ABSTRACT

Title of Dissertation: Fast Timescale Traffic Engineering in MPLS Networks

Surapich Phuvoravan, Doctor of Philosophy, 2004

Dissertation directed by: Professor Mark A. Shayman
Department of Electrical and Computer Engineering

Traffic engineering can be used to solve congestion through efficient traffic distribution. The network with differentiated classes of service is the main focus in this work. In general, providers offer QoS guaranteed in high priority class by serving the packets from that class before the packets from low-priority class. This strategy will work as long as the high priority traffic share is low (5-10%). We are interested in increasing that high priority traffic share. However, the large share of high priority class could lead to a QoS-violation risk. Moreover, it could decrease the goodput of the low-priority traffic as a result of its elastic nature, for example, from the congestion control in TCP. We present two works based on the fast timescale traffic engineering. The first work focuses on the flow migration among the parallel working paths. The potential usefulness of fast timescale migration control is explored with packet-level simulation and experimental testbed. The migration can improve the QoS of real-time

traffic by shifting some of the flows out of the congested links. In the design of the migration controller, we will also consider the interaction between the controller and the TCP congestion control mechanism. We formulate the partially observed Markov Decision Process (MDP) problem and solve it using the dynamic programming technique. To cope with the well-known curse of dimensionality, we also present sub-optimal controllers, which scale well with our large network. Our second work tried to exploit the unused backup path for QoS improvement. Many network providers already setup a backup path for each working path. This backup path will protect the working path upon its failure. Without any failures, the providers fill the backup path with lower-priority traffic to increase their network's throughput. In general, the backup paths will not be used for high priority traffic in normal condition even with the congestion situation of the high priority traffic. We propose the scheme to duplicate the high priority packets to backup path in the congestion condition. With a small throughput degradation of low-priority traffic, we could gain significant QoS improvement of high priority class.

Fast Timescale Traffic Engineering in MPLS Networks

by

Surapich Phuvoravan

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2004

Advisory Committee:

Professor Mark A. Shayman, Chairman/Advisor
Assistant Professor Bobby Bhattacharjee
Assistant Professor Richard J. La
Professor K. J. Ray Liu
Professor A. Udaya Shankar

DEDICATION

To my family

ACKNOWLEDGEMENTS

While this dissertation bears my name, it is equally the product of many people over many years who have challenged me, supported my work, inspired me, and made great sacrifices toward this final goal.

Without the commitment of my advisor and mentor, Dr. Mark Shayman, this dissertation would never have been started or completed. He strove to develop my research logic and my ability to analyze data critically. He was and is always willing to take time to meet me, discuss my research process and my frustration, and carefully read and comment on my draft. If I earned the privileged title “doctor of philosophy”, it is not because I have completed the designated requirement, but because I was fortunate to have worked with Dr. Shayman.

I want to thank all of the committee for shaping my ideas. I am grateful to have worked with Dr. Mark Shayman, Dr. Richard La, Dr. Samrat Bhattacharjee and Dr. Steve Marcus. Each member brought strength and interest to the project that led to a more interesting final dissertation. They also have made thoughtful comments for accuracy and clarity. I am very grateful to Dr. K. J. Ray Liu and Dr. Udaya Shankar for taking time to be on my dissertation committee. I would like to thank Robert Jaeger, who tutored me during my early research life.

I want to thank my family and friends, who constantly support me. I have relied on extraordinary personal and research advice from Dr. Kritchalach Thitikamol, who has

become my big brother, Dr. Apinun Tunpan and Dr. Songrit Maneewongvatana. I also would like to thank Lisa Mangkonkarn (N'Lisa) and Paweena Chittiwuttinon (N'Pui) for checking the correctness of my dissertation. Special thanks to Tuna Guven, my great friend and colleague, for wonder personal and research discussion. I would like to acknowledge the support and encouragement of my little sister and my little brother, Navaporn and Kitjapat Phuvoravan. They are always a special inspiration for me. I also want to give heartfelt thank to special one, Chalermkwan Chantarat, who support and believe in me.

My parents, Somboon and Nuchara Phuvoravan, have been my mind and my inspiration during every minute of my studies. They helped me get around seemingly impossible blockages. They knew from the beginning that I would reach this goal, even when I did not believe it myself. Dad and Mom, I love you.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

1.1 Motivation and Goal

It has been observed that the efficiency and cost structure of the service provider business are affected by how efficiently a service provider utilizes its infrastructure and, specifically, the available bandwidth[7]. The provider should be able to manage the network in such a way to avoid causing some parts of the network to be over-utilized while the other parts are under-utilized. This traffic engineering helps service providers to gain an advantage over their competitors. This work concentrates on dynamic traffic engineering in a provider domain that implements Multiprotocol Label Switching (MPLS). In a MPLS network, a short fixed length, locally significant label is used as an identifier to a stream. The IP packets can be routed based on the label. A router, which supports MPLS, is known as a Label Switching Router (LSR). Tunnels are set up between ingress and egress LSRs. Tunnels, which are called Label Switched Paths (LSPs), can be explicitly routed to satisfy traffic engineering objectives. Multiple LSPs can be established for given ingress-egress pairs. If desired, a bandwidth reservation can be associated with an LSP; the associated bandwidth is referred to as the provisioned bandwidth. Nearly all major providers have either

deployed MPLS or are planning to do so in the future.

There is an approach to do traffic engineering in a MPLS network based on the collected statistics of the network, for example, past day traffic demands. The so-called traffic matrix can be estimated and constructed from the collected statistics. The off-line optimization uses the traffic matrix to compute a globally optimal static set of provisioned LSPs. Algorithms for the optimization are based on both linear and nonlinear programming[3]. If the traffic matrix is a time varying function of the time-of-day, the several sets of tunnel configurations can be computed and used for different periods during the day. A very different approach is proposed in [20]. No prior estimates of traffic are needed in this approach. LSPs are constructed for newly arriving traffic demands in such a way that minimum interference to the unknown future demand is achieved. The combination of these two approaches is in [37]. The estimated traffic demands form the multi-community flow problem, which can be solved numerically. The precomputed solution is used as the guideline to construct LSPs upon the arrival of demand.

Unless the network is significantly overprovisioned, a static configuration of LSPs may not be able to provide for variations of traffic load about the estimates in the traffic matrix. This suggests the need for a fast timescale control. Moreover, this fast timescale controller can be built on top of any static off-line optimization, which is more attractive. From the above motivation, we will focus on fast timescale traffic engineering in this dissertation.

Our fast timescale traffic engineering is divided into two different works. Our two works use different control mechanisms, which can be designed independently. However, they can be implemented together to let us better utilize the existing network resources. The first work uses the flow migration as a mechanism to do fast timescale

traffic engineering. In this work, we focus on a network that already has a provisioned set of parallel LSPs between the ingress-egress node pair. As a result, a flow migration mechanism can exploit these parallel structure by freely moving flows among these paths. In the dissertation, we will start exploring the usefulness of the migration mechanism in high priority traffic network. Then, we will focus on the network with two different classes of service, which are the high priority traffic class and the low priority traffic class.[1] We will design a migration mechanism to help increasing the QoS in high priority traffic without compromising the goodput of low priority traffic. As a result, the network can accept more traffic while keeping the same QoS level. We will formulate the design problem as the Markov decision process and use the dynamic programming technique to obtain a good migration strategy. Note that the control information might be obsolete from the relative long delay (compared to a fast timescale of the problem). Hence, there is also a need to include this long delay into our formulation.

The second work uses a duplication mechanism to do fast timescale traffic engineering. In this work we focus on the MPLS network with a QoS guaranteed backup path. The backup path is provisioned to increase the reliability in the case of failures in the main active path. This backup path is typically unused by high priority traffic when there are no failures. Instead, it is normally filled with low priority traffic. Our goal is to use this backup path in the congestion condition of high priority traffic, even though there are no failures. We use a control mechanism, in which we call a duplication mechanism to exploit this backup path. In the duplication mechanism, we duplicate high priority packets and then send them over to both the main active path and the backup path during the periods of congestion. The duplication mechanism can

[1] We will talk about these two classes in detail with the link model in Section 1.2.

improve the QoS of high priority traffic while it degrades a throughput of low priority traffic. We will use the dynamic programming technique to obtain a good duplication strategy, which determines when we should duplicate packets. The strategy should perform a trade-off so that it provides a good QoS for high priority traffic and gives an adequate low priority throughput.

1.2 Background: Differentiated Service and Link Model

For the entire dissertation, we divide traffic into two classes: one class for normal internet traffic and the other class for real-time traffic. We can describe these two classes based on a Differentiated Service Model as follows:

The Expedited Forwarding Class (EF class) is composed of flows from real-time applications. This class can also be described as a delay and loss-sensitive class. We also call this class a high priority traffic class, which will be mentioned interchangeably with the EF class.

The Best Effort Class (BE class) is the service class that contains normal Internet flows. The recent study showed that 90% of current Internet traffic uses TCP as a transport protocol to adjust the transmission rate. Hence we assume that traffic in this class has the ability to adjust the transmission rate according to available resources of the network. We also call this class a low priority traffic class, which will be mentioned interchangeably with the BE class.

To get better service to the EF class, we use the link model as shown in Figure 1.1. There are two mechanisms in this link model to handle packets. The first mechanism is a classifier. The classifier will send an incoming packet to the queue associated with

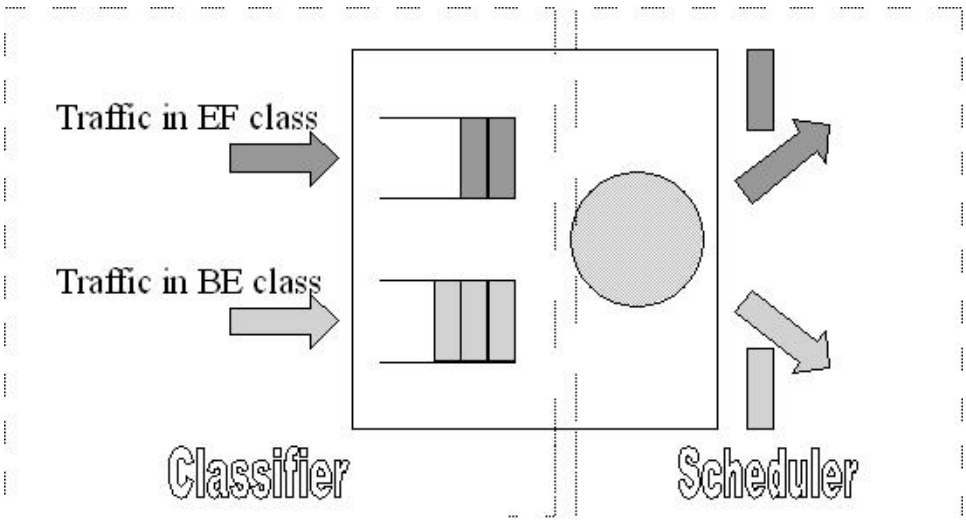


Figure 1.1: The link model with two classes of traffic.

the class of that packet. The second mechanism is a scheduler. The scheduler determines the order of packet transmission. We use the scheduler that always chooses packets in the EF queue before the packets in the BE queue. In other words, the packets from the BE class will not be transmitted unless there are no packets from the EF class waiting in the queue.

With this link mechanism, it is certain that drops and delays of the EF classes are always less than those of the BE class. If an amount of the real-time traffic is always less than a link capacity, the drops of the EF classes can be approximately zero and the delay of it could always be fixed. These conditions are perfect for voice (or video) decoders to replay the good voice (or video) quality at destination. However, there are two drawbacks with this link model. First, there is a chance that real-time traffic cannot meet its QoS requirement even in a carefully designed network. The chance comes from the fact that an amount of packets at any given time has very high variation. At some period of time, the amount of packets is much greater than the amount that link can support. Hence the drops will occur. It leads to the QoS degradation. Second, the

throughput of the BE class can be extremely low even though there are some available resources left. With our link model, the drops of the BE class now depend on the amount of packets in the EF classes. The high variation of packets in EF class could make drops unpredictable. These unpredictable drops can make the TCP protocol behave adversely and response ineffectively to the available resource of the network. Note that the TCP protocol of traffic in the BE class uses drops to adjust the transmission rate.

We can see from our link mechanism that the low priority traffic only gets a link bandwidth of only what is left over from the high priority traffic. Hence, we will call that available bandwidth for the BE class as the TCP service rate, which we will use interchangeably throughout the dissertation. The intuition behind the name, "TCP service rate", can be described by imagining our network having only one class of service (i.e., every link treats all packets the same), and the TCP traffic is the only traffic type there. However, the link capacity or link service rate can fluctuate over time. That link service rate originates the name, TCP service rate. It is simply the service rate that TCP packets will be served in the link.

1.3 Contributions

This dissertation makes the following original contributions:

- We introduce a fast timescale control in a heterogeneous network with two classes of service. We focus on a timescale less than 1 second and possibly on the order of 50 ms.
- We develop a control mechanism that manipulates high priority traffic. The main goal of our control is its QoS's preservation. At the same time, we also consider

the effect of the control to low priority traffic and then incorporate that effect in the design of our control mechanism. By considering both issues at the same time, we can better utilize the network and satisfy both the network provider's need and customer's requirement. Our empirical study shows a clear advantage of the fast timescale traffic engineering, especially when we do traffic engineering intelligently. Moreover, it points out the advantage of having a traffic prediction by reacting in accordance to the prediction. However, this advantage greatly depends on the accuracy of the traffic model in both the voice and video traffic.

Migration Control

- We provide a first step towards a fast timescale migration-based control. This migration-based traffic engineering will move high priority traffic between parallel paths to preserve the QoS of that traffic.

- We develop three controllers for a fast timescale flow migration. These controllers are the LQ, MONTE and IT controllers. The LQ controller uses Linear System with Quadratic Cost (LQ) approach to gain a closed-form approximated solution. This closed-form solution simplifies online calculation and scales well with large state space in our migration problem. Both the MONTE and IT controllers are based on limited lookahead algorithm, and they use a value function from the LQ approach as an approximated value function for the limited lookahead algorithm. To be able to implement online, the MONT controller uses the Monte-Carlo simulation to get an expectation in the limited lookahead algorithm. The IT controller uses the analytical function from the LQ approach in addition to the Monte-Carlo simulation to better estimate the expectation. Hence, the IT controller does not require a large number of simulation samples. As a result, it is well suited for the

fast timescale control, which has only a small processing time and resources in a sample simulation.

- We compare the performance of the designed controllers using both Temperal-Difference (TD) learning and packet-level simulation. We compare them with the other two extreme controllers, which are the NoMig controller and the BAL controller. The NoMig controller is an extreme controller in such a way that we do not have any fast timescale migration control. On the other hand, the BAL controller uses a large amount of migration to keep the network load balanced at all time.

- Our empirical study demonstrates the superiority of both the LQ and IT controllers over the case without migration control, especially in the bursty network environment. Moreover, they yield a greater improvement over the case without migration when the traffic amount is high but not overloaded.

Duplication Control

- To the best of our knowledge, we present the first work that exploits backup paths for congestion control. We explore fast timescale duplication based traffic engineering for this congestion control and the improvement of high priority traffic QoS. This fast timescale control will duplicate high priority packets and send them over both main path and backup to increase QoS.

- We develop three controllers for fast timescale duplication mechanism, which are the NoTCP, MONT and IT controllers. The NoTCP controller is based on Certainty Equivalence Control (CEC) approach. The approximation from the CEC control itself and the ignorance of the TCP congestion control interaction drive

the NoTCP approach away from the optimal solution. However, we gain a closed-form solution that simplifies online calculation and, in turn, scales well with large state space in the duplication problem. Both the MONT and IT controllers use the same concept as the limited lookahead based controllers in the migration problem. However, we use the value function from the NoTCP approach in the lookahead algorithm, instead of that from the LQ approach.

- We compare the performance of the designed controllers using both TD learning and packet-level simulation. We compare them with the other two extreme controllers, the NoDup controller and AllDup controller. The NoDup controller uses an extreme approach that does not send any duplicated packets. On the other hand, the AllDup uses another extreme approach that sends duplicated packets all the time.

- Our empirical study demonstrates the superiority of all intelligent duplication controllers (the NoTCP, MONTE, and IT approaches) over the case without any duplication control, especially in a bursty network environment. Moreover, they yield a greater improvement over the case without duplication when the traffic amount is high.

1.4 Thesis Organization

This dissertation is organized as follows: We begin with describing the various traffic models in Chapter 2. Then, we will present two works based on the fast timescale traffic engineering. We present the first work, which focuses on the migration mechanism among the parallel working path, in Chapter 3, 4, and 5. In Chapter 3, we explore the potential usefulness of fast timescale migration control in the network with

only high priority traffic. Subsequently, we extend our focus to a more realistic network, which is composed of high priority and low priority traffic together in Chapter 4. In this chapter, we also formulate the dynamic programming problem and then provide a state reduction technique to simplify the problem without compromising solution's optimality. In Chapter 5, we provide various sub-optimal solutions to the previous dynamic programming problem. These sub-optimal solutions are designed to cope with the curse of dimensionality in our large problem. We also compare those sub-optimal policies using TD(0) and packet-level simulation. In Chapter 6 and 7, we present our second work. In this work, we exploit the unused backup path for QoS improvement by using a duplication mechanism. We formulate the dynamic programming problem and provide a state reduction technique in Chapter 6. In Chapter 7, we describe various sub-optimal solutions and compare them using TD(0) and the packet-level simulation. Finally in Chapter 8, we summarize the work that has been done in the dissertation and discuss possible future directions of research.

Chapter 2

Traffic Model

There is a long history on traffic modelling for telecommunication networks. The need to guarantee specific quality of service (QoS) levels to users motivate many researchers to attempt capturing the statistical characteristics of actual traffic as a part of developing traffic engineering techniques. This section does not intend to serve as a survey on traffic modelling but it gives an intuitive reason how we chose the traffic model for our experiments and controller design. A detailed discussion and survey on traffic model can be found in [1, 33]. In addition to the intuitive reason in selecting traffic model, we propose two additional TCP models that match the need of our control actions. We will provide a simple validation of these models at the end of the chapter.

