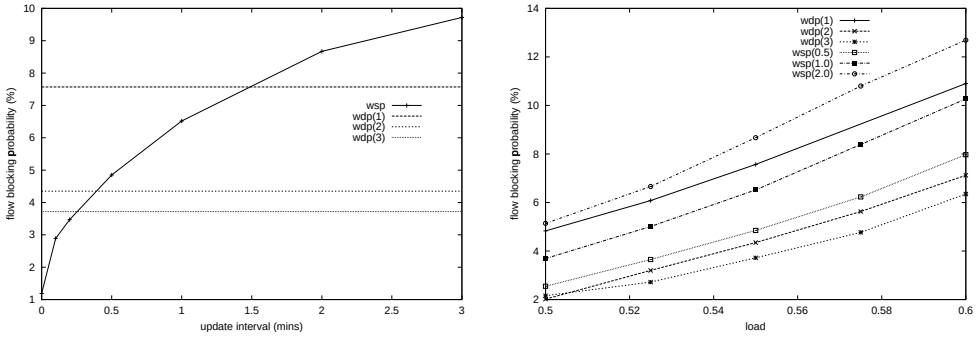
in Figure 7.8(b) the number of path set changes is shown as a function of ψ. As expected, there are fewer changes as the ψ value increases. However this does not effect the blocking significantly since much wider paths are included in the set anyway. Only those paths that may contribute to the width by a small amount are excluded by choosing higher ψ value. To ensure that good paths are not excluded and not so good paths are not included we suggest setting ψ to a value around 0.20.

7.3.3 Comparison of *wsp* and *wdp*

We now compare the performance of hybrid QoS routing scheme *wdp* with a global QoS routing scheme *wsp*. The *wsp* is a well-studied scheme that selects the widest shortest path for each flow based on the global network view obtained through link state updates. The information carried in these updates is the residual bandwidth at the instant of the update. Note that *wdp* also employs link state updates but the information exchanged is average residual bandwidth over a period not its instantaneous value. We use *wsp* as a representative of global QoS routing schemes as it was shown to perform the best among similar schemes such as shortest widest path (*swp*), shortest distance path (*sdp*). In the following, we first compare the performance of *wdp* with *wsp* in terms of flow blocking probability and then the routing overhead.

7.3.3.1 Blocking Probability

Figure 7.9(a) shows the blocking probability as a function of update interval τ used in *wsp*. The τ for *wdp* is fixed at 30 *m*. The offered load on the network ρ was set to 0.55. It is clear



(a) varying update interval

(b) varying load

Figure 7.9: Performance comparison of *wdp* and *wsp*

that the performance of *wsp* degrades drastically as the update interval increases. The *wdp* scheme, using at most two paths per pair and infrequent updates with $\tau = 30$ *m*, blocks fewer flows than *wsp*, that uses many more paths and frequent updates with $\tau = 0.5$ *m*. The performance of *wdp* even with a single path is comparable to *wsp* with $\tau = 1.5$ *m*. Figure 7.9(b) displays the flow blocking probability as a function of offered network load ρ which is varied from 0.50 to 0.60. Once again, the τ for *wdp* is set to 30 *m* and the performance of *wsp* is plotted for 3 different settings of τ: 0.5, 1.0 and 2.0 *m*. It can be seen that across all loads the performance of *wdp* with $\eta = 2$ is better than *wsp* with $\tau = 0.5$. Similarly with just one path, *wdp* performs better than *wsp* with $\tau = 2.0$ and approaches the performance of $\tau = 1.0$ as the load increases. It is also worth noting that *wdp* with two paths rejects significantly fewer flows than with just one path, justifying the need for multipath routing.

It is interesting to observe that even with a single path and very infrequent updates *wdp* outperforms *wsp* with frequent updates. There are several factors contributing to the superior performance of *wdp*. First, it is the nature of information used to capture the link state. The information exchanged about a link is its *average* not *instantaneous* residual bandwidth and hence less variable. Second, before picking the widest disjoint paths, the residual bandwidth on all the links along the current candidate path are adjusted to account for the load offered on that path by this pair. Such a *local adjustment* to the global information makes the network state appear differently to each source. It is as if each source receives a customized update about the state of each link. The sources that are currently routing through a link perceive higher residual bandwidth on that link than other sources.

This causes a source to continue using the same path to a destination unless it finds a much wider path. This in turn reduces the variation in link state and consequently the updated information does not get outdated too soon. In contrast, *wsp* exchanges highly varying instantaneous residual bandwidth information and all the sources have the same view of the network. This results in mass synchronization as every source prefers *good* links and avoids *bad* links. This in turn increases the variance in instantaneous residual bandwidth values and causes route oscillation[1]. The *wdp* scheme, on the other hand, by selecting paths using both local and global information and by employing *ebp* based adaptive proportioning delivers stable and robust performance.

7.3.3.2 Routing Overhead

Now we compare the amount of overhead incurred by *wdp* and *wsp*. This overhead can be categorized into per flow routing overhead and operational overhead. We discuss these two separately in the following.

The *wsp* scheme selects a path by first pruning the links with insufficient available bandwidth and then performing a variant of Dijkstra's algorithm on the resulting graph to find the shortest path with maximum bottleneck bandwidth. This takes at least $O(E \log N)$ time where N is the number of nodes and E is the total number of links in the network. Assuming precomputation of a set of paths R_σ to each destination, to avoid searching the whole graph for path selection, it still needs to traverse all the links of these precomputed paths to identify the widest shortest path. This amounts to an overhead of $O(L_\sigma)$, where L_σ is the total number of links in the set R_σ. On the other hand, in *wdp* one of the candidate paths is chosen in a weighted round robin fashion whose complexity is $O(\eta)$ which is much less than $O(L_\sigma)$ for *wsp*.

Now consider the operational overhead. Both schemes require link state updates to carry residual bandwidth information. However the frequency of updates needed for proper functioning of *wdp* is no more than what is used to carry connectivity information in traditional routing protocols such as OSPF. Therefore, the average residual bandwidth information required by *wdp* can be piggybacked along with the conventional link state updates. Hence, *wdp* does not cause any additional burden on the network. On the other hand, the *wsp* scheme requires frequent updates consuming both network bandwidth and processing

[1]Some remedial solutions were proposed in [2, 3] to deal with the inaccuracy at a source node. However, the fundamental problem remains and the observations made here still apply.

power. Furthermore *wsp* uses too many paths. The *wdp* scheme uses only a few preset paths, thus avoiding per flow path setup. Only admission control decision need to be made by routers along the path. The other overheads incurred only by *wdp* are periodic proportion computation and candidate path computation. The proportion computation procedure is extremely simple and costs no more than $O(\eta)$. The candidate path computation amounts to finding η widest paths and hence its worst case time complexity is $O(\eta N^2)$. However, this cost is incurred only once every ξ period. Considering both the blocking performance and the routing cost, we conclude that *wdp* yields much higher throughput with much lower overhead than *wsp*.

Chapter 8

Hierarchical Proportional Routing

8.1 Introduction

We have shown that proportional routing schemes alleviate congestion in the network by distributing load among multiple "good" paths instead of overloading a single "best" path. However these schemes assume that each router in the network is aware of the topology and the state of the whole network. This is referred to as *flat* routing and under flat routing, each router participates in link state updates and maintains detailed information about the entire network. This introduces significant burden on every router and as the size of the network grows, the overhead at each router increases tremendously. To provide a scalable solution, *hierarchical routing* is suggested [65, 5] as an alternative to flat routing. To make the proportional routing schemes scale well to large networks we need to extend them to provide hierarchical routing.

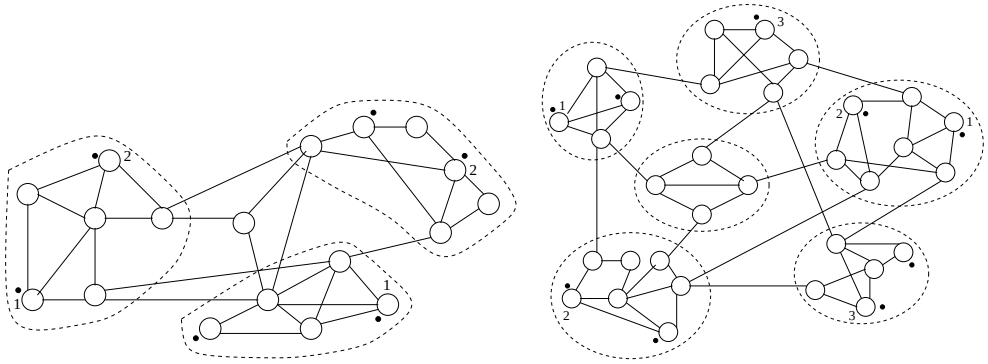
Under hierarchical routing, a network is divided into multiple areas.[1] The routing within the area is flat with each router having detailed information about routers and links in that area. But the routers have only sketchy *aggregate* information about other areas. To route traffic destined for other areas, a source router may select a partial higher level path, based on the aggregate information, that gets expanded, based on the detailed information, at the ingress border router of each area along the path. Such a hierarchical routing reduces the overhead at each router by limiting the scope of link state updates and maintaining only summary information about other areas. Examples of hierarchical routing are inter-area

[1] Also referred to as peer groups in PNNI [5].

routing in OSPF [65] and ATM Forum's PNNI [5].

The hierarchical routing approach while reduces the burden on a router, introduces inaccuracy in the information available for routing. Hence the performance of a hierarchical routing scheme depends heavily on how information about an area is aggregated and how it is utilized in routing across areas. For providing QoS routing across areas, in addition to *topology aggregation, QoS state aggregation* must also be performed. Topology aggregation is concerned with capturing the structure of an area which is relatively static. There are several proposals for topology aggregation such as [52, 48]. QoS state aggregation is concerned with summarizing the QoS state of an area which is quite dynamic. It is somewhat straightforward to summarize the the QoS state when best-path routing is employed. The state of routing between a pair of routers is given by the state of the best path. The best-path based hierarchical routing is studied in [31]. On the other hand, *it is not obvious how to aggregate the topology and state when multiple paths are used to route traffic between a pair of routers*. This chapter addresses the issue of aggregation under multipath proportional routing, and the selection of paths based on the aggregated information.

We propose an aggregation method that summarizes the state of multiple paths between two routers using a single metric. This metric in essence captures the traffic carrying capacity of multiple paths between a pair of routers. Our approach is somewhat similar in spirit to the proposal in [63] but the actual path selection schemes are quite different. We propose two inter-area routing schemes based on this aggregate metric: *hierarchical widest disjoint paths* (hwdp) and *hierarchical widest border routers* (hwbr). The *hwdp* scheme is a hierarchical source routing scheme where a source router selects a set of higher level skeletal paths to the destination as candidates and proportions flows among them. The *hwbr* scheme is a hierarchical next-hop routing scheme that selects only the next-hop border routers which in turn select higher level paths to the destination. Both these schemes use our *widest disjoint paths* (wdp), a flat multipath proportional routing scheme described in the previous chapter, for intra-area routing to expand the skeletal higher level paths to actual physical paths. They essentially differ in the way the network outside an area is aggregated by a border router and propagated to the interior routers. The following sections describe the proposed aggregation metric and the hierarchical multipath proportional routing schemes based on this metric. We also evaluate the performance of the proposed schemes and compare them with best-path based hierarchical routing schemes discussed in [31].



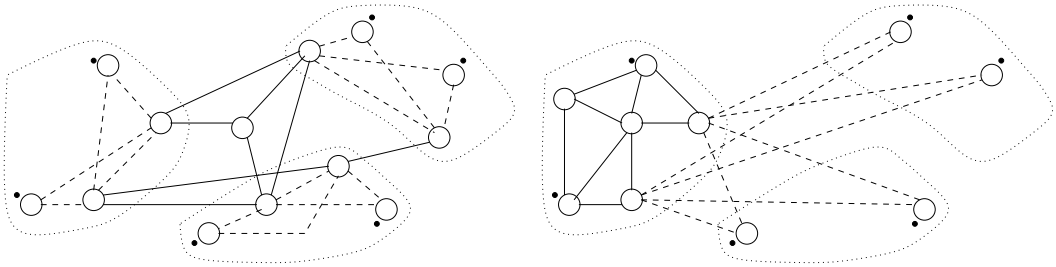
(a) isp (b) rand

Figure 8.1: Topologies used in our study

8.2 Topology and State Aggregation

Topology aggregation is concerned with capturing the structure of an area which is relatively static. There are several proposals for topology aggregation such as [48, 52]. We take a simple approach where a border router makes other routers in its area appear as directly connected to it by a *logical link*. This approach is similar to the one employed by OSPF [65]. Then state aggregation is about summarizing the state of the network by assigning the attributes to these logical links. When best-path routing is used to route traffic within an area, it is straightforward to perform state aggregation: the attributes of a logical link are that of the best-path. For example, when traffic within an area is routed along shortest paths, then the *distance* of the logical link between a pair of routers would simply be the distance of the shortest path. On the other hand, it is not obvious how to summarize the state when multiple paths are used to route traffic between a pair of routers.

The multipath proportional routing scheme *wdp* described in the previous chapter lends itself very well to aggregation. Note that *wdp* selects candidate paths such that the total *width* of these candidates is as large as possible. This width essentially captures the traffic-carrying capacity of all the paths between a pair of routers. Hence, we propose to summarize the state of multiple paths between a pair of routers by a single metric, *the width of its candidate paths*. This metric not only provides more accurate but also more stable description of the state of the network than the best-path metric which could change quite frequently. Given a set of candidate paths R_σ of a pair σ, its width, W_σ can be computed using the procedure shown in Figure 7.1, i.e., $W_\sigma = \text{WIDTH}(R_\sigma)$.



(a) border router view (b) source router view

Figure 8.2: Different views of the isp topology

We propose two hierarchical routing schemes based on this aggregate metric. These schemes are similar in that both employ *wdp* for intra-area routing and exchange aggregate width information among border routers. However, they differ in the amount and thus granularity of aggregate information propagated to interior routers by border routers and consequently in the selection of partial higher level paths to destinations.

8.3 Hierarchical Source Routing

Suppose that a border router injects all the information it receives from other border routers into its area. Then an interior router also would have same level of aggregate information as any border router. For routing to a destination in a different area, an interior source router could then select an higher level skeletal path consisting of border routers. Each border router along the skeletal path would then use intra-area routing to find a path to the next border router along the path till the destination is reached. This is referred to as *hierarchical source routing*.

The *wdp* scheme can be extended naturally to provide hierarchical source routing. At the higher level each area is represented by a set of logical links. The average available bandwidth of a logical link corresponding to a pair σ is set to the width of σ, W_σ. Now we can form a higher level network consisting of physical backbone links and the logical links representing each area. For example, consider the isp topology shown in Figure 8.1(a). The corresponding higher level view of border routers is shown in Figure 8.2(a). Under hierarchical source routing, an interior source router would also have a view similar to border routers since border routers propagate their view as is to interior routers also.

Given this higher level view of the network, a source router can apply *wdp* scheme shown in Figure 7.2 *as is* on this higher level network to identify a set of widest disjoint paths as candidates and perform *ebp* based proportioning among these higher level candidate paths. However, there is one difference in that two paths that are considered disjoint at the higher level may not be really disjoint at the lower level. We can be conservative and treat two higher level paths as disjoint only if they do not pass through the same area. In our study, enforcement of such a constraint did not make much difference since such paths were anyway not chosen due to the sharing of inter-area links. We refer to this hierarchical version of *wdp* where *wdp* is used for both intra-area and inter-area routing as *hwdp*.

8.4 Hierarchical Next-hop Routing

It is likely that a border router would provide much less detailed information about the network to an interior router than what is available to itself. It is desirable that a border router further summarize the information it has before propagating it to the interior routers. This reduces the communication overhead on the network and avoids the information overload at interior routers. In other words, a border router not only provides summarized view of its area to other border routers but also presents aggregated view of the network outside the area to its interior routers. For example, the view of a source router corresponding to the isp topology in Figure 8.1(a) is shown in Figure 8.2(b) while the border routers view it as shown in Figure 8.2(a). Under such a realistic scenario, an interior router may have only sufficient information to select an exit border router to reach a destination. It could first select a border router, and then identify an internal path to the border router. The border router which has more detailed information about the backbone network finds a path to a border router in the destination area which would in turn route to the destination. We refer to this approach of selecting the higher level next-hop instead of complete higher level path as *hierarchical next-hop routing*.

We propose a hierarchical next-hop routing scheme which is referred to as *hierarchical widest border routers* (hwbr). Under *hwbr* scheme, a border router further aggregates multiple higher level logical paths to a destination into a single metric, width of candidate logical paths. It then passes this per destination width information to interior routers. Let $R_{x,d}$ be the set of higher level candidate paths from a border router x to a destination d. Then the border router x would pass the aggregate width $W_{x,d} = \text{WIDTH}(R_{x,d})$ for each destination d to its interior routers. An interior router selects a border router as its next-hop

to a destination based on the width from the border router to the destination and the width of candidate paths from it to the border router.

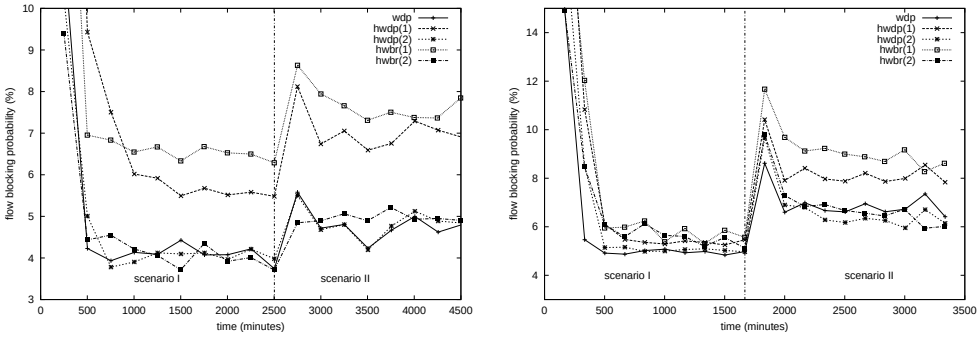
There is a tradeoff in selecting a border router that has better external route to a destination and a border router to which there is a better internal route. However, what matters from the perspective of reducing blocking probability is the width of bottleneck segment. So we define $width(s, d, x)$ from a source s to a destination d via border router x as follows. Let $W_{s,x}$ be the width from source router s to border router x and $W_{x,d}$ be the width from x to destination d. Then $width(s, d, x) = \min(W_{s,x}, W_{x,d})$. Given the widths of all border routers to a destination, a border router with the largest width is chosen as the next-hop to the destination. Note that the next-hop selection is not per flow but periodically recomputed after performing local width adjustment as is the case with selection of candidate paths in *wdp* and *hwdp*. When more than one border router is allowed to be a candidate next-hop, then they are selected in the order of their widths and flows are proportioned among them using the *ebp* strategy.

8.5 Performance Evaluation

In this section, we evaluate the performance of the proposed hierarchical multipath proportional routing schemes. These schemes are compared with the flat *wdp* scheme. We also compare them against the hierarchical best-path routing schemes. We choose *wsp* as a representative of best-path routing approach as it is shown to perform the best among such schemes [2, 31]. We refer to the hierarchical version of *wsp* as *hwsp*. Under *hwsp*, the state of routing between a pair of routers is summarized by the widest shortest path. Hence, the bandwidth and the hop count of a logical link between a pair of routers is given by bottleneck bandwidth and the hop count of the corresponding widest shortest path. The *hwsp* is a hierarchical source routing scheme where a source router selects a widest shortest higher level logical path based on the bandwidths and hop counts of the logical links. This skeletal path is then expanded by the border routers using *wsp* to route within the area. We now describe the simulation environment.

8.5.1 Simulation Environment

Figures 8.1(a) and 8.1(b) show the topologies used in our study. The *isp* topology is a slightly altered version of the topology used in our earlier study and also by others [2, 56]. The *rand* topology is similar to the one used in [31]. In both the topologies the *thin* links



(a) isp (b) rand

Figure 8.3: Convergence and adaptivity of proportional routing schemes

connect routers within an area and the *thick* links are the backbone links. All thin links are assumed to have same capacity of 20 units and all thick links 30 units. Flows arriving into the network are assumed to require one unit of bandwidth. Hence a link with capacity C can accommodate at most C flows simultaneously. The flow dynamics of the network are modeled as follows. The routers labeled with a *dot* are considered to be source or destination routers. Flows arrive into the system according to a Poisson process with rate λ. A pair of source and destination routers are chosen randomly from the set of all such pairs. The holding time of a flow is exponentially distributed with mean $1/\mu$. The overall load offered on the network is then $\rho = \lambda/\mu$. We set the average holding time of a flow, $1/\mu$ to 1 minute and vary the arrival rate λ depending upon the desired overall load ρ.

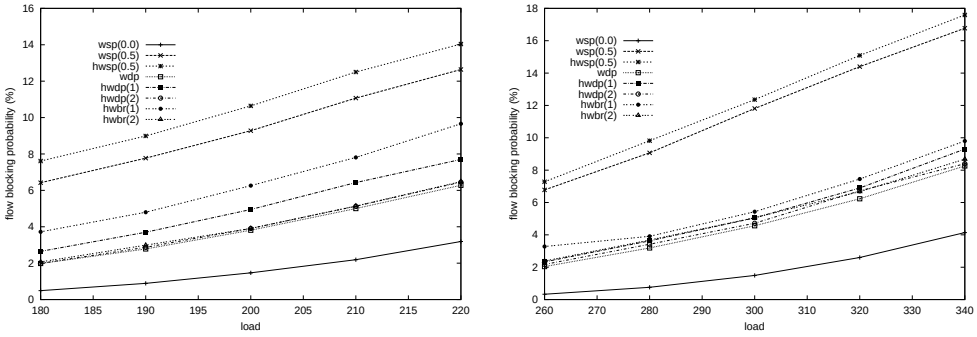
The default values for the configurable parameters of the schemes being simulated are as follows. The update interval in *wsp* and *hwsp* is set to 0.5 minutes or 30 seconds and in the rest of our proportional routing schemes it is set to 30 minutes. The observation interval between recomputations of proportions in the flat *wdp* scheme is set to 40 minutes and candidate path selection is done every 120 minutes. Same settings are used for inter-area routing in both *hwdp* and *hwbr* and the corresponding values for intra-area routing are 20 minutes and 60 minutes respectively. These values are chosen such that inter-area paths are changed more gradually than intra-area paths for the purpose of stability. The number of candidate paths allowed between a pair of nodes is set to 3 for flat routing and intra-area routing.

8.5.2 Convergence and Adaptivity

Figure 8.3 illustrates the convergence and adaptivity of multipath proportional routing schemes. The results corresponding to isp topology are shown in Figure 8.3(a) and that of rand topology are shown in Figure 8.3(b). The performance is measured in terms of the overall flow blocking probability, which is defined as the ratio of the number of blocked flows to the total number of flow arrivals. The overall flow blocking probability is plotted as a function of time. We consider two traffic scenarios. In scenario I, all source-destination pairs are offered an equal amount of load and in scenario II, certain "hot pairs" exchange more traffic than other pairs. We have chosen two hot pairs marked by 1 and 2 in case of isp. Similarly in case of rand three pairs are chosen as hot. In case of isp, we offer a load of 200 in scenario I which is equally split between all source-destination pairs and in scenario II a load of 150 that is equally split among all pairs and a load 50 that is equally split among hot pairs only. The corresponding values for rand case are 300, 200 and 100 respectively.

The performance of *hwdp* is shown for two cases with the number of higher level candidate paths set to 1 and 2. Similarly two plots correspond to *hwbr* with the number of candidate border routers set to 1 and 2. We start the simulations with scenario I and switch to scenario II after sometime. There are several observations that can be made from these results. Starting with an arbitrary set of candidates and proportions, all the multipath proportional routing schemes gradually converge to stable state. This gets disturbed when the traffic scenario changes and the blocking probability shoots up as the candidates and their proportions chosen for one traffic pattern are not perfect for a different traffic pattern. Consequently the proposed proportional routing schemes gradually adapt to the new traffic conditions and converge to stable state again.

With a single candidate path *hwdp* performs better than *hwbr* with a single candidate border router. This is because in *hwbr,* due to lack of sufficient information, many sources may select the same border router as next-hop to many destinations. On the other hand, *hwdp* avoids this problem by using more detailed information in the selection of skeletal path to the destination. However, when the number of candidates is increased to two, there is almost no difference in blocking performance between the source routing scheme *hwdp* and the next-hop routing scheme *hwbr*. These results also indicate that there is significant gain in using multiple candidates for inter-area routing also. The gain is more pronounced in case of isp than in rand and also in scenario II with hot pairs and thus non-uniform load than in scenario I with uniform load. Finally, both the proposed hierarchical schemes with



(a) isp (b) rand

Figure 8.4: Performance comparison with best-path routing schemes

two candidates perform as well as the flat *wdp* scheme.

8.5.3 Blocking Performance

Figures 8.4(a) and 8.4(b) show the performance of these schemes under different load conditions for isp and rand topologies respectively. The blocking probability is plotted as a function of offered load which is varied from 180 to 220 in case of isp and 260 to 340 in case of rand. Note that the update interval in *wsp* and *hwsp* is set to 30 seconds while that in all our proportional routing schemes is set to 30 minutes. We also show the performance of flat *wsp* scheme with update interval of 0 as a reference. This corresponds to the case of instantaneous updates, i.e., an update generated for every change in a link's available bandwidth or a server that keeps track of the current state of the network and performs path selection and admission control in a centralized manner.

First, let us look at the impact of update interval and state aggregation on the performance of *wsp* scheme. There is a drastic increase in the blocking probability when the update interval is changed from an unrealistic setting of 0 to a more realistic value of 30 seconds. This shows the impact of inaccuracy on best-path routing schemes and that they work well only with very frequent updates. Essentially, best-path routing is suitable for centralized routing but not appropriate for distributed routing. The hierarchical version of *wsp*, *hwsp* fares worse than *wsp*. This is because the impact of inaccuracy introduced by the best-path based aggregation gets further amplified by the selection of the best higher level logical path based on inaccurate information.

Now, let us compare the proposed hierarchical proportional routing schemes with best-path routing schemes *wsp* and *hwsp*. It is expected that with always up-to-date information (update interval of 0), the flat *wsp* scheme would perform better than proportional routing schemes. However, it is surprising that with only aggregate information and update interval of 30 minutes, hierarchical proportional routing schemes perform much better than even the flat *wsp* scheme with update interval of 30 seconds. It is worth noting that this is the case even when the number of candidates is limited to one in both *hwdp* and *hwbr* schemes. Furthermore, the gap in blocking performance between the proposed schemes and *wsp* widens with the larger rand topology. These results demonstrate that hierarchical multipath proportional routing schemes with their minimal update overhead are more suitable than best-path routing schemes for routing in large networks.