2.1 Video Traffic Model

There has been a fair amount of discussion on the impact of long-term correlation of variable-bit-rate(VBR) video traffic on traffic engineering. The discussion is based on the recent discovery of long-range dependence in VBR video traffic. The definition and the detail of long-range dependence and short-range dependence is in [29]. Because a buffer overflow probability of long-range dependent (LRD) video traffic decays

hyperbolically, the notion of effective bandwidth based on Markov models does not apply. This led many researchers to believe that this LRD property of VBR video traffic will have critical impact on traffic engineering in general.

Heyman and Lakshman[16] have shown that the LRD is not a crucial property in determining buffer behavior of VBR video traffic. They observed large short-range dependence (the auto-correlation function for lag one is greater than 0.95) in many video trace sets. Comparing the Markov model with the data traces, a mean queue length of Markov chain model matches with a mean queue length of data trace quite well especially at low to middle traffic intensity.[16, Figure 10] In conclusion, they showed that the Markov chain traffic model can be used to estimate the buffer occupancy well when the buffer sizes are not too large. Small buffer size is desirable in practical network due to the real-time application requirements.

The paper by Bong K. Ryu and Anwar Elwalid[31] strengthened the importance of the short-range dependent traffic component. They introduced the notion of Critical Time Scale (CTS) as the number of frame correlations that contribute to cell loss rate. They used the large deviation theory(with infinite number of sources assumption) to show that CTS is finite and small even in a presence of LRD (under realistic ranges of buffer size and cell loss rate). They also used simulations to get a buffer overflow probability. The results show that the LRD property is not practically important in determining cell loss rate for ATM networks under a realistic buffer dimensioning. Moreover, the short-term correlation have dominant impact on network performance.

Since the Markov model has a lot of nice properties and has been proved for the traffic modelling as described above, we use Markov Modulated Fluid (MMF) model for our experiment. MMF is one type of Markov traffic models that is sufficient for aggregated video traffic. The fluid models are simple and good for simulation and

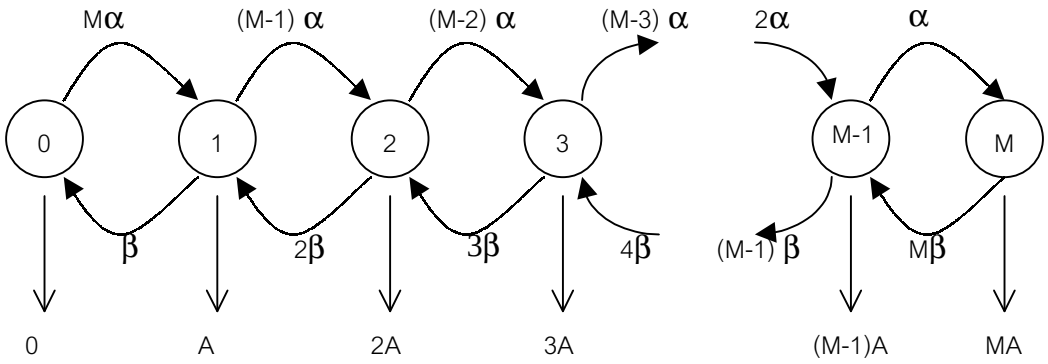


Figure 2.1: State transition diagram for birth-death MMF model

analysis. The continuous bit rate is quantized into a finite set of discrete levels and sampled at random Poisson points or inter-sample time is exponentially distributed. The birth-death Markov chain version is used for its simplicity in [24]. The details of MMF model can be found in [33, 24]. We will briefly describe birth-death version here. The state transition diagram for birth-death MMF model is shown in Fig 2.1. At state i, the bit rate will be constantly generated at iA bps where A is quantization step. The transition from state i to state $(i+1)$ is exponentially distributed with rate $(M-i)\lambda$ and the transition from state i^{th} to state $(i-1)^{th}$ is exponentially distributed with rate $i\beta$

This birth-death MMF model can also be viewed as an aggregation of identical ON-OFF mini-sources[33]. Each ON-OFF mini-source has exponentially distributed ON time with mean of $1/\beta$ seconds and exponentially distributed OFF time with mean of $1/\alpha$ seconds. At ON state, each mini-source generates constant bit rate stream at A bps. The MMF is the sum of the traffic from M identical independent ON-OFF mini-sources. Let $\epsilon = \frac{\alpha}{\beta+\alpha}$ be the steady state probability of ON-period. The traffic average rate is $\bar{A} = \epsilon \cdot A \cdot M$

To understand the implication of each parameters (ie A, M, α, β) we refer to the

works in [22, 23] which studied the Markov model that is similar to the MMF model, the Markov Modulated Poisson Process (MMPP) model. Two main issues, the Link capacity allocation and model state reduction in finite buffer networks, are extensively analyzed in these studies. They analyzed the model using signal processing theory to get traffic characteristic in frequency domain. We will discuss their work using the similar notation of birth-death MMF that we already described above.

The MMPP process is built by the superposition of M i.i.d. 2-state Markov chains mini-sources which have exponentially ON time with mean of $1/\beta$ seconds and exponentially distributed OFF time with mean of $1/\alpha$ seconds. Unlike the MMF model, at ON state, the mini-source generates poisson distributed stream with average rate of A bps. This is the only difference between MMF and MMPP model.

Also, let $\epsilon = \frac{\beta}{\beta+\alpha}$ be the steady state probability of ON-period. The steady state distribution of each mini-source is Bernoulli with probability ϵ at rate A and $(1-\epsilon)$ at rate 0. Hence, the steady state distribution of our MMF model $X(t)$ is binomial:

$$P[X = kA] = \begin{pmatrix} M \\ k \end{pmatrix} \epsilon^k (1-\epsilon)^{M-k} \tag{2.1}$$

So the traffic average rate is also $\bar{A} = \epsilon \cdot A \cdot M$. They also defined input bandwidth and variation coefficient which are respectively given by $B = 2(\alpha+\beta),\ C = \sqrt{\frac{1-\epsilon}{M\epsilon}}$

So the power spectrum of the video model is expressed by

$$P(\omega) = 2\pi\bar{A}^2\delta(\omega) + \frac{BC^2\bar{A}^2}{(B/2)^2+\omega^2} \tag{2.2}$$

From equation (2.2) and the fact that we send this process to the finite buffer MMPP/M/1/K queue, increasing C means up-scaling the input power in the entire frequency band. Similarly, reducing B has the effect of shifting input power from high

frequency band to the low frequency band, given fixed average service rate of the queue.

We could think of increasing C is trying to increase the burstiness of the video model for the entire frequency band while reducing B is trying to make the burstiness of the video traffic in low frequency more.

Now let's go back to our MMF model, we define the **frequency factor** $(\gamma = \alpha + \beta)$ as a way to emphasize the burstiness in high frequency band which is similar to B. Also we define the **Number of ON-OFF mini-source** M as a way to adjust the burstiness for the entire frequency band which is similar to $1/C$ above.(given fixed traffic average rate and ϵ)

2.2 Voice Traffic Model

We model the voice call with constant bit rate stream at 64 Kbps. This is the stream rate of G.711 codec. The voice calls have Poisson arrival with rate λ and exponentially distributed duration time with mean of $1/\mu$. (We set it to three minutes throughout the experiment.)

The voice call process can be viewed as $M/M/\infty$ queueing system. Therefore, in the steady state, the number of voice calls in the system is Poisson distributed with parameter λ/μ. The average number in the system is $N = \lambda/\mu$. Using constant bit rate stream (CBR), the average bandwidth consumption of voice traffic is $64N = 64\frac{\lambda}{\mu}$ Kbps.

We can change the voice load or the bandwidth consumption of the voice traffic from varying the call arrival rate (λ), given fixed average duration time ($1/\mu$).

2.3 TCP Traffic Model

The majority of Internet traffic is transported over TCP protocol. TCP is the window-based closed-loop congestion control, which adjusts its sending rate as a function of network conditions. Note that our traffic engineering changes the network parameters, which in turn affect TCP performance. TCP sender maintains a window of packets that have been sent and not yet acknowledged. After setup a connection, TCP start with a window size of one packet. It follows a slow-start algorithm to increase the window size multiplicatively. The slow-start ends as the window size reaches a predefined threshold. After the slow-start phase, TCP follows congestion avoidance algorithm, which can also be described as an "Additive-Increase-Multiplicative-decrease" (AIMD) algorithm. In congestion avoidance phase, the window is increased linearly in time as transmission progress without errors and the window is halved when the missing ACK condition is detected (from Triple-Duplicate ACK). The frequent packet drops can cause TCP sender to stop sending for a while until time-out. The transmission eventually resumes after the time-out with a window of one packet, and if this is successful, AIMD is resumed, otherwise another time-out occurs with double duration. Note that the time-out degrades TCP performance a lot, so traffic engineering must be aware of the situation that causes the time-out and prevent the occurrence of such situation. We measure a TCP performance from the satisfaction of TCP users. Hence the TCP performance is just the ability of TCP to response to the requests of users.

Our model specification and general TCP model

Assume that TCP is the only protocol used in low priority traffic class. As mentioned in Section 1.2, we use a class name, Best Effort (BE) class, interchangeably with low priority traffic class. Also Recall that the link scheduler always serves high

priority packets before low priority packets. Hence, low priority traffic only get a link bandwidth only what is left over from high priority traffic. As a result, we will call that available bandwidth for BE class as TCP service rate.

From the mentioned link schedule, we should look for a TCP model that specify TCP performance in term of TCP service rate. This TCP model will help us identify the performance of TCP traffic relative to the traffic characteristic of high priority traffic. Moreover we need a model for aggregated TCP flows because we will have a number of TCP flows in the link. Finally, we would like to have a performance measure from a user's point of view. We will compare these two performance measure, TCP throughput and TCP goodput, to illustrate our last requirement. Unlike the throughput , which represent all transmission (including retransmission), the goodput represent the actual transmitted amount of information, directly from the user's point of views (or application's point of view). In conclusion, we need a transient behavior model representing the performance of aggregate TCP flows. The model should be a function of TCP service rate, which indirectly relates to high priority traffic fluctuation. From our model requirements, we will explore and discuss TCP models in the literature as follows:

Recently, researchers have proposed a number of analytic models that characterize TCP performance in term of round-trip delay and packet loss rate. These models[15, 4, 9, 26] characterize the steady state throughput in long-lived flows and ignore the time-out, which is major performance degradation and occurred frequently in recent study. Padhye et al.[28] include time-out in their model to better fit the TCP behavior. Some get the throughput with interaction of active queue management[12]. These models originally were proposed to aid the design of active queue management schemes and TCP friendly protocol. Note that All models above are inspired by the

Floyd et.al[13]'s proposal to identify the unresponsive flows as a way to isolate the cause of the congestion collapse. Hence, these models show the throughput, which is the amount seen by the router instead of the actual amount seen by applications as in our requirements. So these models is not the model we look for.

Misra et al.[27] present the transient throughput of long lived TCP interacting with AQM. Even though, this work is a good model of aggregated TCP flows for transient analysis. Again, this model originally proposed for router design, which use a throughput as a performance measure.

In the very different objective of TCP modelling, the goodput of TCP is considered through the TCP latency [5, 35, 25]. These models represent the TCP performance seen by end users instead of the throughput seen by the router in identifying non TCP-friendly problem. Moreover, they can capture the short-lived flow behavior. This is desirable because the recent studies showed that mean size of transferred data by each TCP flow is around 10KB[14]. Capturing only the long-lived behavior clearly is not enough. However, these models represent only behavior of individual flow instead of the behavior of aggregate flows that we want.

To the best of our knowledge, there is no model that matches our need. However, we learn from all models in the literature that time-out is the most degradation factor in TCP performance. Therefore, we will present two models along with their validation. The first TCP model is a model for a service rate with small fluctuation. This model represents a performance of TCP traffic in the case that high priority traffic have a continuous but small fluctuation characteristic. The second TCP model is a model for a service rate with large fluctuation.

Comparing to the first model, this model represents a performance of TCP traffic that have larger fluctuation of the service rate, i.e., the service rate can change between

very low value and very high value abruptly. The fluctuation is large enough to cause a significant amount of drops, which leads to unavoidable time-out.

2.3.1 TCP Model for a Service Rate with Small Fluctuation

we propose a model that shows TCP performance as a function of TCP service rate as follows:

$$f_{\mathrm{TCP}}(C_1, C_2) = D_{\mathrm{tcp}} \cdot \frac{1}{[C_2]^+} \cdot (C_1 - C_2 - \eta_{\mathrm{tcp}})^+ \tag{2.3}$$

Where, C_1 is the TCP service rate at the last time slot, C_2 is the TCP service rate at the current time slot, η_{tcp} is the threshold of change, D_{tcp} is the arbitrary constant and $[x]^+ = \max(x, 0)$.

This function $f_{\mathrm{TCP}}(\cdot, \cdot)$ actually represents the situations that leads to TCP time-out. Again, we learned from TCP model literature that time-out is the most degradation factor in TCP performance. Hence, the function aim to represent the degradation of the TCP performance, i.e., we try to represent that TCP performance is bad when $f_{\mathrm{TCP}}(\cdot, \cdot)$ is large.

There are two terms in $f_{\mathrm{TCP}}(C_1, C_2)$, which are $(C_1 - C_2 - \eta_{\mathrm{tcp}})^+$ term and $\frac{1}{[C_2]^+}$ term. The $(C_1 - C_2 - \eta_{\mathrm{tcp}})^+$ term represents the service rate reduction, which leads to reduction of TCP goodput. As mentioned before, the reduction of the service rate can lead to the packet drops. Intuitively, the number of packet drops depends on how large the service rate reduces, i.e. $C_1 - C_2$. The large amount of packet drops make many TCP flows going to time-out; thus in turn reduces TCP goodput.[1] We added the threshold η_{tcp} in the reduction to represent the sensitivity of TCP to the reduction. Note

[1] we include the reduction of service rate in current time-step only. However, the fluctuation of service rate in the previous time slots may effect in TCP goodput also. We will ignore that for a moment and use $f_{TCP}(.,.)$ as our first step to attack this problem.

that TCP would notice the reduction if that reduction is large enough. Hence any reduction that less than the threshold, η_{tcp}, would not affect the TCP performance. The $\frac{1}{[C_2]^+}$ term represents the fact that lower the available bandwidth (service rate) for TCP, the higher chance to get time-out. We knew from models in literature that available bandwidth, C_2, is proportional to window size, given fixed number of connection. As the window size is small, the chance of get time-out is large, thus in turn largely degrade the goodput of TCP. Moreover, the concave of $\frac{1}{[C_2]^+}$ represents *fast retransmit* mechanism that can prevent the huge loss from time-out.[2] To activate fast retransmit mechanism, we need three duplicated ACKs. Hence, we need at least window size of three or more to activate this mechanism. If C is larger, window size of each TCP flow could be larger. Hence, it would be easier to activate fast retransmit mechanism, which in turn make the chance of getting time-out significantly less. As a result, the marginal increment of degradation will be less (TCP perform better) as C is getting larger, in which we could see the concave nature of this degradation.

As mentioned earlier, this function only captures the situations that cause the time-out. Hence it may not good representation of TCP performance in the network, especially the network that has only the long-lived TCP flows with the dramatically large buffer in each router. Long-lived TCP flows increase their transmission rate linearly which makes the drops rarely happen when the large buffer can keep most overloaded packets to transmit later. In addition to small drops, the sufficiently large window of long-lived TCP triggers fast retransmission from Triple-Duplicate ACK.

[2]The fast retransmit mechanism will retransmit the packet as soon as TCP received three duplicated ACK without waiting for a time-out. The drops will be corrected earlier than waiting for a retransmission at the end of time-out. Hence, we can prevent a huge loss from time-out with this fast retransmit mechanism.

The fast retransmission prevents the time-out, which make our heuristic function inaccurate. However, a majority of TCP flows are short lived with the average and the median length no longer than 10KB. The short-lived TCP still stay in the slow-start phase, which aggressively increase the sending rate. So the buffers in the routers tend to be crowded[3]. Then there is a higher probability of drops when the service rate is reduced. Moreover, the congestion window (of each TCP flow) has a relatively small value. Hence, it does not have enough packets to get triple duplicate ACK in order to activate the fast retransmit mechanism. As a result, packet loss always requires a time-out to resume a transmission. All properties of short-lived TCP flow above lead us to intuitively believe that the reduction of the service rate in our heuristic function could correctly predict the goodput degradation.

In conclusion, our model claims that TCP performance degradation depends mainly on a sharp reduction of TCP service rate. In other words, we have a cost associated with a sharp reduction of TCP service rate. We can infer from the claim that the TCP performance will be best if the TCP service rate is smooth at all time. This inference is intuitively correct with most of transport layer congestion control. Because it can estimate channel accurately and behaves effectively according to that channel estimation.

Model Validation

Recall the model in Equation (2.3) has a cost associated with the reduction of TCP service rate (available bandwidth for TCP traffic). In this section, we use ns-2 simulator[39], which is packet-level simulator, to validate this model. The validation has two parts. First, we prove that TCP goodput depends on the reduction of TCP

[3]Note that the main purpose of slow-start is to quickly exploit the unused available BW. Many short-lived flows would try to competitively exploit the BW. Hence, the buffer of the link will be crowded

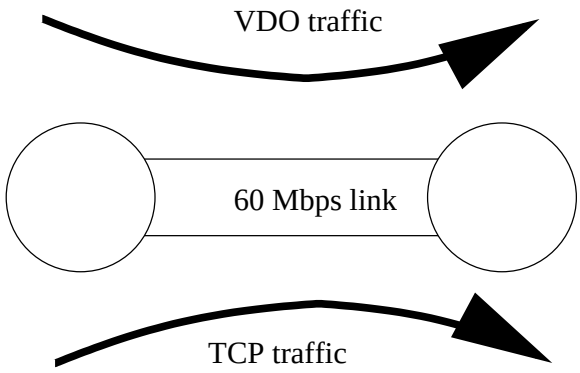


Figure 2.2: network topology in the validation of TCP model for TCP service rate with small fluctuation.

service rate. Second, we prove that TCP goodput also depends on how large the reduction is, i.e., the reduction size of TCP service rate.

We setup a small network shown in Figure 2.2. This small network captures a bottleneck link in a real network. There are two traffic types sent through this bottleneck link. First is video traffic, which is high priority traffic type. As shown in Figure 2.3, the video traffic is a CBR (Constant Bit Rate) traffic that periodically changes its transmission rate between ($B + \Delta B$) Mbps and ($B - \Delta B$) Mbps every ΔT second. Second traffic type is low priority traffic type, which consists of TCP traffic flows. Recall that every link serves video packets (high priority packets) before any TCP packets (low priority packets). Hence, the available bandwidth left from video traffic, i.e., TCP service rate have characteristic as show in Figure 12b. This TCP service rate periodically changes its rate between ($C - B - \Delta B$) Mbps and ($C - B + \Delta B$) Mbps every ΔT second. In this validation, $C = 40$Mbps.Each TCP flow sends small files with a uniformly distributed size of 20-40 Kbytes. Upon the end of each transmission, TCP sources will setup a new connection and begin to transmit a new file. Recall that the new connection setup will use a initial value for ITO. The

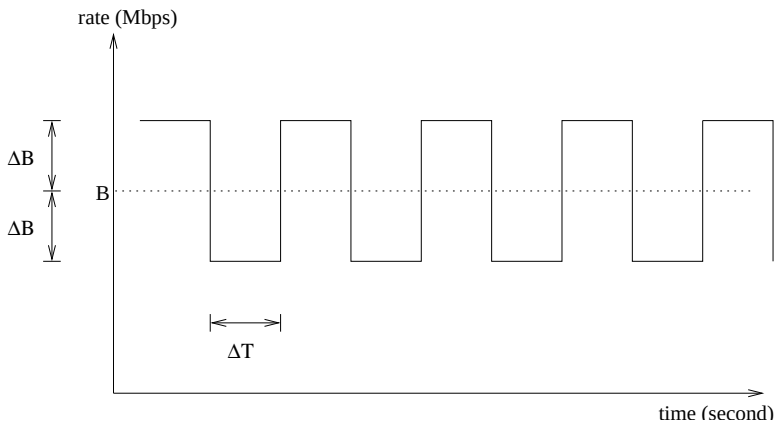


Figure 2.3: VDO traffic characteristic used in the validation of TCP model for TCP service rate with small fluctuation.

small file transmission and re-connection of flows capture a characteristic of short-lived TCP. We have four different sets of TCP flows as shown in Table 2.1. We named those four sets of TCP flows as FlowsSet A, B, C and D. As listed in the Table, TCP flows in FlowsSet A have three distinct RTT value, 10ms, 30ms and 50ms. There are 15 TCP flows with each RTT value. Therefore, there are totally 45 TCP flows in SET A. If you follow the Table, the FlowsSet B, C and D can be described similar to the description of FlowsSet A above. The different value of RTT represents different locations of end-users, while a number of flows with same RTT value indicate the fact that each location can have multiple users. Hence, we can run our simulation over these different sets of TCP flows to show that our model suits for various network setups.

The two parts of our validation have parameter adjusted and their result as follows:

1. Reduction Dependence:

   we fix $B = 28$Mbps and $\Delta B = 5$Mbps. With a fixed simulation length, $T_{\text{sim}} = 500$seconds, the number of occurrences that the reduction of TCP service rate happen, Δn, is $T_{\text{sim}}/(2 \cdot \Delta T)$. Hence we vary ΔT in such a way that Δn is

FlowsSet	RTT values	number of sources per RTT value.
A	10ms, 30ms, 50ms	15
B	10ms, 30ms, 50ms	20
C	10ms, 30ms, 50ms, 70ms	15
D	10ms, 30ms, 50ms, 70ms	20

Table 2.1: Different sets of TCP flows in the validation of TCP model for TCP service rate with small fluctuation.

varied from 12 to $2,500$. If the TCP goodput depends on the reduction of TCP service rate, we would expect TCP goodput to decrease as Δn increases (number of reduction increases), i.e, TCP goodput depends on a number of reduction, Δn.

After we ran the simulation, we got the result as shown in Figure 2.4. We could see that the experimental result follows our expectation. Hence we can conclude that the TCP goodput depends on a reduction of TCP service rate.

2. Reduction Size Dependence:

   we fix $\Delta T = 0.05$second and $B = 28$Mbps. Then we vary ΔB from 5Mbps to 11.2Mbps. If the TCP goodput depends on how large the reduction is, we would expect TCP goodput to decrease as ΔB increases (reduction size increases).

   After we ran the simulation, we got the result as shown in Figure 2.5. We could see that the experimental result follows our expectation. Hence we can conclude that the TCP goodput depends on a reduction size of TCP service rate.

In conclusion, both parts of validation strengthen the claim that the TCP performance depends on the TCP service rate reduction and the size of that reduction.

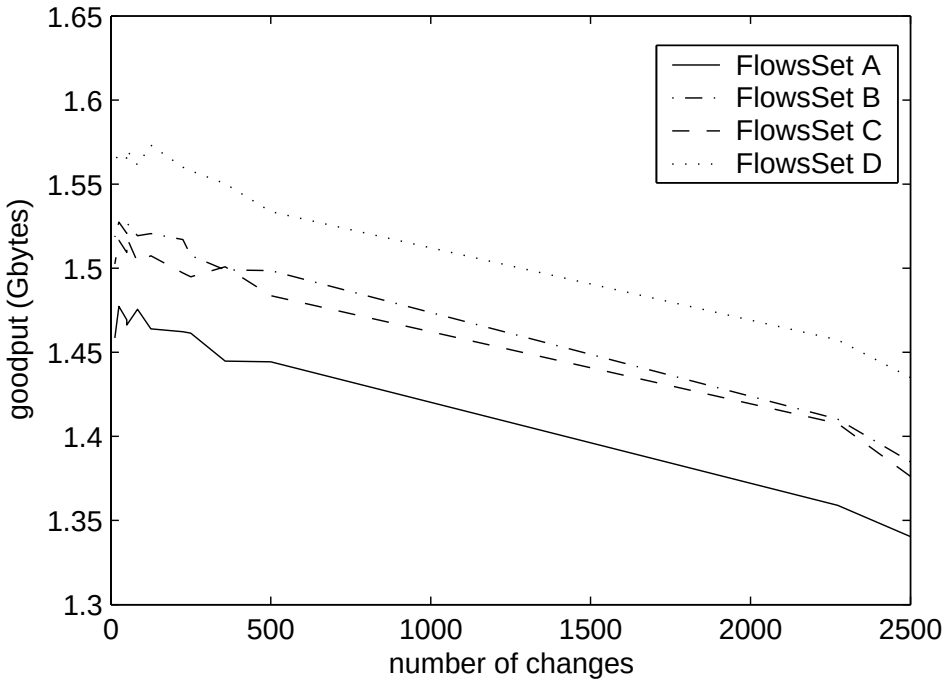


Figure 2.4: The relationship of TCP goodput and the number of reduction, Δn

2.3.2 TCP Model for a Service Rate with Large Fluctuation

Compare to the service rate characteristic in previous section, we now focus on the service rate with larger fluctuation, i.e., the service rate can change between very low value and very high value abruptly. This abrupt change of TCP service rate could cause a significant amount of drops, which leads to unavoidable time-out. So, (also as our assumption) we focus on the service rate characteristic that causes unavoidable time-out

With this special TCP service rate characteristic, we will use a chance of getting to the second and subsequent initial time-out (ITO) as our performance measure in this TCP model. Instead considering all time-out, we considered only Initial time-out (time-out of the connection setup procedure) because this time-out is significantly

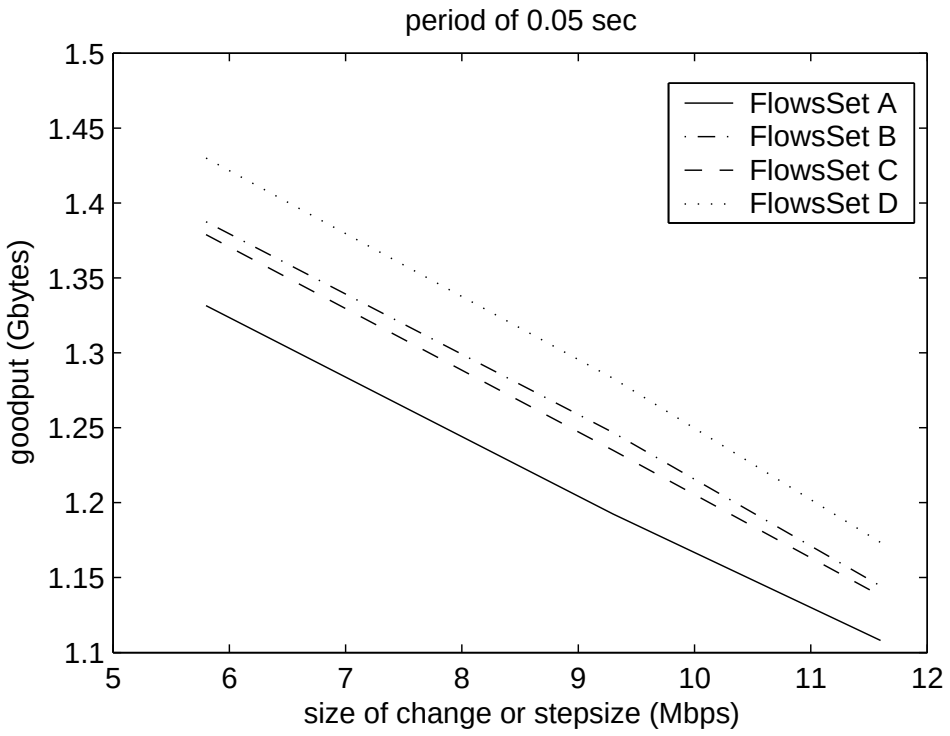


Figure 2.5: The relationship of TCP goodput and the size of reduction, ΔB

larger than other time-out. Note that TCP relies on its own packet samples to estimate an appropriate-retransmission timeout (RTO) value[14]. For the first control packets (SYN, SYN-ACK), and the first data packet, since no sampling data is available, TCP has to use a fixed conservative initial timeout (ITO) value as RTO. Hence ITO is significantly larger than other time-out. To make it even worse, the length of the next time-out will be twice of the length of previous time-out, following Karn's algorithm in TCP congestion control. Hence, the second time-out (ITO) will be really large considering the large first ITO (as large as 3 seconds). The Internet user may have to wait a long time if he gets into the time-out during the connection setup.[4] When we

[4]If there is the second or subsequence ITO, we could imagine how long the user have to wait after he click at his browser. Given a normal first ITO of 3 seconds, the second ITO would be 6 seconds. If

assume that the time-out is unavoidable, the performance of TCP would depend on whether it goes to second or subsequent time-out or not. As a result, we will use a chance of getting the second and subsequent ITO as our performance measure in this model.

We propose our TCP model as shown in Equation (2.4).

$$h_{\mathrm{TCP}}(\vec{w}(k)) = D_{\mathrm{setup}} \cdot w(k) \cdot w(k-k_s) + D_{\mathrm{setup2}} \cdot w(k) \cdot w(k-2k_s) \cdot w(k-3k_s) \quad (2.4)$$

Where k_s is the time-out length of connection procedure (ITO) in the unit of time slots. D_{setup} and D_{setup2} are arbitrary constant values.

We define k^{th} time slot as an interval from time k to $k+1$. Note that we let The length of time slot is arbitrary but are equals for any time slot. Then, we let $w(k)$ as a binary variable indicate the level of TCP service rate during k^{th} time slot, i.e., $w(k) = 1$ indicates a low TCP service rate during k^{th} time slot, while $w(k) = 0$ indicates a high TCP service rate during k^{th} time slot. Finally, we define a vector $\vec{w}(k) = [w(k-3k_s), w(k-3k_s+1), \ldots, w(k)]$ as a trajectory of TCP service rate from time $(k-3k_s)$ to time $(k+1)$.

The model in Equation (2.4) actually represents the TCP service rate pattern that lead to the second and the third ITO. So any sequence $\vec{w}(k)$ that lead to second and the third ITO will have a corresponding large value of TCP performance degradation. The detail of TCP service rate pattern in our model can be discussed as follows:

Before going to the detail of TCP service rate pattern, we should talk about the assumption we made to ease a modelling process. We assume that there are TCP packet drops when we have low TCP service rate in that time slot. From the nature of

his request gets into second ITO, he will have to wait for $3 + 6 = 9$ seconds. It will be even worse with subsequence ITO. The long waiting time can make user furious and judge the network as a slow and unreliable one.

this TCP service rate described earlier, the time slot with low TCP service rate normally have a very small TCP service rate, comparing to a time slot with high TCP service rate. Given fixed number of users, attempts to see WebPages can make congestion over that time slot; thus in turn induce the loss of the packets. So we will use the time slot with low TCP service rate as a sign of lost packets of TCP flows, i.e., we assume TCP packet drops when we have low TCP service rate in that time slot. This assumption will not hold if our time slot is very small. However, we focus on the time slot of 50ms, which is long enough to cause packet drops.

The first term of $h_{\mathrm{TCP}}(\vec{w}(k))$ in Equation (2.4), i.e., the $D_{\mathrm{setup}} \cdot w(k) \cdot w(k-k_s)$ term, represents a TCP service rate pattern that leads to second ITO. We use Figure 2.6 to illustrate this pattern. We are now at time k and recalled that k_s is the ITO in the time slot unit. If $w(k-k_s)$ is one, i.e., we have a low TCP service rate over $(k-k_s)^{\mathrm{th}}$ time slot, many flows during $(k-k_s)^{\mathrm{th}}$ time slot have had gone to time-out from packet losses.[5] The timers in TCP will start their clock during that time slot. Eventually, those timers will expire and trigger the retransmission of the packets during our current k^{th} time slot.[6] If the TCP service rate in current time slot is again low ($w(k)=1$), the retransmitted packets will be dropped. which in turn lead to second ITO. This TCP service rate pattern, which leads to second ITO, makes TCP users to wait for a long time. This long waiting time represents a poor TCP performance as we

[5]Recalled that we assume drops when the TCP service rate is low

[6]To be exact, the timers of the TCP flows, in which their packets is dropped in the bottleneck link during $(k-k_s)^{\mathrm{th}}$ time slot, will expire and trigger a retransmission of packets. The retransmission packets will arrive bottleneck link during k^{th} time slot, given a fixed delay from TCP sources to the bottleneck link. To ease the explanation, we let that fixed delay to be zero, i.e., we describe everything as if the TCP sources attached to the bottleneck link.

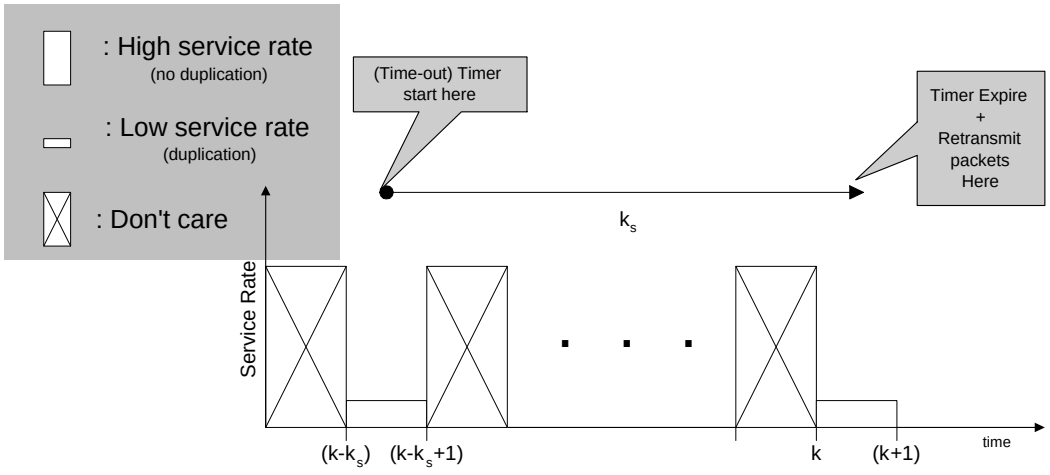


Figure 2.6: The TCP service rate pattern that leads to the second time-out

mentioned earlier.[7]

Similar to the first term, the second term of Equation (2.4), i.e., the $D_{\mathrm{setup2}} \cdot w(k) \cdot w(k-2k_s) \cdot w(k-3k_s)$ term, represents a TCP service rate pattern that leads to the third ITO. Figure 2.7 exhibits our idea. Suppose we have a low TCP service rate during $(k-3k_s)^{\mathrm{th}}$ time slot and $(k-2k_s)^{\mathrm{th}}$ time slot, i.e., "$w(k-3k_s)=1$" and "$w(k-2k_s)=1$". The timers that start their clocks during $(k-3k_s)^{\mathrm{th}}$ time slot might expire and retransmit packets during $(k-2k_s)^{\mathrm{th}}$ time slot. However, the low TCP service rate in $(k-2k_s)^{\mathrm{th}}$ time slot can cause the loss of those retransmitted packet. In that case, the timers will restart their clocks over and will expire again during our current k^{th} time slot. Recall that the second timers stay twice in length from the first time-out length. Again, if the TCP service rate in current time slot

[7]This TCP model assume that there is a fixed delay from sources to bottleneck link. This assumption might not be able to hold at all time in a real network. For example, packets may wait in many queues (of the intermediate link) before reaching bottleneck link. These queues can create a variable jitter, which prevent the retransmitted packets to reach bottleneck link k_s second after the first drops. In this case, our model may not be able to predict TCP performance very well.