These results further support the observations made earlier about the relative performance of multipath proportional routing schemes. Across all loads and in both the topologies, the performance of *hwdp* is better than *hwbr* when the number of candidates permitted are only one. However, when the number of candidates allowed is two, there is not much difference in performance between these two schemes. More importantly, with two candidates, both these schemes perform as well as the flat *wdp* scheme. Considering the overhead involved in the propagation and the maintenance of the summary information about other areas, we conclude that the *hwbr* scheme with multiple candidate border routers is a suitable choice for hierarchical routing.

Chapter 9

Conclusions and Future Work

In this thesis, we focussed on *proportional routing* approach as an alternative to *best-path routing* approach for providing QoS routing. While best-path routing schemes select the best path for each incoming flow, proportional routing schemes proportion flows among a set of candidate paths. The best-path routing schemes require frequent exchange of network state, imposing both the communication overhead on the network and processing overheads on core routers. On the other hand, a proportional routing scheme can select a few good candidate paths using infrequently exchanged global information and proportion flows among candidates using only locally collected information. We have described one such scheme that selects *widest disjoint paths* as candidates and proportions flows among these paths using a simple local *equalization of blocking probabilities* strategy. We have shown that our proportional routing scheme yields higher throughput with lower overhead than best-path routing schemes.

We have extended our proportional routing approach to provide hierarchical routing across large networks that are divided into multiple areas. We proposed an aggregation method that summarizes the state of multiple paths between two routers in an area using a single metric. We presented two hierarchical multipath routing schemes *hierarchical widest disjoint paths* (hwdp) and *hierarchical widest border routers* (hwbr) that are based on this aggregate metric. We evaluated their performance and shown that the proposed hierarchical multipath routing schemes perform as well as flat multipath routing scheme *wdp*. Furthermore, we demonstrated that these schemes with only aggregate information outperform even the flat best-path routing scheme *wsp* having detailed information about the network.

Based on our results, we conclude that *hwbr* scheme with multiple candidates, due to its low overhead and high throughput, is a suitable choice for hierarchical routing across large networks.

We have used analytical methods and simulation models to verify the proposed proportional routing schemes and compare them with other schemes. However it would be ideal if we could implement and validate them in a real setting. But it is not possible to deploy them on a large scale and conduct experiments with an actual network. So we plan to simulate the core network while using FreeBSD or Linux machine implementations of our schemes to act as edge routers. Since the proposed schemes essentially add more intelligence at the edge routers and leave the functionality of core routers untouched, this serves the purpose. We are specifically interested in the amount of complexity introduced by these schemes at the edge routers. This experimental setup would help measure the storage requirements and processing speeds of these schemes. The real traffic conditions would be simulated by running actual traffic traces.

This thesis assumes a service model where admission of flows is controlled such that every admitted flow receives the expected quality of service. While such a service is very much desirable, there is an immediate need for providing what is referred to as *better best-effort service*. Under such a service, there are neither admission controls nor per-flow guarantees. Instead, a better service is provided to all the traffic by balancing the load across the network and by utilizing the resources efficiently. This service based on multipath routing is likely to be more readily adopted by network service providers as an alternative to the existing shortest path routing based best-effort service. Provision of this *better best-effort service* entails tackling many of the issues that are addressed in this thesis in the context of QoS routing: How to select a few good candidate paths and how to split traffic among them. The ideas presented in this thesis such as local adjustment to global updates and selection of widest paths that are mutually disjoint w.r.t. bottleneck links as candidates are very much applicable in this setting too. One key difference is that due to lack of admission control, a source would not be able to gather information such as blocking probability of flows locally. So, what requires to be addressed is how to proportion traffic (in the absence of local information) among a set of candidate paths selected by *wdp*. This and other related problems in deploying *better best-effort service* is the focus of our current research.

Part II

Stored Video Delivery across Resource Constrained Networks

Chapter 10

Selective Frame Discard Algorithms

10.1 Introduction

The playback of stored video over a network is required by several applications such as digital libraries, distance learning and collaboration, video and image servers and interactive virtual environments. Stored video typically has high bandwidth requirements and exhibits significant rate variability [24, 49, 50]. This is particularly the case when variable bit rate encoding schemes are used. In a network where resources such as the network bandwidth and buffering capacity are constrained, it is a major challenge to design an efficient stored video delivery system that can achieve high resource utilization while maximizing users' perceived quality-of-service (QoS).

Video smoothing techniques [97, 114, 23, 116, 60, 88, 37, 85] have been proposed for reducing the network bandwidth requirement of bursty video streams by taking advantage of client buffering capabilities. Similar techniques have also been developed when network bandwidth is constrained instead of the client buffer [21, 89, 96, 101]. In reality, however, both network bandwidth and client buffering capacity are likely to be limited. Under such circumstances, there may *not* be a feasible server transmission schedule that can deliver video streams to clients without incurring loss of data. Instead of being denied service, clients may choose to receive lower quality video streams with occasional frame losses. This may arise, for example, in the case of constant-bit-rate (CBR) service, where for a client with a limited buffer, the network may not have sufficient bandwidth to support the peak rate of a smoothed video stream, or in the case of renegotiated CBR (RCBR) service [28], where bandwidth renegotiation fails in the middle of a video transmission.

When delivering a video stream across a resource-constrained network, a naive approach at the server may attempt to transmit each frame with no awareness of the resource constraints. As a result the network may drop packets causing frame losses. In addition, the client may be forced to drop frames that arrive too late for playback. This results in wastage of network bandwidth and client buffer resources. In this chapter, we introduce the concept of *selective frame discard*[1] *(SFD)* at the server which *preemptively* discards frames in an intelligent manner by taking network constraints and client QoS requirements into consideration. The proposed server selective frame discard has two advantages. First, by taking the network bandwidth and client buffer constraints into account, the server can make the best use of network resources by selectively discarding frames in order to minimize the likelihood of future frames being discarded, thereby increasing the overall quality of the video delivered. Second, unlike frame dropping at the network or the client, the server can also take advantage of *application-specific* information such as information content of a frame and inter-dependencies, in its decision in discarding frames. As a result, the server optimizes the perceived quality of service at the client while maintaining efficient utilization of the network resources.

In this chapter we develop various selective frame discard algorithms for stored video delivery across a network where both the network bandwidth and the client buffer capacity are limited. We begin by formulating the problem of *optimal selective frame discard* using the notion of a cost function. The cost incorporates the QoS metrics of clients. Given network bandwidth and client buffer constraints, we develop an $O(N \log N)$ algorithm to find the minimum number of frames that must be discarded in order to meet these constraints. For a given cost function, an optimal algorithm for solving the optimal selective frame discard problem can be designed using dynamic programming. Since the computational complexity of this optimal algorithm is prohibitively high in general, we also develop several efficient heuristic algorithms which take both resource constraints and cost into consideration. These algorithms are evaluated using JPEG video traces. Through the performance evaluation, we find that the proposed *minimum cost maximum gain* heuristic algorithm yields near-optimal performance for JPEG encoded video.

The rest of this chapter is organized as follows. In Section 10.1, we briefly discuss related

[1] In this work we assume that frames are basic *application-level* data units for server selective discard. This assumption is not necessary. The algorithms developed in this chapter do not hinge on this assumption at all. In practice, other (preferably) application-level data units such as slices, blocks or macro blocks in JPEG and MPEG can also be used as the basis for server selective discard.

work. Section 2 describes the problem setting and formulates the optimal selective frame discard problem. The minimum frame discard algorithm is described and its correctness is proved in Section 10.3. Section 10.4 introduces several efficient selective frame discard heuristics and presents performance evaluation based on JPEG traces. These algorithms are extended to MPEG in Section 10.5. We conclude with Section 10.6.

Related Work

Rate adaptation and control for compressed video has been extensively studied, in particular, in the context of joint source and channel adaptive encoding (see, e.g., [18, 19, 33, 38, 51, 54, 102]). For example, in [33, 102], the problem of finding an optimal transmission schedule with leaky bucket constraints is studied some form of cost functions. In [19, 38, 51], video rate adaptation through quantization parameter adjustment over networks with feedback-based congestion control mechanisms. These techniques are mostly designed for *live* video transmission, and may not be well suitable to delivery of stored video, due to a number of reasons (e.g., absence of encoders at a stored video server, or the processing overhead/delay incurred).

Packet discarding schemes which take advantage of application specific information have been used in many different contexts. For example, in [83] a frame-induced packet discarding scheme at the network level is introduced. In this scheme upon detection of loss of a threshold number of packets belonging to a video frame, the network attempts to discard all the remaining packets of that frame. Similarly, the continuous media toolkit (CMT) [94, 95] also uses application-specific information to assign priority in packet discarding. Our problem setting, however, is considerably different from these existing studies. In designing efficient server selective frame discard algorithms, we leverage application-specific information to optimize the client QoS while at the same time taking both network bandwidth and client buffer constraints into account.

10.2 Problem Formulation

In this section we provide an overview of the stored video delivery system and motivate the notion of *selective frame discard* at the server for a resource constrained network. The idea of a cost function is introduced to incorporate a QoS metric and is used to formulate the selective frame discard problem.

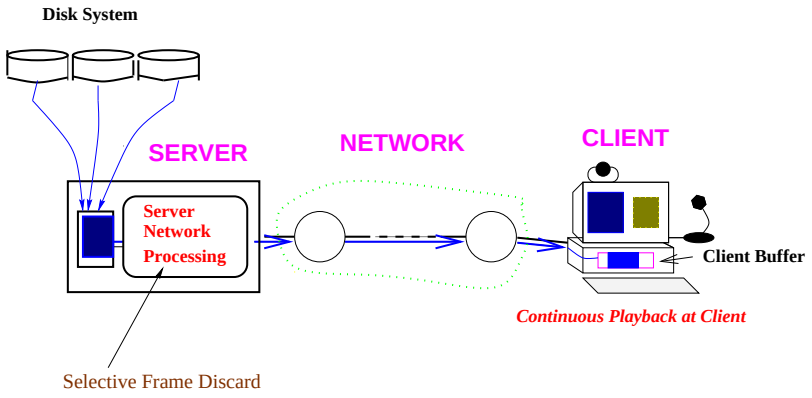


Figure 10.1: Overview of the problem setting

Figure 1 depicts a server transmitting a stored video stream to a client across a network. The video data is retrieved from the disk subsystem into the server memory and moved onto the network as per some server transmission schedule. The client has a buffer which can be used for the work ahead of video data by the server. The client plays back the video frames periodically as determined by the frame rate. Each video frame has a deadline constraint associated with it. Since the frames are being played back at a periodic rate, the frame has to be available at the client when the decoding process attempts to display it. If the frame is not available, the playback is paused, resulting in a *playback discontinuity*.

We assume a network environment where a fixed amount of network bandwidth can be reserved for a video stream (i.e., the CBR service). However, the network resource may be constrained (this constraint is referred to as *rate constraint*). Furthermore, the buffer resource at a client may also be constrained (this constraint is referred to as *client buffer constraint*). In such a resource constrained system, there may not be sufficient resources to ensure the continuous playback of the video at the client. While the rate constraint regulates the amount of data that can be transmitted in one time unit, the client buffer constraint limits the amount of work ahead by the server into the client buffer. In the presence of both rate and buffer constraints, a feasible server transmission schedule which satisfies both constraints simultaneously may not exist. Hence in these circumstances, frame dropping is unavoidable.

A naive approach at the server may attempt to transmit each frame with no cognizance of the resource constraints. This may cause packet loss and delay in the network or buffer overflow at the client. As a result the client may receive incomplete frames which cannot be played back. Also the client may be forced to drop a frame if it arrives late. The system

N	:	length of video in frames.
f_i	:	size of i^{th} frame.
B	:	client buffer capacity for storing unplayed frames.
C	:	network bandwidth.
$\mathcal{N}$	:	set of all frames, i.e, $\{1, \ldots, N\}$
S	:	a subset of frames, i.e., $S \subseteq \mathcal{N}$.
$A(S)$	:	a transmission schedule w.r.t. set S
$A_i(S)$	:	cumulative data sent by the server over $[1, i]$
$a_i(S)$	:	amount of data sent by the server in slot i
$D(S)$	:	underflow curve w.r.t set S
$D_i(S)$	:	cumulative data consumed by the client over $[1, i]$
$U(S)$	:	overflow curve w.r.t set S
$U_i(S)$	:	maximum cumulative data that can be received by the client over $[1, i]$
$B_i(S)$	:	buffer occupancy at the end of time slot i.
$\hat{A}(S)$	:	greedy transmission schedule w.r.t set S
$\hat{A}_i(S)$	:	cumulative data sent by the server over $[1, i]$ according to the greedy schedule
$\hat{a}_i(S)$	:	amount of data transmitted in slot i under $\hat{A}(S)$.

Table 10.1: Notation

resources consumed by these dropped frames are effectively wasted.

Selective frame discard aims at optimizing the utilization of the network resources by *pre-emptively* discarding frames at the server. A frame is transmitted only if it can meet its playback deadline. Otherwise the frame is discarded thereby increasing the likelihood of other frames meeting their playback deadlines. By effectively utilizing the resources, selective frame discard improves the playback continuity.

In formulating the selective frame discard problem, we consider a discrete-time model at the frame level. Each time slot represents the unit of time for playing back a video frame. For simplicity of exposition, we assume zero startup delay, i.e., the time the server starts video transmission and the time the client starts playback is the same. We also ignore the network delay. Table 10.1 summarizes the notation we introduce in this section.

Consider a video stream with N frames. For $i \in \mathcal{N} = \{1, \ldots, N\}$, the size of i^{th} frame is denoted by f_i. Let C denote the bandwidth of the network (i.e., server transmission rate is limited by C per unit of time), and B is the client buffer size. For $S \subseteq \mathcal{N}$, $\mathbf{1}_{\{j \in S\}}$ is the indicator function: $\mathbf{1}_{\{j \in S\}} = 1$ if $j \in S$ and 0 if $j \notin S$. Let $D(S) = \{D_0(S), D_1(S), \ldots, D_N(S)\}$ where $D_i(S) = \sum_{j=0}^{i} f_j \, \mathbf{1}_{\{j \in S\}}$, and let $U(S) = \{U_0(S),$

$U_1(S), \ldots, U_N(S)\}$ where $U_i(S) = D_i(S) + B$. We refer to $D(S)$ as the *(client) buffer underflow curve with respect to* S, and $U(S)$ as the *(client) buffer overflow curve with respect to* S. A server transmission schedule $A(S)$ associated with S is a schedule which only transmits frames included in S, namely, frame i is transmitted under $A(S)$ if and only if $i \in S$. Let $a_i(S)$ be the amount of video data transmitted during time slot i, $i = 1, \ldots, N$. In accordance with the notation for $D(S)$ and $U(S)$, the schedule $A(S)$ is denoted by $A(S) = \{A_0(S), A_1(S), \ldots, A_N(S)\}$ where $A_0(S) = 0$ and $A_i(S) = \sum_{j=0}^{i} a_j(S)$. Examples of $D(S)$, $U(S)$ and $A(S)$ are shown in Figure 10.2.

A server transmission schedule $A(S)$ is said to be *feasible* with respect to S if and only if for $i = 0, 1, \ldots, N$, 1) *rate constraint is not violated*, i.e., $a_i(S) \leq C$; 2) *buffer constraint is not violated*, i.e., $A_i(S) \leq U_i(S)$; and 3) *playback constraints are not violated*, i.e, $D_i(S) \leq A_i(S)$. In other words a transmission schedule is feasible if it lies within the buffer underflow curve $D(S)$ and the buffer overflow curve $U(S)$, having slope no more than C (see Figure 10.2 for an illustration). A set $S \subseteq \mathcal{N}$ is said to be *feasible* if and only if there exists a feasible transmission schedule $A(S)$ with respect to S. For a given pair of rate and buffer constraints (C, B), we denote the collection of all feasible sets by $\mathcal{SFD}(C, B)$.

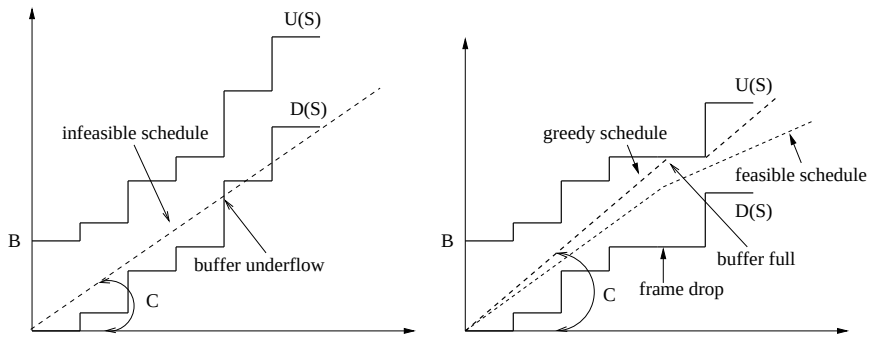
Given a schedule $A(S)$, the buffer occupancy at the end of time slot i (namely, immediately after frame i has been retrieved from the client buffer if $i \in S$) is denoted by $B_i(S)$. $B_i(S)$ satisfies the following recurrence relation:

$$B_i(S) = \max\{\min\{B_{i-1}(S) + a_i(S), B\} - f_i \mathbf{1}_{\{i \in S\}}, 0\}$$

where $B_0(S) = 0$

If $B_{i-1}(S) + a_i(S) > B$, the *buffer overflow* occurs at time i. If $B_{i-1}(S) + a_i(S) < f_i$, then *buffer underflow* occurs at time i. Clearly for a feasible schedule, $B_i(S) = B_{i-1}(S) + a_i(S) - f_i \mathbf{1}_{\{i \in S\}}$.

Associated with each S, we define a special schedule $\hat{A}(S)$, referred to as the *greedy transmission schedule* with respect to S. Under $\hat{A}(S)$, the amount of data transmitted in time slot i, $i = 1, \ldots, N$, is given by $\hat{a}_i(S) = min\{B - B_{i-1}(S), C)$, where $B_0(S) = 0$ and $B_i(S) = B_{i-1}(S) + \hat{a}_i(S) - f_i \mathbf{1}_{\{i \in S\}}$. Hence $\hat{A}(S) = \{\hat{A}_0(S), \hat{A}_1(S), \ldots, \hat{A}_N(S)\}$, where $\hat{A}_i(S) = \sum_{j=0}^{i} \hat{a}_j(S)$. It is clear that $\hat{A}(S)$ transmits at the rate C whenever possible without overflowing the buffer (see Figure 10.2 for an example). In other words, it attempts to



(a) An infeasible schedule (b) A feasible schedule

Figure 10.2: Relation of $D(S)$, $U(S)$ and a server transmission schedule $A(S)$

keep the buffer as full as possible. By definition $\hat{A}(S)$ always lies below the buffer overflow curve $U(S)$. Hence $\hat{A}(S)$ is feasible if it stays above the underflow curve $D(S)$, i.e., if $\hat{A}_i \geq D_i(S)$, $i = 0, 1, \ldots, N$.

The greedy schedule $\hat{A}(S)$ has the following property, the proof of which is straightforward.

Proposition 1 *For any $S \subseteq \mathcal{N}$, if $A(S)$ is a schedule conforming to the rate constraint, then $A_i(S) \leq \hat{A}_i(S)$, $i = 0, 1, \ldots, N$.*

Since $\hat{A}(S)$ bounds the amount of data that can be transmitted from the above, any feasible transmission schedule has to stay below $\hat{A}(S)$. Hence if $\hat{A}(S)$ is not feasible, then any other schedule $A(S)$ is not feasible. As a result, for any $S \subseteq \mathcal{N}$, *S is a feasible set if and only if $\hat{A}(S)$ is feasible with respect to S.*

For a given pair of rate and buffer constraints (C, B), there are in general more than one feasible set. For example, trivially $S = \emptyset$ is always a feasible set. Obviously the perceived quality of the playback at the client would depend on the frames transmitted by the server. It is likely that the greater the number of frames dropped, the lesser the perceived video quality. In addition, consecutive losses of frames or a cluster of lost frames in near proximity would have a more pronounced impact on the perceived video quality than dispersed losses of frames. In order to reflect the perceived video quality at the client, we introduce the notion of a *cost function*, $\phi(S)$, to quantify the "desirability" of different feasible sets. Such a function associates a certain cost with each discarded frame. The cost of a feasible

set $\phi(S)$ is the cost associated with the frames that are not part of the set. For an appropriately defined cost function, $\phi(S)$ should reflect the perceived quality of playing back the set S. Thus minimizing the cost is equivalent to optimizing the QoS at the client.

For a given cost function ϕ, the *optimal selective frame discard problem* therefore is to find a feasible set S^* which minimizes the associated cost $\phi(S^*)$, formally

> *Find a set* S^* *such that* $S^* \in \mathcal{SFD}(C, B)$ *and* $\phi(S^*) = \min\{\phi(S) : S \in \mathcal{SFD}(C, B)\}$.

S^* is referred to as an *optimal feasible set* with respect to ϕ.

For a given cost function ϕ, a general optimal algorithm to determine S^* can be designed using dynamic programming. The optimal algorithm proceeds in stages, where stage i corresponds to the ith frame, $i = 1, 2, \ldots, N$. In each stage, a set of appropriate states is maintained. A transition from a state in stage i-1 to another state in stage i represents whether frame i is included or discarded at stage i while no constraints are violated. The incurred cost of the transition is computed accordingly using the cost function. The optimal selective frame discard problem can thus be reduced to a shortest path problem and solved using dynamic programming. The computational complexity of the algorithm is $O(NBW)$, where W is the largest size of the states in each stage, which in the worst case can be as large as 2^N.

10.3 Upper Bound on the Size of Feasible Sets

Before we address the problem of finding an optimal set that minimizes a given cost function, we first consider a more fundamental question:

> *What is the minimum number of frames to be discarded so that the remaining frames that are transmitted by the server can meet their respective playback time under the known network bandwidth (rate) and client buffer constraints? Or in other words, what is the largest possible cardinality of any feasible set* $S \in \mathcal{SFD}(C, B)$*?*

The solution to this question is not only of interest in its own right, but, as we will see, also sheds light on the design of efficient selective frame discard algorithms in Section 10.4. In

```
1.  PROCEDURE MINFD(C, B)
2.    Initialization (i = 0): S# = ∅, B_i = 0, i_0 = 0.
3.    For i = 1 to N
4.      â_i = C
5.      If B_{i-1} + C > B, i.e., is buffer full?
6.        â_i = B − B_{i-1} and i_0 = i
7.      Else
8.        If B_{i-1} + C < f_i, i.e., is deadline of frame i violated?
9.          For j = i_0 + 1 to i
10.           Compute the gain Δ^i_j
11.         Choose the frame k with the largest gain max Δ_i
12.         Discard frame k and include frame i, i.e.,
13.         S# := (S# ∪ {i}) \ {k}
14.         Update buffer occupancy at B_i, i.e.,
15.         B_i := B_{i-1} + C + max Δ_i − f_i
16.         Update i_0 if necessary
17.       Else
18.         S# := S# ∪ {i}
19.   Output S#
20. END PROCEDURE
```

Figure 10.3: The minimum frame discard (MINFD) algorithm.

this section, we first present an algorithm for solving this problem, and then establish its correctness. This algorithm is referred to as the *minimum frame discard* (in short MINFD) algorithm. In Subsection 10.3.1 we present the details of the MINFD algorithm for video streams encoded using an intra-frame encoding scheme such as JPEG, where there is no inter-frame dependence among the frames. Extension of the MINFD algorithm to handle inter-frame dependence is discussed in Subsection 10.3.2, using the MPEG encoding scheme as an example.

10.3.1 The MINFD Algorithm for Intra-Frame Encoded Video

Consider a video stream encoded using an intra-frame encoding scheme such as JPEG. Following the notation introduced in Section 10.2, f_i denotes the size of the i^{th} frame of the video stream. Let C denote the available network bandwidth (i.e. the rate constraint) and B, the size of the client buffer.

The following observations play a key role in the development of the MINFD algorithm.

1. As long as the buffer constraint is not violated, always try to send as much data as

possible (i.e., send at rate C)

2. Whenever the buffer is full, delay transmission until the buffer is no longer completely filled and then resume transmission at rate C. Note that *it is never necessary to discard frames because of buffer overflow*.

3. Whenever a playback deadline cannot be met, either the current frame or an earlier frame must be discarded. This is because the total size of the currently included frames is more than that can be transmitted using the available bandwidth *subject to the buffer constraint*. In deciding the frames to be discarded, we should choose those that would optimize the likelihood of the deadlines of future frames being met.

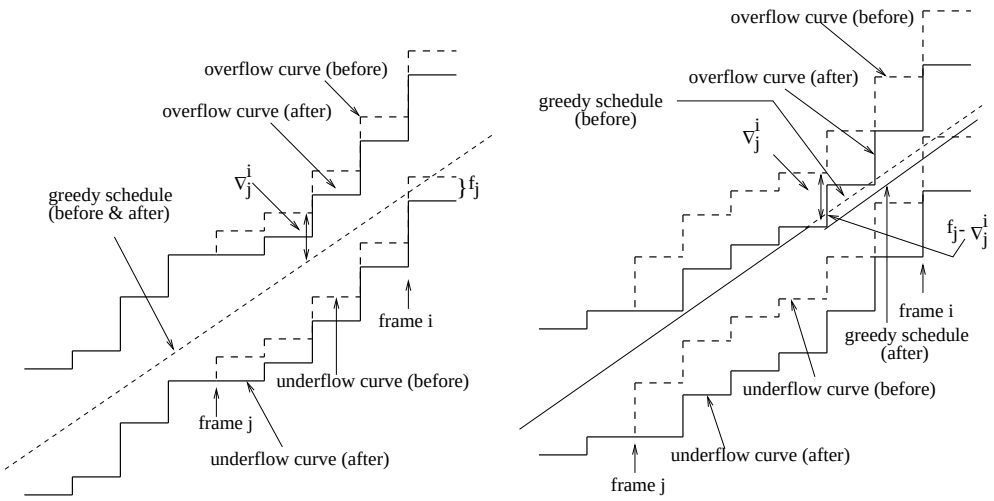
The first two observations state that we should follow the greedy schedule in transmitting the video data. Based on the third observation, we devise a strategy which discards the frame that maximizes the *buffer occupancy* at the time when a playback deadline is violated. In Theorem 1, we show that this strategy is optimal in the sense that it minimizes the total number of frames discarded.

The algorithm is presented in pseudo-code in Figure 10.3. It proceeds in stages, $i = 0, 1, \ldots, N$, and constructs a feasible set $S^{\#}$ iteratively.

At stage 0 (line 2 in the algorithm), we start with $S^{\#} = \emptyset$. At this point, the buffer occupancy $B_0 = 0$. The variable i_0 is used to keep record of the most recent buffer full point if any, and is initialized to 0.

At any stage i (lines 3-16), $i = 1, \ldots, N$, we follow the greedy schedule $\hat{A}$, and transmit as much data as possible, namely, $\hat{a}_i = \min\{C, B - B_{i-1}\}$ (lines 4-6). If the buffer is full at this point, set $i_0 = i$. Otherwise (lines 7-16), we check to see whether the playback deadline of frame i is met by the greedy schedule $\hat{A}$ with respect to the current feasible set $S^{\#}$ (line 8). If $B_{i-1} + C < f_i$, the playback deadline of frame i is violated, a frame needs to be discarded. In order to decide which frame to discard, for each j, $1 \leq j \leq i$, we introduce the notion of *gain* in the buffer occupancy at time i if frame j is discarded. We denote this gain by Δ_j^i; its definition will be given shortly. The frame discarded, say, frame k, is thus the one which yields the largest gain, namely, $\Delta_k^i = \max_{1 \leq j \leq i} \Delta_j^i$. This is done in lines 9-11.

Formally, let $S_{i-1}^{\#}$ denote the feasible set constructed at stage $i-1$. Recall that $D_j(S_{i-1}^{\#})$, $U_j(S_{i-1}^{\#}) = D_j(S_{i-1}^{\#}) + B$ and $\hat{A}_j(S_{i-1}^{\#})$ represent the buffer underflow curve, buffer over-



(a) $f_j < \nabla_j^i$ (b) $f_j \geq \nabla_j^i$

Figure 10.4: Effect of discarding a frame j on D, U and $\hat{A}$

flow curve and the amount of data transmitted by the greedy schedule up to time j with respect to $S_{i-1}^{\#}$. For $j = 1, \ldots, i-1$, define

$$\nabla_j^i = \min_{j \leq l \leq i-1} \{U_l(S_{i-1}^{\#}) - \hat{A}_l(S_{i-1}^{\#})\} = \min_{j \leq l \leq i-1} \{D_l(S_{i-1}^{\#}) + B - \hat{A}_l(S_{i-1}^{\#})\}. \quad (10.1)$$

∇_j^i represents the minimal difference between the buffer overflow curve $U(S_{i-1}^{\#})$ and the greedy schedule $\hat{A}(S_{i-1}^{\#})$ in the time interval $[j, i-1]$. Intuitively, it is the maximal amount that we can shift the segment $[j, i]$ of $U(S_{i-1}^{\#})$ downwards towards $\hat{A}(S_{i-1}^{\#})$ without crossing $\hat{A}(S_{i-1}^{\#})$ (see Figure 10.4).

Now we are in a position to define Δ_j^i.

$$\Delta_j^i = \begin{cases} f_i, & j = i, \\ \min\{f_j, \nabla_j^i\}, & j = 1, \ldots, i-1. \end{cases} \quad (10.2)$$

We now show that Δ_j^i is the gain in the buffer occupancy at time i if frame j is discarded. More precisely,

$$B_{i-1}(S_{i-1}^{\#} \setminus \{j\}) = B_{i-1}(S_{i-1}^{\#}) + \Delta_j^i. \quad (10.3)$$

This is shown pictorially in Figure 10.4 where the two cases: (a) $f_j \leq \nabla_j^i$ and (b) $f_j > \nabla_j^i$ are depicted. As a result of discarding frame j, the segment $[j+1, i-1]$ of the new buffer overflow curve $U(S_{i-1}^{\#} \setminus \{j\})$ and underflow curve $D(S_{i-1}^{\#} \setminus \{j\})$ are the original ones ($U(S_{i-1}^{\#})$ and $D(S_{i-1}^{\#})$) shifted f_j amount downwards. Consider the case where $f_j \leq \nabla_j^i$. This can only occur if $j > i_0$ where i_0 is the last time before i the buffer is full. The greedy schedule can transmit exactly the same amount of data as the original schedule, i.e., $(i-1-j)C$, during the time interval $[j+1, i-1]$. Therefore, (10.3) holds at time $i-1$. On the other hand, if $f_j > \nabla_j^i$, the amount of data transmitted by the greedy schedule during the same interval is only $(i-1-j)C - (f_j - \nabla_j^i)$. This is because the buffer becomes full at some point. Hence the greedy schedule needs to stop transmission for a duration of $(f_j - \nabla_j^i)/C$ time. Thus (10.3) also holds at time $i-1$. Note that in either case, we have

$$\hat{A}_{i-1}(S_{i-1}^{\#} \setminus \{j\}) - \hat{A}_j(S_{i-1}^{\#} \setminus \{j\}) = \hat{A}_{i-1}(S_{i-1}^{\#}) - \hat{A}_j(S_{i-1}^{\#}) - (f_j - \Delta_j^i). \quad (10.4)$$

From (10.1), $\nabla_j^i = 0$ for any $j \leq i_0$. Therefore, discarding any frame before time i_0 will result in zero gain, i.e., $\Delta_j^i = 0$. In other words, *discarding any frame before the last buffer full point will not help meet the playback deadline of frame* i. This is the reason in line 9 of the algorithm in Figure 10.3, we only search in the range of $[i_0+1, i]$ for a frame to discard. Let k, $i_0 + 1 \leq k \leq i$ be such that $\Delta_k^i = \max_{i_0+1 \leq j \leq i} \Delta_j^i$. Hence discarding frame k yields the maximal gain at time i. Denote this maximal gain by $\max \Delta_i$, i.e., $\max \Delta_i = \Delta_k^i$. As $\Delta_i^i = f_i$, we have $\max \Delta_i \geq f_i$. Hence from (10.3)

$$B_{i-1}(S_{i-1}^{\#} \setminus \{k\}) \geq B_{i-1}(S_{i-1}^{\#}) + f_i.$$

Therefore, if $k \neq i$, discarding frame k will help meet the playback deadline of frame i. As a result of discarding frame k from $S_{i-1}^{\#}$ and including frame i at stage i, i.e., setting $S_i^{\#} := S_{i-1}^{\#} \cup \{i\} \setminus \{k\}$ (lines 12-13), we have

$$B_i(S_i^{\#}) = B_{i-1}(S_{i-1}^{\#}) + C + \max \Delta_i - f_i. \quad (10.5)$$

Note that the above equation also holds when $k = i$.

In lines 14-16 of the algorithm, the buffer occupancy B_i is updated using (10.5), and i_0 is set to k^* if discarding k results in a full buffer at time k^*, where k^* is the point where the minimum in (10.1) is attained. If the deadline of frame i is met, it is included in $S_i^{\#}$ by

setting $S_i^{\#} := S_{i-1}^{\#} \cup \{i\}$ (line 18). The algorithm stops after stage N and outputs the set $S^{\#}$.

The feasible set $S_i^{\#}$ constructed at stage i of the MINFD algorithm has the following important property.

Lemma 1 *Let S be any feasible set, i.e., $S \in \mathcal{SFD}(C, B)$. Then*

$$|S_i^{\#}| \geq |S \cap \{1, 2, \ldots, i\}| \tag{10.6}$$

where $|\cdot|$ denote the cardinality of a set.

Moreover, for any $j = 1, 2, \ldots, i$, if

$$|S_i^{\#} \cap \{1, \ldots, j\}| = |S \cap \{1, \ldots, j\}|, \tag{10.7}$$

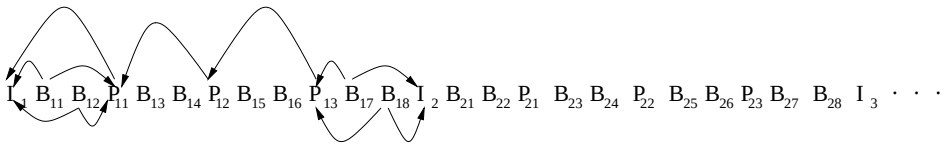
then $B_j(S_i^{\#}) \geq B_j(S)$.

Intuitively, Lemma 1 states that the number of frames included in the (partial) feasible set $S_i^{\#}$ constructed at stage i is at least as large as the number of frames (up to time i) that are included in any other feasible set. Moreover, among all feasible sets *that discard the same number of frames up to time j*, $S_i^{\#}$ maximizes the buffer occupancy at time j. Hence, $S_i^{\#}$ maximizes the chance of future frames to meet their playback deadlines. As a consequence of this lemma, the transmission schedule $S^{\#}$ produced by the MINFD algorithm results in the minimum number of discarded frames for any cost function, or equivalently, $|S^{\#}|$ is maximized.

Theorem 1 *Let $S^{\#}$ be the feasible set produced by the MINFD algorithm. Then*

$$|S^{\#}| = \max\{|S| : S \in \mathcal{SFD}(C, B)\}. \tag{10.8}$$

By using a clever data structure for maintaining and updating the gain Δ_j^i, we can design an $O(N \log N)$ algorithm to construct $S^{\#}$. Due to space limitation, we will not describe it here.



(a) Playback order with dependencies (e.g., B_{17} depends on P_{13} and I_2)

I 1 P11 B11 B12 P12 B13 B14 P13 B15 B16 I 2 B17 B18 P21 B21 B22 P22 B23 B24 P23 B25 B26 I 3 B27 B28 . . .

(b) Decoding order

Figure 10.5: Sequence of MPEG frames with a GOP of size 12. Frame P_{ij}/B_{ij} refers to jth P/B frame in the ith GOP

10.3.2 The MINFD Algorithm for MPEG Encoded Video

The MINFD algorithm described in Figure 10.3 can be extended to handle video streams with inter-frame dependencies such as those encoded using the MPEG encoding scheme. In this section, we illustrate how to modify the MINFD algorithm to handle inter-frame dependencies using MPEG as an example.

The MPEG standard defines three types of frames: I frames, P frames and B frames. I frames are coded autonomously, while P frames are coded with respect to the previous I or P frame. B frames use both previous and future I or P frames as a reference. The MPEG standard also defines a group-of-pictures (GOP) as a consecutive sequence of frames (pictures) containing a single I frame, which is the first frame of the group, and upon which the rest of the frames in the group depend. An example of GOP of size 12 is $IBBPBBPBBPBB$ (see Figure 10.5(a) for an illustration). Because of the inter-frame dependencies, discarding an I frame results in the loss of an entire GOP. Similarly, discarding a P frame results in the loss of the P and B frames that depend on it.

Another complication arising from these inter-frame dependencies is that the client playback order differs from the client decoding order. Figure 10.5 illustrates this key difference between the playback order and decoding order for a sequence of MPEG frames with a GOP of size 12. Both the playback order and the decoding order play a role in determining whether a frame can be played back in time or not. For example, whether a B frame can be played back depends not only on its own playback deadline (which is determined by the playback order), but also on the in-time arrival of its future reference frame (which is

determined by the decoding order).

We now describe how to extend the MINFD algorithm to handle the inter-frame dependencies in MPEG, using a GOP of size 12 as an example. For simplicity of exposition, we assume that *a sequence of MPEG frames are transmitted from the server to the client in their decoding order*. We proceed by considering transmission of all or part of the frames in a GOP at each stage. Note that because the last two B frames in a GOP of size 12 depend not only on the previous P frame, but also on the I frame of the next GOP, these two B frames need to be treated specially: they are *only* included for consideration of transmission in the *next* GOP (after the I frame of the next GOP, on which they depend), but they *are* eligible for consideration of discarding in the current GOP. This special treatment will be further expanded on, as we sketch the extended MINFD algorithm for MPEG video below.

Suppose we have a MPEG video sequence of frames with N GOPs[2] of size 12. At stage i, $i = 1, 2, \ldots, N$, we consider transmission of the i^{th} GOP, $I_i[B_{i-1,7}B_{i-1,8}]P_{i,1}B_{i,1}B_{i,2}P_{i,2}$ $B_{i,3}B_{i,4}P_{i,3}B_{i,5}B_{i,6}(B_{i,7}B_{i,8})$. Here either of the last two B frames from the previous GOP, $B_{i-1,7}B_{i-1,8}$, is included for consideration of possible transmission with the ith GOP *provided* that the following two conditions are met: 1) $i \geq 2$ and its previous reference frame, i.e., the last P frame from the GOP, $P_{i-1,3}$, is included for transmission in the previous GOP at stage $i-1$; and 2) it is not selected for discarding in the previous GOP at stage $i-1$. The last two frames in the current GOP are included here for consideration of discarding only. They will be considered for transmission in the next GOP at stage $i+1$, because their future reference frame, the I frame in the next GOP, has to be transmitted before them.

For the frames included in the ith GOP (excluding $B_{i,7}B_{i,8}$), $I_i[B_{i-1,7}B_{i-1,8}]P_{i,1}B_{i,1}B_{i,2}$ $P_{i,2}B_{i,3}B_{i,4}P_{i,3}B_{i,5}B_{i,6}$, we check to see whether they can be transmitted using the greedy schedule in such a manner that *all of them can be played back in time at the client*. If this is the case, we move to the next stage and consider transmission of the $i+1$th GOP. If this is not the case, then one or more frames must be discarded due to the network bandwidth constraint. Note that no more than 14 frames needed to be discarded at each stage. This is because in the worst case we can always discard all the frames in the current (i.e., the ith) GOP including $B_{i,7}B_{i,8}$ and move to the next stage. Now the question is to find the minimum number of frames, m_i^*, $1 \leq m_i^* \leq 14$, that must be discarded at stage i so that some of the frames in the ith GOP (excluding $B_{i,7}B_{i,8}$) can be transmitted using the greedy schedule in such a manner that *these frames can be played back in time at the client*

[2]The algorithm described below does not require the video sequence to have a fixed GOP pattern.

(we refer to this condition as the *maximum playback constraint*). Furthermore, the m_i^* discarded frames are chosen in such a manner that *the buffer gain at the end of the ith GOP is maximized* (we refer to this condition as the *maximum buffer gain criterion*).

To find m_i^*, we start with $m_i^* = 1$. Among all the frames that have been included for transmission from the previous stages and the frames in the current GOP, we check to see whether we can find a single frame to discard so that the maximum playback constraint can be satisfied. (Frames that are eligible to discard as a single frame include all the B frames and those "isolated" P or I frames whose dependent B and P frames have already been discarded in the previous stages.) If the maximum playback constraint *can* be satisfied by discarding a single frame, we choose to discard the one among all the eligible single frames that meets the maximum buffer gain criterion. If the maximum playback constraint *cannot* be satisfied by discarding a single frame, we proceed to consider $m_i^* = 2$. More generally, consider the case $m_i^* = m$, $1 \leq m \leq 14$. Among all the frames that have been included for transmission from the previous stages and the frames in the current GOP, we check to see whether we can choose *exactly* m frames (including the dependent frames of any chosen frame) to discard so that the maximum playback constraint can be satisfied. If this cannot be done, we proceed to consider $m_i^* = m + 1$. If this can be done, we choose those m frames to discard such that the maximum buffer gain criterion be met. Note that this procedure stops when $m_i^* = 14$, as we can always discard the current GOP to meet the maximum playback constraint. Thus we can always find 14 frames to discard such that the maximum buffer gain criterion is met.

Based on a similar argument as used in Theorem 1, it can be shown that the modified MINFD algorithm described above produces a feasible transmission schedule that minimizes the total number of frames discarded. The time complexity of the algorithm is polynomial in N, the number of GOPs in an MPEG video stream. However, the exponent of the polynomial is in the order of the GOP size.

The modified MINFD algorithm for MPEG video minimizes the total number of frames discarded. A variation of this MINFD algorithm can also be devised using a different metric — one that, first of all, minimizes the total number of I frames discarded, then the total number of P frames discarded, and then the total number of B frames discarded. In other words, a P frame is discarded only if it cannot be included by discarding any number of B frames. Similarly, an I frame is discarded only if it cannot be included by discarding any number of B and P frames. Using this metric, we can ensure that each GOP has at

```
1.  PROCEDURE JITFD(N)
2.    For i = 1 to N
3.      Increment the buffer by â_i
4.      If buffer occupancy > f_i
5.        Decrement the buffer by f_i and display frame i
6.      Else
7.        Discard frame i
8.  END PROCEDURE
```

Figure 10.6: The JITFD selective frame discard algorithm.

least its I frame preserved whenever possible, thereby ensuring certain level of perceived video quality at the client.

10.4 Heuristic Selective Frame Discard Algorithms for JPEG

As mentioned earlier the complexity of the optimal selective frame discard algorithm is $O(BNW)$. For large values of B and N, this can result in very high complexity. In this section we design a set of efficient heuristic algorithms that aim at minimizing the cost associated with the discarded frames. Most of these heuristics are designed based on the MINFD algorithm and hence have a low computational complexity.

Recall that the MINFD algorithm finds the minimum number of frames that must be discarded for a feasible schedule. However it may tend to discard consecutive frames if large frames are clustered together. Hence the *playback discontinuity* at the client may be very high. In order to provide a measure of this *playback discontinuity*, we define a cost function, $\phi(S)$, that takes two aspects of playback discontinuity into consideration: the *length of a sequence of consecutive discarded frames* and the *spacing or distance between two adjacent but non-consecutive discarded frames*.

The cost function $\phi(S)$ assigns a cost c_i to a discarded frame i depending on whether it belongs to a sequence of consecutive discarded frames or not. If frame i belongs to a sequence of consecutive discarded frames, then the cost c_i is defined to be l_i, if frame i is the l_i^{th} consecutively discarded frame in the sequence. Otherwise, the cost c_i is defined based on its distance d_i to the previous discarded frame and given by the formula $c_i = 1 + \frac{1}{\sqrt{d_i}}$. Therefore, for a set $S \in \mathcal{N}$, the total cost of S is $\phi(S) = \sum_{j \in \mathcal{N} \setminus S} c_j$.

Obviously there are many other ways to define a cost function. We believe that the two

aspects of playback discontinuity considered by $\phi(S)$, namely the cost due to consecutive discard and that due to spacing between discarded frames, are important measures of the perceived quality. Any other cost function should reflect these two aspects of playback discontinuity in one way or another. More study[3] is needed in this area to come up with a more realistic cost function based on perceptual quality of video playback [112]. In the rest of this section we will describe a set of heuristic algorithms based on the cost function $\phi(S)$ defined above and results of performance evaluation are then presented. Our algorithms can be easily modified to incorporate the specifics of other cost functions.

10.4.1 Heuristic Algorithms

The heuristic algorithms aim at finding a low cost feasible set S by taking either the cost of discarding a frame directly into consideration or indirectly. They differ in the criteria used in selecting a frame to discard. All the heuristics use the greedy schedule to determine the amount of data to be transmitted in each time slot.

As a simple baseline algorithm, we first introduce the *just-in-time* selective frame discard heuristic, JITFD. JITFD is perhaps the simplest and most intuitive selective frame discard approach. It always discards the *current* frame whenever its playback deadline cannot be met, irrespective of its cost. The algorithm is shown in Figure 10.6. At each time i, the buffer is increased by $\hat{a}_i$ = min($B - B_{i-1}$, C) (line 3), as per the greedy transmission schedule. If the buffer occupancy is smaller than the size of the current frame i, i.e. $B_i + \hat{a}_i < f_i$, the frame is discarded as in lines 4-7. The computational complexity of this algorithm is linear in N.

The *distance* based selective frame discard algorithm, DISTD(λ), uses a parameter λ to indirectly control the cost of discarded frames. The basic structure of the algorithm is the same as the MINFD algorithm. For any given $\lambda \geq 1$, DISTD(λ) attempts to space the discarded frames λ distance apart by incorporating a distance based priority in selecting a frame to discard. The procedure to select a frame to discard is presented in Figure 10.7. At each time i, if the playback deadline of frame i is violated, the procedure is invoked. The procedure finds a frame, k, with highest priority p_k, among all frames selected for transmission since the last buffer full point i_0. Here the priority p_k of a frame is defined

[3]Although a lot of attention has been devoted to development of accurate perceptual models for video encoding [107], unfortunately most of these models are not directly applicable to our study here. We need a relative frame-based perceptual model to design a meaningful cost function as a basis for selective frame discarding.

```
1.  PROCEDURE DISTD_Select(i, λ )
2.     Set k = i; p_k = min(d_k, λ)
3.     For j = i_0 + 1 to i-1
4.        Do not consider j if j ∉ S
5.        p_j = min(d_j, λ)
6.        If p_j > p_k
7.           Set k = j
8.        Else If p_j = p_k and Δ^i_j > Δ^i_k
9.           Set k = j
10. END PROCEDURE
```

Figure 10.7: Procedure to select the frame to discard for DISTD.

based on its distance d_k from the previously discarded frame: $p_k = \min\{\lambda, d_k\}$ (line 2). Hence all frames with a distance at least λ are treated with the same priority. Frames are considered for discarding in the order of decreasing priority. Frames with highest priority, namely, $p_j = \lambda$, are considered first. If such a frame cannot be found, all frames with distance λ -1 are considered, and so forth. Among the frames with the same priority, the frame with the largest gain Δ_k^i is chosen (line 8). Finally, the selected frame k is chosen for discarding only if its gain Δ_k^i is bigger than the size of the current frame, f_i (this criterion is not shown in Figure 10.7). Otherwise, the current frame i is discarded.

The *minimum cost* based selective frame discard algorithm, MINCD, takes the cost of discarding a frame directly into consideration. The procedure for selecting the frame to discard is given in Figure 10.8. At time i, if the playback deadline of frame i is violated, a frame k with lowest incurred cost c_k^i is chosen for discarding. Let S_{i-1} be the feasible set constructed at time $i-1$. The incurred cost c_k^i is defined to be the cost incurred if frame k is discarded at time i, i.e., $c_k^i = \phi(S_{i-1}) - \phi((S_{i-1} \cup \{i\}) \setminus \{k\})$. As shown in lines 3-6, a frame with the smallest incurred cost is chosen for discarding. If two frames have the same incurred cost, the one that yields larger gain Δ_j^i is chosen (lines 7-8).

The last heuristic we consider is the *minimum cost maximum gain* based selective frame discard heuristic, MCMGD. In selecting a frame to discard, it takes both the gain Δ_j^i from discarding a frame and the cost c_j^i incurred thereof into consideration. The procedure for selecting the frame to discard is shown in Figure 10.9. It discards a frame k with the largest gain to the incurred cost ratio, i.e., Δ_j^i/c_j^i (lines 5-6). By discarding frames with the largest gain to cost ratio, the MCMGD heuristic uses in effect the steepest gradient search for an optimal solution.

```
1.  PROCEDURE MINCD_Select(i)
2.    Set k = i
3.    For j = i_0 + 1 to i-1
4.      Do not consider j if j ∉ S_i
5.      If c_j^i < c_k^i
6.        Set k = j
7.      Else If c_j^i = c_k^i and Δ_j^i > Δ_k^i
8.        Set k = j
9.  END PROCEDURE
```

Figure 10.8: Procedure to select the frame to discard for MINCD.

```
1.  PROCEDURE MCMGD_Select(i)
2.    Set k = i
3.    For j = i_0 + 1 to i-1
4.      Do not consider j if j ∉ S_i
5.      If Δ_j^i/c_j^i > Δ_k^i/c_k^i
6.        Set k = j
7.  END PROCEDURE
```

Figure 10.9: Procedure to select the frame to discard for $MCMGD$.

The computational complexity of the DISTD, MINCD and MCMGD heuristics is $O(N^2)$. This is much smaller than the computational complexity of the optimal algorithm OPTFD.

10.4.2 Performance Evaluation

In this section we evaluate the performance of the heuristic selective frame discard algorithms using JPEG video traces. For given bandwidth and client buffer size constraints, the number of frames discarded and the cost incurred by these algorithms are compared. The cost is computed using the heuristic cost function defined earlier. The impact of each constraint on the performance of these algorithms is also studied by varying one constraint

Title	Length (min)	No. of Frames	Ave. Rate (Mbps)	Peak Rate (Unsmoothed)	Peak Rate (Smoothed)
Sleepless in Seattle	101	181457	2.28	3.99	3.30
Beauty and Beast	80	143442	3.04	7.29	6.54
Jurassic Park	122	220061	2.73	5.73	4.78

Table 10.2: Characteristics of JPEG video traces

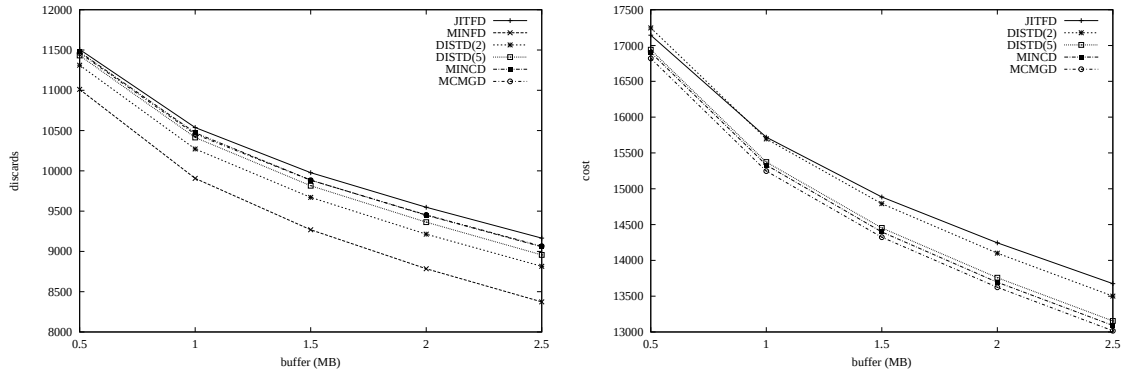
Selective Frame Discard Algo	Sleepless in Seattle		Beauty and the Beast		Jurassic Park	
	Discards	Cost	Discards	Cost	Discards	Cost
JITFD	10538	15720.21	8778	13355.89	13457	20269.55
DISTD(2)	10272	15696.25	8602	13370.69	13141	20262.52
DISTD(5)	10414	15372.89	8692	13196.46	13294	19909.13
MINCD	10473	15332.03	8742	13170.24	13384	19845.25
MCMGD	10455	15246.31	8712	13130.756	13342	19746.68
MINFD	9907	128797.77	8183	106951.31	12516	148921.86

Table 10.3: Comparison of various selective frame discard algorithms for JPEG.

while keeping the other constraint fixed. We present the results for three representative traces, *Sleepless in Seattle*, *Beauty and the Beast* and *Jurassic Park*. Table 10.2 lists the characteristics of these traces [22], where among other things, the average rate, the peak rate of the video traces are shown. Also included is the peak rate of the optimal smoothed schedule [97] using a client buffer size of 1 MB and zero startup delay.

Table 10.3 compares the performance of various selective frame discard algorithms. The rate constraint C in each case is set to the average rate of the video trace, while the client buffer size B is set to 1 MB. As shown in Table 10.2, the peak rate of the optimal smoothed schedule is considerably higher than the chosen rate constraint. Hence continuous playback is not possible, forcing the server to discard frames. Consider the performance of the heuristic algorithms when applied to the video trace *Sleepless in Seattle*. JITFD discards 10538 frames with a cost of 15720.21. DISTD(2) drops 10272 frames, while DISTD(5) drops 10414 frames, larger than that of DISTD(2). However, the cost of DISTD(5) is 15372.894, lower than that of DISTD(2), which is 15696.250. This is due to the fact the discarded frames in DISTD(5) are more distributed than those of DISTD(2), incurring a lower cost despite a larger number of discarded frames. For the same trace, MINCD discards 10473 frames with a cost of 15332.03, and MCMGD incurs a cost of 15246.31 by discarding 10455 frames. All the heuristic discard schemes that take cost into consideration incur less cost than JITFD does. Among them, MCMGD performs best, as expected. We also ran the optimal algorithm[4], OPTFD, on this trace. It discarded 10455 frames with a cost of 15245.8. We see that the results produced by MCMGD are near optimal. It is also worth pointing out that MINFD indeed gives the lowest number of discards. However, the

[4]In order to speed up the execution of OPTFD and reduce the memory space, a branch and bound strategy is used to control the number of states by pruning states which are unlikely to lead to an optimal cost. Hence the results produced may not be completely accurate, but we believe they give a good approximation to the true optimal values.