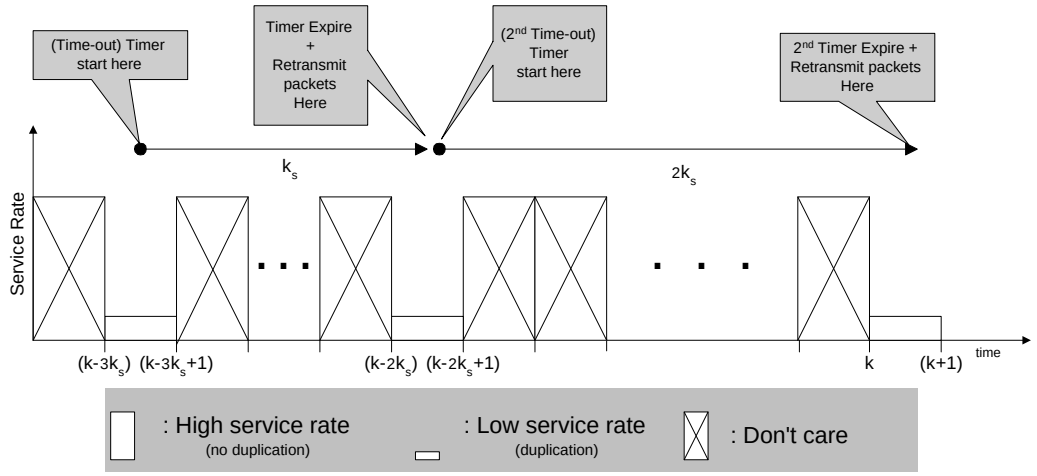


Figure 2.7: The TCP service rate pattern that leads to third time-out

is again low ($w(k) = 1$), the retransmitted packets will be dropped. which in turn lead to the third ITO. This TCP service rate pattern, which leads to the third ITO, makes TCP users to wait for a long time. Again we can say that this long waiting time represents a poor TCP performance as we mentioned earlier.

Normally we will let $D_{\text{setup2}} \gg D_{\text{setup}}$. The fact behind this setup is that the third time-out is much longer than the second time-out. So the third time-out exhibits worse TCP performance.

Model Validation

There are two parts in this validation. First, we will prove a TCP service rate pattern that lead to the second ITO. That particular pattern can simply be described as the two time slots with low TCP service rate, in which time separation between them equals to the first ITO length. In the second part, we will prove a TCP service rate pattern that lead to the third ITO. Note that most hosts in current Internet use first ITO value of 3 seconds. Hence we will assume from this point that first ITO value is 3 seconds. Again, we use ns-2 simulator[39], which is packet-level simulator, to validate the model.

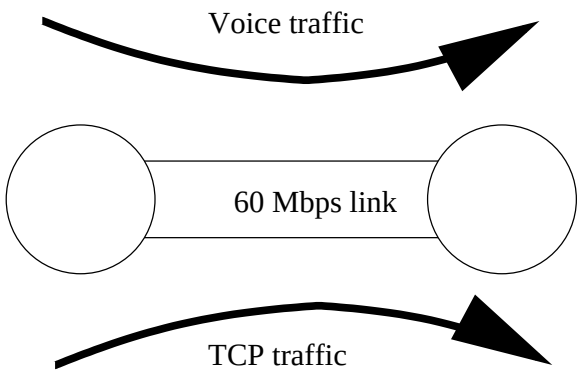


Figure 2.8: network topology in the validation of TCP model for TCP service rate with large fluctuation.

We setup a small network shown in Figure 2.8. This small network captures a bottleneck link of backup LSP in a real network. There are two traffic type sent through this bottleneck link. First is voice traffic, which is high priority traffic type. The voice traffic will be used to explored various patterns of TCP service rate. We will have different voice traffic for the two different validation parts, in which we will discuss later.

The second traffic type is low priority traffic, which consists of short-lived TCP flows. Similar to Section 2.3.1, we use two different sets of TCP flows as shown in Table 2.2. We named those two sets of TCP flows as FlowsSet A and B. As listed in the Table, TCP flows in FlowsSet A have four distinct RTT value, 10ms, 30ms, 50ms and 70ms. There are 25 TCP flows with each RTT value. Therefore, there are totally 100 TCP flows in FlowsSet A. If you follow the Table, the FlowsSet B can be described similar to the description of FlowsSet A above. As we mentioned earlier, we ran our simulation over these different sets of TCP flows to show that our model suits for various network setups.

1. Second ITO validation:

FlowsSet	RTT values	number of sources per RTT value.
A	10ms, 30ms, 50ms, 70ms	25
B	10ms, 30ms, 50ms, 70ms	38

Table 2.2: Different sets of TCP flows in the validation of TCP model for TCP service rate with large fluctuation.

In this validation part, we will use a voice traffic, which has a function of its rate over time as shown in Figure 2.9. This rate function helps us to explore various pattern of TCP service rate related to Second ITO. As seen in the Figure, this function has only two intervals that have ΔB Mbps rate. The rest of the time, we keep the voice rate at zero Mbps, i.e., there is no voice traffic in bottleneck link. The first non-zero rate interval start ΔT seconds before the second non-zero interval as shown in the Figure. We can vary ΔT to generate different patterns of TCP service rate. The length of non-zero interval is 50msec, which is our preselected time-step.

In this validation part, we would like to show that the three-second separation of two time-steps with low TCP service rate can cause Second ITO. In other words, number of TCP flows that get into Second ITO is huge at $\Delta T = 3$second. (We will use "number of Second ITO" instead of number of TCP flows that get into Second ITO from this point) Hence, we vary ΔT in the simulation and monitor the number of Second ITO. We will expect to get the result that have relatively large number of Second ITO at $\Delta T = 3$second, comparing to other value of ΔT

We vary ΔB from 40Mbps to 60Mbps. The value $\Delta B = 60$Mbps represents the case that high priority traffic takes all bandwidth away from TCP traffic, while The value $\Delta B = 40$Mbps represents the case that high priority traffic still leaves

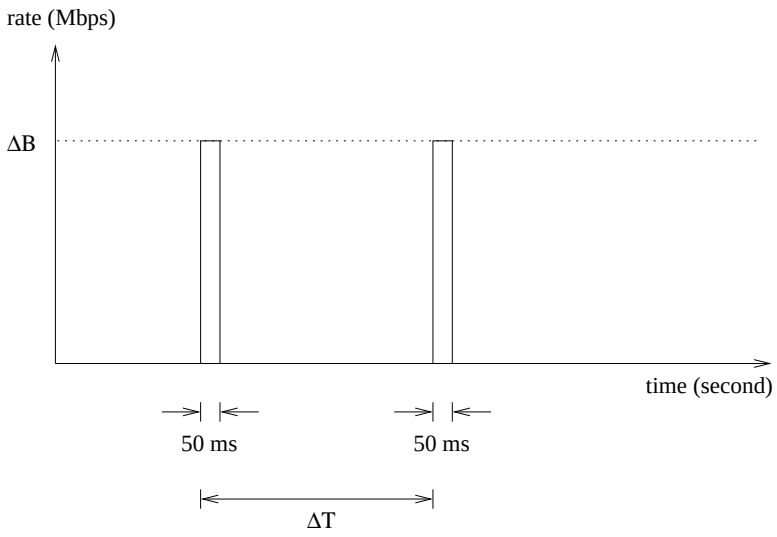


Figure 2.9: Voice traffic characteristic used in Second-ITO validation: TCP model for TCP service rate with large fluctuation.

some bandwidth for TCP traffic.

After we ran our simulation, we got the result shown in Figure 2.11 and 2.10.[8] Figure 2.10 shows the percentage of connection setup that goes to Second ITO. For example, at $\Delta T = 3$second and $\Delta B = 60$Mbps, we can see value 10.5 percent from the graph. This value can be interpreted as follow; out of 100 connection setup, there are 10.5 of them have to get Second ITO. In other words, 10.5 out of 100 connection setup will suffer a long waiting time (before the connection setup can be completed and ready to acquire data).[9]

[8]Both Figure is a result using FlowsSet B only. We do not show a result using FlowsSet A because both FlowsSets exhibit similar result.

[9]Note that $\Delta T = 0$ in both Figure 2.11 and 2.10 shows the case that the TCP service is high all the time (without a time-step of low TCP service rate). We include the $\Delta T = 0$ case in the Figures to show that ITO normally happen even we have smooth and high TCP service rate. However, the low TCP service rate still induce much more ITO

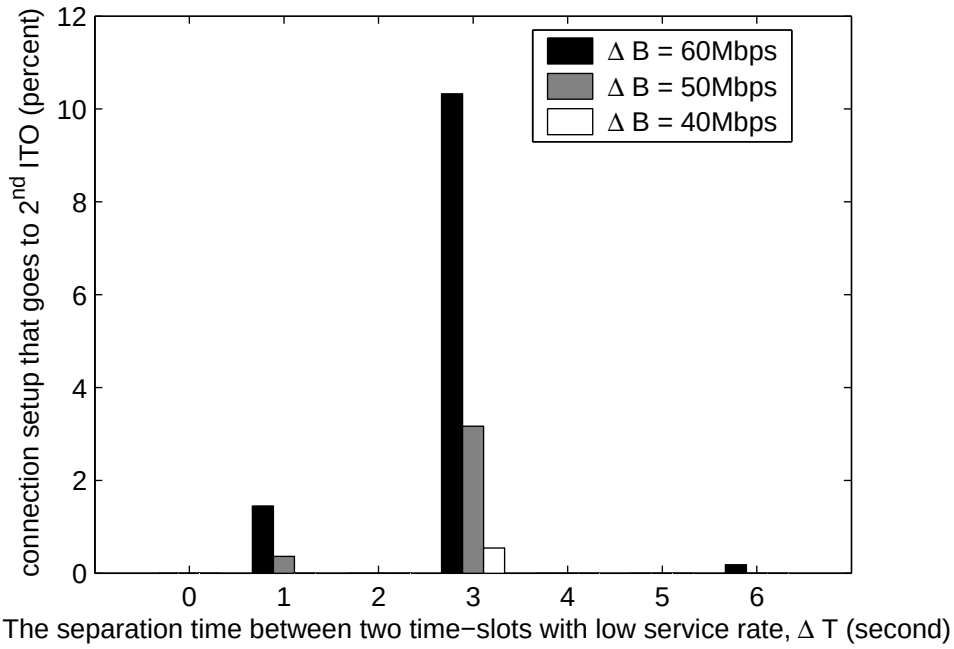


Figure 2.10: Percentage of TCP connection setup that goes to Second ITO

The result in the graph show that $\Delta T = 3$ case has significant number of Second ITO comparing to other value of ΔT. Hence we can conclude that TCP performance is badly degrading if there are three seconds separation between two time-step with low TCP service rate. This conclusion validates our TCP model in Equation (2.4).

The by-product of this experiment is shown in Figure 2.11. The Figure can be used to strengthen our assumption that low TCP service rate induces first time-out. From the Figure, we can see that 100% of connection setup, which try to start its connection when TCP service rate is zero ($\Delta B = 60$Mbps), go to the first ITO.

In conclusion, our simulation result help strengthen our claim that a three-second separation of time-steps with low TCP service rate can lead to the Second ITO.

2. Third ITO validation:

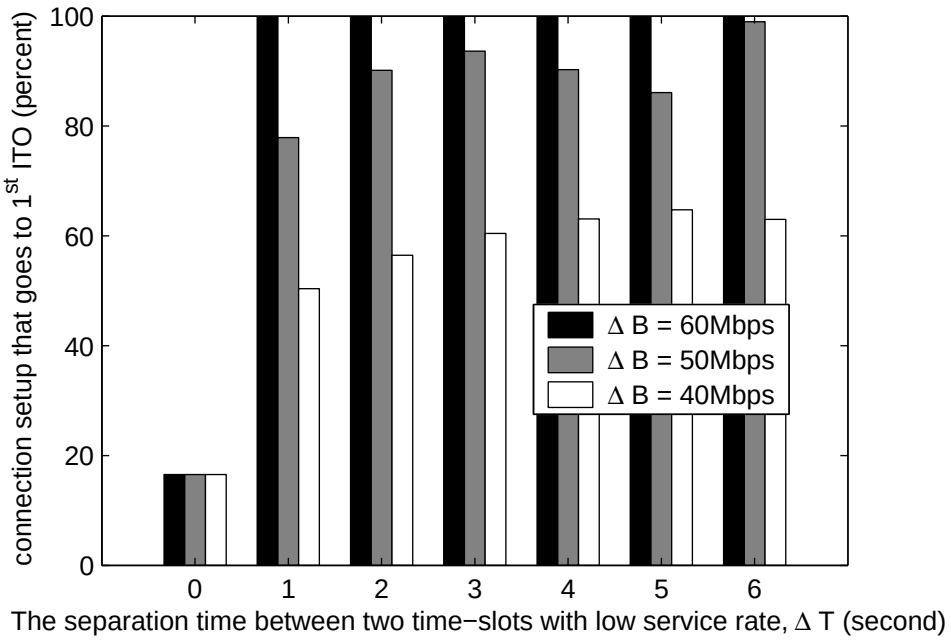


Figure 2.11: Percentage of TCP connection setup that goes to first ITO

In this validation part, we will use a voice traffic, which has a function of its rate over time as shown in Figure 2.12. This rate function helps us to explore various pattern of TCP service rate related to Third ITO. As seen in the Figure, this function has three intervals that have ΔB Mbps rate. The rest of the time, we keep the voice rate at zero bps. The first non-zero rate interval start 3 seconds[10] before the second non-zero interval, while the third non-zero rate interval start ΔT seconds after the second non-zero intervals shown in the Figure. We can vary ΔT to generate different patterns of TCP service rate. Again, The length of non-zero interval is 50msec, which is our preselected time-step.

In this validation part, we would like to validate the model by showing the a

[10]we keep it fixed at 3 second because we would like to make sure that TCP sources have already gone to Second ITO. This pattern come from assumption that the separation of 3 seconds lead to Second ITO as proved in the first experimental part.

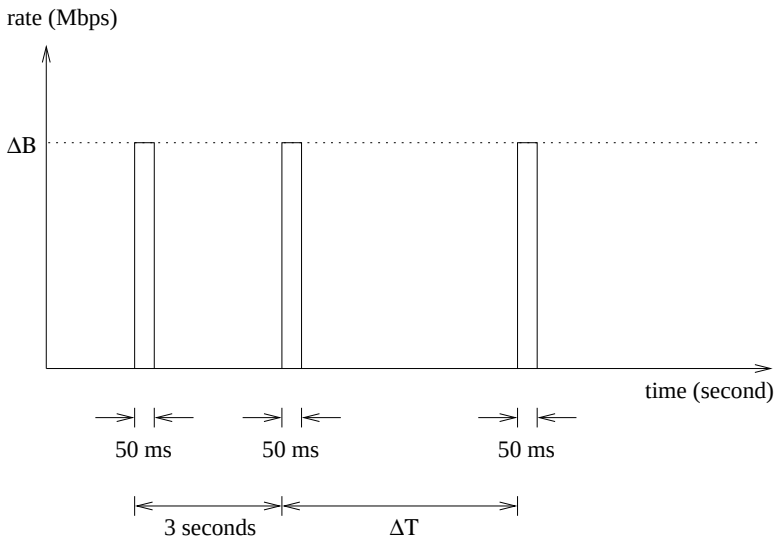


Figure 2.12: Voice traffic characteristic used in Third-ITO validation: TCP model for TCP service rate with large fluctuation.

pattern with $\Delta T = 6$second can cause Third ITO. In other words, number of TCP flows that get into Third ITO is huge at $\Delta T = 6$second. (Again, We will use "number of Third ITO" instead of number of TCP flows that get into Third ITO from this point) Hence, we vary ΔT in the simulation and monitor the number of Third ITO. We will expect to get the result that have relatively large number of Third ITO at $\Delta T = 6$second, comparing to other value of ΔT

Like the first experimental part, we vary ΔB from 40Mbps to 60Mbps. The value $\Delta B = 60$Mbps represents the case that high priority traffic takes all bandwidth away from TCP traffic, while The value $\Delta B = 40$Mbps represents the case that high priority traffic still leaves some bandwidth for TCP traffic.

After we ran our simulation, we got the result shown in Figure 2.13.[11] Figure 2.13

[11] Again, the Figure is a result using FlowsSet B only. We do not show a result using FlowsSet A because both FlowsSets exhibit similar result.

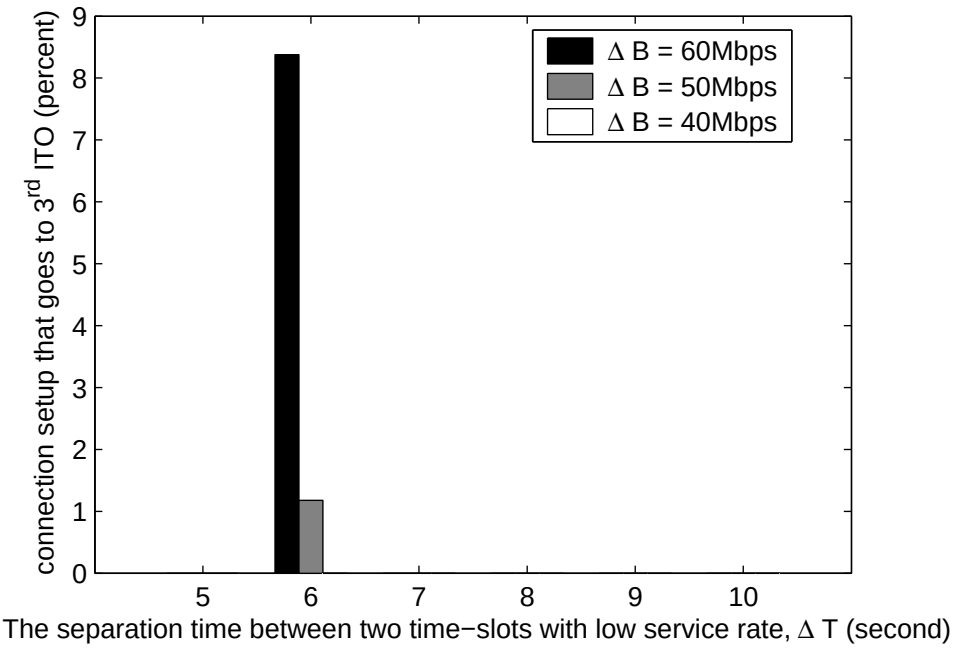


Figure 2.13: Percentage of TCP connection setup that goes to Third ITO

shows the percentage of connection setup that goes to Third ITO. For example, at $\Delta T = 6$second and $\Delta B = 60$Mbps, we can see value 8.5 percent from the graph. This value can be interpreted as follow; out of 100 connection setup, there are 8.5 of them have to get Third ITO. In other words, 8.5 out of 100 connection setup will suffer a very long waiting time (before the connection setup can be completed and ready to acquire data).

The result in the graph show that $\Delta T = 6$ case has significant number of Third ITO comparing to other value of ΔT. Hence we can conclude that TCP performance is badly degrading if the TCP service rate is low, six seconds right after the three-second separation between first two time-steps with low TCP service rate. This conclusion validates our TCP model in Equation (2.4).

Chapter 3

Reactive Control with Flow Migration. (in High Priority Network)

As described earlier, we will explore the fast timescale migration control-e.g., in the response times less than 1 second and possibly on the order of 50ms. If it proves feasible to respond quickly to congestion by migrating flows off of paths that traverse bottleneck links, then the network can be operated in very high utilization even for the high priority traffic.

The first is an off-line static algorithm (Bernoulli Splitting), which provides long-term optimal flow to LSP assignments. The second scheme is more dynamic, and assigns flows to LSPs which are “least-loaded”, when the flow starts at the ingress router. The detail of migration algorithm and non-migration algorithms is described in Section 3.1.

We evaluate the benefits of these three different control strategies in context of streaming media applications, specifically voice-over-IP and variable bit-rate video. We present results from both packet-level simulations and a detailed implementation of each scheme. We chose these applications since they have similar QoS requirements, and are also likely to be significant traffic components in the next generation of IP

networks. Further, the video flows have high burstiness and represent a challenging test case for the control mechanisms.

3.1 Control Algorithm

In this section, we briefly describe two algorithms —"Bernoulli Splitting" (BSplit), "Least Load Routing" (LLR) — that we consider as non-migration algorithms, and our migration scheme. Both non-migration algorithms apply upon arrival of a new call by assigning it to particular LSP. An assigned call will remain in a selected LSP until it departs.

Bernoulli Splitting (BSplit) This controller is based on an optimal off-line analysis that assumes that the slow timescale average traffic rate between each ingress-egress pair is known. BSplit is a randomized policy such that each new arriving call is dispatched among LSPs according to a split rule (probability distribution).

Least Load Routing (LLR) Whenever a call arrives into a particular ingress node, this non-migration controller in the node finds which LSP has the least load among the LSPs that the ingress node is using for the particular egress. The new call is simply joined into the least loaded LSP. If there is a tie, we break this with a random assignment with equal probabilities among LSPs. The load for an LSP is defined to be the maximum load among the links it traverses. In our experiments, each LSP has a single bottleneck link, and each ingress node gets the current load information by a feedback packet from the LSR at the head end of the bottleneck link for each LSP that the ingress node is using.

Migration Algorithm (MIG) This online centralized controller is built upon the Bernoulli

Splitting controller (BSplit). The algorithm has two configurable parameters: the migration threshold (Γ%) and safety margin(δ%). The idea is that the controller is triggered whenever link utilization exceeds the migration threshold; in this case, it attempts to migrate flows to reduce the utilization of the bottleneck link to the migration threshold minus the safety margin. In doing so, it will not cause other link utilizations to exceed the migration threshold minus the safety margin which in turn stabilize the system. Pseudo-code of the migration algorithm is shown in Algorithm 1. Figure 3.1 illustrates notations used in the algorithm and their relations.

1. In each controller time slot, get the link l with maximum utilization that is over the threshold Γ%. Record amount over safety margin level (A_{move})
2. If l exists, then discover the set ζ of LSPs using it.
   (we exclude LSPs that do not have any traffic at that time)
3. Find alternative LSPs for each $\rho \in \zeta$
4. For each $\rho \in \zeta$, calculate
   $S(\rho) = \sum$[available BW - safety margin (δ%)]$^+$ of its alternative LSPs,
   where $[\cdot]^+ \doteq \min(\cdot, 0)$
5. Choose $\gamma \in \zeta$, that maximizes $S(\gamma)$. The Ingress migrates amount of A_{move} from γ to its alternative LSPs proportional to their [available BW - δ%]$^+$ values.

Algorithm 1: The migration algorithm in the network which has only real-time traffic.

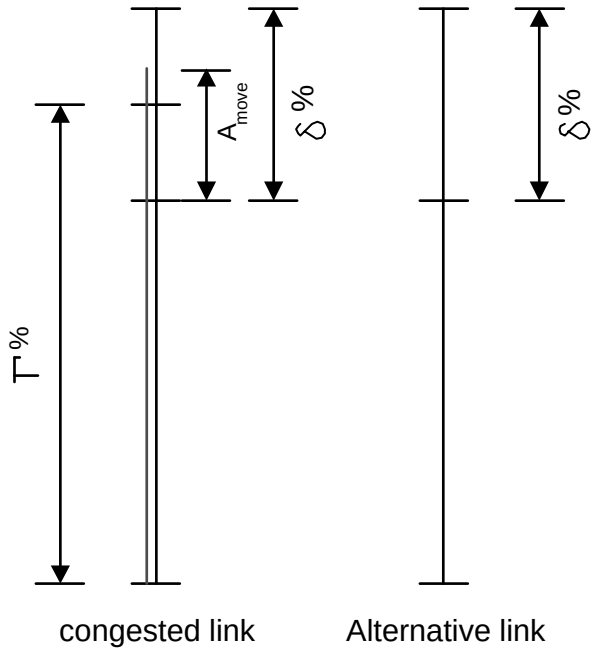


Figure 3.1: Notations used in migration algorithm and their relations.

3.2 Simulator Implementation

We used ns-2 simulator[39] for our simulations. To meet our simulation requirement, we incorporated some new features including the multiple-path capability between ingress and egress node pair and bandwidth guaranteed LSPs. The intelligence is added in the ingress node to be able to classify the flows to LSPs. Due to the need to capture the per-flow statistics, the voice traffic generator needs to be written in tcl to simulate the Poisson distributed arrivals and exponentially distributed duration. Despite the limited number of flow identifiers, the flow identifiers have been carefully reused without sacrifice of the per-flow statistics and traffic properties.

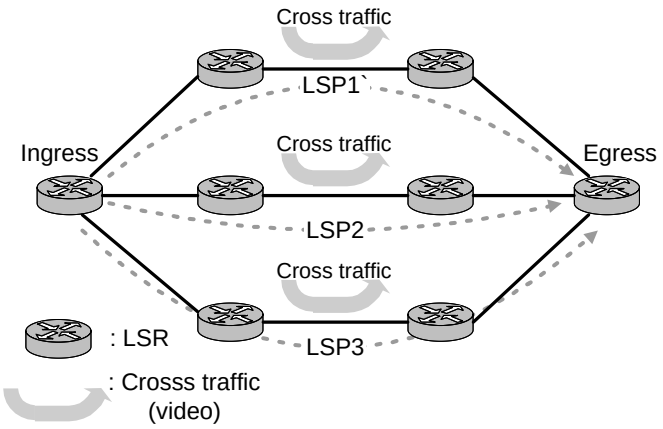


Figure 3.2: Network Topology.

3.3 Experiment setup

We considered the network topology shown in Figure 3.2 for our experiments. Inside the network, we have two types of traffic, video and voice that belong to the same priority class and hence have the same QoS requirements. Due to per behavior aggregate queuing, both traffic types share the same queue inside the network.

Throughout our simulation, we fixed the packet size of video and voice traffic at 400 bytes. All links have a buffer size of 24Kbytes. This is twice the bandwidth-delay product for a 90Mbps link with 1ms propagation delay. The ingress LSR monitors the congested link utilization every 50ms. Based on this, it decides whether to migrate or not at the end of the interval.

In order to study the performance of the controller algorithms better, we varied the incoming traffic characteristics. First, we varied the voice load, i.e. the bandwidth consumption of the voice traffic by varying the call arrival rate (λ) while fixing the average duration time ($1/\mu$) at 3 minutes. Then, we varied the video traffic characteristics using the following three parameters:

1. Video Load: By fixing the number of on-off sources (M) and the steady state probability of ON-period ($epsilon = 1/2$), we varied the average bandwidth consumption (video load) by changing its step size (A) according to the equation (3.1).

$$\text{Average BW} = \frac{1}{2} A \cdot M \tag{3.1}$$

2. The Fluctuation or Frequency Factor (γ): As mentioned in modeling section, this parameter is similar to the input-bandwidth-factor B of [22, 23] which affects the burstiness in the frequency band. As γ is reduced, the low frequency burstiness of the video traffic increases.

3. The number of ON-OFF sources (M): As mentioned earlier, it is a way to adjust the burstiness of traffic in the entire frequency domain.

The explanations on the parameters γ and M, stated above and in [22, 23], are from the Signal Processing perspective. From the Networking perspective, γ can be imagined as the frequency of bulk-traffic arrival and the duration that it remains queued in the network, while M represents how large each bulk is.

We considered minimum drop rate as our performance metric rather than end-to-end delay because the buffer size is so small as to keep the end-to-end delay bounded.

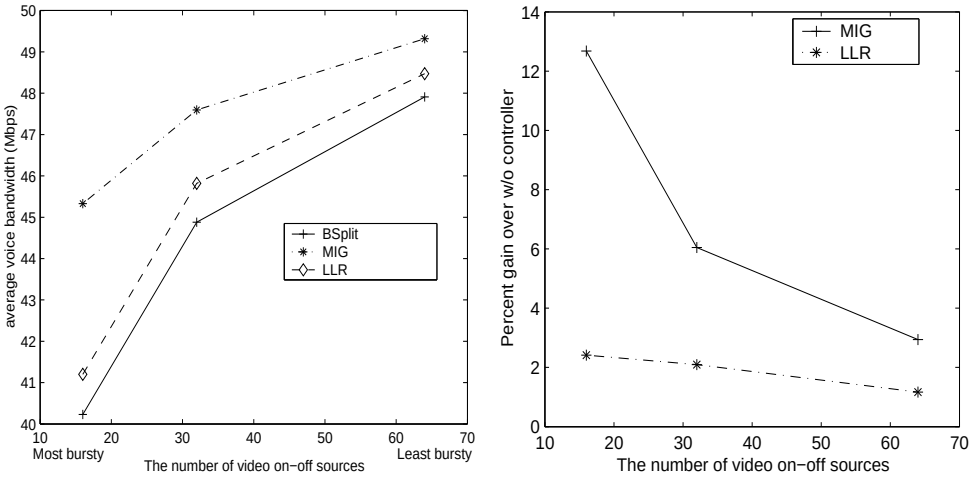
The maximum number of migration requests for each 30 minute-simulation run was 36000. Every data point was obtained by repeating the simulation for different random number seeds to get a reasonably accurate measure of the drop rate. We ran the simulations a number of times to get statistics sufficiently accurate.

3.4 Experimental result

We started with a test case to study how much voice traffic the network can sustain under BSplit, MIG and LLR controllers, given an one percent drop rate constraint and different burstiness levels of the video traffic. We use the one percent drop rate constraint as an example of the QoS requirement from the application. Assume that the application e.g. voice CODEC in our case will produce a bad sound quality if the drop rate is larger than one percent. We varied the average number of voice calls fixing the average video cross traffic bandwidth at 40Mbps and γ at 0.5, and obtained the voice bandwidth consumption. As expected, the results showed that we can indeed fit more voice traffic in the network as the value of M increases (that is as burstiness of video decreases).

Figure 3.3(a) gives the average voice bandwidth in the system to get 1% average drop rate using BSplit, MIG (with 100% threshold and 2% safety factor) and LLR controllers. We could see that under bursty video cross traffic, MIG can accommodate voice traffic up to 45.3 Mbps while the LLR and BSplit can sustain up to 41.2 Mbps and 40.2 Mbps, respectively. Figure 3.3(b) gives the percentage voice bandwidth gain of MIG and LLR controllers over the BSplit controller. The gain of MIG controller is more than that of LLR controller for small M, which implies that MIG controller is more beneficial when the cross traffic is more bursty. This come from the fact that migration can move many calls in a short time (every 50 milliseconds) while the LLR controller moves the call only when there is a new arrival (one call per 70-100 millisecond on an average). Therefore, the MIG controller can adapt to the temporary but abrupt imbalance in load better than LLR controller.

We also consider the effect of voice load to the controller performance under the same video traffic condition above, i.e., fixing the video traffic bandwidth at 40 Mbps



(a) The maximum voice bandwidth to get 1% drop rate requirement

(b) The relative bandwidth gain

Figure 3.3: The effect of burstiness (step size) of the cross traffic to the controller performance.

and parameter γ at 0.5. We changed the voice bandwidth by varying the call arrival rate λ for different burstiness level (M) of video. From Figure 3.4, we can infer that the MIG controller out-performs the LLR controller at light loads. At very high voice loads (i.e., large λ) the LLR controller performs slightly better than MIG controller. This is because LLR updates its decisions at the rate of call arrivals and hence balances the load across the LSPs faster. But it is highly unlikely that any provider will operate at that high a load.

We also ran the simulation to see the controller performance under different network condition, by varying M and γ. We also changed the threshold and safety factor of the migration mechanism and explored its effect on the controller performance. The combination of MIG and LLR mechanisms is also implemented and compared its performance with MIG and LLR separately. (The results is not shown

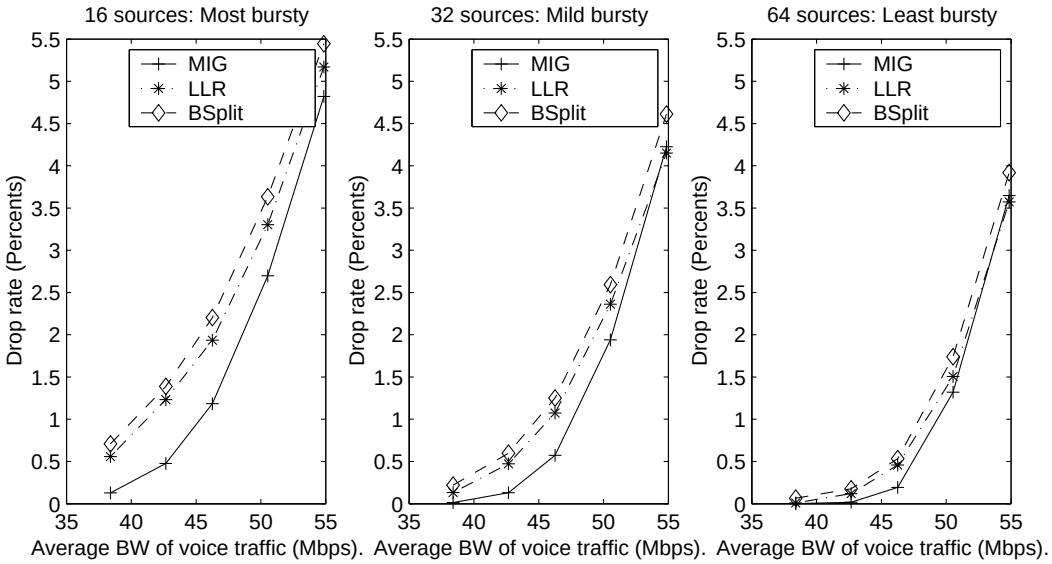


Figure 3.4: The drop rate at different average voice load using MIG (100% threshold and 2% safety factor), LLR and BSplit controllers.

here)[1] One of the interesting results is when we vary γ; MIG performs better in the high values of γ. Therefore we can interpret from the meaning of γ that MIG performs best in frequently changed network.

Some of experimental result is verified from testbed implementation in the MENTER project. The LSR in the testbed is developed from FreeBSD-3.3 and NISTswitch software. The detail implementation and the result present in our paper and technical report[21, 30].

3.5 Conclusion

The simulation and testbed prove the advantage of fast timescale migration controller under different network conditions. Under different network load and traffic burstiness,

[1]The complete results and detail discussions are shown in our technical report[30].

we have compared our migration scheme that operates at 50ms timescales with two other non-migration algorithms: “Bernoulli Splitting”(no adaptation) and “Least Load Routing” (call arrival rate timescale). Both our simulation and testbed results showed that the end-to-end voice traffic drop rates and consecutive drops were improved under various traffic loads when a finer timescale controller is applied. We also demonstrated that when the cross traffic is burstier, our migration controller has better voice traffic bandwidth gains. Also the migration controller performs better at frequently changed network.

Overall, this work is a first step towards migration-based control. Our results show that such an approach does have significant benefits compared to traditional approaches. On the other hand, all controllers including migration controller suffer from the obsolete information that sends back to it. This is due to the relatively long delay (comparing to fast timescale of controller). So there is a need for the dynamic programming technique to predict the traffic matrix. The dynamic programming takes into account predictions by the traffic model in determining the action to take.

Chapter 4

Proactive Control with Flow Migration

Many service providers guarantee low delay and QoS of real-time traffic by considering it as high priority traffic and by configuring the queue discipline to serve high priority traffic before Best-Effort traffic. It works fine as long as the high priority traffic share is low (normally 5-10%). This constraint is undesirable because it limits the amount of high priority traffic that users can send. Moreover, users are willing to pay for their high priority traffic's QoS guarantee. This fact can lead to a great amount of profit if the providers can support more high priority traffic in their existing network. The constraint comes from dramatic fluctuation of its bandwidth utilization. If there is a lot of high priority traffic in the network, this large fluctuation could lead to unpredictable high traffic rate. The network might not be able to immediately support this unpredictable high traffic rate, which could easily jeopardize QoS of the high priority traffic.

Moreover, the fluctuation of available bandwidth to Best-Effort works adversely with TCP congestion avoidance mechanism. As mentioned in Section 1.2, the low priority traffic only gets a link bandwidth of what is left over from high priority traffic. This left-over bandwidth will fluctuate if the high priority traffic fluctuates. Hence, the low priority traffic will see the bad and fluctuated channel, in which its TCP congestion

avoidance mechanism could work adversely. To better illustrate the above problem, we look at the case when the high priority load decreases, stays at low load for a fairly long time and then significantly increases. The reduction of the high priority load induces the TCPs, especially short-lived TCP flows, to aggressively exploit their larger available bandwidth. TCP increases its window size faster as the network sends back the acknowledgement packets faster without losses. More packets accumulate at the buffers of the routers. This leads to packet drops when the service rate of Best-Effort traffic decreases from the result of the increment in high priority loads. The drops can cause the TCP to reduce its window by half or cause TCP to time-out. Therefore, the throughput of TCP flows decrease significantly especially when they are caught up in time-out phase.