(a) buffer size *vs.* num of discarded frames

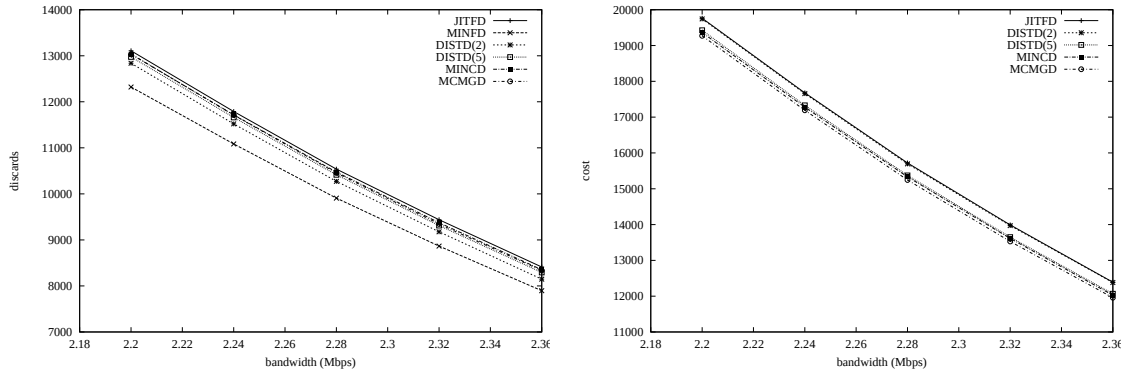
(b) buffer size *vs.* cost

Figure 10.10: Performance under varying buffer sizes with C fixed at 2.28 Mbps for *Sleepless in Seattle*

incurred cost is quite high as it tends to discard consecutive large frames. Clearly, there is a trade-off between reducing the total number of discarded frames and distributing discarded frames in a video stream.

We now study the impact of varying buffer size while fixing the rate constraint on the performance of the selective frame discard algorithms. Figure 10.10 shows the number of discarded frames as well as the incurred cost as a function of buffer size for the trace *Sleepless in Seattle*. The bandwidth C is fixed at 2.28 Mbps, and the client buffer size B is increased from 0.5 MB to 2.5 MB. It can be seen that all the other four heuristic algorithms perform better than JITFD. The difference in performance among the heuristics widens as the buffer size increases. This phenomenon can be explained as follows. Recall that frames which come before a buffer full point are not considered for discarding for a deadline violation after the buffer full point. Hence with increased buffer size, the number of frames from which a frame can be selected for discarding increases. It therefore enhances the effectiveness of the selection criteria used in the heuristics such as MINCD and MCMGD. Among all the heuristics, it is quite evident that MCMGD performs best at all buffer sizes.

Figure 10.11 shows the impact of bandwidth variation for the trace *Sleepless in Seattle*. The bandwidth is varied from 2.96 Mbps to 3.12 Mbps with the client buffer size fixed at 1 MB. As the bandwidth increases, the difference in performance between the JITFD and the other four heuristic algorithms narrows slightly. This is because at a higher bandwidth, the playback deadline of fewer frames are violated. As a result, discarded frames



(a) bandwidth *vs* num of discarded frames

(b) bandwidth *vs* cost

Figure 10.11: Performance under varying bandwidth with B fixed at 1MB for *Sleepless in Seattle*

are more likely to be distributed and the advantage of more sophisticated heuristics is less pronounced. The MCMGD algorithm still has the best performance across the bandwidth range,

We have run the heuristic algorithms on other JPEG traces. The results obtained are very similar. We conclude that the proposed heuristic algorithms work well in improving the perceived quality as measured by the proposed cost function. Among them, the MCMGD heuristic has the best performance.

10.5 Heuristic Selective Frame Discard Algorithms for MPEG

In this section, we extend the heuristic algorithms developed for JPEG encoded video to handle inter-frame dependencies for MPEG encoded video. The evaluation based on MPEG video traces is then presented.

The following modifications are made to the selective frame discard heuristics in order to take the MPEG inter-frame dependencies into account. If the current frame is a P or B frame, it is discarded if its previous reference frame is not included in the transmission schedule. In addition, a B frame is also discarded if its future reference frame cannot be transmitted before the playback deadline of the current B frame.

The above modifications are the only changes needed for the *just-in-time* (JITFD) heuristic. For the *minimum cost* (MINCD) and *min-cost-max-gain* (MCMGD) heuristics, an addi-

Title	Length (min)	I Frames	P Frames	B Frames	Total Frames	Avg. Rate (Mbps)	Peak Rate (Mbps)
Starwars	121	14505	43514	116036	174055	0.374	4.446

Table 10.4: Characteristics of *Starwars* MPEG video trace.

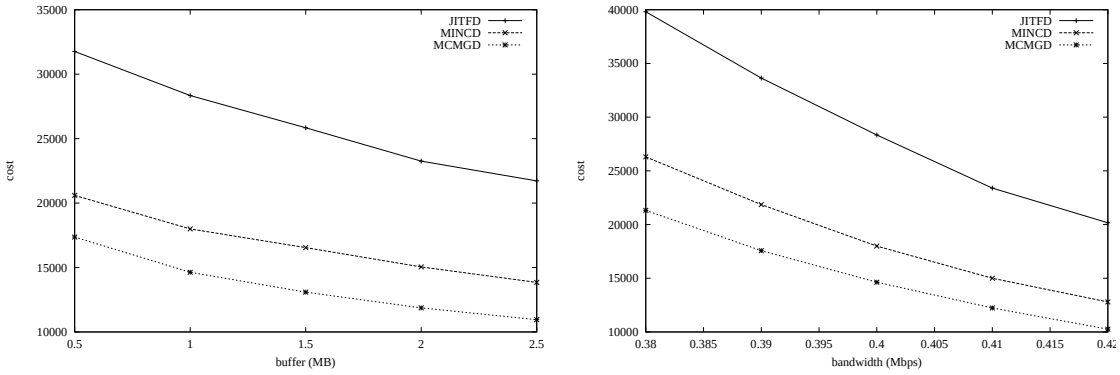
Selective Frame Discard Algo	Discarded Frames				Cost
	I	P	B	Total	
JITFD	210	827	5924	6961	31761.86
MINCD	1	18	11948	11967	20597.93
MCMGD	1	18	9860	9879	17364.47

Table 10.5: Comparison of the various selective frame discard algorithms for MPEG.

tional modification is incorporated to attach a relative importance to I, P and B frames. In deciding a frame to discard, frames are considered in the following order of relative importance: B, P_3, P_2, P_1, I. In other words, eligible frames of types B are always considered first. If no eligible frames of type B are left, eligible frames of type P_3 are considered, and so forth. As described in Section 10.4, MINCD discards a frame with lowest incremental cost, while MCMGD chooses a frame with largest ratio of gain and incremental cost. Similar extensions can also be incorporated in the distance-based (DISTD) heuristic, where the distance between frames is now defined for each type of frames. For clarity of exposition, we do not include the DISTD heuristic in the following performance evaluation.

The performance of the JITFD, MINCD and MCMGD heuristics is evaluated and compared in Table 10.5 using the *Star Wars* MPEG video trace. The statistics of the MPEG trace is listed in Table 10.4. The rate constraint C is set to 0.4 Mbps and the client buffer size B is fixed at 0.5 MB. From Table 10.5, we see that JITFD performs significantly worse than MINCD and MCMGD, unlike the case of JPEG. This is due to the fact that JITFD does not take into account the frame dependencies and the relative importance of frame types. Hence it is prone to discard a large number of I frames, incurring a much higher cost, as compared to that of MINCD and MCMGD. Whereas MINCD and MCMGD attempt to minimize the chance of discarding I and P frames by considering frame types in the order of their relative importance. MCMGD discards fewer B frames than MINCD, while the same number of I and P frames are discarded by both heuristics. MCMG performs better than MINCD, because it takes both cost and gain into consideration.

The impact of varying client buffer size and network bandwidth on the performance of



(a) buffer size *vs.* cost (C = 0.4 Mbps)

(b) bandwidth *vs.* cost (B = 1 MB)

Figure 10.12: Performance under (a) varying buffer sizes (b) varying bandwidth for *Star-wars*

these heuristics is shown in Figure 10.12 (a) and (b) respectively. In both cases, as the client buffer size or network bandwidth increases, the cost for each heuristic decreases. This is because with larger buffer or higher network bandwidth, the likelihood of discarding I or P frames is reduced, thereby reducing the overall cost. We have observed similar results for other MPEG traces.

10.6 Summary

In this chapter, we have developed various selective frame discard algorithms for stored video delivery across a network where both the network bandwidth and the client buffer capacity are limited. We began by formulating the problem of optimal selective frame discard using the notion of a cost function. The cost function captures the perceived video quality at the client. Given network bandwidth and client buffer constraints, we developed an $O(N \log N)$ algorithm to find the minimum number of frames that must be discarded in order to meet these constraints. The correctness of the algorithm is also formally established. We presented a dynamic programming algorithm for solving the optimal selective frame discard problem. Since the computational complexity of the optimal algorithm is prohibitively high in general, we also developed several efficient heuristic algorithms for selective frame discard. These algorithms are evaluated using JPEG and MPEG video traces. We found that the *minimum cost maximum gain* algorithm performs best for both JPEG and MPEG encoded video. The basic ideas presented here can be extended to layered

video also. In the next chapter, we develop layer selection schemes that in effect selectively discard frames of layers in an attempt to provide smoother quality video stream.

Chapter 11

Layer Selection Algorithms

The selective frame discard algorithms presented in the previous chapter essentially adjust the frame rate of a video by selectively playing frames and thus effectively reducing the bandwidth required for its playback. But this provides only a coarser control on perceived quality. Furthermore, these algorithms are not aware of the content of video played and thus do not take into account the visual importance of each frame. Layered encoding is proposed to provide finer control on video quality. Layered encoding separates the video signal into components of differing visual importance: the base layer contains coarser details while upper layers contain increasingly finer details of the video. A prefix of these layers is chosen such that the resource constraints are met [59]. However it is not a trivial task to select layers such that better but consistent quality playback is ensured when the network conditions are constantly varying.

In this work, we address the layer selection problem in stored layered video delivery and show how smoother[1] quality video playback can be provided by utilizing the client buffer for prefetching. We first define smoothness criteria, design metrics for measuring it, and then develop off-line algorithms to maximize smoothness for the case where the network bandwidth is varying but known *a priori*. We then describe an adaptive algorithm for providing smoothed layered video delivery that doesn't assume any knowledge about future bandwidth availability. The results of our experiments for measuring and comparing the performance these schemes are then presented. We conclude the chapter with a brief discussion on our future work.

[1] Here smoothing refers to the perceived video quality, not bandwidth.

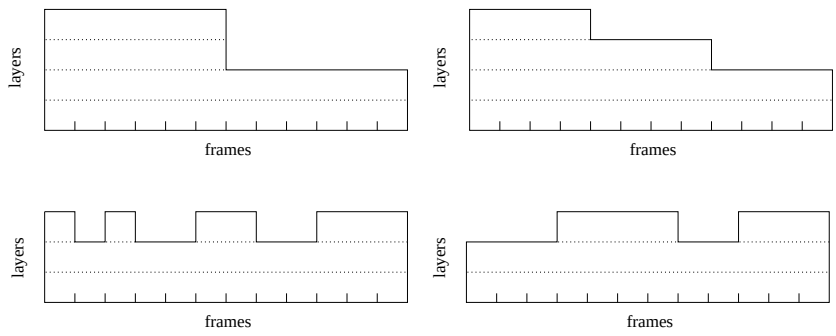


Figure 11.1: Illustration of smoothness criteria

11.1 Smoothness Criteria and Quality Metrics

One of the problems in assessing the performance of a video delivery scheme is the lack of a good metric that captures the user's perception of video quality. In general, the higher the amount of detail in the played video, the better is its quality. However, it is generally agreed that it is visually more pleasing to watch a video with consistent, albeit lower, quality than one with highly varying quality. Thus, a good metric should capture both the amount of detail per frame as well as its uniformity across frames.

Consider the example sequences of layers in a video stream shown in Figure 11.1. The top two streams consume the same amount of network resources, as the bottom two streams do. However, the sequences on the right are "smoother" than the ones on the left. In the first case, the degradation in quality is more gradual in the "smoother" sequence. In the second case the "smoother" sequence has fewer changes in quality. These smoothness criteria can be captured in a metric by giving higher weight to lower layers and to longer runs[2] of continuous frames in a layer.

We propose a family of metrics that attempt to measure the smoothness in the perceived quality of a layered video. They represent the smoothness, M, of a video stream by a vector of the form $M = (m_1, m_2, \ldots, m_L)$, where L is the total number of layers and m_i is the measure associated with layer i. Using these metrics, two video streams can be compared by using lexicographic ordering of the corresponding vectors. A stream with smoothness measure M^1 is considered better than a stream with M^2 if $\exists i, m_j^1 = m_j^2, j < i$ and $m_i^1 > m_i^2$. In other words, the stream with a higher measure at a lower layer is considered smoother.

[2]Hereafter *run* refers to a sequence of consecutive frames shown in a layer.

Each metric in the family computes its m_i differently. We consider three such metrics in our work: *avgrun*, *minrun*, *exprun*. The *avgrun* metric measures the mean length of a run in layer i and *minrun* its minimum run. The *exprun* metric measures the expected run length in layer i. In other words, the *exprun* metric gives the run length in layer i that can be expected to be seen around an arbitrary frame in the layer.

These metrics can be computed as follows:

$$avgrun = \frac{n_1+n_2+...+n_k}{k}$$
$$minrun = \min\{n_1, n_2, \ldots, n_k\}$$
$$exprun = \frac{n_1^2+n_2^2+...+n_k^2}{N}$$

where k is the total number of runs in a layer and $n_1, n_2, \ldots, n_k$ are the lengths of these runs. These values are then normalized by the length of the video sequence and presented as a value between 0 and 1.

The following table lists the smoothness of each sequence in Figure 11.1, as measured by each of the above metrics. It is assumed that the length of the video is 12 frames.

	top left	top right
avgrun	$(1.00, 1.00, 0.50, 0.50)$	$(1.00, 1.00, 0.67, 0.33)$
minrun	$(1.00, 1.00, 0.50, 0.50)$	$(1.00, 1.00, 0.67, 0.33)$
exprun	$(1.00, 1.00, 0.25, 0.25)$	$(1.00, 1.00, 0.44, 0.11)$
	bottom left	bottom right
avgrun	$(1.00, 1.00, 0.15)$	$(1.00, 1.00, 0.30)$
minrun	$(1.00, 1.00, 0.08)$	$(1.00, 1.00, 0.25)$
exprun	$(1.00, 1.00, 0.10)$	$(1.00, 1.00, 0.17)$

Consider for example the bottom left sequence in Figure 11.1. The total number of runs is $1, 1, 4$ respectively for the layers $1, 2, 3$. For layers 1 and 2 the average run length and minimum run length is same as the length of the video. Hence their normalized measures are each 1.0. The layer 3 has 4 runs of length $1, 1, 2, 3$ respectively. Thus the average run length is 1.75 and the corresponding *avgrun* measure is $\frac{1.75}{12} = 0.145$. Similarly the minimum run length is 1 and the *minrun* measure is $\frac{1}{12} = 0.083$. The expected run length of layer 3 is $\frac{1^2+1^2+2^2+3^2}{12} = 1.25$ and hence the corresponding *exprun* measure is $\frac{1.25}{12} = 0.104$. As can be seen, the relative smoothness of the sequences in Figure11.1 is reflected by each of the metrics in the above table.

The metrics *avgrun* and *minrun* are easy to understand and each measure a different statistic on the runs present. But they fail to take into account the absence of runs. For example, given a sequence of runs of a layer, by dropping all the runs except the longest run we can generate a new sequence with larger *avgrun* and *minrun* values. Such a new sequence may not necessarily be perceptually better than the original sequence. In order to address this we also need to consider the absence of a frames in a run. The *exprun* metric captures this notion by taking both the sum of all runs and the length of each individual run into account.

We now proceed to formulate the layer selection problem in video delivery and develop algorithms that choose layers such that the smoothness of the resulting sequence is maximized as measured by these metrics.

11.2 Problem Formulation

The objective of a layer selection scheme is to optimize the perceived video quality, as measured by metrics described earlier, given the resource constraints. In formulating this problem, we consider a discrete-time model at the frame level. Each time slot represents the unit of time for playing back a video frame. For simplicity of exposition, we assume startup delay of one slot, i.e., the server starts video transmission one time slot ahead of the time the client starts playback. In other words, server starts transmission at time 0 and the client starts displaying the frame 1 at time 1. We also ignore the network delay. Table 11.1 summarizes the notation we introduce in this section.

Consider a video stream with N frames and L layers. The size of j^{th} layer of i^{th} frame is denoted by f_i^j. Let $\hat{C}^j = (\hat{C}_0^j, \hat{C}_1^j, \ldots, \hat{C}_N^j)$ and $\hat{B}^j = (\hat{B}_0^j, \hat{B}_1^j, \ldots, \hat{B}_N^j)$, where $\hat{C}_i^j$ denote network bandwidth and $\hat{B}_i^j$ client buffer capacity during time slot i at a layer j. Let $S = (S_1, S_2, \ldots, S_N), 0 \leq S_i \leq L$ be a layer sequence.

Let $D(S) = (D_0(S), D_1(S), \ldots, D_N(S))$ where $D_i(S) = \sum_{k=0}^{i} \sum_{j=0}^{S_i} f_k^j$, and let $U(S) = (U_0(S), U_1(S), \ldots, U_N(S))$ where $U_i(S) = D_i(S) + \hat{B}_i^1$. We refer to $D(S)$ as the *(client) buffer underflow curve with respect to* S, and $U(S)$ as the *(client) buffer overflow curve with respect to* S. A server transmission schedule $A(S)$ associated with S is a schedule which only transmits layers of frames included in S, namely, layer j of frame i is transmitted under $A(S)$ if and only if $j \leq S_i$. Let $a_i(S)$ be the amount of video data transmitted during time slot i, $i = 1, \ldots, N$[3]. The schedule $A(S)$ is denoted by $A(S) =$

[3]When optimizing *avgrun* and *minrun* metrics, to account for the absence of runs, we need to add an

Table 11.1: Notation

N	:	length of video in frames
L	:	number of layers of video
f_i^j	:	size of j^{th} layer of i^{th} frame
B_i^j	:	buffer occupancy by layer j at slot i
$\hat{B}_i^j$	:	buffer constraint during slot i layer j
$\hat{C}_i^j$	:	bandwidth constraint during slot i layer j
S	:	a layer sequence
$\hat{S}$	:	an optimal feasible layer sequence
$A(S)$	:	a transmission schedule w.r.t. sequence S
$A_i(S)$	:	cumulative data sent by server over $[1, i]$
$a_i(S)$	:	amount of data sent by server in slot i
$D(S)$	:	underflow curve w.r.t sequence S
$D_i(S)$	:	cumulative data consumed by the client over $[1, i]$
$U(S)$	:	overflow curve w.r.t sequence S
$U_i(S)$	:	maximum cumulative data that can be received by the client over $[1, i]$

$(A_0(S), A_1(S), \ldots, A_N(S))$ where $A_0(S) = 0$ and $A_i(S) = \sum_{k=0}^{i} a_k(S)$.

A server transmission schedule $A(S)$ is said to be *feasible* with respect to S if and only if for $i = 0, 1, \ldots, N$, 1) *rate constraint is not violated*, i.e., $a_i(S) \leq C_i^1$; 2) *buffer constraint is not violated*, i.e., $A_i(S) \leq U_i(S)$; and 3) *playback constraints are not violated*, i.e, $D_i(S) \leq A_i(S)$. For a given rate and buffer constraints $(\hat{C}, \hat{B})$, we denote the collection of all feasible layer sequences by $\mathcal{FLS}(\hat{C}, \hat{B})$.

Now the optimal layer selection problem can be stated as follows: find a feasible sequence $\hat{S}$ that maximizes the associated metric *quality*($\hat{S}$), formally

> *Find a sequence* $\hat{S}$ *such that* $\hat{S} \in \mathcal{FLS}(\hat{C}, \hat{B})$ *and* $quality(\hat{S}) = \max\{quality(S) : S \in \mathcal{FLS}(\hat{C}, \hat{B})\}$.

11.3 Optimal Layer Selection

In this section, we discuss the potential approaches to designing optimal layer selection algorithms for maximizing each of the metrics defined earlier. We first make some simplifying assumptions about the problem setting. We then describe a generic layer selection

additional constraint that a transmission schedule has to be work-conserving, i.e., $a_i(S) = min(\hat{B}_i^1 - \sum_{j=1}^{L} B_i^j, \hat{C}_i^1)$

```
1.  PROCEDURE OPTLAYERS(Ĉ, B̂)
2.     Initialization : Ĉ^1 = Ĉ, B̂^1 = B̂
3.     For j = 1 to L
4.        (Ĉ^{j+1}, B̂^{j+1}, Ŝ^j) = maxanyrun(Ĉ^j, B̂^j)
5.  END PROCEDURE
```

Figure 11.2: The generic optimal layer selection procedure

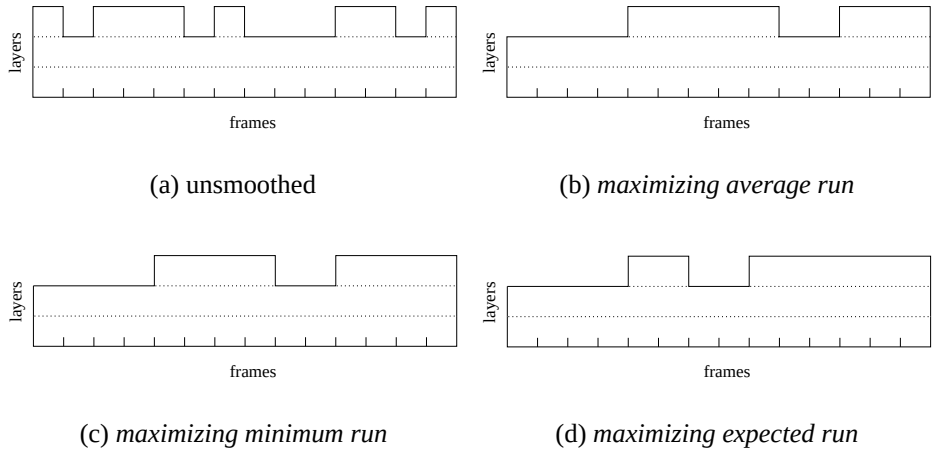
procedure which uses a metric-specific procedure for selecting frames within a layer. The frame selection procedures for these metrics namely, MAXAVGRUN, MAXMINRUN, and MAXEXPRUN are then presented.

We assume that each layer in the video is of CBR, i.e., all frames in a layer are of the same size. This enables us to maximize the given metric for each layer in isolation. For simplicity of exposition, we further assume that all layers of equal bit rate[4]. Now, without loss of generality, all the units can be scaled such that size of each frame in a layer is 1, i.e., $f_i^j = 1$ and buffer and bandwidth values are all integers.

The generic optimal layer selection procedure shown in Figure 11.2 can be described as follows. Consider each layer j from lower to higher starting with layer 1. Select an optimal subset, $\hat{S}^j$, as measured by the metric, of all frames at that layer j treating it as the only layer of the video. A metric-specific procedure is used in place of *maxanyrun* to perform the selection of frames in a layer. This optimal selection has to satisfy the resource constraints given by the residual bandwidth, $\hat{C}^j$ and residual buffer, $\hat{B}^j$ that is available after considering all the layers up to $j-1$. Since it is always preferable, according to our metrics, to select lower layer frames and also because all frames are of equal size, with optimal selection of frames at each layer, the resulting layer sequence would also be optimal.

The Figure 11.3 shows an example unsmoothed sequence and the corresponding smoothed sequences that would result after applying the MAXAVGRUN, MAXMINRUN, and MAXEXPRUN algorithms. The maximum available buffer is assumed to be 3. While the unsmoothed sequence has 5 runs, all the smoothed sequences have only 2 runs and thus longer runs. However, the run lengths are different in each case so as to optimize the respective metrics. In the following subsections we explain these algorithms in detail.

[4]Note that the algorithms described here do not hinge on this assumption



(a) unsmoothed (b) *maximizing average run*

(c) *maximizing minimum run* (d) *maximizing expected run*

Figure 11.3: Optimal layer sequences: (a) buffer=0; (b)(c)(d) buffer=3

11.3.1 Maximizing the average run length

The average run length in a sequence can be maximized by minimizing the number of runs while keeping the sum of all the runs as high as possible. We propose MAXAVGRUN algorithm that achieves this by delaying the start of a run as late as possible and stretching its end as far as possible. Intuitively, a new run is not initiated unless the buffer is accumulated adequately and it is not terminated until the buffer is drained completely.

The Figure 11.4 shows the details of this algorithm operating at a layer j. It consists of two scans across the length of the video: one in forward direction (lines 2-11) and one in backward direction (lines 13-23). The forward scan can be viewed as a step that identifies the end of each run and the minimal number of runs. The backward scan essentially extends each run towards the front of it while at the same time maximizing the residual buffer made available to higher layers.

The algorithm during a forward scan switches between *select phase* in which frames are selected (line 6) and *discard phase* in which frames are discarded (line 8). It starts with empty buffer and in discard phase (line 2). In each slot, the buffer occupancy, B_i^j is updated (line 4) such that bandwidth constraint is not violated. It enters select phase if the buffer is full even after consuming a frame (line 5) and switches to discard phase if the buffer is empty even before consuming a frame (line 7). If the buffer occupancy stays within the bounds, the same phase is continued (lines 9-10). The current frame is either selected or discarded based on the current phase (line 11).

At the end of a forward scan it is possible that accumulated buffer is not completely drained. Furthermore, before each run the buffer may be filled up too early and hence rendered unusable by higher layers. The backward scan addresses these concerns by accumulating the buffer as late as possible and only when needed. It keeps track of the buffer requirement at a frame i, $\check{B}_i^j$, for the selected frames beyond i in future. Since no buffer is required beyond frame N, it is initialized to 0 (line 13). The current frame is selected whenever the current buffer occupancy is more than the future buffer requirement (lines 15-16). The residual buffer available for layer $j+1$ is set by subtracting the buffer required for layer j from the the total amount available to it (line 17). If enough bandwidth is available at slot i to satisfy the future buffer requirements of layer j, then some residual bandwidth is made available to layer $j+1$ and the buffer requirement beyond slot $i-1$ is also adjusted accordingly (lines 18-23). Thus by filling the buffer closer to where it is consumed, larger residual buffer is made available to higher layers and longer runs are made possible at each layer.