Our main goal will be to design a migration control to preserve QoS when we increase the amount of high priority traffic in the network. Besides that main goal, our migration control should also provide a smooth channel for TCP traffic such that TCP can get high throughput over that channel. This work does not intend to find the multiple feasible paths between an ingress and egress pair; instead, we assume the existence of parallel LSPs and only consider the problem of migrating flows between these LSPs.

The main difference between the migration control in this chapter and the previous chapter is that we designed the migration control in this chapter to be proactive. We will proactively migrate flows to avoid a predicted congestion. On the other hand, the migration control in Chapter 3 will not have a prediction and will migrate flows only if there is a congestion in the network already. Recall that we usee a particular threshold value to indicate congestion in the network. If the utilization is over that threshold, we will assume a congestion and act upon that sign. Hence, the migration control in

Chapter 3 is more reactive to the congestion in contrast to our proactive control in this chapter.

Even though our main focus in low priority traffic is TCP, as stated earlier, our approach may apply to other transport layer protocols which have reactive congestion control. This is due to the fact that the reactive congestion controls have to determine the available bandwidth in the network and send the traffic according to that estimation. In general, the smoothness of the channel makes that estimation scheme perform better.

4.1 Related Works

The migration mechanism has been used in the work by Elwalid et al.[10] to dynamically load balance the Best-Effort traffic among parallel LSPs. From the fact that they focus on a much longer timescale and single class of service, these make their work different from ours

Firoiu et al.[11] use the TCP model to do a traffic engineering in the IP network. They also include the CBR flow model in their network, which makes it a more realistic situation. They use both steady-state throughput model and latency model of TCP to predict the drop rate and queue length in the network. This in turn can get the end-to-end delay bound in the network so they can have an admission control policy. However, they focus on the traffic engineering in a longer timescale than our work. Moreover, they focus on the single class of service as opposed to the two different classes of service in our work.

4.2 System Model

We start with a relatively simple network in Figure 4.1. Similar to Chapter 3, we have two types of high priority traffic, video and voice traffic, which have the same QoS requirements. Due to behavior aggregate queuing, the high priority traffic types share the same queue inside the network. The high priority queue is always served before the low priority queue, which is filled with Best-Effort traffic. We will consider the TCP Reno version, which is the most implemented version, as a protocol used in the Best-Effect traffic. The voice traffic is routed through multiple LSPs from an ingress to egress pair in the network while the video traffic and the Best-Effort traffic share the same bottleneck links in all LSPs. Since the migration mechanism is already proven to be useful in Chapter 3, the voice calls will be migrated between multiple LSPs as a way to do fast timescale traffic engineering.

Although we use the simple network as a starting point, we consider locating the controller to the ingress LSR, which is the point of migration. This can provide a chance to consider the distributed migration algorithm that can be investigated in the future.

There are N LSPs from ingress LSR to egress LSR. There is a propagation delay from the ingress LSR to the bottleneck link in the upper LSP (1^{st} LSP), which will be divided into p_1 time slots. With the same slot length, we will divide the propagation delay from the ingress LSR to the bottleneck link in the other LSP (i^{th} LSP) into p_i time slots, for $i = 2, \ldots, N$. The bottleneck links of all LSPs periodically feed the video utilization information back to the controller. Delay of the i^{th} bottleneck information to be seen by the controller is q_i time slots. So the controller sees the i^{th} LSP's bottleneck information after the delay of q_i time slots,[1] for $i = 1, \ldots, N$. Without loss of

[1]We use different notation on p_i and q_i to represent the fact that the path from ingress node to bottle-

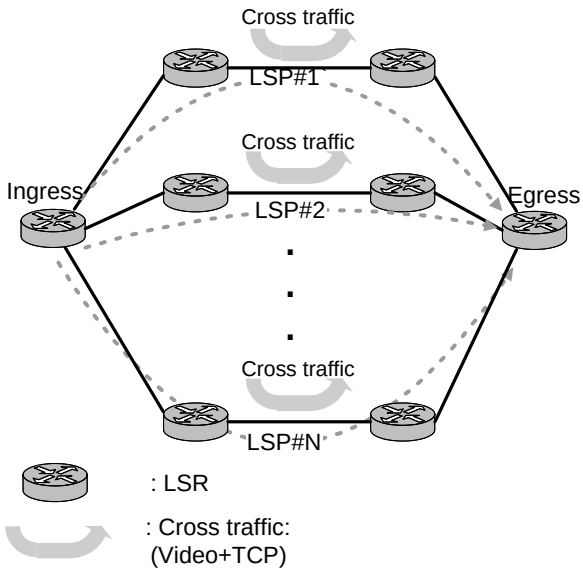


Figure 4.1: The single ingress-egress pair network.

generality, we index the LSPs such that $0 \leq p_1 + q_1 \leq p_2 + q_2 \leq \cdots \leq p_N + q_N$. To further simplify our notation, we define total delay d_i in i^{th} LSP as $d_i \stackrel{\triangle}{=} p_i + q_i$ for $i = 1, \ldots, N$. Now, we have $0 \leq d_1 \leq d_2 \leq \cdots \leq d_N$.

Below are the details of traffic models that we consider:

Voice Traffic We assume that each voice flow has a 64Kbps constant bit rate. The new voice calls arrive according to the Poisson process with rate λ_v. Each voice call has an exponentially distributed duration time with a mean of $\frac{1}{\mu_v}$ seconds. We use a randomized policy such that each new arriving call is dispatched among LSPs according to the statistically splitting rule. In the statistically splitting rule, we assign the probability value $P^{(i)}$ for i^{th} LSP, $i = 1, \ldots, N$. Hence, we will have the new voice call arrival in i^{th} LSP according to the Poisson process with rate

neck link and the path from the bottleneck link to ingress node can be different.

$P^{(i)}\lambda_v$. As a result, the number of voice calls in each LSP is clearly a discrete-time Markov chain with state space $S_{v_i} = \{0, \ldots, C_i\}$ for i^{th} LSP, where C_i is the capacity of bottleneck link in i^{th} LSP, $i = 1, \ldots, N$. (Note that we represent C_i as the unit of voice calls.) A transition probability matrix $M_{v_i} : (S_{v_i} \times S_{v_i}) \rightarrow [0, 1]$ can be calculated from the above voice traffic parameters.

Video Cross Traffic Since the video and voice traffic share the same queue, this cross traffic determines the service rate that voice traffic can be served at the bottleneck link before some of them get dropped. For convenience, instead of specifying the cross traffic distribution directly, we specify the distribution of "service process", which is the difference between the rate of cross traffic and the capacity of bottleneck link C. In i^{th} LSP, $i = 1, \ldots, N$, the service process is represented by a Markov chain with state space $S_i = \{b_{i_1}, \ldots, b_{i_{m_i}}\}$, a transition probability matrix $M_i : (S_i \times S_i) \rightarrow [0, 1]$, and a set of m_i distinct rate values, which are $b_{i_1}, \ldots, b_{i_{m_i}}$.

Low priority Traffic We will use a heuristic function as discussed in Section 2.3 as our model for low priority traffic, which mainly use the TCP protocol.

4.3 MDP State Formulation and Notation

We will formulate our problem as a partially observed Markov Decision Process (MDP). This MDP consists of an action space, a state space, an observation, a state-transition structure, and a reward structure. We describe each of these components in detail as follows:

Action Space Let $u_i(k) \in [-C_i, C_i], i = 1, \ldots, N$, denote the control action made at

time k to the voice calls in i^{th} LSP while C_i is the bottleneck link capacity in that LSP. u_i represents the number of calls that we "migrate **out** of i^{th} LSP". The negativity of $u_i(k)$ means "migration **to** that i^{th} LSP". $\vec{u}(k) = [u_1(k) \dots u_N(k)]$ depend on the current state at time k. Because the calls that migrate from one LSP must go to any of the LSPs, the constraint $\sum_{i=1}^{N} u_i(k) = 0$ is needed. The admissible state-dependent action space $\mathbf{U}(\vec{X}(k))$ is a subset of $\{[-C_1, C_1] \times [-C_2, C_2] \times \cdots \times [-C_N, C_N]\}$ determined by the constraint $\sum_{i=1}^{N} u_i(k) = 0$ together with the regulation that the "out"-migrated flows cannot be greater than the existing voice flows in each LSP.

State Space The system state has two components. The first component is voice traffic variables $\vec{X}(k)$. This $\vec{X}(k)$ is a vector representing the number of voice calls in our sampling interval (time slot) at ingress LSR, i.e.,

$\vec{X}(k) = [X_1(k), X_2(k), \dots, X_N(k)]$. Each element $X_i(k) \in \Re$ is a number of voice calls that is assigned to i^{th} LSP in our sampling interval (time slot) at ingress LSR.[2] Moreover, $X_i(k)$ is bounded by the link capacity C_i. In other words, $0 \leq X_i(k) \leq C_i$ for $i = 1, \dots, N$. More specifically, the vector $\vec{X}(k)$ is the number of voice calls at the ingress LSR on all LSPs at time k. From the above definition, the voice traffic variables takes a value from $\mathbf{S_L} = \{[0, C_1] \times [0, C_2] \times \cdots \times [0, C_N]\}$

The second component is the recently received service rate information, designated by $\vec{b}(k) = [b_1(k) \dots b_N(k)]$, where $b_i(k)$ is the service rate information (at time k) taken from S_i, $i = 1, \dots, N$, as stated in section 4.2. (So $\vec{b}(k)$ take value from $\mathbf{S_V} = \mathbf{S_1} \times \mathbf{S_2} \times \dots \times \mathbf{S_N}$). The state of service process is the recently

[2]We can obtain this information by counting a number of voice packets at ingress LSR in our sampling interval, e.g., from time $k-1$ to time k. Then we divide the number of voice packets with 64Kbps to get a number of voice calls in that sampling interval (time slot).

received state information, instead of the real state at the bottleneck links. We use this recent information because of the information delay, which prevents our controller to gain an immediate access to a real state at the bottleneck link. Hence this $\vec{b}(k)$ is the information that controller has at time k.

To gain a better understanding, we can discuss the relationship of our service rate information and the actual state at the bottleneck link. We first refer to the service process in i^{th} LSP with the above state variable $b_i(k)$ as a service process B_i. Then we define an associated service process $\hat{B}_i$ to be a real service process, which actually occurs the in bottleneck link. We can see that our service process B_i is essentially the actual service process $\hat{B}_i$ with the q_i delay. Note that both service processes are uncontrollable from our control action, i.e., they are unaffected by the control action made by our controller. Hence, we can use a simple equation $b_i(k) = \hat{b}_i(k - q_i)$ to represent the relationship between these two service processes. Where $\hat{b}_i(k)$ is a state at time k of process $\hat{B}_i$.

Using the above two components, we can define our system state variable $Y(k)$ as

$$Y(k) = \begin{bmatrix} \vec{X}(k) \\ \vec{b}(k) \\ \vec{b}(k+1) \\ \vdots \\ \vec{b}(k+d_N) \end{bmatrix}$$

We could see that this state variable is a combination of current voice state, current service rate state, and all of d_N future service rate states. Actually, we could view the state variable as an internal state, which some of them cannot be observed. These future service rate states are required in order to represent the cost function,

which we will define later in this section.

So the complete state space is $\mathbf{S_L} \times \mathbf{S_V}^{(d_N+1)}$.

Observation As we mentioned before, we can partially observe the state variable $Y(k)$. We define observation $Z(k)$ as

$$Z(k) = \begin{bmatrix} \vec{X}(k) \\ \vec{b}(k) \end{bmatrix}$$

In other words, we could observe only the current voice state $\vec{X}(k)$ and current service rate state $\vec{b}(k)$.

State Transition From the current state $Y(k)$, we apply a control $\vec{u}$ so that the system makes a transition to the new state $Y(k+1)$. The transition of each component is as follows:

- The state of service rate information make a transition from $\vec{b}(k)$ to $\vec{b}(k+1)$ with probability $P(b_i(k), b_i(k+1)), i = 1, \ldots, N$, given by the $(b_i(k), b_i(k+1))^{\text{th}}$ entry in the given matrix M_i.[3]
- The voice traffic state $\vec{X}(k)$ makes a transition from $X_i(k)$ to $X_i(k+1)$ (given control action u_i) with probability $P((X_i(k) - u_i), X_i(k+1))$, specified by the $((X_i(k) - u_i), X_i(k+1))^{\text{th}}$ entry in the matrix M_{v_i} for $i = 1, \ldots, N$.

MDP Cost and Value Function We define the one-step cost function at state $Y(k)$ and

[3]Notice that transition of $\vec{b}(k)$ does not depend on control action $\vec{u}(k)$

control action $\vec{u}(k)$ by

$$\begin{aligned} & R\left(Y(k), \vec{u}(k), Y(k+1)\right) \\ & = R_a(Y(k), \vec{u}(k), Y(k+1)) + D_1 \cdot R_b(Y(k), \vec{u}(k), Y(k+1)) \\ & \quad + D_2 \cdot R_c(Y(k), \vec{u}(k), Y(k+1)). \end{aligned} \tag{4.2}$$

This cost function is considered as a penalized cost, in which we would like to minimize. It depends on the current state $Y(k)$, future state $Y(k+1)$ and current control decision $\vec{u}(k)$. Note that we will assign D_1 to be greater than D_2 ($D_1 > D_2$) to reflect higher priority nature of real-time traffic. Each individual term of the cost function $R(Y(k), \vec{u}(k), Y(k+1))$ can be described as follows:

a. Control Actions This cost term depicts the fact that the migrated flows can be impaired, for example, from the out-of-order packets after they are migrated. Also this cost term prevents frequent migration. This prevention is desirable because the frequent flow migration may cause the network to become unstable. The associated cost is represented by

$$R_a(Y(k), \vec{u}(k), Y(k+1)) = ||\vec{u}(k)||^2,$$

where we define $||\vec{u}||^2$ as $\sum_{i=1}^{N} u_i^2$.

b. Drops of Priority Traffic Note that $X_i(k)$ will share the same bottleneck link with $\hat{b}_i(k+p_i)$. Hence, the competition of $X_i(k)$ and $\hat{b}_i(k+p_i)$ over the bottleneck link can cause packet drops. We will use these drops as a penalized cost in this high priority drops term. Recall that $b_i(k) = \hat{b}_i(k-q_i)$. Hence, the drops can be calculated from the value of $X_i(k)$ and $b_i(k+p_i+q_i)$ (or $b_i(k+d_i)$) using a formula $[X_i(k) - b_i(k+d_i)]^+$. As a result, our high

priority drop cost is denoted by

$$R_b(Y(k), \vec{u}(k), Y(k+1)) = \sum_{i=1}^{N} \left([X_i(k+1) - b_i(k+d_i+1)]^+ \right)^2 .$$

c. Reduction of TCP Goodput The reduction of TCP goodput is a result of congestion control in the TCP traffic. Similar to the discussion in the high priority drops cost term above, we can get an available bandwidth in LSP from $[b_i(k+d_i) - X_i(k)]^+$. With the TCP goodput function proposed in Section 2.3.1, we can get a cost associated with TCP goodput reduction as follows:

$$\begin{aligned} & R_c(Y(k), \vec{u}(k), Y(k+1)) \\ & = \sum_{i=1}^{N} f_{\mathrm{TCP}} \left([b_i(k+d_i) - X_i(k)]^+ , [b_i(k+d_i+1) - X_i(k+1)]^+ \right) . \end{aligned}$$

Now we obtain a cost function for our H-stage MDP problem as,

$$E^{[Y(0)]} \left[\sum_{k=0}^{H-1} \gamma^k R(Y(k), u(k), Y(k+1)) \right] ,$$

where $E^{[Y]}[X] \stackrel{\triangle}{=} E[X|Y]$.

4.4 Optimal Policy and Its Difficulty

In this section, we will discuss an optimal policy in our partially observed MDP problem. In order to apply the DP algorithm, we will use state augmentation to reformulate our problem to the one with perfect information. Then we will discuss the difficulty with the DP algorithm for finding the optimal policy.

We define information vector $I(k)$ as

$$I(k) = [Z(0), Z(1), \ldots, Z(k), \vec{u}(0), \vec{u}(1), \ldots, \vec{u}(k-1)], \; I(0) = Z(0).$$

We consider the class of policies that consists of a sequence of functions

$$\pi = \{\mu_0, \mu_1, \ldots, \mu_{H-1}\},$$

where μ_k maps information vector $I(k)$ into control $\vec{u}(k) = \mu_k(I(k))$ and is such that $\mu_k(I(k)) \in \mathbf{U}$ for all $I(k)$. Such policies will be called "admissible". We want to find an admissible policy π that minimizes the value function,

$$J_H^\pi(Z(0)) = J_H^\pi(I(0)) = E^{[I(0)]}\left[\sum_{k=0}^{H-1} \gamma^k R(Y(k), \mu_k(I(k)), Y(k+1))\right].$$

Hence, an optimal policy π^* is one that minimizes the finite horizon value function; that is,

$$J_H^{\pi^*}(Z(0)) = \min_{\pi \in \Pi} J_H^\pi(Z(0)),$$

where Π is the set of all admissible policies.

By using state augmentation, we can reformulate our imperfect information (partially observed) problem to the one with perfect information as follows: With the information vector, we get an evolution of a new system as

$$I(k+1) = (I(k), Z(k+1), \vec{u}(k))$$

and get the dynamic programming algorithm,

$$J_{H-1}(I(H-1)) = \min_{\vec{u}(H-1)} E^{[I(H-1), \vec{u}(H-1)]}\left[R(Y(H-1), \vec{u}(H-1), Y(H))\right] \tag{4.3}$$

$$\begin{aligned} &J_k(I(k)) \\ &= \min_{\vec{u}(k)} E^{[I(k), \vec{u}(k)]}\left[R(Y(k), \vec{u}(k), Y(k+1)) + \gamma \cdot J_{k+1}(I(k), Z(k+1), \vec{u}(k))\right]. \end{aligned} \tag{4.4}$$

Hence, the optimal policy $\pi^* = \{\mu_0^*, \mu_1^*, \ldots, \mu_{H-1}^*\}$ can be obtained by minimizing the right-hand side of the DP Equation (4.3) and (4.4).