Figure 11.5 shows the MAXAVGRUN algorithm in operation at layer 3 on the example unsmoothed sequence in Figure 11.3(a) while generating the smoothed sequence in Figure 11.3(b). The top curve represents the rate constraint corresponding to the residual bandwidth available at layer 3 as given by unsmoothed sequence. The bottom curve gives the buffer constraint which is essentially the rate constraint shifted down by the buffer size which is 3 in this example. It is assumed that data transmitted in a slot is added to the buffer only at the end of the slot. The middle curve is the consumption curve corresponding to the frames selected by the MAXAVGRUN procedure. The crossover of consumption curve and the buffer constraint curve implies a buffer overflow while that of consumption and rate constraint curves means a buffer underflow.

The MAXAVGRUN algorithm, as described earlier, starts in discard phase and continues dropping frames till the 5^{th} frame where upon it enters select phase. Otherwise, i.e, if we had dropped frame 5 also, it would have caused a buffer overflow. Once in select phase it continues selecting frames up to 9 at which point it is forced to switch to discard phase lest buffer would underflow. It then selects another run of frames 12-14, even before the overflow point, during the backward scan.

We now illustrate the operation of the MAXAVGRUN algorithm using an example. Consider a layered video clip of length 20 frames with 3 layers. Let us assume that client can buffer up to 2 units of the video. Suppose that the curve given in Figure 11.6(b) represents

```
1.  PROCEDURE MAXAVGRUN(Ĉ^j, B̂^j)
2.    Initialization : B_0^j = 0, Θ = 0
3.    For i = 1 to N
4.      Update buffer: B_i^j = B_{i-1}^j + Ĉ_i^j − Θ
5.      If B_i^j > B̂_i^j, i.e., buffer overflow?
6.        Select phase: Θ = 1; B_i^j = B̂_i^j
7.      Else If B_i^j < 0, i.e., buffer underflow?
8.        Discard phase: Θ = 0; B_i^j = 0
9.      Else
10.       Continue same phase
11.     Note frame status: Ŝ_i^j = Θ
12.
13.   Initialization : B̌_N^j = 0
14.   For i = N to 1
15.     If B_i^j > B̌_i^j, i.e., buffered enough?
16.       Select frame: Ŝ_i^j = 1
17.     Residual buffer limit: B̂_i^{j+1} = B̂_i^j − B̌_i^j
18.     If Ĉ_i^j > B̌_i^j + Ŝ_i^j, i.e., bandwidth enough?
19.       Residual bandwidth: Ĉ_i^{j+1} = Ĉ_i^j − B̌_i^j − Ŝ_i^j
20.       Buffer not needed for future: B̌_{i-1}^j = 0
21.     Else
22.       Adjust future buffer: B̌_{i-1}^j = B̌_i^j − Ĉ_i^j + Ŝ_i^j
23.       No residual bandwidth: Ĉ_i^{j+1} = 0
24. END PROCEDURE
```

Figure 11.4: The algorithm for maximizing average run length

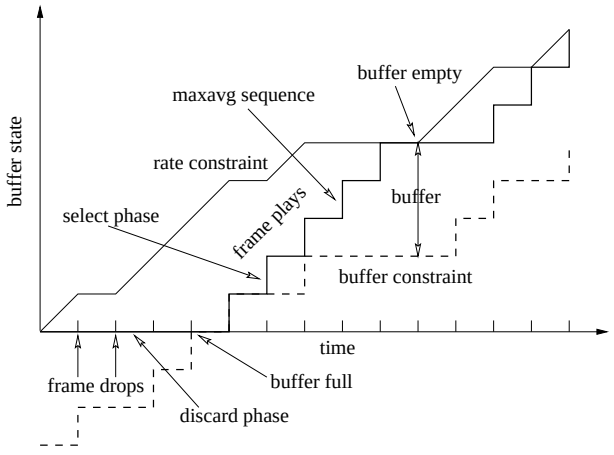
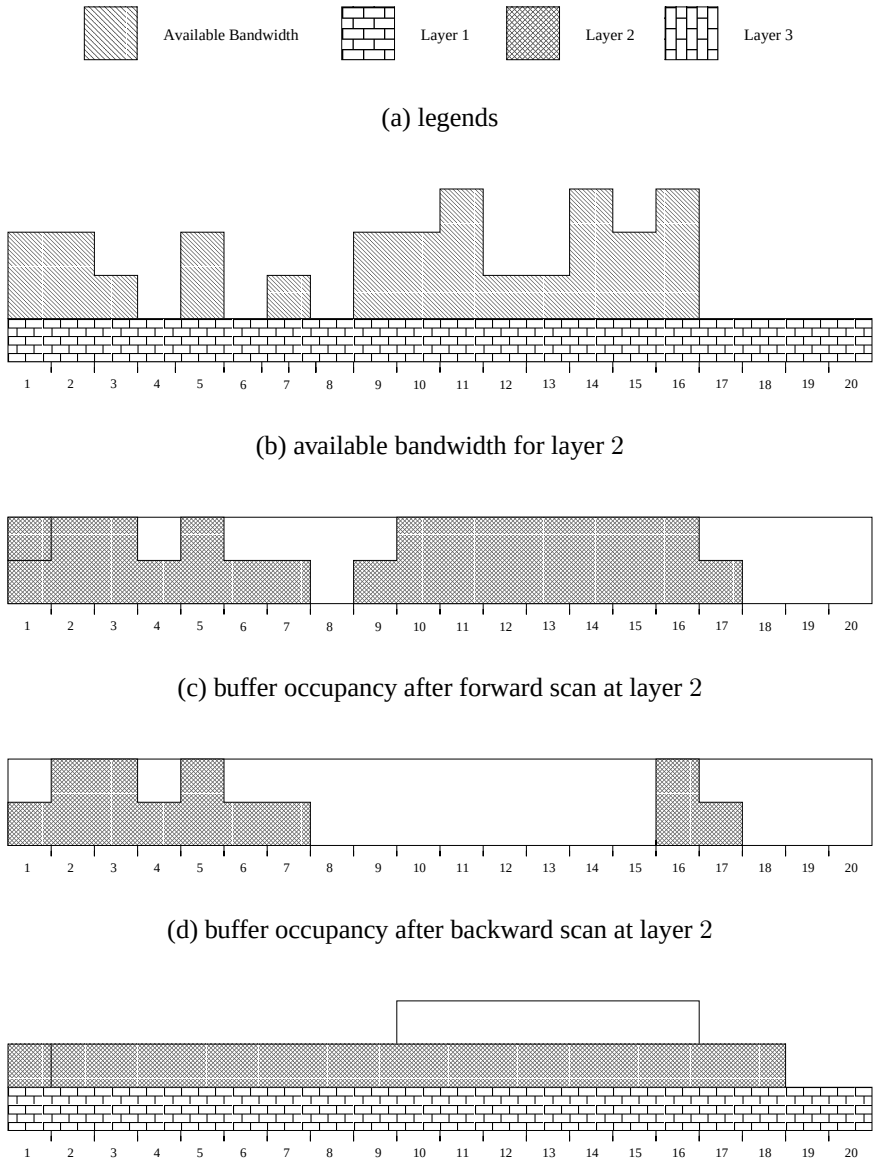


Figure 11.5: Illustration of MAXAVGRUN sequence



(a) legends

(b) available bandwidth for layer 2

(c) buffer occupancy after forward scan at layer 2

(d) buffer occupancy after backward scan at layer 2

(e) selected frames up to layer 2

Figure 11.6: Illustration of the operation of MAXAVGRUN at layer 2

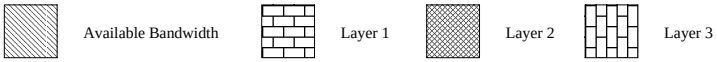
the available bandwidth in each frame slot. The bandwidth used to transmit layer 1 and the residual bandwidth after processing layer 1 are marked differently in the figure. Since the available bandwidth in each slot is greater than 1, the layer 1 can be delivered completely without using any buffer for prefetching.

Figure 11.6(c) shows the buffer occupancy in each frame slot as computed during the forward scan at layer 2. In slot 1, it uses the 2 units of bandwidth available for building the buffer for layer 2. In slot 2, since the buffer is full and bandwidth is available, it initiates a run and starts selecting frames. The run is terminated at slot 18 beyond which it can not be continued due to lack of bandwidth and buffer size limitation. In this scan, the buffer is filled whenever additional bandwidth available and kept as full as possible.

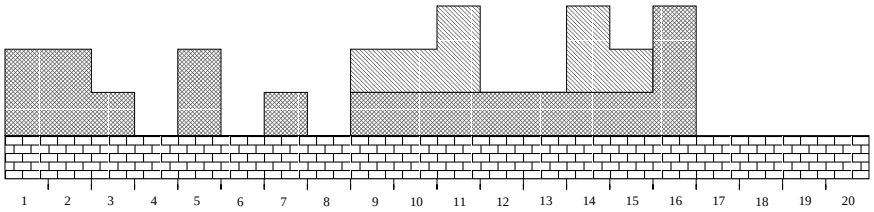
There are two improvements possible to the sequence generated by forward scan. First, the length of each run may be extended at the front since in forward scan buffer might have been built up earlier than necessary. In this example, the run of layer 2 can be started at slot 1 itself instead if at slot 2. Second, larger residual buffer can be made available to higher layers by buffering frames of a layer only when needed. The layer 2 requires 2 units of buffer only at the end of slot 16 which can be built up in that slot instead of building it too early by slot 10. This would increase the residual buffer for layer 3. This is precisely what is done during a backward scan. The buffer occupancy and the selected layer sequence after the backward scan on layer 3 are shown in Figures 11.6(d) and 11.6(e) respectively. The bandwidth utilized by layer 2 and the residual bandwidth for layer 3 are shown in Figure 11.7(b).

Figures 11.7(c) and 11.7(d) give the buffer occupancy at the end of forward scan and backward scan respectively on layer 3. Once again the run of layer 3 is extended at the front during the backward scan. It can also be seen that by making larger amount of residual buffer available to layer 3, its run is continued for longer period. Otherwise, without the backward scan on layer 2, layer 3 would have two shorter runs of length 1 and 3 instead of a single run of length 7. The final resulting layer sequence given by MAXAVGRUN algorithm is shown in Figure 11.7(b).

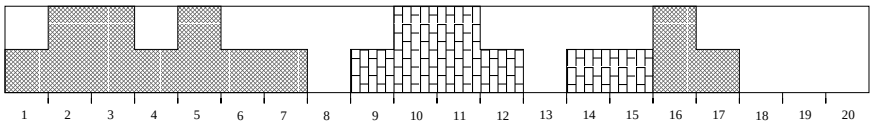
It is quite obvious from the description and illustration that this algorithm ensures that neither buffer nor bandwidth constraint is violated at each slot. Hence it selects only feasible subset of frames. Furthermore, since during the forward scan a run is started only after the buffer is full and it is ended only when the buffer is empty, the length of is each run (except the very last run) is guaranteed to be at least buffer size. The backward scan attempts to



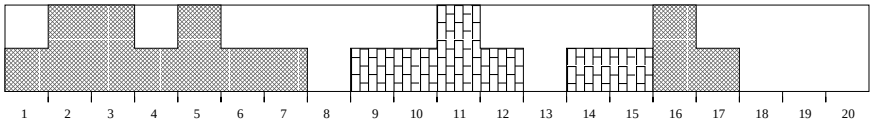
(a) legends



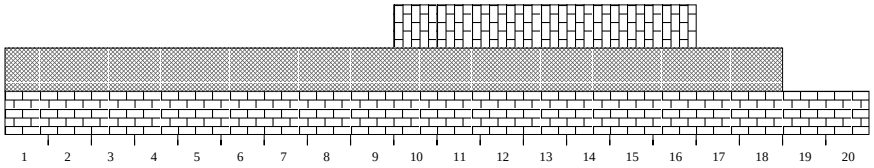
(b) available bandwidth for layer 3



(c) buffer occupancy after forward scan at layer 3



(d) buffer occupancy after backward scan at layer 3



(e) selected frames up to layer 3

Figure 11.7: Illustration of the operation of MAXAVGRUN at layer 3

further extend the length of each run. It can be easily shown that MAXAVGRUN algorithm minimizes the number of runs and hence maximizes the average run length.

11.3.2 Maximizing the minimum run length

The algorithm described above yields the minimum number of runs with the length of each run being at least the size of buffer. However there could be some variation in relative lengths of these runs. One way to maximize the minimum run length would be to reduce the variance among the runs. In other words, we can pull up the minimum run length in the overall sequence by growing the shorter runs while shrinking their longer neighboring runs.

Given a MAXAVGRUN sequence, it is not possible to grow a run at the end since the buffer is empty at that point. However a run can be made longer by selecting a chunk of frames just before the run and discarding an equal number of frames from the end of a previous run. The extent of growth is limited obviously by the maximum buffer and also the buffer accumulation pattern right before the run. It should be noted that though a sequence with higher than the minimum number of runs may also have the largest minimum run length, it is possible to find a sequence that has the same minimum run length but with lesser number of runs. So starting with a MAXAVGRUN sequence we can find a MAXMINRUN sequence by readjusting the lengths of each run without increasing the number of runs.

The MAXMINRUN algorithm first applies the MAXAVGRUN algorithm on the unsmoothed sequence to generate a sequence that has the minimum number of runs. It then considers each pair of consecutive runs in order and tries to bring their lengths as close as possible. Lets say their lengths are n_k and n_{k+1}. Also, let x_{k+1} is the limit on the length by which run $k+1$ can be grown. If $n_k \leq n_{k+1}$, continue with the next pair. Otherwise select $min((n_k - n_{k+1} + 1)/2, x_{i+1})$ number of frames before the beginning of run $i+1$ and discard same number of frames from the end of run k. Accordingly adjust the counter x_{k+1}. We are not done with just one iteration. This procedure is somewhat similar to bubble sort. It proceeds with another iteration if there was any change to any of the runs in the previous iteration. Otherwise the procedure terminates.

Figure 11.3(c) shows the resulting MAXMINRUN sequence for a simple case of unsmoothed sequence given in Figure 11.3(a) and the corresponding MAXAVGRUN sequence in Figure 11.3(b). By applying the above procedure the minimum run length at layer 3 is increased from 1 in unsmoothed sequence, 3 in MAXAVGRUN sequence to 4 in MAXMIN-

RUN sequence. While this heuristic algorithm works towards maximizing the minimum run length, further investigation is needed to develop a provably optimal algorithm.

11.3.3 Maximizing the expected run length

It is quite clear that the longer the runs are in a sequence, the higher it's expected run length is. So in order to maximize the expected run length we need to make each run as long as possible. Furthermore, extension of a longer run contributes more towards the expected run length than that of a shorter run. Hence, a reasonable heuristic approach for maximizing the expected run length is to start with a MAXAVGRUN sequence and to make the longer sequences even longer while further shortening the shorter runs.

The MAXEXPRUN algorithm first applies the MAXAVGRUN algorithm. It then considers each pair of consecutive runs in order and tries to grow the longer run even more. Let the length of two consecutive runs be n_k and n_{k+1}. Also, let x_{k+1} is the limit on the length by which run $k+1$ can be grown. If $n_k - x_{k+1} > n_{k+1} + x_{k+1}$, continue with the next pair. Otherwise select x_{i+1} number of frames before the beginning of run $i+1$ and discard same number of frames from the end of run k. Unlike in MAXMINRUN case, MAXEXPRUN terminates with just one scan over each of the runs given by MAXAVGRUN. Figure 11.3(d) shows the sequence with maximum expected run length corresponding to the unsmoothed one in Figure 11.3(a).

11.4 Adaptive Layer Selection

The algorithms discussed so far assume that the bandwidth availability for the entire duration of the video is known *a priori*. Based on the insights gained from these algorithms, in this section we develop an adaptive scheme for layer selection that assumes no knowledge about future bandwidth availability. However, we assume the presence of a bandwidth estimator that gives the precise current bandwidth in each frame slot.

The key questions that needs to be addressed by any layer selection scheme for smoother quality are: 1) which layers and 2) which frames of a layer be prefetched; when to initiate a run for a layer and where to position the run. The *off-line*[5] algorithm addresses these issues by taking advantage of the complete knowledge about future bandwidth availability.

[5]Hereafter we refer to the MAXAVGRUN algorithm described in the previous section as the off-line algorithm.

With the aid of a forward scan and a backward scan for each layer, it prefetches the frames of layers from lower to higher as late as possible and only when needed. But when future availability of bandwidth is not known, the decision on which layer and frame to transmit next has to be made in an online fashion. In such a case it is not possible to precisely compute the buffer needed for each layer and prefetch just in time. In the following we present an *adaptive algorithm* to predict the buffer requirements of a layer to tide over the potential future bandwidth droughts and to position the runs of a layer to accumulate sufficient cushion before it is displayed.

The adaptive algorithm attempts to estimate future bandwidth availability based on the past history. It associates a *target buffer cushion* ($\tilde{B}^j$) with each layer j and adjusts this value dynamically based on the bandwidth availability. The target buffer cushion of a layer j corresponds to the number of frames of layer j that need to be prefetched in order to continue a run at layer j under the current bandwidth conditions. If the amount of cushion is too less, even small dips in available bandwidth could cause discontinuity of a run at this layer. If it is too much, this may result in inefficient use of buffer which otherwise could help cushion other layers. In the case of off-line algorithm, the forward scan ensures that the prefetched amount is not too less and the backward scan ensures that it is not too much. In the absence of knowledge about future bandwidth availability, the key task of an adaptive algorithm is to estimate the minimum amount of buffer cushion that is sufficient for a layer.

A run of layer j can be sustained only if the average available bandwidth is greater than j and the amount of buffer available for layer j is sufficient to cushion the variations in bandwidth. It is possible to estimate the buffer requirements for an uninterrupted run of a layer j by keeping track of how often the available bandwidth goes below j and how long it stays below j. By monitoring the fluctuations in available bandwidth, we can identify a *critical layer* ($\hat{j}$), i.e., the highest layer that can be subscribed but may not have sufficient buffer available to cushion the bandwidth fluctuations. Given a critical layer $\hat{j}$, ideally we would like to protect the runs of all the lower layers up to $\hat{j} - 1$ from bandwidth droughts and extend the run at layer $\hat{j}$ as long as possible. We may not initiate a run at layer $\hat{j}$ unless it can be sustained for a certain minimum period. In the following, we present a simple adaptive algorithm that addresses these issues in an indirect way and adjusts the target buffer cushion for each layer by tracking only buffer usage.

The adaptive algorithm dynamically adds layers and allocates the buffer among them based

```
1.  PROCEDURE ADJUST-CUSHION(i)
2.    If B_k = B̂, i − τ + 1 ≤ k ≤ τ
3.      For j = 1 to ĵ
4.        B̃^j = βB̃^j, i.e., decrease target cushion
5.        X^j = i
6.
7.    For j = 1 to ĵ
8.      If B_i^j < δB̃^j and B_k^j >= B̃^j for some k, X^j < k < i
9.        B̃^j = αB̃^j, i.e., increase target cushion
10.       X^j = i
11. END PROCEDURE
```

Figure 11.8: The cushion adjustment at time i

```
1.  PROCEDURE ADD-LAYER()
2.    If B^j ≤ B̃^j, 1 ≤ j ≤ ĵ and C̄ ≥ ĵ + 1
3.      If B̃^(ĵ+1) < B̃^ĵ
4.        B̃^(ĵ+1) = B̃^ĵ
5.      λ^(ĵ+1) = i + B̃^(ĵ+1)
6.      ĵ = ĵ + 1
7.  END PROCEDURE
```

Figure 11.9: The procedure to add a new layer

on how buffer is built and drained. The target buffer cushion for a layer is increased if the current target cushion was found to be inadequate to avoid discontinuity in the run due to a bandwidth drought period. It is decreased if the current target buffer cushion was filled and remained undrained for a certain observation window period. Figure 11.8 shows the procedure for adjusting cushion at slot i. The target buffer cushion for all the active layers is decreased by β if the buffer was full for a certain duration τ (lines 2-5). There are two reasons for doing this. First, the buffer being full for sustained period indicates that bandwidth is certainly sufficient to accommodate $\hat{j}$ layers and hence the lower layers can be protected with much less cushion. Second, it also implies that buffer is scarce and hence we need to use it more efficiently by prefetching less conservatively for lower layers. The target buffer cushion $\tilde{B}^j$ for an active layer j is increased by α if the current target cushion was filled earlier after the last cushion update time X^j and then gets drained below a threshold δ (lines 7-10). This way we react in advance for any onset of drought period by conservatively increasing the target cushion even before it is drained completely.

```
1.  PROCEDURE NEXT-TO-XMIT()
2.    For j = 1 to ĵ
3.      If B^j < B̃^j
4.        j' = j
5.        Return
6.
7.    B_min = B̂
8.    For j = 1 to ĵ
9.      If B^j − B̃^j < B_min
10.       B_min = B^j − B̃^j
11.       j' = j
12. END PROCEDURE
```

Figure 11.10: The procedure to determine the layer for transmission

The adaptive algorithm starts a new run at an higher layer $\hat{j}+1$ only if there is sufficient cushion for all the lower layers up to $\hat{j}$ and sufficient bandwidth to accommodate one more layer. The procedure to determine whether to add a new layer and where to start the run is given in Figure 11.9. A new run at a layer $\hat{j}+1$ is initiated only if the target buffer cushion is already filled for all the layers up to $\hat{j}$ and the long term bandwidth is sufficient to accommodate layer $\hat{j}+1$ also (line 2). The long term mean bandwidth ($\bar{C}$) is measured by exponential averaging the current and past bandwidth values. The target buffer cushion for $\hat{j}+1$ is ensured to be at least as large as the layer below it (lines 3-4). This is because higher layers need more cushion to sustain a run than lower layers. The new run is positioned at frame $\lambda^{\hat{j}+1}$, i.e., a distance of $\tilde{B}^j$ away from the current frame slot i (line 5). This is intended to allow sufficient time for the cushion buffer to be filled before the new layer gets played back at the client. The new layer $\hat{j}+1$ thus becomes the critical layer (line 6).

The adaptive scheme determines which layer to be transmitted next based on the target buffer cushion and the current buffer cushion values for each layer. The corresponding procedure is given in Figure 11.10. A frame of a layer j' is transmitted only if the current prefetched amount for each layer up to $j'-1$ is greater than the corresponding target buffer cushion value (line 2-5). In other words the lowest layer with cushion less than its target is chosen for transmission. If all the active layers have their target cushions built up, one more layer may be added using the procedure described in Figure 11.9. When there is additional bandwidth available even after filling up target cushions of all the active layers, the layer with the least additional cushion above its target buffer cushion is chosen for transmission (lines 7-11). In other words the extra bandwidth is shared fairly between all the active

layers.

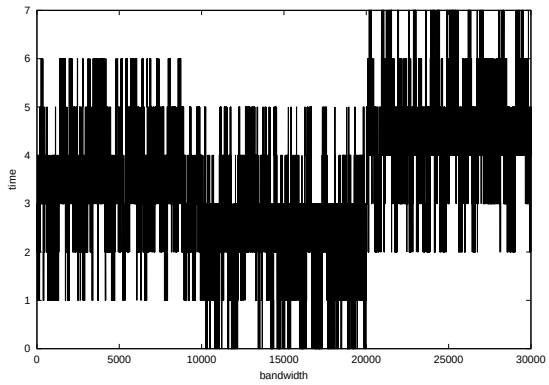
11.5 Experimental Results

We conducted simulations to study the performance of off-line and adaptive algorithms described above. As mentioned in Section 11.3 we assume that the video is CBR with linearly spaced layers, i.e., the size of all frames is the same and is scaled to be 1 unit. Correspondingly the client buffer and the network bandwidth are scaled to be integers. The video consists of 4 layers. The length of the sequence is 30000 frames. The playback rate is set to 30 frames/sec and hence the whole video lasts for a period of 1000 secs.

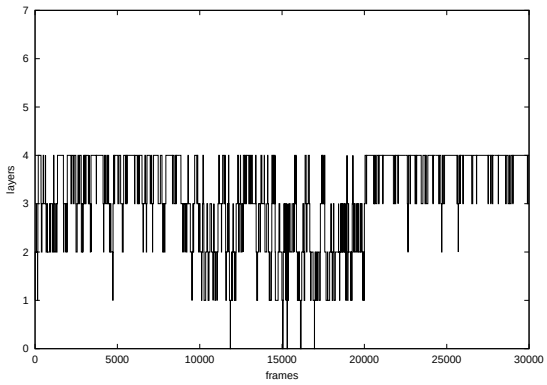
The bandwidth is varied such that the mean bandwidth was 3.5 during the first 10000 frame slots, 2.5 during the next 10000 slots and 4.5 during the last 10000 slots. The faster time scale variation around these mean values is modeled using a Markov chain. The resulting bandwidth curve used in our experiments is shown in 11.11(a).

We study the performance of off-line and adaptive schemes under the above bandwidth conditions by varying the amount of client buffer. The configurable parameters for the adaptive scheme were set as follows. The target cushion increase and decrease factors α and β were set to 2.0 and 0.75 respectively. In other words, the target cushion is doubled whenever the current target was found to be too less and is decreased by a quarter if it was found to be too much. The threshold δ to trigger cushion increase was set to 0.5, i.e., whenever the current buffer for a layer drains below half its target buffer, the target cushion is increased. The observation window period τ is set to 300, i.e., the target cushion values are decreased if the buffer was full for 10 secs. These values are set such that the adaptive scheme is conservative in protecting the lower layers by maintaining higher cushions.