We could let H go to infinity and get a value function $J^{\mu^*}(Y(0))$ for a corresponding infinite time horizon problem; that is

$$J^{\mu^*}(Z(0)) = \lim_{H\to\infty} J_H^{\pi^*}(Z(0)).$$

Then, the optimal policy π^* for this infinite-time horizon is $\{\mu^*, \mu^*, \ldots\}$.

The main difficulty with the DP algorithm in Equation (4.3) and (4.4) is that it is carried out over a state space of expanding dimension. As a new observation is added at each stage k, the dimension of the state (the information vector $I(k)$) increases accordingly. This difficulty highlights the importance of state reduction in the next section.

4.5 State Reduction

We already mentioned the difficulty from the expanding state space in the last section. This difficulty motivates an effort to reduce the state that is truly necessary for control purposes. We will use the concept of sufficient statistics to reduce our state space in this section.

First we can write a definition of sufficient statistics[2] as follows

Sufficient statistic is a function $S_k(I(k))$ of Information vector, such that a minimizing control in Equation (4.3) and (4.4) depends on $I(k)$ via $S_k(I_k)$. By this we mean that the minimization in the right-hand side of DP algorithm (4.3) and (4.4) can be written in terms of some function G_k as

$$\min_{u(k)} G_k(S_k(I(k)), u(k)).$$

The salient feature of sufficient statistic is that an optimal policy obtained by the

preceding minimization can be written as

$$\mu_k^*(I(k)) = \overline{\mu}(S_k(I(k))). \tag{4.5}$$

We will propose the observation $Z(k)$ to be the sufficient statistic for our problem. If we can prove that $Z(k)$ is the sufficient statistic following the above definition, we will be able to reformulate our MDP problem using $Z(k)$ as our state. This new formulation will indirectly give the optimal policy of the original problem using Equation (4.5).

Following the above definition, we can prove that $Z(k)$ is a sufficient statistic by writing the minimization in the right hand side of the DP algorithm (4.3) and (4.4) in terms of function G_k as

$$\min_{\vec{u}(k)} G_k(Z(k), \vec{u}(k)).$$

We shall start from Equation (4.3):

$$\begin{aligned}
&J_{H-1}(I(H-1)) \\
&= \min_{\vec{u}(H-1)} E^{[I(H-1),\vec{u}(H-1)]} \left[R(Y(H-1), \vec{u}(H-1), Y(H))\right] \\
&= \min_{\vec{u}(H-1)} E^{[I(H-1),\vec{u}(H-1)]} \\
&\quad \left[\begin{array}{l} ||\vec{u}(H-1)||^2 \\ +D_1 \cdot \sum_{i=1}^{N} \left([X_i(H) - b_i(H+d_i)]^+\right)^2 \\ +D_2 \cdot \sum_{i=1}^{N} f_{\mathrm{TCP}} \left([b_i(H-1+d_i) - X_i(H-1)]^+, [b_i(H+d_i) - X_i(H)]^+\right) \end{array} \right] \\
&= \min_{\vec{u}(H-1)} E^{[Z(H-1),Z(H-2),\ldots,Z(0),\vec{u}(H-2),\ldots,\vec{u}(0),\vec{u}(H-1)]} \\
&\quad \left[\begin{array}{l} ||\vec{u}(H-1)||^2 \\ +D_1 \cdot \sum_{i=1}^{N} \left([X_i(H) - b_i(H+d_i)]^+\right)^2 \\ +D_2 \cdot \sum_{i=1}^{N} f_{\mathrm{TCP}} \left([b_i(H-1+d_i) - X_i(H-1)]^+, [b_i(H+d_i) - X_i(H)]^+\right) \end{array} \right]
\end{aligned}$$

Now we will analyze each term above as follows:

$$
\begin{aligned}
& E^{[Z(H-1),Z(H-2),\ldots,Z(0),\vec{u}(H-2),\ldots,\vec{u}(0),\vec{u}(H-1)]} \\
& \quad \left[\sum_{i=1}^{N} f_{\mathrm{TCP}} \left([b_i(H-1+d_i) - X_i(H-1)]^+ , [b_i(H+d_i) - X_i(H)]^+ \right) \right] \\
& = E^{[\vec{X}(H-1),\vec{X}(H-2),\ldots,\vec{X}(0),\vec{b}(H-1),\vec{b}(H-2),\ldots,\vec{b}(0),\vec{u}(0),\ldots,\vec{u}(H-1)]} \\
& \quad \left[\sum_{i=1}^{N} f_{\mathrm{TCP}} \left([b_i(H-1+d_i) - X_i(H-1)]^+ , [b_i(H+d_i) - X_i(H)]^+ \right) \right] \\
& = E^{[\vec{X}(H-1),\vec{b}(H-1),\vec{u}(0),\ldots,\vec{u}(H-1)]} \\
& \quad \left[\sum_{i=1}^{N} f_{\mathrm{TCP}} \left([b_i(H-1+d_i) - X_i(H-1)]^+ , [b_i(H+d_i) - X_i(H)]^+ \right) \right] \\
& = E^{[\vec{X}(H-1),\vec{b}(H-1),\vec{u}(H-1)]} \\
& \quad \left[\sum_{i=1}^{N} f_{\mathrm{TCP}} \left([b_i(H-1+d_i) - X_i(H-1)]^+ , [b_i(H+d_i) - X_i(H)]^+ \right) \right]
\end{aligned}
$$

The third equality comes from Markov Property of our Markov Decision Process (MDP) problem. The last equality comes from the fact that $\vec{b}(\cdot)$ does not depend on $\vec{u}(\cdot)$. At the same time, $\vec{X}(H)$ depends only on $\vec{X}(H-1)$ and $\vec{u}(H-1)$. Hence, that function of TCP cost does not depend on any of $\{\vec{u}(0), \ldots, \vec{u}(H-2)\}$.

Similar to the TCP cost term above, we can analyze and get the other two terms,

$$
\begin{aligned}
& E^{[Z(H-1),Z(H-2),\ldots,Z(0),\vec{u}(H-2),\ldots,\vec{u}(0),\vec{u}(H-1)]} \left[\sum_{i=1}^{N} \left([X_i(H) - b_i(H+d_i)]^+ \right)^2 \right] \\
& = E^{[\vec{X}(H-1),\vec{b}(H-1),\vec{u}(H-1)]} \left[\sum_{i=1}^{N} \left([X_i(H) - b_i(H+d_i)]^+ \right)^2 \right]
\end{aligned}
$$

and

$$
\begin{aligned}
& E^{[Z(H-1),Z(H-2),\ldots,Z(0),\vec{u}(H-2),\ldots,\vec{u}(0),\vec{u}(H-1)]} \left[||\vec{u}(H-1)||^2 \right] \\
& = E^{[\vec{X}(H-1),\vec{b}(H-1),\vec{u}(H-1)]} \left[||\vec{u}(H-1)||^2 \right] .
\end{aligned}
$$

Next, we can go back to our DP algorithm; That is

$$
\begin{aligned}
& J_{H-1}(I(H-1)) \\
& = \min_{u(H-1)} \\
& \left[\begin{array}{l}
E^{[\vec{X}(H-1),\vec{b}(H-1),\vec{u}(H-1)]} \left[||\vec{u}(H-1)||^2 \right] \\
+D_1 \cdot E^{[\vec{X}(H-1),\vec{b}(H-1),\vec{u}(H-1)]} \left[\sum_{i=1}^{N} \left([X_i(H) - b_i(H+d_i)]^+ \right)^2 \right] \\
+D_2 \cdot E^{[\vec{X}(H-1),\vec{b}(H-1),\vec{u}(H-1)]} \\
\quad \left[\sum_{i=1}^{N} f_{\mathrm{TCP}} \left([b_i(H-1+d_i) - X_i(H-1)]^+ , [b_i(H+d_i) - X_i(H)]^+ \right) \right]
\end{array} \right] \\
& = \min_{u(H-1)} E^{[Z(H-1),\vec{u}(H-1)]} \\
& \left[\begin{array}{l}
||\vec{u}(H-1)||^2 \\
+D_1 \cdot \sum_{i=1}^{N} \left([X_i(H) - b_i(H+d_i)]^+ \right)^2 \\
+D_2 \cdot \sum_{i=1}^{N} f_{\mathrm{TCP}} \left([b_i(H-1+d_i) - X_i(H-1)]^+ , [b_i(H+d_i) - X_i(H)]^+ \right)
\end{array} \right] \\
& = \overline{J}_{H-1}(Z(H-1))
\end{aligned}
$$

Therefore, we can write the DP algorithm (4.3) in term of a function G_{H-1} as

$$\min_{\vec{u}(H-1)} G_{H-1}(Z(H-1), \vec{u}(H-1)).$$

Now, we continue our work on the DP algorithm (4.4). Using an induction, we assume

$$J_k(I(k+1)) = \overline{J}_k(Z(k+1)).$$

Then,

$$
\begin{aligned}
& J_k(I(k)) \\
& = \min_{\vec{u}(k)} E^{[I(k),\vec{u}(k)]} \left[R(Y(k), \vec{u}(k), Y(k+1)) + \gamma \cdot J_{k+1}(I(k), Z(k+1), \vec{u}(k)) \right] \\
& = \min_{\vec{u}(k)} E^{[I(k),\vec{u}(k)]} \\
& \quad \left[\begin{array}{l} ||\vec{u}(k)||^2 \\ +D_1 \cdot \sum_{i=1}^{N} \left([X_i(k+1) - b_i(k+d_i+1)]^+ \right)^2 \\ +D_2 \cdot \sum_{i=1}^{N} f_{\mathrm{TCP}} \left([b_i(k+d_i) - X_i(k)]^+, [b_i(k+d_i+1) - X_i(k+1)]^+ \right) \\ +\gamma \cdot J_{k+1}(I(k), Z(k+1), \vec{u}(k)) \end{array} \right] \\
& = \min_{\vec{u}(k)} E^{[Z(k),Z(k-1),\ldots,Z(0),\vec{u}(k-1),\ldots,\vec{u}(0),\vec{u}(k)]} \\
& \quad \left[\begin{array}{l} ||\vec{u}(k)||^2 \\ +D_1 \cdot \sum_{i=1}^{N} \left([X_i(k+1) - b_i(k+d_i+1)]^+ \right)^2 \\ +D_2 \cdot \sum_{i=1}^{N} f_{\mathrm{TCP}} \left([b_i(k+d_i) - X_i(k)]^+, [b_i(k+d_i+1) - X_i(k+1)]^+ \right) \\ +\gamma \cdot \overline{J}_{k+1}(Z(k+1)) \end{array} \right]
\end{aligned}
$$

Again, we will analyze each terms. First, the next stage cost becomes

$$
\begin{aligned}
& E^{[Z(k),Z(k-1),\ldots,Z(0),\vec{u}(k-1),\ldots,\vec{u}(0),\vec{u}(k)]} \left[\overline{J}_{k+1}(Z(k+1)) \right] \\
& = E^{[Z(k),\vec{u}(k)]} \left[\overline{J}_{k+1}(Z(k+1)) \right].
\end{aligned}
$$

The equality comes from the fact that $\vec{b}(k+1)$ and $\vec{X}(k+1)$ in $Z(k+1)$ are conditionally independent of the other variables given $Z(k)$ and $\vec{u}(k)$.

For the other terms, we could analyze them the same way we did in $\overline{J}_{H-1}(Z(H-1))$. So we complete the analysis.

Next, we can go back to the DP algorithm,

$$
\begin{aligned}
& J_k(I(k)) \\
& = \min_{\vec{u}(k)} \\
& \left[\begin{array}{l}
E^{[\vec{X}(k),\vec{b}(k),\vec{u}(k)]} \left[||\vec{u}(k)||^2 \right] \\
+D_1 \cdot E^{[\vec{X}(k),\vec{b}(k),\vec{u}(k)]} \left[\sum_{i=1}^{N} \left([X_i(k+1) - b_i(k+d_i+1)]^+ \right)^2 \right] \\
+D_2 \cdot E^{[\vec{X}(k),\vec{b}(k),\vec{u}(k)]} \\
\left[\sum_{i=1}^{N} f_{\mathrm{TCP}} \left([b_i(k+d_i) - X_i(k)]^+ , [b_i(k+d_i+1) - X_i(k+1)]^+ \right) \right] \\
+\gamma \cdot E^{[\vec{X}(k),\vec{b}(k),\vec{u}(k)]} \left[\overline{J}_{k+1}(Z(k+1)) \right]
\end{array} \right] \\
& = \min_{\vec{u}(k)} E^{[Z(k),\vec{u}(k)]} \\
& \left[\begin{array}{l}
||\vec{u}(k)||^2 \\
+D_1 \cdot \sum_{i=1}^{N} \left([X_i(k+1) - b_i(k+d_i+1)]^+ \right)^2 \\
+D_2 \cdot \sum_{i=1}^{N} f_{\mathrm{TCP}} \left([b_i(k+d_i) - X_i(k)]^+ , [b_i(k+d_i+1) - X_i(k+1)]^+ \right) \\
+\gamma \cdot \overline{J}_{k+1}(Z(k+1))
\end{array} \right] \\
& = \overline{J}_k(Z(k)).
\end{aligned}
$$

Therefore, we can write the DP algorithm (4.4) in term of function G_k as

$$\min_{\vec{u}(k)} G_k(Z(k), \vec{u}(k))$$

QED.

So, $Z(k)$ is a sufficient statistic of $I(k)$, and we have the new dynamic

programming algorithm as,

$$
\begin{aligned}
&\overline{J}_{H-1}(Z(H-1)) \\
&= \min_{u(H-1)} E^{[Z(H-1),\vec{u}(H-1)]} \\
&\left[\begin{array}{l}
||\vec{u}(H-1)||^2 \\
+D_1 \cdot \sum_{i=1}^{N} \left([X_i(H) - b_i(H+d_i)]^+\right)^2 \\
+D_2 \cdot \sum_{i=1}^{N} f_{\mathrm{TCP}} \left([b_i(H-1+d_i) - X_i(H-1)]^+, [b_i(H+d_i) - X_i(H)]^+\right)
\end{array}\right]
\end{aligned}
$$

$$
\begin{aligned}
&\overline{J}_{k}(Z(k)) \\
&= \min_{\vec{u}(k)} E^{[Z(k),\vec{u}(k)]} \\
&\left[\begin{array}{l}
||\vec{u}(k)||^2 \\
+D_1 \cdot \sum_{i=1}^{N} \left([X_i(k+1) - b_i(k+d_i+1)]^+\right)^2 \\
+D_2 \cdot \sum_{i=1}^{N} f_{\mathrm{TCP}} \left([b_i(k+d_i) - X_i(k)]^+, [b_i(k+d_i+1) - X_i(k+1)]^+\right) \\
+\gamma \cdot \overline{J}_{k+1}(Z(k+1))
\end{array}\right]
\end{aligned}
$$

Now, we can reformulate the problem; we will define the new state variable as[4]

$$\tilde{Y}(k) = Z(k) = \begin{bmatrix} \vec{X}(k) \\ \vec{b}(k) \end{bmatrix}.$$

We have a new system with a infinite horizon cost,

$$\lim_{H\to\infty} E^{[\tilde{Y}(0)]} \left[\sum_{k=0}^{H-1} \gamma^k R'(\tilde{Y}(k), u(k)) \right].$$

where, $R'(\tilde{Y}(k), u(k))$ is a combination of control cost, high priority drop cost and TCP goodput cost, i.e.,

$$\begin{aligned} & R'(\tilde{Y}(k), u(k)) \\ & = E^{[\tilde{Y}(k), \vec{u}(k)]} \\ & \begin{bmatrix} ||\vec{u}(k)||^2 \\ +D_1 \cdot \sum_{i=1}^{N} \left([X_i(k+1) - b_i(k+d_i+1)]^+ \right)^2 \\ +D_2 \cdot \sum_{i=1}^{N} f_{\mathrm{TCP}} \left([b_i(k+d_i) - X_i(k)]^+, [b_i(k+d_i+1) - X_i(k+1)]^+ \right) \end{bmatrix} \end{aligned}$$

[4]We replace observation variable $Z(k)$ with a new state variable $\tilde{Y}(k)$ to prevent a confusion with z_k in subsequence sections.

Chapter 5

Subobtimal Policy: Flow Migration Problem

In Chapter 4, we initially formulated the migration problem as a partially observed MDP problem. Later in the chapter, we showed that this partially observed MDP problem can be reduced to the perfect information MDP problem as far as control is concerned. That is, we will automatically get the optimal control policy in the partially observed MDP problem if we solve the equivalent perfect information MDP problem. Moreover, the equivalent perfect information MDP has a much smaller state space. As a result, it is wise to work on the perfect information MDP problem in this chapter, instead of working on the original partially observed MDP problem.

Although we achieved a large reduction in state space by focusing on the perfect information MDP problem, we still got into a well-known curse of dimensionality problem. With this problem, the number of states often grows exponentially with the number of state variables. This exponential growth can easily induce large state space. The resulting large state space could in turn create a difficulty in finding an optimal policy (even after we apply state reduction technique). Hence, it is nice to design policies that yield a good approximation to the optimal policy and can be easily calculated. In this chapter, we will propose some sub-optimal policies for the migration problem. Then, we will evaluate and compare their performance using the Temporal

Difference (TD) learning and the packet-level simulation.

5.1 Linear System with Quadratic Cost Approach

The linear system with quadratic cost problem has been well studied in the control community[2]. Its analytically closed-form solution makes this problem very attractive. Moreover, the closed-form solution has the potential to work well in a large-scale problem. Hence, it could help us cope with state explosion in our migration MDP problem.