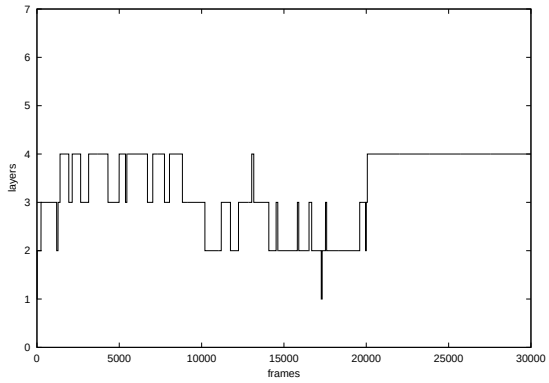
Figure 11.11 compares the number of layers selected, and hence the corresponding smoothness achieved by the off-line and adaptive schemes. The output of the off-line scheme with a small client buffer of 30 is shown in Figure 11.11(b). We expect that this relatively unsmoothed stream is similar to one generated by a greedy layer selection scheme that adds a new layer as and when bandwidth is available. Clearly such a sequence with frequent transitions between layer levels is undesirable. Figures 11.11(c) and 11.11(d) show the layer selection by off-line and adaptive schemes respectively when the client buffer is 300. It is very reasonable to expect a buffer at the client that can accommodate up to 10 secs of one layer of the video. Even with not so large a buffer the off-line algorithm yields a



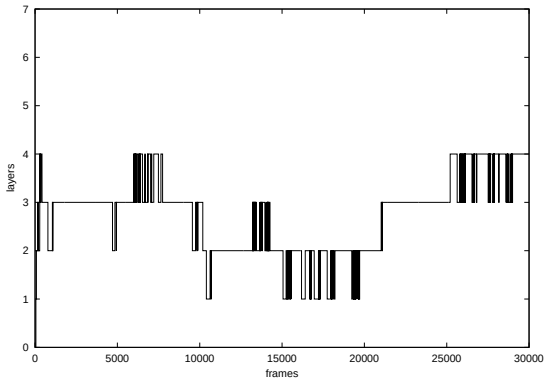
(a) varying bandwidth



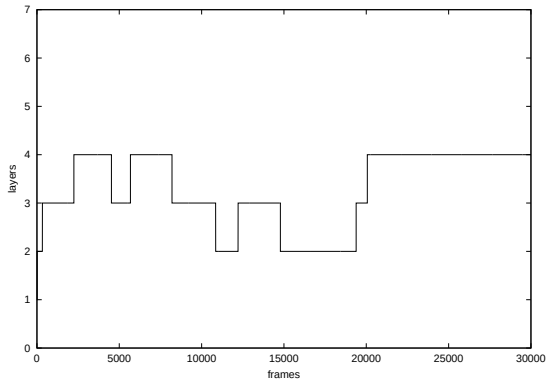
(b) off-line: buffer of 30



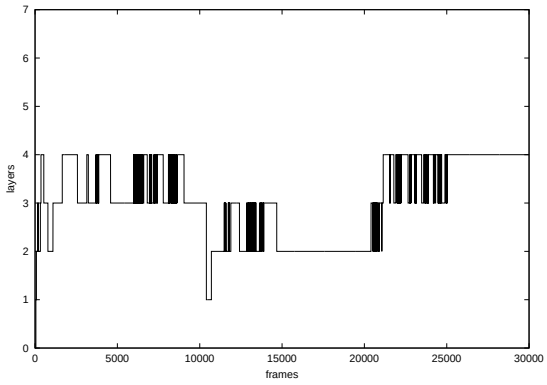
(c) off-line: buffer of 300



(d) adaptive: buffer of 300



(e) off-line: buffer of 900



(f) adaptive: buffer of 900

Figure 11.11: The comparison of smoothness achieved by off-line and adaptive schemes

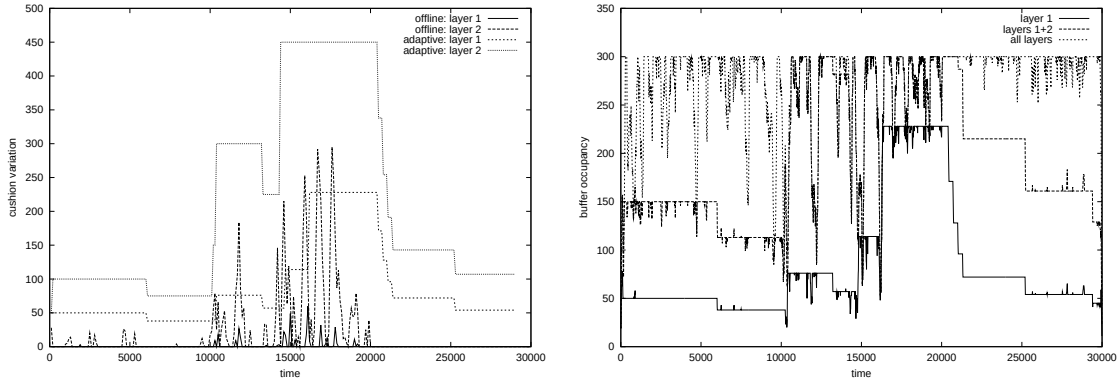
considerably smoother sequence. Though the adaptive scheme generates a less smoother sequence than the off-line scheme, it is still significantly better than the result of a greedy selection scheme.

When the buffer is increased to a size of 900 that can accommodate 10 secs of 3 layers of the video, both the schemes produce much smoother sequences as shown in Figures 11.11(e) and 11.11(f). In particular, the output of the off-line algorithm is very smooth. This is because it has complete knowledge and utilizes the buffer in the most efficient manner. The output of adaptive scheme, while it resembles that of the off-line algorithm in terms of layer subscription level, is not as smooth: there is fluctuation about the critical layer. However, the increased buffer does improve the output sequence of the adaptive scheme. In particular, the segment between 10000 and 20000 is much smoother and displays more layers.

It is worth noting that the sequence in Figure 11.11(f) would appear much smoother if the fluctuations about the critical layer could be avoided. For example, consider the segment between frame slots 21000 and 25000. This could have been improved by not selecting layer 4 when its run cannot be sustained for a certain minimum duration. Currently the adaptive scheme, though it attempts to fill the cushion for a layer before the start of a run, still does not attempt to avoid potentially short runs. We need to further improve the algorithm to estimate the length of a run and initiate one only when it is likely not to be short-lived.

The off-line algorithm also exhibits a similar behavior in that it doesn't explicitly avoid short runs. For example, in Figure 11.11(c) at around frame slot 12500, layer 4 is chosen briefly. This would not contribute to quality viewing and discarding that run would make the sequence smoother. This behavior can be attributed to the work-conserving nature of the off-line algorithm: available bandwidth is always utilized and thus higher layers are selected momentarily when the buffer is already full. The algorithm needs to be modified to avoid shorter runs even by resorting to not being work-conserving. Further, we need to alter the *exprun* metric to discount runs shorter than a certain length.

We now describe the operation of the adaptive scheme given the bandwidth curve shown in Figure 11.11(a). Since the adaptive scheme is not aware of future bandwidth availability it has to dynamically adjust the target buffer cushion for each layer based on past history on the variability in available bandwidth. This cushion adaptation process is illustrated in Figure 11.12(a). It shows how the target buffer cushion is adjusted over time for the layers



(a) target buffer cushion (b) buffer occupancy

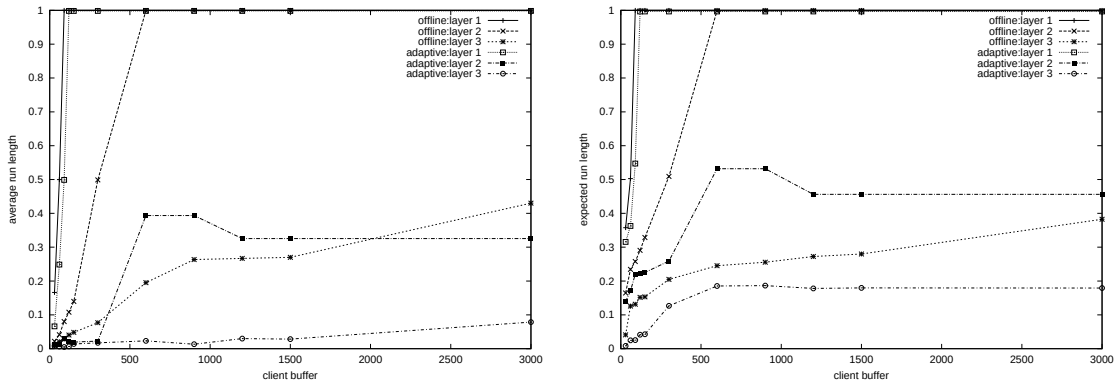
Figure 11.12: The illustration of cushion adaptation process

1 and 2. It also gives the ideal amount of buffer required for these layers as computed by the off-line algorithm. Figure 11.12(b) shows the actual buffer occupied by layers 1 and 2 in the adaptive scheme. It also shows the total buffer occupied by all the layers.

By contrasting the Figures 11.12(a) and 11.12(b) we can see the times when the adaptive scheme changes the target buffer cushion values. For example, at around frame slot 10350, the actual buffer occupied by layer 1 frames is less than half its target cushion of 38. In response to this, the adaptive scheme doubles the target cushion of layer 1 . Similarly the layer 2 target cushion is also doubled. At around 20400, the actual buffer stayed full for at least 300 frame slots, hence causing the target cushion for both layers to be decreased. It can be seen that target buffer cushion values in the adaptive scheme have the same trend as the ideal buffer cushions of the corresponding layers. However, the adaptive scheme is more conservative while being reactive.

We also measured, using the average run length and expected run length metrics, the relative smoothness of sequences resulting from these algorithms. Figure 11.13 shows the performance of these schemes, as measured by these metrics, under various buffer settings. The average run lengths for layers 1, 2 and 3 are shown in Figure 11.13(a) and expected run lengths in Figure 11.13(b).

It can be observed that metrics for layer 1 under both the algorithms approach unity even at very small buffer sizes. With the off-line algorithm, the metrics for layer 2 also approach unity for moderate buffer sizes. On the other hand, the metrics for layer 2 in the adaptive



(a) average run length

(b) expected run length

Figure 11.13: The performance of off-line and adaptive schemes

scheme increase up to a point as the buffer increases and stagnate after a certain buffer size. This is because the discontinuity at around frame 10350 in the run of layer 2 persists even with large buffer sizes (as evident in Figure 11.11(f)). This discontinuity is an artifact of the adaptive scheme. Though the adaptive schemes reacts in advance to an onset of congestion, it still may not be able to build sufficient cushion to tide over a sudden but prolonged dip in available bandwidth. Further, in an attempt to use the buffer and bandwidth more effectively, the adaptive scheme chooses not to be too conservative in prefetching and thus may not build a larger cushion even with a larger available buffer. This does not, however, fully explain the decrease in the metrics with increased buffer sizes beyond 900 and requires further investigation. The effect of larger buffers is less pronounced with higher layers since their selection is limited more by the lack of bandwidth than the lack of buffer space.

It is evident that metrics improve faster with the off-line scheme, reflecting its efficient use of buffer. As can be seen, both metrics capture the relative smoothness: the sequences generated by the off-line algorithm score higher than the corresponding sequences from the adaptive algorithm. Moreover, the graphs of both the metrics appear quite similar. This indicates that a work conserving algorithm that maximizes average run length would also come close to maximizing the expected run length metric.

11.6 Related Work

The problem of layer selection for delivering layered video has received a lot of attention recently. One of the earliest works in this area presented in [59] proposes a receiver-driven layered multicast (RLM) protocol for transmitting layered video to heterogeneous receivers. The number of layers subscribed by a receiver is dynamically varied based on the perceived loss rate and thru join experiments. The later studies [26] have shown that receiver-driven schemes such as RLM exhibit significant instability. There were some proposals [7, 27] on addressing this problem inside the network by assigning higher priority to lower layers and providing priority based dropping at routers in case of congestion. These approaches introduce some additional complexity at core routers.

In [98] the available bandwidth is modeled as a stochastic process and optimal allocation of bandwidth between base and enhancement layers is studied. It was assumed that client buffer is unlimited. The work most relevant to ours was reported in [87]. They assume that TCP-friendly congestion control [86] was employed and hence the available bandwidth curve has a sawtooth shape. They address the problem of buffer allocation between layers such that it is used efficiently in absorbing the short-term fluctuations in bandwidth. Though our work is quite similar in spirit to theirs, our approach has been quite different. Our focus has been on designing metrics to capture smoothness criteria and on developing algorithms to maximize these metrics.

11.7 Conclusions and Future work

In this chapter we addressed the issue of layer selection for maximizing the perceived video quality under given resource constraints. We defined smoothness criteria and designed metrics namely, *avgrun*, *minrun*, and *exprun* for measuring smoothness. We then developed an optimal offline algorithm MAXAVGRUN to find a sequence with maximum average run length when the network conditions are known *a priori*. We also presented heuristic algorithms, MAXMINRUN and MAXEXPRUN for maximizing the minimum and expected run lengths respectively. We then described a simple adaptive algorithm for providing smoothed layered video delivery that doesn't assume any knowledge about future bandwidth availability. We conducted simulations to study the performance of these schemes and shown that even with a small client buffer it is possible to provide significantly smoother quality video playback.

There are several simplifying assumptions made in this preliminary work on providing smoother quality layered video stream. Specifically, in the adaptive layer selection scheme, we assume the presence of a bandwidth estimator that gives the precise current bandwidth in each frame slot. It is likely that there would be some amount of error in the estimation of available bandwidth. This could lead to packet losses. The packet losses can be handled using retransmissions. Since the frames are prefetched there is sufficient time for error recovery through retransmissions. The trade-off between retransmission of the lost packets and the prefetching of new frames should be investigated.

The schemes, as presented, provide smoother video by favoring longer runs in a layer. They need to be enhanced to further smoothen the video streams by also avoiding short runs. Similarly, the *exprun* metric also needs to be refined to discount short runs. Real experiments need to be conducted to ascertain the relative merit of the proposed metrics and effectiveness of the proposed algorithms. Finally, the schemes presented here work on layered CBR video streams, and need to be extended for VBR video streams too.

Part III

Channel Assignment and Admission Control in Cellular Networks

Chapter 12

Compact Local Packing

12.1 Introduction

A mobile *cellular system* divides the geographical region it serves into smaller regions called *cells*. A radio channel used in one cell can be used in another if they are separated by *co-channel reuse distance*. Concurrent use of a channel in cells within this distance causes *co-channel interference*. Frequency reuse without causing interference is the core concept of cellular systems.

A *base station* (BS) serves *mobile hosts* (MH) in each cell. The mobile cellular system is centrally coordinated and administered by a *Mobile Telecommunication Switching Office* (MTSO). Base stations are connected to the MTSO and each other by a wired network. To establish a communication session, a MH requests the BS in its cell for a channel. BS, probably in consultation with MTSO, assigns an unused channel if available. Otherwise the call is blocked. When a MH moves to a different cell during a session, the responsibility of continuation of service is handed over to the new BS. For a successful *hand-off* the new BS must find an idle channel. Otherwise the session is terminated. *Reduction of blocking and forced termination probabilities* is the main goal of channel assignment schemes.

The channel assignment schemes suggested in the literature [39] range from fixed to dynamic, centralized to distributed, and timid to aggressive. Fixed assignment strategies assign a set of channels permanently to a cell, while dynamic schemes assign them on-demand. In centralized schemes, MTSO makes the assignment decisions using the complete information it has about the system state. Distributed channel allocation algorithms

run at individual BSs, which are collectively responsible for the channel assignment. Aggressive schemes allow channel reassignments to make room for a new request, while timid ones do not permit such reconfigurations.

Local Packing (LP) is a distributed dynamic channel allocation (DDCA) method [12], where each base station assigns channels to calls using an augmented channel occupancy (ACO) table. When a base station receives an access request, it searches for an empty column (available channel) in its ACO table. If more than one candidate exists, a channel is selected arbitrarily. In this chapter, we investigate the impact of channel selection on packing. We propose two new schemes for channel selection. *Coaxial* selection uses a co-cell channel occupancy (CCO) table in addition to the ACO table. *Residual* selection requires the ACO table to be extended (EACO) with channel usability information. While the *coaxial* scheme chooses a channel that is being used by the most number of co-cells, the *residual* scheme selects a channel that is usable in the least number of interfering cells. Simulation results show that these schemes block fewer calls than *sequential* and *random* selection strategies. While *coaxial* scheme achieves this at the expense of increased broadcast overhead, the *residual* scheme avoids the overhead while achieving almost identical blocking performance.

12.2 Local Packing

The Local Packing algorithm is a cell-based distributed dynamic channel allocation method that adapts to local traffic demands. Each base station assigns channels to new or handoff calls using its *Augmented Channel Occupancy* (ACO) table. This, as shown in Figure 12.1(a), is a matrix of $k_i + 1$ rows and $M + 1$ columns, where M is the number of channels in the system and k_i is the number of interfering cells for the cell i. The check marks (*X*) in the first row indicate the channels occupied by cell i and the remaining k_i rows indicate the channel occupancy pattern in the neighborhood of cell i. Thus an empty column indicates the availability of the corresponding channel which can be assigned to a call in cell i. The entries in the last column of the matrix show the number of currently available channels in the corresponding cell.

When a base station receives a request for a channel, it searches for an empty column in its ACO table. If successful, that channel is assigned to the request. If more than one such channel exists, the first one is selected. If there are no free channels, the base station looks for a channel that is being used in no more than one neighboring cell. This corresponds to

Cell-site	Channel Number								Num Free
	1	2	3	4	5	6	…	M	
i		X			X		…		2
i_1	X		X				…		0
i_2							…	X	2
i_3	X		X				…		1
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
i_{k_i}			X				…	X	3

(a) ACO Table

Cell-site	Channel Number								
	1	2	3	4	5	6	…	M	
c_1	X					X	…		
c_2				X			…		
c_3	X		X				…		
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
c_{l_j}			X				…	X	
#Used	2	0	2	1	0	1	…	1	

(b) CCO Table

Cell-site	Channel Number								Num Free
	1	2	3	4	5	6	…	M	
i		X		O	X	O	…		2
i_1	X		X				…		0
i_2	O					O	…	X	2
i_3	X		X	O			…		1
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
i_{k_i}	O		X			O	…	O	3
#Usable	2	0	0	1	0	2	…	1	

(c) EACO Table

Figure 12.1: Information at cellsite i

a column with a single check mark in the ACO table. If such a channel exists, it checks to see, by referring to the last column, if the cell using this channel has any free channels. If that is the case, a request is sent to that cell to relocate the call currently holding that channel to another channel. It then assigns the freed channel to the access request in its cell. Thus LP attempts packing of channels by allowing up to one local reassignment to accommodate a new call and achieves the same performance as that of the most persistent polite aggressive DDCA [13, 14].

The content of the ACO table is updated by collecting channel occupancy information from all the interfering cells. Whenever a base station captures or releases a channel, it has to broadcast this information to all the interfering cells. A base station on receiving such a broadcast updates its own ACO table and recomputes the number of free channels in its cell. It should also send out an update if the number of available channels in its cell changes as a result of a neighbor occupying or freeing a channel. It is assumed that fast and reliable communication protocols for the ACO table update are in place to ensure database consistency.

In LP, the base station assigns the first available channel to an access request. Though *random* selection is a viable option, the authors of LP felt that *sequential* selection is a natural choice and may also result in better packing. The base stations in LP have very few selection alternatives due to the limited amount of information in their ACO tables about the available channels. While the ACO table allows a base station to identify the free channels, it doesn't offer any clues on discriminating between them.

A natural extension would be to supplement the ACO table of each cell with channel occupancy information of its *co-cells*, i.e, cells at co-channel reuse distance. This information, referred to as *co-cell channel occupancy* (CCO) table, can aid in differentiating between available channels based on their usage pattern in the co-cells. The *coaxial* selection scheme presented in Section 12.3, is based on this approach.

Alternatively we can extend the ACO table to include channel usability information along with channel occupancy information for each interfering cell. This knowledge can be utilized to distinguish between available channels. The extended ACO (EACO) table increases the choices for selection and enables a base station to pick channels more judiciously. The *residual* selection scheme based on this approach is presented in Section 12.4.

12.3 Coaxial Selection

Coaxial selection is based on the observation that reusing the channels that are currently being used in the co-cells results in better packing. The requisite channel occupancy information for each co-cell is maintained in the CCO table. This table, as shown in Figure 12.1(b), is similar to the ACO table. Each row corresponds to a co-cell and the column to a channel. A check mark (*X*) in this table indicates the occupancy of that channel by the respective co-cell. Additionally, the last row contains the count of co-cells currently using a given channel.

The update procedure for CCO table is similar to that of ACO table. Whenever a base station acquires or releases a channel, it has to notify its co-cells along with its neighbors so that each base station is aware of the channel occupancy pattern in both its interfering cells and co-cells. However, a change in the count of free channels due to a neighbor, need to be communicated only to interfering cells, not to co-cells. This update procedure requires the same information to be sent, as in the case of LP, but to twice the number of base stations, involving an additional processing overhead at each base station.

Coaxial scheme selects, from the available channels in a cell, a channel that is currently being used by the maximum number of its co-cells. This channel corresponds to the column that has the maximum count in the last row of CCO table, among the empty columns in the ACO table. A channel that is being used by the maximum number of co-cells is less likely to be usable by the interfering cells. By making such a selection, each cell collaborates with its co-cells in an attempt to achieve a globally compact pattern.

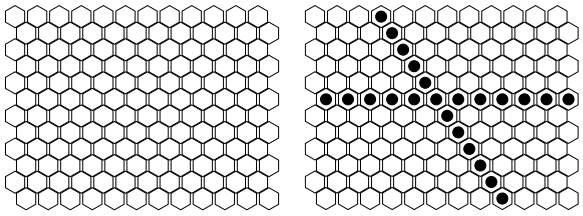
12.4 Residual Selection

Residual selection is based on the observation that channel occupancy pattern in the co-cells can be inferred from the channel usability information in the neighborhood. The EACO table, as shown in Figure 12.1(c), contains channel usability information along with

channel occupancy information. This table is very much like the ACO table except that the EACO table has one extra row and another type of mark, a circle (*O*). Similar to a check mark, which when placed in an entry corresponding to a cell l and a channel n indicates that channel n is occupied by cell l, a circle mark in that entry would mean that channel n is available for use in cell l. While the last column gives the count of free channels for each interfering cell, the circle marks identify those channels. The entries in the last row keep the count of circle marks in their corresponding columns (excluding the first row), i.e., the number of neighboring cells where the corresponding channels are usable. So a circle mark in the first row and a zero in the last row for column n implies that channel n is available for use in this cell and not usable in any of the neighboring cells.

The entries in the EACO table are updated through a procedure similar to the one used to maintain the ACO table. Whenever a base station grabs or frees a channel, it has to notify its neighbors so that each base station is aware of the channel occupancy pattern in its neighborhood. It should also send out an update broadcast whenever there is a change in its cell's usability status of any channel (circle marks in the first row). While it may seem that the maintenance of the EACO table consumes more update broadcasts, it actually needs no more than what is required by the ACO table. An available channel is indicated by an empty column in the ACO table and similarly by a circle mark in the first row of the EACO table. But in both the tables, the first entry of the last column indicates the number of available channels, and any change in the usability status of a channel affects the number of available channels. Hence every update broadcast by a base station employing EACO table has an analogous one in the case of ACO table. While the number of broadcasts is the same in both cases, the difference is in their content. Whereas the number of free channels is broadcast in the case of ACO table, the messages for EACO table carry the channel number and the current status of the affected channel. From this, the number of free channels in each cell and the number of cells in which a channel is usable can be computed.

The channel usability information contained in the EACO table can be utilized to discriminate between two available channels, giving rise to several options for selection. The *residual selection* is a scheme that opts for channels that are least usable in its interference region. This scheme selects, among the available channels, a channel that is usable in the least number of neighboring cells, i.e., a channel corresponding to the column that has the minimum count in the last row, among the columns having a circle mark in their first row. The intention of such a choice is to reduce the impact of this assignment in the neighbor-



(a) uniform traffic (b) hotspots

Figure 12.2: The simulated cellular system

hood, in terms of the number of interfering cells in which the assigned channel is rendered unusable. The extreme case being the existence of an available channel that is currently not usable by any of the neighboring cells, in which case it would be ideal to pick such a channel since this assignment doesn't affect any of the neighbors.

This approach of opting for a channel that is unusable by neighbors, can help form a compact pattern among the co-cells. A channel is available in this cell but not in the neighboring cells, possibly because the same channel is being used by the cells within the neighbor's interference region but just outside this cell's interference region. By assigning that channel to a call in this cell, *residual* selection scheme attempts to keep the co-channel distance as close to the allowed minimum as possible. Another advantage of this scheme is the reduced number of update broadcasts. While the messages carrying channel occupancy information remain unchanged, fewer broadcasts are required to propagate usability status. This is a direct fallout of choosing a channel that is already unusable in most of the neighboring cells. Such an assignment doesn't cause a change in usability status of that channel in many of the interfering cells, and thus limits the ripple effect of occupancy update broadcasts.

12.5 Simulation and Analysis

We have studied the impact of selection strategy on the performance of LP by simulating different channel selection schemes and measuring their call blocking probabilities. Apart from the *residual* and *coaxial* methods described above, we have simulated *sequential* and *random* selection schemes. As the names suggest, the *sequential* method picks the first available channel and the *random* method randomly (uniformly) chooses one among all the available channels. Each method is simulated in both aggressive and timid modes, i.e., with and without allowing reassignments. While these schemes vary in their selection among

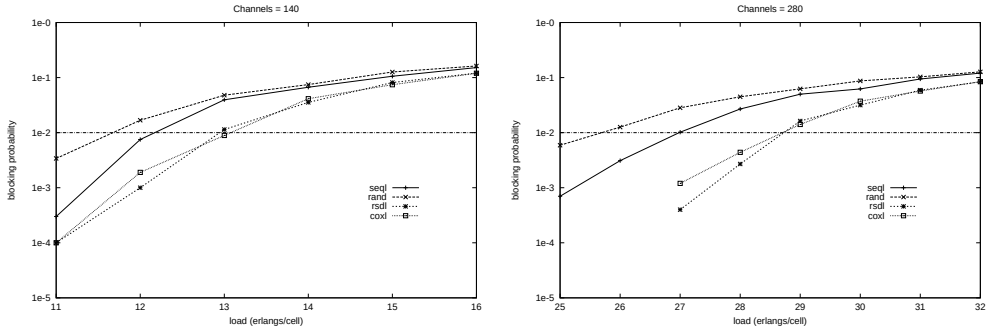
many idle channels, in aggressive mode they all follow the same procedure as described in LP for identifying a channel for relocation, when there are no free channels.

We have measured the call blocking probability of all these methods by developing a simulation model using CSIM library [100]. Our cellular system consists of 144 hexagonal cells arranged as a 12x12 array as shown in Figure 12.2(a). To avoid the boundary effect, this array is wrapped-around in both the dimensions, i.e., left-most and right-most columns are made neighbors and so are the top and bottom rows. This is intended to make the results, representative of an infinite system and applicable to a typical large cellular network. We have assumed the size of a *cluster*, the set of neighboring cells that cause mutual co-channel interference, to be 7, which is the typical reuse constraint in current cellular systems. This implies that a channel that is being used in a cell, cannot be simultaneously used either by the immediate or second-layer neighbors, amounting to a total of 18 interfering cells surrounding each cell.

These simulations were carried out assuming that the total number of channels in the system is 140 and 280. Each run simulates arrival of 10,000 mobiles, where the average call arrival rate is λ per cell and the average call holding time, $1/\mu$, is normalized to 1. Assuming that the call inter-arrival time and call holding time are exponentially distributed, and also assuming infinite population, the load per cell, λ/μ, would be same as λ. Thus the call blocking probability under various loads is determined by varying just the call arrival rate. Apart from the uniform traffic, we also model the traffic hot spot conditions on two intersecting highways as shown in Figure 12.2(b). Our simulation model is quite similar to the one presented in [12].

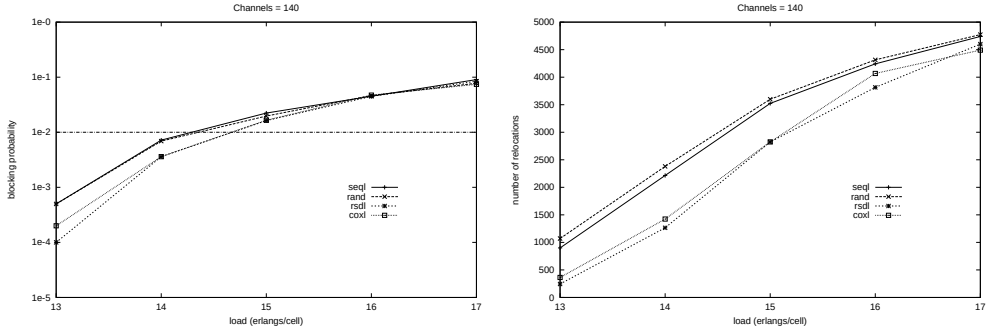
Figure 12.3 shows the blocking performance of these selection schemes, in timid mode, under uniform traffic conditions. It clearly shows that both *residual* and *coaxial* selections result in lower blocking probability than *sequential* and *random* selections in both the cases. Further, the blocking performance of *residual* and *coaxial* selection schemes are so close that their curves are almost indistinguishable. It also shows that the *sequential* selection yields better results than *random* selection. Overall, considering that 1% blocking is a reasonable design target, the proposed schemes carry 10% more traffic, at that blocking level, than the other two schemes.

The blocking performance of these schemes running in aggressive mode under uniform traffic conditions is shown in Figure 12.4. It also gives the number of relocations necessitated by each of these schemes. With reassignments, there doesn't seem to be considerable

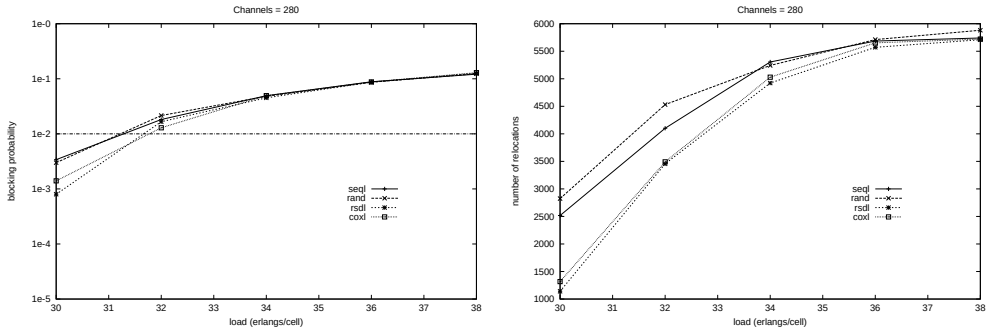


(a) (b)

Figure 12.3: Performance comparison of timid schemes

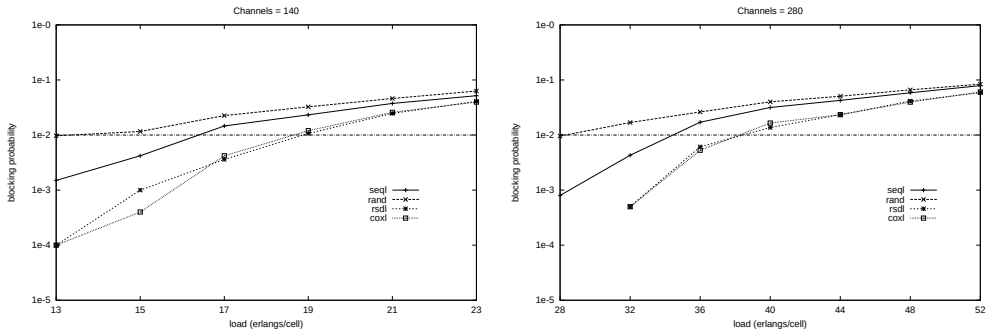


(a) (b)



(c) (d)

Figure 12.4: Performance comparison of aggressive schemes

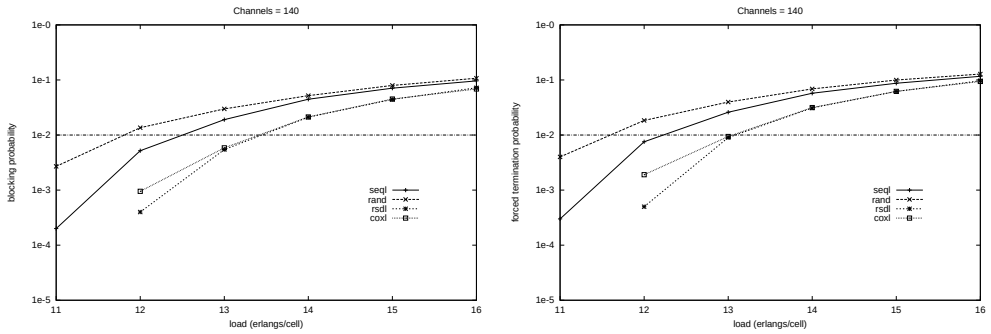


(a) (b)

Figure 12.5: Performance under traffic with hot spots

difference between different selections, in terms of the number of calls blocked. But both *sequential* and *random* methods trigger more relocations than the other two schemes. For example, in the case of a system with 280 channels and the traffic load of 32 Erlangs/cell, the number of calls blocked by *sequential* and *residual* schemes are 182 and 166 respectively. While the *sequential* method relocates 4102 calls, the *residual* approach does it only 3456 times. In essence, in aggressive mode the *sequential* and *random* methods catch up with the proposed schemes, but at the cost of increased number of relocations. It is interesting to note that *random* selection which performs quite poorly in timid mode, does fairly well in aggressive mode. This brings to our attention, the interaction of the assignment and reassignment strategies. It is possible that the channel packing attempted by selection schemes may behave differently due to the relocations, which needs further investigation.

Figure 12.5 shows the blocking performance of these selection schemes, in timid mode, under traffic conditions shown in Figure 12.2.b. The traffic in cells other than hot spots is fixed at a load where all the schemes have less than 1% blocking probability. This was 11 and 25 Erlangs for systems having 140 and 280 channels respectively. It is quite apparent that the proposed schemes are better at alleviating hot spots. Figure 12.6 shows the performance of these selection schemes, in timid mode, in the presence of hand-offs. We simulated the mobiles with an average of two hand-offs and a uniform probability of migrating to any of the adjacent cells. Hand-offs and originating calls were treated similarly. Again, the proposed schemes resulted in lower probability of both blocking and forced termination than the other two schemes. These results demonstrate that the proposed schemes either carry more traffic or reduce the number of reallocations.



(a) (b)

Figure 12.6: Performance under traffic with hand-offs

Following is a comparison of runtime of these schemes in selecting a channel. Let the total number of channels available in the system be m. To pick a free channel, both *sequential* and *random* selection schemes need to scan, in the worst case, the entire first row of the ACO table once, resulting in $o(m)$ time complexity. The *residual* scheme requires keeping track of the count of cells in which a channel is usable, which can be updated incrementally. The first row and the last row of EACO table are scanned to find an available channel with the *minimum usable* count. Similarly *coaxial* scheme needs a scan of the first row of the ACO table and the last row of CCO table to identify a free channel with the *maximum used* count. Both schemes require $o(m)$ time for selection.

It is clear that the proposed schemes, *coaxial* and *residual*, outperform *sequential* and *random* selection strategies. Though *coaxial* and *residual* schemes have almost identical blocking performance, the former incurs an extra update broadcast overhead to maintain the CCO table. However the *residual* selection does not involve any additional cost to maintain the EACO table. Hence we conclude that *residual* scheme is the most suitable selection strategy.

12.6 Summary

Local Packing (LP) is a distributed dynamic channel allocation (DDCA) method where each base station assigns channels to calls using an augmented channel occupancy (ACO) table. We have shown that the ACO table can be extended to include channel usability information, and argued that this extra information can be maintained without increasing

the number of update broadcasts. We proposed *residual* selection scheme that makes use of this channel usability information and *coaxial* scheme that uses a CCO table in addition to the ACO table. These schemes were compared with *sequential* and *random* approaches, and found to either carry more traffic or reduce the number of relocations needed. We have found that *residual* scheme has almost identical blocking performance to that of *coaxial* scheme, while incurring significantly less update broadcast overhead. These schemes essentially attempt to reduce both the call blocking and dropping probabilities by increasing frequency reuse. They do not distinguish between handoff requests and originating calls. Since forced termination of an existing call is less desirable than blocking a new call, call admission control schemes have been proposed that give higher priority to handoff requests. In the next chapter, we study how reassignments can be used to ensure continuity of service to admitted calls without wasting spectral resources.

Chapter 13

Reassignment based Admission Control

13.1 Introduction

Call admission control (CAC) schemes in wireless cellular networks attempt to reduce call dropping probability possibly at the expense of increased call blocking probability. Several CAC schemes have been proposed [32, 53, 66, 84] in the literature. Most CAC schemes are based on the guard channel concept [32]. This approach offers a generic means of improving the probability of successful handoffs by simply reserving a few channels in each cell exclusively for handoffs. This may, however, result in low spectrum utilization. A recently proposed scheme, based on the concept of shadow cluster [53], performs CAC decisions using estimates of future resource requirements of mobiles. Besides its complexity, this scheme requires knowledge about the mobility pattern of users, which may not always be available.

In this chapter, we explore the use of channel *reassignments* to reduce handoff failures. It has been pointed out that dynamic channel assignment schemes cannot maximize channel reuse as they serve randomly offered call attempts. This leaves enough room for channel reconfigurations which can be used to service otherwise unsatisfiable handoffs. Additionally, a handoff results in releasing a channel in the old cell which can potentially be used for reassignment to free up a channel in the new cell. But reassignments may inconvenience users and incur communication and processing overheads. However, at low loads the additional number of calls serviced more than compensates the overhead due to reassignments. As the system gets overloaded, the number of reassignments grows rapidly without a corresponding increase in the number of serviced calls. Hence reassignments must be used

prudently to be beneficial.

We propose using guard channels to control reassignments. Reassignment in the neighborhood is used as an indication of congestion by a cell. The number of guard channels in a cell is dynamically adjusted based on the reassignment frequency of its neighbors. This significantly reduces the need for further reassignments by having the cell react prior to onset of congestion, while maintaining high spectrum utilization. It is possible to conceive various adaptive approaches that utilize the knowledge of reassignments in the neighborhood of a cell. A simple scheme would be to keep the reassignments under a given target. An ideal, but perhaps impractical, scheme is one that maximizes the revenue, given that each call serviced brings in some reward and each reassignment incurs some penalty. We propose a scheme that attempts to balance the number of reassignments and the number of admitted calls by dynamically adjusting guard channels. Simulation results show that this scheme uses reassignments profitably, while achieving near zero dropping probability.

The rest of the chapter is organized as follows. Section 13.2 describes the simulation set-up that is used for all the experimental results reported in this chapter. Section 13.3 discusses the use of guard channels to reduce call dropping and their effect on bandwidth utilization. We then show in Section 13.4, how reassignments can be used to almost eliminate handoff failures and investigate the effect of load and mobility on reassignments. A simple scheme that attempts to keep the number of reassignments under a specified target is described. We then present a revenue based scheme. We conclude the chapter with some remarks about future extensions to this work.

13.2 Experimental Setup

All our experiments were conducted using a simulation model described here. The cellular system consists of 144 hexagonal cells arranged as a 12x12 array. To avoid the boundary effect, this array is wrapped-around in both the dimensions. The size of a *cluster*, the set of neighboring cells that cause mutual co-channel interference, is assumed to be 7, amounting to a total of 18 interfering cells surrounding each cell. This interference zone is referred to as the cell's *neighborhood*.

All simulations were carried out assuming that the total number of channels in the system is 70. Local Packing [12], a distributed dynamic channel allocation algorithm, is used to allocate channels to new call or handoff requests. Each cell maintains an augmented

Cell-site	Channel Number								Num Free
	1	2	3	4	5	6	...	M	
i		X			X		...		0
i_1	X		X				...		0
i_2						X	...	X	2
i_3	X		X				...		1
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
i_{k_i}			X				...	X	3

Figure 13.1: ACO table at cellsite i

channel occupancy (ACO) table that tracks channels used and the number of free channels available in each neighboring cell. Figure 13.1, shows an example ACO table at cell i. Channels not in use in a cell's neighborhood are eligible for assignment to calls in that cell. When no such channel is available, a channel that is being used in exactly one neighboring cell having free channels is considered eligible for reassignment. For example, given the ACO table of cell i in Figure 13.1, the mobile currently using channel 6 in neighbor i_2 can be relocated to one of the 2 free channels available in i_2, releasing channel 6 to satisfy a request in cell i.

Call interarrival time and call holding time are assumed to be exponentially distributed. The mean call holding time, $1/\mu$, is fixed at 1 and thus the average load per cell λ/μ is varied by varying the mean arrival rate per cell, λ. The number of handoffs per call is assumed to be geometrically distributed with mean $\bar{h}$ and hence the probability of a call moving out of a cell is $\bar{h}/(\bar{h}+1)$. Mobiles are assumed to migrate to any of the adjacent cells with equal probability. A mobile resides for the same amount of time at each visit to a cell during the lifetime of the call. Except when studying the effect of varying mobility, $\bar{h}$ was set to 3 for all the experiments. Similarly the load is fixed at 6 Erlangs/cell when studying the effect of mobility. Each experiment simulated 100,000 calls.

13.3 Utility of Guard Channels

Admission control is based on the principle that denial of service to new calls is better than unreliability of service to admitted calls. In other words, dropping calls in progress due to handoff failures is less desirable than blocking new calls. A simple way of giving priority to handoffs is to reserve a number of channels (say g) exclusively for servicing handoffs.

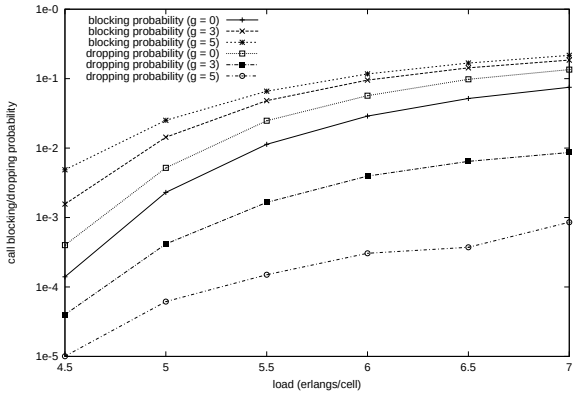


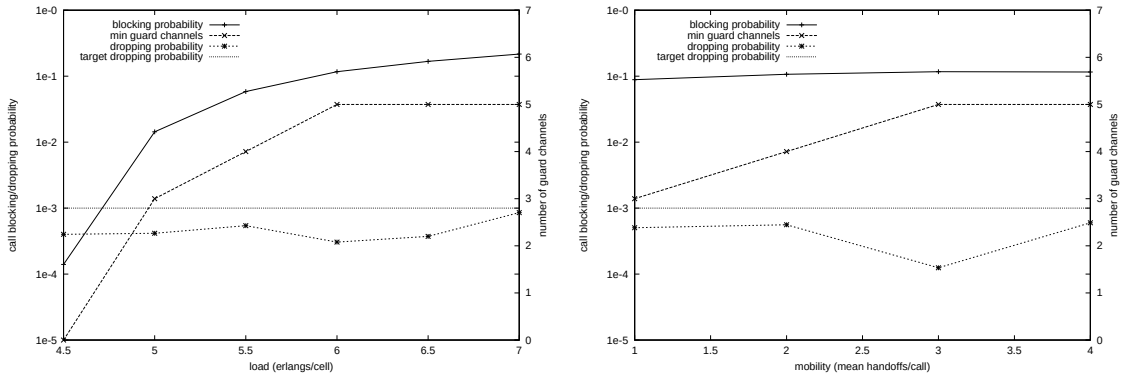
Figure 13.2: Effect of guard channels on blocking and dropping probabilities

These reserved channels are called *guard channels*. A cell rejects new calls whenever the number of free channels available in that cell goes below g. While this reduces the chances of handoff failures, it may result in under-utilization of scarce bandwidth.

Unlike fixed channel allocation schemes, with dynamic schemes like LP guard channels are not exclusive to a cell. A single unused channel in a neighborhood may be counted as a free channel by more than one cell. Hence, g guard channels per cell does not imply that two neighboring cells have a combined total of $2{\cdot}g$ channels reserved for handoffs. In fact, this could be as low as g when both cells have the same set of free channels available. A new call is admitted only if the number of free channels available is more than g. In this chapter, GUARD refers to a guard channel based scheme using LP for channel allocation.

Figure 13.2 shows the blocking and dropping probabilities under varying load for different values of g. As the number of guard channels increases the dropping probability decreases while the blocking probability increases. When g is zero, the dropping probability is even more than the blocking probability. This is due to the fact that there are more handoffs ($\bar{h}$ is 3) than new calls and handoffs are not given any preferential treatment over new calls. As the load increases both blocking and dropping probabilities increase across all values of g. A similar trend can be seen (not shown here) as mobility increases.

The objective of admission control schemes is to minimize the call blocking probability while keeping call dropping probability below an acceptable limit. For guard channels based schemes, this is equivalent to guaranteeing a given upper bound on call dropping probability using as few guard channels as possible. However, statically determining the right number of guard channels to achieve this is not possible, since it depends on traffic



(a) varying load

(b) varying mobility

Figure 13.3: Minimum number of guard channels required to keep dropping probability below $1/1000$

conditions such as load and mobility.

Figure 13.3(a) shows the minimum number of guard channels per cell required to keep call dropping probability below $1/1000$, for different values of load. It also shows the corresponding blocking probabilities. These values were obtained for each load by conducting a series of experiments with progressively increasing numbers of guard channels until the dropping probability fell below $1/1000$. The number of guard channels is shown on the right vertical axis. It can be seen that the minimum number of guard channels required varies with load. For example, at a load of 5 Erlangs/cell the number of guard channels required is 3 while it is 5 when load is 6 Erlangs/cell.

Figure 13.3(b) shows the minimum number of guard channels per cell required to keep call dropping probability below $1/1000$ for different values of mobility. Again, the minimum number of guard channels required varies with mobility. For example, when mobility ($\bar{h}$) is 2, the number of guard channels required is 4 while it is 5 when $\bar{h}$ is 3.

From these figures, it is evident that fixing the number of guard channels statically will result in either under-utilization of bandwidth or poor quality of service. Hence it is desirable to have an adaptive scheme that dynamically adjusts the number of guard channels in each cell to suit the prevailing traffic conditions. In the following section, we show how reassignments could be used for this purpose.

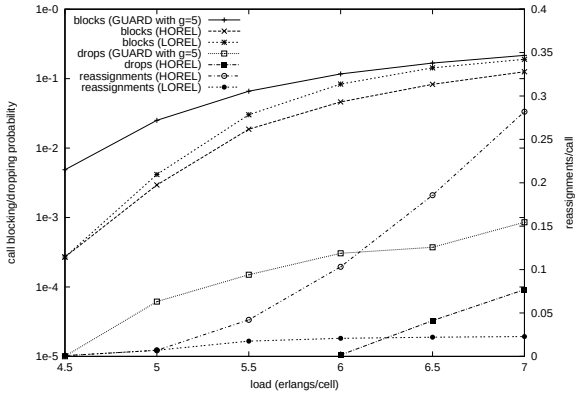


Figure 13.4: Illustration of the effectiveness of reassignments

13.4 Effectiveness of Reassignments

When no free channels are available in a cell for a new request, it may be possible to relocate an on-going call in a neighboring cell to a different unused channel in that cell and release the currently used channel. This released channel can then be assigned to the new request. Such a reallocation of channels to calls in progress to make room for a new request is called *reassignment*. This has been shown to increase the capacity of the system to carry more load [16]. But reassignments may inconvenience users and incur communication and processing overheads. Hence reassignments should be used sparingly and judiciously.

We propose using reassignments to reduce handoff failures. Dynamic channel assignment schemes cannot maximize channel reuse as they serve randomly offered channel requests. As a result, it is almost always possible to reassign channels to calls in progress to make room for a successful handoff. Further, a handoff involves releasing a channel in the old cell which can potentially be reassigned to free up a channel in the new cell. While reassignments do have an associated cost they are much more preferable than dropping calls. Thus, relying on reassignments to avoid call dropping is justifiable. However, it is not advisable to use reassignments to admit a new call since the need for reassignment indicates overload in the neighborhood. Admitting a new call in such a scenario is likely to cause many more reassignments. Moreover, it may lead the system to a state of saturation where no further reassignments are possible resulting in failed handoffs.

A simple scheme based on reassignments, called HOREL, rejects a new call only if there are no free channels available (i.e., no guard channels) but use reassignments if needed to satisfy a handoff request. Figure 13.4 compares the performance of GUARD with g

of 5 and HOREL. The average number of reassignments per call is shown on the right vertical axis. This indicates the expected number of times a mobile is forced to switch channels due to reassignments. HOREL drops less than 1 in 10000 calls while blocking much fewer calls than GUARD. But as the load increases HOREL causes quite a few reassignments with more than 1 reassignment per every 4 calls admitted, at load 7. At low loads the additional number of calls serviced more than compensates the overhead due to reassignments. As the system gets overloaded, the number of reassignments grows rapidly without a corresponding increase in the number of serviced calls.

It is clear that the uncontrolled use of reassignments even for handoffs could be counter-productive. However, used in a controlled manner, apart from ensuring success of handoffs, reassignments also provide an indication of congestion in the neighborhood. One way to control reassignments is by employing guard channels. Reserving a few channels for handoffs obviates the need for frequent reassignments. On the other hand, reassignments can be thought of as a feedback mechanism to determine the right number of guard channels to eliminate handoff failures. The number of guard channels can be dynamically adjusted based on the extent of congestion as indicated by the reassignment frequency. This allows the system to respond just in time adapting to the existing traffic conditions rather than conservatively allocating more guard channels. Thus, it is possible to use both reassignments and guard channels synergistically. Reassignments mitigate the negative effect of guard channels on bandwidth utilization while guard channels prevent excessive reassignments.

We shall now see two such adaptive schemes LOREL and MXREV that utilize the knowledge of reassignments in the neighborhood differently. LOREL attempts to contain the number of reassignments under a specified limit while MXREV tries to maximize the revenue by balancing the penalty for reassignments with the reward for serviced calls. From a quality of service point of view it is desirable to place an upper bound on the number of times a mobile is forced to switch channels. From a service provider's point of view it is desirable to maximize the revenue generated by the service. LOREL and MXREV respectively address these two perspectives.

In both these schemes a cell uses its local neighborhood information in deciding the number of guard channels. It is assumed that each cell is aware of the number of reassignments, the number of admitted calls and the number of blocked calls in the neighborhood. Based on this information each cell periodically adjusts its number of guard channels. It is assumed that this period is big enough to capture steady state behavior but much smaller than the

time between changes in traffic conditions. Note that these schemes are characterized by distributed decision making using local information. While each cell acts independently, collectively they have the effect of achieving a common objective. Following subsections describe these schemes in detail.

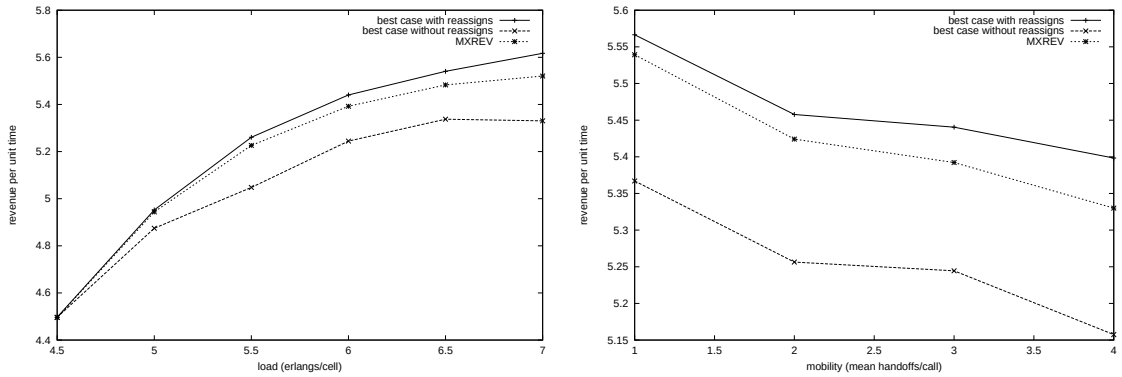
13.4.1 Bounding Reassignments

The bounded reassignment scheme LOREL attempts to contain reassignment frequency below a specified limit r^* as follows. Each cell periodically adjusts the number of guard channels g^*, based on reassignment frequency in the neighborhood. It keeps track of the number of calls admitted, c, and the number of reassignments, r, in the neighborhood since the last adjustment to its g^*. If either the number of new call requests in that cell reaches a threshold value n' or r reaches a threshold value r', then g^* is recomputed as follows.

$$g^* \leftarrow \begin{cases} g^* + 1, & r/c > r^*, \\ \max(g^* - 1, 0), & r/c < r^* - \delta, \\ g^*, & otherwise. \end{cases} \tag{13.1}$$

In each period the number of guard channels is increased if the reassignment frequency is above the given limit r^* and it is decreased if the frequency is below $r^* - \delta$. When the reassignment frequency is within the range $(r^* - \delta, r^*)$ the number of guard channels remains unchanged. The parameter δ is used to insure against under-utilization by not allowing the number of reassignments to fall too far below the specified limit. By making cells recompute their g^* values whenever the number of reassignments reaches the threshold value, LOREL ensures that corrective action is taken whenever congestion is seen in the neighborhood. On the other hand, by recomputing g^* periodically based on the number of new call arrivals, it ensures that the bandwidth does not go underutilized because of excess guard channels.

Figure 13.4 compares the performance of LOREL with GUARD and HOREL. For the purpose of this experiment, we chose, $r^* = 0.02$, $\delta = 0.005$, $n' = 25$, and $r' = 20$. Thus, the number of guard channels g^* in a cell is adjusted after every 25 new call arrivals in that cell or after 20 reassignments in its neighborhood. If the reassignment frequency is above 0.02, g^* is incremented and it is decremented if reassignment frequency falls below 0.015. g^* stays unchanged if the reassignment frequency is between 0.015 and 0.02. From the figure, it can be seen that reassignment frequency is contained under the given limit



(a) varying load

(b) varying mobility

Figure 13.5: Revenue as a function of load and mobility

of 0.02 at all loads. Comparing the blocking probabilities and reassignment frequencies of LOREL with that of HOREL, it can be said that LOREL achieves substantial reduction in the reassignment frequency for a relatively small increase in the blocking probability. Compared to GUARD, the blocking probability of LOREL is significantly lower at low loads and the difference narrows at high loads. Further, it should be noted that there were no dropped calls under LOREL. To sum up, the combination of controlled reassignments and dynamically determined guard channels performs well by adapting to the traffic conditions.

13.4.2 Maximizing Revenue

Reassignments effectively eliminate handoff failures and also make it possible for the system to be less conservative in reserving guard channels resulting in higher utilization. But reassignments may inconvenience users and incur communication and processing overheads. A service provider would like to use reassignments only if they are profitable. From this point of view, an ideal scheme is one that maximizes the revenue V, given that each call serviced brings in reward R and each reassignment incurs penalty P. We now describe a scheme, called MXREV, that attempts to maximize the revenue given R and P. We do not consider the penalty for call dropping since reassignments almost completely eliminate them.

Each cell keeps track of the number of new calls blocked b and the number of reassignments r in its neighborhood. To maximize the revenue, MXREV tries to minimize L, the sum of the penalty due to reassignments and the loss in revenue due to blocked calls, $b \times R + r \times P$,

by dynamically adjusting the number of guard channels, g^*. Note that changing g^* has opposing effects on b and r. As in the case of LOREL, the guard channels are adjusted whenever the r reaches a threshold r' or the number of new call requests n reaches a threshold n'.

$$g^*_{i+1} \leftarrow \begin{cases} 2 \times g^*_i - g^*_{i-1}, & L_i < L_{i-1}, \\ g^*_{i-1}, & otherwise. \end{cases} \tag{13.2}$$

The cells remember the values of L and g^* for the last two periods i and $i-1$. If the loss in revenue decreased from L_{i-1} to L_i, the number of guard channels, g^* will be moved in the same direction for the next period $i+1$ as was done in transitioning from period $i-1$ to i. Otherwise, g^* will be moved in the opposite direction. In any case, g^* is always moved in steps of 1.

Intuitively, this scheme performs a blind hill-climbing. For a given load and mobility, the revenue peaks at a specific number of guard channels. But due to changing traffic conditions this is a moving peak. In general, it is not possible to determine if the peak has been reached. The scheme reverses direction whenever it senses that it is moving down the hill (when loss in revenue increases) and moves in the same direction as long as it is climbing the hill (when loss in revenue decreases). Thus the number of guard channels hovers around the optimal value for the current traffic conditions.

Figure 13.5(a) compares the revenue generated by MXREV with the best observed revenues with and without reassignments under varying load when both P and R are 1. The best revenue with reassignments was obtained for each load by selecting the maximum revenue from the experiments with different guard channels. The best revenue without reassignments was computed by considering the case that achieved a dropping probability less than $1/10000$ with the minimum number of guard channels. This limit on dropping probability was chosen for fair comparison since the dropping probability for MXREV is almost zero. Figure 13.5(b) shows a similar comparison under varying mobility. In both these figures, the difference in revenue between the two best cases shows the potential benefit from using reassignments. The figures show that MXREV closely approximates the performance of the best case with reassignments. Fixed guard channel schemes and even adaptive schemes that do not use reassignments are unlikely to achieve the performance of the best case without reassignments (the bottom curve). The use of reassignments allows the luxury of reacting to the traffic conditions just in time. This explains the superior performance of MXREV.

Note that the revenue increases with increasing load and gradually flattens out. As the offered load increases the amount of calls serviced also increases. But the capacity to admit more calls decreases gradually and hence the incremental gains tend to grow smaller. On the contrary, the revenue decreases with increasing mobility. Due to increase in the number of handoffs more guard channels are required thus increasing blocking probability. The number of reassignments also increases. Each contributes to the decline in revenue.

13.5 Conclusions

We showed that reassignments when used in a controlled manner increase the channel utilization without sacrificing the quality of service. We also showed how guard channels can be used to control reassignments. We presented two schemes that are based on reassignments which dynamically adjust guard channels in each cell adapting to the local traffic conditions. Simulation results showed that these schemes perform well in balancing the number of reassignments and the number of admitted calls while ensuring that almost no dropping probability of admitted calls. Issues involved in employing these schemes in real cellular systems need to be addressed. An important aspect of these schemes is their local decision making in fixing the number of guard channels. Its effect on fairness in resource distribution among different cells need to be analyzed.

Bibliography

[1] A. Albanese, J. Blomer. J. Edmonds, M. Luby and M. Sudan, “Priority Encoding Transmission”, IEEE Transactions on Information Theory, 42(6):1737-1744, November 1996.

[2] G. Apostolopoulos, R. Guerin, S. Kamat, S. Tripathi, “Quality of Service Based Routing: A Performance Perspective”, ACM SIGCOMM 1998.

[3] G. Apostolopoulos, R. Guerin, S. Kamat, S. Tripathi, “Improving QoS Routing Performance under Inaccurate Link State Information,” ITC’16, pp. 1351-1362, June 1999.

[4] G. Ash, *Dynamic Routing in Telecommunications Networks*, McGraw-Hill, 1998.

[5] ATM Forum, Private Network-Network Interface, Specification Version 1 (PNNI 1.0), March 1996.

[6] D. Awduche, J. Malcolm, J. Agogbua, M. O’Dell, J. McManus, “Requirements for Traffic Engineering over MPLS,” RFC-2702, September 1999.

[7] S. Bajaj, L. Breslau, and S. Shenkar, “Uniform versus Priority Dropping for Layered Video”, Proc. ACM SIGCOMM’98, Sep 1998.

[8] R.Callon, P. Doolan, N. Feldman, A. Fredette, G. Swallow, and A. Viswanathan, “A Framework for Multiprotocol Label Switching,” IETF Internet Draft, Work in Progress, September 1999.

[9] S. Chen and K. Nahrstedt, “An Overview of Quality-of-Service Routing for the Next Generation High-Speed Networks: Problems and Solutions,” IEEE Network Magazine, Special Issue on Transmission and Distribution of Digital Video, 1998.

[10] S. Chen and K. Nahrstedt, “Distributed Quality-of-Service Routing in High-Speed Networks Based on Selective Probing,” LCN’98, October 1998.

[11] J. Chen, P. Druschel, and D. Subramanian, “A New Approach to Routing with Dynamic Metrics”, IEEE INFOCOM 1999.

[12] Chih-Lin I and Pi-Hui Chao, “Local Packing - Distributed Dynamic Channel Allocation at Cellular Base Station,” *IEEE GLOBECOM* ’93.

[13] Chih-Lin I, “Distributed Dynamic Channel Allocation Algorithms in Microcells Under Light Traffic Loading,” in *Proc. IEEE ICC*, 1993.

[14] L. J. Cimini, G. J. Foschini, and Chih-Lin I, “Call Blocking Performance of Distributed Algorithms for Dynamic Channel Allocation in Microcells,” in *Proc. IEEE ICCC*, 1992.

[15] T. Coleman and Y. Li, “An Interior, Trust Region Approach for Nonlinear Minimization Subject to Bounds,” in *Journal on Optimization, Vol. 6*, pp. 418–445, SIAM, 1996.

[16] D. Cox, and D.Reudnik, “Increasing Channel Occupancy in Large-Scale Mobile Radio Systems: Dynamic Channel Reassignment”, *IEEE Transactions on Communications*, 21(11), November 1973.

[17] E. Crawley, R. Nair, B. Rajagopalan, H. Sandick, “A Framework for QoS-Based Routing in the Internet”, RFC 2386, August 1998.

[18] W. Ding, “Joint Encoder and Channel Rate Control of VBR Video over ATM”, IEEE Trans. Circuit Syst. Video Technol., vol. 7, no. 2, pp. 266-278, Apr. 1997.

[19] N. G. Duffield, K. K. Ramakrishnan, and A. Reibman, “SAVE: An Algorithm for Smoothed Adaptive Video over Explicit Rate Networks”, IEEE INFOCOM’98, San Francisco, CA, March 1998.

[20] R. F. Farmer and I. Kaufman, “On the Numerical Evaluation of Some Basic Traffic Formulae,” in *Networks, Vol 8.*, pp. 153–186, John Wiley and Sons, Inc., 1978.

[21] Wu-chi Feng, “Rate-constrained Bandwidth Smoothing for the Delivery of Stored Video”, SPIE Multimedia Computing and Networking 1997.

[22] Wu-chi Feng, “Video-on-Demand Services: Efficient Transportation and Decompression of Variable Bit Rate Video”, Ph.D. Thesis, Univ. of Michigan, April 1996.

[23] W. Feng and J. Rexford, “A Comparison of Bandwidth Smoothing Techniques for the Transmission of Prerecorded Compressed Video,”, in Proc. IEEE INFOCOM, pp 58-66, April 1997.

[24] M. Garrett and W. Willinger, “Analysis, Modeling and Generation of Self-Similar VBR Video Traffic”, Proc. ACM SIGCOMM, pp. 269-280, Aug. 1994.

[25] R.J. Gibbens, F.P. Kelly, and P.B. Key, “Dynamic Alternative Routing: Modelling and Behaviour”, Teletraffic Science, pp. 1019-1025, Elsevier, Amsterdam, 1989.

[26] R. Gopalakrishnan, J. Griffoen, G. Hjalmtysson, and C. Sreenan, “Stability and Fairness Issues in Layered Multicast”, Proc. NOSSDAV’99, Jun 1999.

[27] R. Gopalakrishnan, J. Griffoen, G. Hjalmtysson, C. Sreenan, and S. Wen, ‘A Simple Loss Differentiation Approach to Layered Multicast”, Proc. IEEE INFOCOM’00, Tel-Aviv, Israel, Mar 2000.

[28] M. Grossglauser, S. Keshav and D. Tse, “RCBR : A Simple and Efficient Service for Multiple Time-Scale Traffic”, Proc. ACM SIGCOMM, pp. 219-230, Aug 1995.

[29] R. Guerin, S. Kamat, A. Orda, T. Przygienda, D. Williams, “QoS Routing Mechanisms and OSPF Extensions”, *Work in Progress*, Internet Draft, March 1997.

[30] R. Guerin, A. Orda, “QoS-Based Routing in Networks with Inaccurate Information: Theory and Algorithms”, IEEE Infocom 1997.

[31] F. Hao, and E.W. Zegura, “On Scalable QoS Routing: Performance Evaluation of Topology Aggregation”, IEEE INFOCOM 2000. 1997.

[32] D. Hong and S.S. Rappaport, “Traffic Model and Performance Analysis for Cellular Mobile Radio Telephone Systems with Prioritized and Nonprioritized Handoff Procedures,” *IEEE Transactions on Vehicular Technology*, 35(3), 1986.

[33] C. Hsu, A. Ortega and A. Reibman, “Joint Selection of Source and Channel Rate for VBR Video Transmission under ATM Policing Constraints”, IEEE Journal on Selected Areas in Communication, 1997.

[34] A.A. Jagers and E.A. Van Doorn, “On the Continued Erlang Loss Function,” Op. Res. Lett., vol. 5, pp. 43-46, 1986.

[35] D.L. Jagerman, “Some properties of the Erlang Loss function,” Bell Systems Technical Journal, vol. 53, No. 3, March 1974.

[36] D.L. Jagerman, “Methods in traffic calculation,” AT&T Bell Lab Technical Journal, vol. 63, No. 7, September 1984.

[37] Z. Jiang and L. Kleinrock, “A General Optimal Video Smoothing Algorithm”, INFOCOM 1998.

[38] H. Kanakia, P. P. Mishra, and A. Reibman, “Packet Video Transport in ATM Networks with Single-bit Feedback”, In Proc. of the Sixth International Workshop on Packet Video, Portland, Oregon, Sept. 1994.

[39] I. Katzela and M. Naghshineh, “Channel Assignment Schemes for Cellular Mobile Telecommunication Systems: A Comprehensive Survey”, *IEEE Personal Communications*, June 1996.

[40] J.S. Kaufman, “Blocking in a Shared Resource Environment,” IEEE Trans. Commun. vol. COM-29, pp. 1474-1481, 1981.

[41] F.P. Kelly, “Routing in Circuit-Switched Networks: Optimization, Shadow Prices and Decentralization”, Advances in Applied Probability 20, 112-144, 1988.

[42] F.P. Kelly, “Routing and capacity Allocation in Networks with Trunk Reservation”, Mathematics of Operations Research, 15:771-793, 1990.

[43] F.P. Kelly, “Dynamic Routing in Stochastic Networks”, In *Stochastic Networks*, ed. F.P. Kellyand R.J. Williams, Springer-Verlag, 169-186, 1995.

[44] F.P. Kelly, “Loss Networks,” The Annals of Applied Probability, vol. 1, pp. 319-378, 1991.

[45] F.P. Kelly, “Fixed Point Models of Loss Networks,” J. Austr. Math. Soc., Ser. B, 31, pp. 204-218, 1989.

[46] P.B. Key, and G.A. Cope, “Distributed Dynamic Routing Schemes”, IEEE Communucations Magazine, pp. 54-64, Oct 1990.

[47] Murali Kodialam, and T. V. Lakshman, “Minimum Interference Routing with Applications to MPLS Traffic Engineering”, INFOCOM 2000.

[48] T. Korkmaz and M. Krunz, “Source-oriented Topology Aggregation with Multiple QoS Parameters in Hierarchical ATM Networks”, IWQOS 1999.

[49] M. Krunz and S.K. Tripathi, “On the Characteristics of VBR MPEG Streams”, in Proc. ACM SIGMETRICS, pp. 192-202, June 1997.

[50] T.V. Lakshman, A. Ortega and A.R. Reibman, “Variable bit-rate (VBR) video: Tradeoffs and Potentials,” Proceedings of the IEEE, vol. 86, May 1998.

[51] T. V. Lakshman, P. P. Mishra, and K. K. Ramakrishnan, “Transporting Compressed Video over ATM Networks with Explicit Rate Feedback Control”, In Proc. of the IEEE INFOCOM’97 Conference, Kobe, Japan, Apr. 1997.

[52] W.C. Lee, “Topology Aggregation for Hierarchical Routing in ATM Networks”, Computer Communications Review, vol. 25, no. 2, pp. 82-92.

[53] D. Levine, I. Akylidz, and M. Naghshineh, “A Resource Estimation and Call Admission Algorithm for Wireless Multimedia Networks Using the Shadow Cluster Concept,” *IEEE/ACM Transactions on Networking*, 5(1), February 1997.

[54] W. Luo and M. El Zarki, “Quality control for VBR video over ATM networks”, IEEE Journal on Selected Areas in Communication, vol. 15, no. 6, pp. 1029-1039, Aug. 1997.

[55] Q. Ma, P. Steenkiste, and H. Zhang, “Routing High Bandwidth Traffic in Max-Min Fair Share Networks,” In *Proc. ACM SIGCOMM’96*, October 1996, Stanford, CA.

[56] Q. Ma, P. Steenkiste, “On Path Selection for Traffic with Bandwidth Guarantees”, IEEE ICNP 1997.

[57] Q. Ma, P. Steenkiste, “Routing Traffic with Quality-of-Service Guarantees in Integrated Services Networks”, NOSSDAV 1998.

[58] Q. Ma, “Quality-of-Service Routing in Integrated Services Networks”, Ph.D Dissertation, School of Computer Science, Carnegie Mellon University, January 1998.

[59] S. McCanne, V. Jacobson, and M. Vetterli, “Receiver-driven Layered Multicast,” ACM SIGCOMM 1996

[60] J.M. McManus and K.W. Ross, “Video on Demand over ATM: Constant-rate Transmission and Transport”, Proc. IEEE INFOCOM, pp. 1357-1362, March 1996.

[61] D. Mitra, and J.B. Seery, “Comparative Evaluations of Randomized and Dynamic Routing Strategies for Circuit-Switched Networks”, IEEE Trans. on Communications, vol. 39, no. 1, pp. 102-116, January 1991.

[62] D. Mitra, J.A. Morrison, and K.G. Ramakrishnan, “ATM Network Design and Optimization: A Multirate Loss Network Framework,” IEEE/ACM Transactions on Networking, vol. 4, no. 4., pp. 531-543, August 1996.

[63] M. Montgomery and G. de Veciana, “Hierarchical Source Routing through Clouds”, IEEE INFOCOM 1998.

[64] J.A. Morrison, K.G. Ramakrishnan, and D. Mitra, “Refined asymptotic approximations to loss probabilities and their sensitivities in shared unbuffered resources,” SIAM J. Appl. Math., 1998.

[65] J. Moy, “OSPF Version 2”, Request For Comments 2328, Internet Engineering Task Force, April 1998.

[66] M. Naghshineh and M. Schwartz, “Distributed Call Admission Control in Mobile/Wireless Networks,” *IEEE Journal on Selected Areas in Communications*, 14(4), May 1996.

[67] K.S. Narendra and P. Mars, “The Use of Learning Algorithms in Telephone Traffic Routing - A Methodlogy”, Automatica, vol. 19, no. 5, pp. 495-502, 1983.

[68] K.S. Narendra and M.A.L. Thathachar, “On the Behavior of a Learning Automaton in a Changing Environment with Application to Telephone Traffic Routing”, IEEE Trans. on Systems, Man and Cybernetics, vol. SMC-10, no.5, pp. 262-269, May 1980.

[69] S. Nelakuditi, D.H.C Du, and Z.-L. Zhang, “Channel Selection Strategies for Compact Local Packing,” Technical Report, July 1997.

[70] S. Nelakuditi, Z.-L. Zhang, and D.H.C. Du, “Reassignment based Call Admission Control for Cellular Networks,” IEEE LANMAN’98, May 1998.

[71] S. Nelakuditi, R.R. Harinath, S. Rayadurgam and Z.-L. Zhang, ”Revenue Based Call Admission Control for Wireless Cellular Networks”, ICPWC’99, Jaipur, India, Feb 1999.

[72] S. Nelakuditi, R.R. Harinath, S. Varadarajan, and Z.-L. Zhang, “Adaptive Layer Discard Schemes for Stored Video Delivery over Wireless Networks”, IEEE LAN-MAN’99, November 1999.

[73] S. Nelakuditi, R.R. Harinath, E. Kusmierek, Z.-L. Zhang, “Providing Smoother Quality Layered Video Stream”, In *Proceedings of NOSSDAV’00*, Raleigh, June 2000.

[74] S. Nelakuditi, R.P. Tsang, Z.-L. Zhang, “Quality-of-Service Routing without Global Information Exchange”, IWQOS 1999.

[75] S. Nelakuditi, Z.-L. Zhang, and R.P. Tsang, “Adaptive Proportional Routing: A Localized QoS Routing Approach”, IEEE INFOCOM’00, March 2000.

[76] S. Nelakuditi, Z.-L. Zhang, R.P. Tsang, and D.H.C. Du, “Adaptive Proportional Routing: A Localized QoS Routing Approach”, Submitted to Transactions on Networking.

[77] S. Nelakuditi, S. Varadarajan, and Z-L. Zhang, “On Localized Control in Quality-of-Service Routing”. Submitted to IEEE Transactions on Automatic Control, Special Issue on Systems and Control Methods for Communication Networks.

[78] S. Nelakuditi, and Z.-L. Zhang, “On Selection of Paths for Multipath Routing”, IWQOS’01, June 2001.

[79] S. Nelakuditi, and Z.-L. Zhang, “Localized Adaptive Proportioning Approach to QoS Routing”. Submitted to IEEE Communications Magazine.

[80] S. Nelakuditi and Z.-L. Zhang, “Hierarchical Multipath Routing”, Submitted to ICC’02, April 2002.

[81] N. Taft-Plotkin, B. Bellur, and R. Ogier, “Quality-of-Service Routing using Maximally Disjoint Paths”, IWQOS 1999.

[82] M. Powell, “A Fast Algorithm for Nonlinear Constrained Optimization Calculations,” in *Numerical Analysis, ed. G.A. Watson, Lecture Notes in Mathematics, Vol 630*, Springer Verlag, 1978.

[83] S. Ramanathan, P.V. Rangan and H.M. Vin, “Frame-Induced Packet Discarding: An Efficient Strategy for Video Networking”, In Proceedings of the Fourth International Workshop on Network and Operating Systems Support for Digital Video and Audio, Lancaster, UK, pp. 175-187, November 1993.

[84] R. Ramjee, D. Towsley, and R. Nagarajan, “On Optimal Call Admission Control in Cellular Networks,” *ACM/Baltzer Journal of Wireless Networks*, 3(1), 1997.

[85] A.R. Reibman and B.G. Haskell, “Constraints on Variable Bit-Rate Video for ATM Networks”, IEEE Transactions on Circuits and Systems for Video Technology, 2(4):361-372, Dec. 1992.

[86] R. Rejaie, M. Handley, and D. Estrin, “RAP: An End-to-End Rate-based Congestion Control Mechanism for Realtime Streams in the Internet”, Proc. IEEE INFOCOM’99, Mar 1999.

[87] R. Rejaie, D. Estrin, and M. Handley, “Quality Adaptation for Congestion Controlled Video Playback over the Internet”, Proc. ACM SIGCOMM’99, Cambridge, Sep 1999.

[88] J. Rexford, S. Sen, J. Dey, W. Feng, J. Kurose, J. Stankovic, and D. Towsley, “Online Smoothing of Live, Variable-bit-rate Video”, in Proc. Workshop on Network and Operating System Support for Digital Audio and Video, pp. 249-257, May 1997.

[89] J. Rexford and D. Towsley, “Smoothing variable-bit-rate video in an internetwork,” in *Proc. SPIE Symposium on Voice, Video, and Data Communications: Multimedia Networks: Security, Displays, Terminals, and Gateways*, November 1997.

[90] J.W. Roberts, “Teletraffic Models for the Telecom 1 Integrated Services Network,” in Proc. Internet Teletraffic Congress-10, Session 1.1, paper #2.

[91] J. Roberts, U. Mocci, and J. Virtamo, “Broadband Network Teletraffic,” LNCS 1155, Springer Verlag, 1996.

[92] E. Rosen, A. Viswanathan, and R. Callon, “Multiprotocol Label Switching Architecture,” IETF Internet Draft, Work in Progress, August 1999.

[93] K.W. Ross, “Multiservice Loss Models for Broadband Telecommunication Networks”, Springer-Verlag, 1995.

[94] L.A. Rowe, K.D. Patel, B.C. Smith and K. Liu, “MPEG Video in Software : Representation, Transmission and Playback“, IS&T/SPIE, Symp. on Elec. Imaging Sci. & Tech., San Jose, CA, February 1994.

[95] L.A. Rowe and B.C. Smith, “A Continuous Media Player”, Proceedings 3rd International Workshop on Network and Operating System Support for Digital Audio and Video, San Diego, CA, November 1992.

[96] S. Sahu, Z.-L. Zhang, J. Kurose and D. and Towsley, “On the efficient retrieval of VBR video in a multimedia server”, In *Proc. IEEE International Conference on Multimedia Computing and Systems'97*, pp. 46-53, June 1997, Ottawa, Ontario, Canada.

[97] J.D. Salehi, Z.-L. Zhang, J. F. Kurose and D. Towsley, “Supporting Stored Video: Reducing Rate Variability and End-to-End Resource Requirements through Optimal Smoothing”, Proc. ACM SIGMETRICS, May 1996.

[98] D. Saparilla, and K. Ross, “Optimal Streaming of Layered Video”, Proc. IEEE INFOCOM'00, Tel-Aviv, Israel, Mar 2000.

[99] M. Saquib, R. Yates, “Optimal Call Admission to a Mobile Cellular Network,” *IEEE Vehicular Technology Conference*, 1995.

[100] H. Schwetman, “CSIM Reference Manual (Revision 15),” June 1991.

[101] S. Sen, D. Towsley, Z.-L. Zhang and J. Dey, “Optimal Multicast Smoothing of Streaming Video over an Internetwork”, Technical Report, Computer Science Department, University of Massachusetts, July, 1998.

[102] S. Servetto, K. Ramachandran, K. Nahrstedt and A. Ortega, “Optimal Segmentation of a VBR Source for its Parallel Transmission over Multiple ATM Connections”, In Proceedings of the International Conference on Image Processing, 1997.

[103] A. Shaikh, J. Rexford, K. Shin, “Evaluating the Overheads of Source-Directed Quality-of-Service Routing”, ICNP 1998.

[104] A. Shaikh, J. Rexford, K. Shin, “Efficient Precomputation of Quality-of-Service Routes”, NOSSDAV 1998.

[105] A. Shaikh, J. Rexford, K. Shin, “Load-Sensitive Routing of Long-Lived IP Flows”, ACM SIGCOMM 1998.

[106] P.R. Srikantakumar and K.S. Narendra, “A Learning Model for Routing in Telephone Networks”, SIAM J. Control and Optimization, vol. 20, no. 1, pp. 34-57, January 1982.

[107] O. Verscheure, X. Garcia, G. Karlsson, and J.P. Hubaux, “User-Oriented QoS in Packet Video Delivery”, IEEE Network, November/December 1998.

[108] C. Villamizar, “OSPF Optimized Multipath (OSPF-OMP), ” Internet Draft, February 1999.

[109] C. Villamizar, “MPLS Optimized Multipath (MPLS-OMP)”, Internet Draft, February 1999.

[110] Z. Wang, J. Crowcroft, “Quality-of-Service Routing for Supporting Multimedia Applications”, IEEE JSAC Sept 1996

[111] A. Webster *et al*., “An Objective Video Quality Assessment System based on Human Perception”, Proc. SPIE-Human Vision, Visual Processing and Digital Display, vol. 1913, pp. 15-26.

[112] D. Wijesekera and J. Srivastava, “Quality of Service (QoS) Metrics for Continuous Media”, Multimedia Tools and Applications, Vol 2, No 3, Sept 1996, pp. 127-166.

[113] E. W. Zegura, K.L. Calvert, and S. Bhattacharjee, “How to Model an Internetwork,” IEEE INFOCOM 1996.

[114] J. Zhang and J. Y. Hui, “Traffic Characteristics and Smoothness Criteria in VBR Video Traffic Smoothing”, IEEE International Conference on Multimedia Computing and Systems, June 1997.

[115] Z. Zhang, C. Sanchez, B. Salkewicz, E. Crawley, “Quality of Service Extensions to OSPF”, *Work in Progress*, Internet Draft, September 1997.

[116] Z.-L. Zhang, J. Kurose, J.D. Salehi and D. Towsley, “Smoothing, Statistical Multiplexing and Call Admission Control for Stored Video”, IEEE Journal on Selected Areas in Communication, Special Issue on Real-Time Video Services in Multimedia Networks, August 1997.

[117] Z.-L. Zhang, S. Nelakuditi, R. Aggarwal, and R.P. Tsang, “Efficient Selective Frame Discard Algorithms for Stored Video Delivery across Resource Constrained Networks”, IEEE INFOCOM’99, March 1999.

[118] Z.-L. Zhang, S. Nelakuditi, R. Aggarwal, and R.P. Tsang, “Efficient Selective Frame Discard Algorithms for Stored Video Delivery across Resource Constrained Networks”, Special Issue on Adaptive Real Time Multimedia Transmission over Packet Switching Networks in Journal of Real Time Imaging, June 2001.