Moreover, the linear system with quadratic cost (LQ) problem and our migration problem have very similar intuitive objectives and both objectives have a strong correlation. The objective in our problem is “Keep drops close to zero with small amount of migration”. Note that we use migration as our control action. On the other hand, the objective of the LQ problem is “Keep state variables close to zero with small amount of control action”. From the above two objectives, we could easily notice their similarity. Now, we will try to illustrate their relationship. We slightly change the objective of the linear system with quadratic cost (LQ) problem by elevating the system with a constant value, e.g., mean of link utilization. The updated objective becomes “Keep state variables close to their means with small amount of control action”. Note that drops will occur when the amount of traffic is over link capacity. So, small drops can be obtained if we keep traffic utilization close to its mean. This fact yields a nice relationship between our migration’s objective and the objective of the linear system with quadratic cost problem. If we pursue the objective of the LQ problem by keeping traffic near the mean, this could lead to our migration’s objective getting minimum drops. Therefore, we could get a good approximation to our migration problem

working on the context of the linear system with quadratic cost problem.

Before we go into detail, we establish the useful notation for our LQ approach:

I_N is identity matrix with dimension $N \times N$,

0_N is all zeroes matrix with dimension $N \times N$,

1_N is all ones matrix with dimension $N \times N$,

where N is a total number of LSPs.

5.1.1 Linear System with Quadratic Cost Formulation

We will now formulate our migration problem in a context of the LQ problem. Then we will argue that this formulation yields a good approximation to migration problem.

State Variables

$$X_k \stackrel{\triangle}{=} \begin{bmatrix} x_1(k) \\ x_2(k) \\ \vdots \\ x_N(k) \\ v_1(k+d_1) \\ v_2(k+d_2) \\ \vdots \\ v_N(k+d_N) \end{bmatrix}$$

We define X_k as a vector of $2N$ state variables; N variables associate with voice calls $\{x_1(k), \ldots, x_N(k)\}$ and other N variables associated with video $\{v_1(k), \ldots, v_N(k)\}$. For $i = 1, \ldots, N$, $x_i(k)$ is a difference of voice calls from its mean in i^{th} LSP, i.e.,

$$x_i(k) \stackrel{\triangle}{=} X_i(k) - \bar{X}_i,$$

where $X_i(k)$ is the amount of voice calls at time k in i^{th} LSP (in unit of voice calls) and $\bar{X}_i$ is the mean of voice calls when there is no migration, which we assume to be fixed at all time k.

In the same manner, for $i = 1, \ldots, N$, we define $v_i(k)$ as the difference of video traffic from its mean in i^{th} LSP, i.e.,

$$v_i(k) \triangleq v_i^r(k) - \bar{v}_i,$$

where $v_i^r(k)$ is the real amount of video traffic at time k in i^{th} LSP (in unit of voice calls), i.e., $v_i^r(k) \triangleq C_i - b_i(k)$.[1] In addition, $\bar{v}_i$ is the mean of video traffic $v_i^r(k)$, which we assume to be fixed at all time k. Note that we use a future video states $v_i(k + d_i)$, for $i = 1, 2, \ldots, N$, as our state variables. These future states are required in order to represent the cost function, which we will discuss later in this section.

In conclusion, the state variables, X_k, represent the deviation of voice and video traffic from their means.

Control Actions

$$U_k \triangleq \begin{bmatrix} u_1(k) \\ u_2(k) \\ \vdots \\ u_{N-1}(k) \end{bmatrix}$$

We define U_k as a vector of $N - 1$ independent control actions $\{u_1(k) \ldots u_{N-1}(k)\}$ at time k. For $i = 1, \ldots, N$, each $u_i(k)$ indicates the amount of calls migration *into* that i^{th} LSP. In other words, $u_i(k)$ is a control action to voice traffic in i^{th} LSP. Note that we

[1] Recall from Chapter 4, we defined the state variable b_i as a service rate information, which represents the amount of bandwidth left after the usage of video traffic.

have a constraint on control actions $\sum_{i=1}^{N} u_i(k) = 0$. The constraint represents the fact that voice calls must go from one LSP to any of the other LSPs without loss from the migration mechanism. With that constraint, we can obtain the control action for N^{th} LSP immediately from our vector U_k, i.e., $u_N(k) = -\sum_{i=1}^{N-1} u_i(k)$. So it is logical not to include $u_N(k)$ in U_k.

State Equation

$$X_{k+1} = AX_k + BU_k + W_k,$$

where

$$A = I_{2N},\ B = \begin{bmatrix} & & & & \\ & I_{N-1} & & & \\ & & & & \\ -1 & -1 & \cdots & -1 & -1 \\ & & & & \\ & 0_{N-1} & & & \\ & & & & \\ 0 & 0 & \cdots & 0 & 0 \end{bmatrix},\ W_k \triangleq \begin{bmatrix} w_1(k) \\ w_2(k) \\ \vdots \\ w_N(k) \\ w_{v_1}(k) \\ w_{v_2}(k) \\ \vdots \\ w_{v_N}(k) \end{bmatrix}.$$

We define W_k as a vector of noise variables associated with each element in X_k. These noises emulate all uncertainty in the system. The first N variables, $\{w_1(k) \ldots w_N(k)\}$, emulate the voice calls arrival and departure, while the last N variables, $\{w_{v_1}(k) \ldots w_{v_N}(k)\}$, emulate the fluctuation of the video traffic model. We let the means of all noise variables be set equal to zero so it would emulate the fluctuation of state variables around their means. In fact, in the real system, W_k has to depend on X_k. (Call departure depends on the number of current voice calls, and video fluctuation depends on the current state of the video traffic model.) However, to obtain

a closed-form solution, noise variables W_k in the LQ problem context have to be independent random variables with a finite second moment. Moreover, W_k should not depend on X_k and U_k[2]. Therefore, we will have to assume independence and assign the second moment of W_k to be finite.[2] This assumption yields another approximation to our original migration problem.

From the state equation above, we can see that the next state variable X_{k+1} is a weighted sum of current state variables X_k, control actions U_k, and uncertainty variables W_k. Our migration handles only voice calls. So, the last N rows of B associated with video is all zeros. Moreover, the N^{th} row of B are all -1 to represent the fact that $u_N(k)$ can be obtained from other $\{u_1(k) \dots u_{N-1}(k)\}$ with formula $u_N(k) = -\sum_{i=1}^{N-1} u_i(k)$ as described above. Recall that $u_i(k)$ is a control action to voice traffic in i^{th} LSP.

Cost Function

$$E_{w_k}\left[X_0' Q X_0 + \gamma \sum_{k=0}^{\infty} \gamma^k (X_{k+1}' Q X_{k+1} + U_k' R U_k)\right], \tag{5.1}$$

[2]Note that, for $i = 1, \dots, N$, the voice traffic in i^{th} LSP has a long-term poisson distributed number of calls with mean and variance $\frac{p_i \lambda_v}{\mu}$. Hence, we assign the noise variables associated with voice variables to have their second moment $E[|w_i(k)|^2] = \frac{p_i \lambda_v}{\mu} + (\frac{p_i \lambda_v}{\mu})^2$. Recall from Section 2.1, video traffic is modelled by birth-death Markov chain with mean $p_\alpha MA$ and variance $MA^2 p_\alpha (1 - p_\alpha)$, where $p_\alpha = \frac{\alpha}{\alpha+\beta}$. Hence, we let the noise variables associated with video traffic to have their second moment $E[|w_{v_i}(k)|^2] = MA^2 p_\alpha (1 - p_\alpha) + (p_\alpha MA)^2$.

where

$$\begin{aligned} R &= 1_{N-1} + I_{N-1}, \\ Q &= C'C, \\ C &= \sqrt{D_1} \cdot \mathrm{diag}\left([\gamma^{d_1/2}\ \gamma^{d_2/2}\ \cdots\ \gamma^{d_N/2}]\right) \cdot [I_N\ \ I_N], \\ D_1 &> 1. \end{aligned}$$

The cost function in Equation (5.1) is a nice approximation to the cost function in our migration problem. It represents two important kinds of terms, the drops cost terms (with $X_k'QX_k$)and control cost terms (with $U_k'RU_k$). Note that we do not have any terms representing TCP cost. However, TCP cost is significantly less important than other costs because TCP traffic is in the lower priority class, as we mentioned in Section 4.3. So, Equation (5.1) could be a good approximation to our migration problem's cost function. We will now describe the relationship between the cost function in Equation (5.1) and the cost function of our original migration problem. We start with the control cost terms represented by $U_k'RU_k$. From $R = 1_{N-1} + I_{N-1}$, we could simplify $U_k'RU_k$ into scalar form as,

$$\begin{aligned} U_k'RU_k &= U_k'1_{N-1}U_k + U_k'I_{N-1}U_k \\ &= \left[-\sum_{i=1}^{N-1} u_i(k)\right]^2 + \sum_{i=1}^{N-1}[u_i(k)]^2 \\ &= [u_N]^2 + \sum_{i=1}^{N-1}[u_i(k)]^2 = \sum_{i=1}^{N}[u_i(k)]^2. \end{aligned}$$

We could see that this $U_k'RU_k$ is exactly the control cost term in our original migration problem.[3]

[3]Recall that we want to minimize the following cost in our migration problem,

$$\lim_{H\to\infty} E^{[\tilde{Y}(0)]}\left[\sum_{k=0}^{H-1}\gamma^k R'(\tilde{Y}(k), u(k))\right],$$

Now, we will discuss about drop cost terms represented by $X_k' Q X_k$, $k \in \{1, \ldots, \infty\}$. These terms actually are an approximation of the high priority drop cost term in our original migration problem. Recall that $C = \sqrt{D_1} \cdot \text{diag}\left([\gamma^{d_1/2} \ \gamma^{d_2/2} \ \cdots \ \gamma^{d_N/2}]\right) \cdot [I_N \ \ I_N]$. Hence,

$$X_k' Q X_k = X_k' C' C X_k = D_1 \cdot \sum_{i=1}^{N} \gamma^{d_i} \cdot \left(x_i(k) + v_i(k+d_i)\right)^2 .$$

Recall that $\bar{X}_i$ is a mean of voice traffic, and $\bar{v}_i$ is a mean of video traffic in i^{th} LSP, for $i \in \{1, \ldots, N\}$. So the mean of total traffic utilization in i^{th} LSP is $\bar{X}_i + \bar{v}_i$ in unit of voice calls. Then,

$$\begin{aligned} \left(x_i(k) + v_i(k+d_i)\right) &= \left(X_i(k) - \bar{X}_i\right) + \left(v_i^r(k+d_i) - \bar{v}_i\right) \\ &= \left(X_i(k) + v_i^r(k+d_i)\right) - \left(\bar{X}_i + \bar{v}_i\right). \end{aligned}$$

We could see that it represents the deviation of the traffic utilization from the utilization mean. Note that our cost function penalizes this deviation. So the deviation will be kept close to zero with optimal control action. This in turn leads to small drops as we wanted in our migration problem's objective. As we already discussed in Chapter 4 when we defined a cost function for our original problem, $x_i(k)$ will share the same bottleneck link with $v_i(k+d_i)$. Hence, $x_i(k) + v_i(k+d_i)$ will correctly represent the deviation of the traffic utilization from the utilization mean at the bottleneck link.

---

where $R'(\tilde{Y}(k), u(k))$ is a combination of control cost, high priority drop cost, and TCP goodput cost,

$$\begin{aligned} & R'(\tilde{Y}(k), u(k)) \\ &= E^{[\tilde{Y}(k), \vec{u}(k)]} \left[\begin{array}{l} ||\vec{u}(k)||^2 \quad + D_1 \cdot \sum_{i=1}^{N} \left([X_i(k+1) - b_i(k+d_i+1)]^+\right)^2 \\ +D_2 \cdot \sum_{i=1}^{N} f_{\text{TCP}}\left([b_i(k+d_i) - X_i(k)]^+, [b_i(k+d_i+1) - X_i(k+1)]^+\right) \end{array} \right]. \end{aligned}$$

In conclusion, the cost function in this LQ formulation is close to the cost function in our migration problem.

5.1.2 Solution to Linear System with Quadratic Cost Formulation

The cost function in Equation (5.1) can be rearranged to

$$E_{w_k}\left[\sum_{k=0}^{\infty}\gamma^k(X_k'QX_k+\gamma U_k'RU_k)\right].$$

We let $A_\gamma=\sqrt{\gamma}A$, $B_\gamma=\sqrt{\gamma}B$ and $R_\gamma=\gamma R$. Then, we get the stationary control vector U_k from Equation (5.2),

$$\begin{aligned} U_k &= \mu^*(X_k) \\ &= -(B_\gamma'PB_\gamma+R_\gamma)^{-1}B_\gamma'PA_\gamma X_k, \end{aligned} \tag{5.2}$$

and value function $\hat{V}_{\mathrm{LQ}}(X_k)$ from Equation (5.3),

$$\hat{V}_{\mathrm{LQ}}(X_k) = X_k'PX_k+\tfrac{\gamma}{1-\gamma}E[W_k'PW_k], \tag{5.3}$$

where we can obtain P from Riccati's Equation [8]:

$$P=A_\gamma'[P-PB_\gamma(B_\gamma'PB_\gamma+R_\gamma)^{-1}B_\gamma'P]A_\gamma+Q.$$

To guarantee the existence of P, we need the following requirements:

1. $Q\geq 0,\ R_\gamma>0$
2. (A_γ,B_γ) stabilizable
3. (A_γ,C) detectable; $C'C=Q$

All parameters in our LQ formulation satisfy all of the above requirements. For the second and third requirements, A has eigenvalue 1, which makes A_γ to have eigenvalue $\sqrt{\gamma}$. Hence, (A_γ,B_γ) is stabilizable, and (C,A_γ) is detectable.

For the first requirement, we have to prove that Q is positive semi-definite and R is positive definite. Note that C has full rank and $Q = C'C$. Hence, Q is positive semi-definite.

To prove that R is positive definite, we first claim that the $N-1$ eigenvalues of 1_{N-1} are $\{N-1, 0, 0, 0, \ldots, 0\}$. This claim follows the fact that 1_{N-1} has rank one. So it will have only one non-zero eigenvalue. This fact explains all zero eigenvalues. Note that the trace of matrix equals a summation of its eigenvalues, i.e., $tr(A) = \sum_{i=1}^{N} \lambda_i$, where $\{\lambda_1, \lambda_2, \ldots, \lambda_N\}$ are eigenvalues of N by N matrix A. Hence, we can get a single eigenvalue as $N-1$ because $tr(1_{N-1}) = N-1$.

Then we can further claim that the eigenvalues of $R_\gamma = \gamma \cdot (1_{N-1} + I_{N-1})$ are $\{\gamma N, \gamma, \gamma, \ldots, \gamma\}$. This claim follows the fact that eigenvalues of $A + cI$ are $\{\lambda_1 + c, \ldots, \lambda_N + c\}$ if $\{\lambda_1, \lambda_2, \ldots, \lambda_N\}$ are eigenvalues of N by N matrix A, where c is a scalar and I is an identity matrix. Using this fact and the previous claim, we will know that the eigenvalues of $1_{N-1} + I_{N-1}$ are $\{N, 1, 1, \ldots, 1\}$. Hence, the eigenvalues of $R_\gamma = \gamma \cdot (1_{N-1} + I_{N-1})$ are $\{\gamma N, \gamma, \gamma, \ldots, \gamma\}$.

From all positive eigenvalues of R_γ, we can conclude that R_γ is positive definite ($R_\gamma > 0$).

5.1.3 Example of a Two LSP Case

We assume that the total delay in the 1^{st} LSP is 2, and the total delay in the 2^{nd} LSP is 1, i.e., $d_1 = 2$ and $d_2 = 1$. Moreover, we let the discount factor $\gamma = 0.7$, and $D_1 = 0.3$.

Then, we got

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},\ X_k = \begin{bmatrix} x_1(k) \\ x_2(k) \\ v_1(k+2) \\ v_2(k+1) \end{bmatrix},$$

$$B = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0 \end{bmatrix},\ U_k = \begin{bmatrix} u_1(k) \end{bmatrix},\ W_k = \begin{bmatrix} w_1(k) \\ w_2(k) \\ w_{v_1}(k) \\ w_{v_2}(k) \end{bmatrix}.$$

and

$$Q = D_1 \cdot \begin{bmatrix} \gamma^{d_1} & 0 & \gamma^{d_1} & 0 \\ 0 & \gamma^{d_2} & 0 & \gamma^{d_2} \\ \gamma^{d_1} & 0 & \gamma^{d_1} & 0 \\ 0 & \gamma^{d_2} & 0 & \gamma^{d_2} \end{bmatrix}.\ \text{So, } C = \sqrt{D_1} \cdot \begin{bmatrix} \gamma^{d_1/2} & 0 & \gamma^{d_1/2} & 0 \\ 0 & \gamma^{d_2/2} & 0 & \gamma^{d_2/2} \end{bmatrix}.$$

Let $A^\gamma = \sqrt{\gamma} \cdot A$, $B^\gamma = \sqrt{\gamma} \cdot B$ and $R = 1$ (since the dimension of U_k is one).

We can use MATLAB to obtain a solution P for the Riccati's equation,

$$P = \begin{bmatrix} 0.41238 & 0.11089 & 0.41238 & 0.11089 \\ 0.11089 & 0.54158 & 0.11089 & 0.54158 \\ 0.41238 & 0.11089 & 0.41238 & 0.11089 \\ 0.11089 & 0.54158 & 0.11089 & 0.54158 \end{bmatrix}.$$

Then,

$$u_1(k) = \begin{bmatrix} -0.11035 & 0.15764 & -0.11035 & 0.15764 \end{bmatrix} \cdot \begin{bmatrix} x_1(k) \\ x_2(k) \\ v_1(k+2) \\ v_2(k+1) \end{bmatrix}$$

5.1.4 Separation Theorem for Linear System with Quadratic Cost

We can notice that the optimal control in our LQ formulation is a function of current voice state $x_i(k)$ and future video state $v_i(k+d_i)$. However, we have only current voice state information $x_i(k)$ and current video state $v_i(k)$ available at time k. So we need to associate our LQ solution with the state information at time k. Fortunately, we work on a nice system, a linear system with quadratic cost. There is a useful theorem for this system called *Separation Theorem*, which occupies a central position in modern automatic control theory[2]. The theorem stated that the optimal controller in a linear system with quadratic cost can be decomposed into two parts, which are the estimator part and the actuator part. The estimator part uses the data $I(k)$ to estimate state $X(k)$. The optimal estimate is a conditional expectation $E[X(k)|I(k)]$. The actuator part applies the optimal solution from linear system with quadratic cost formulation to the system.

With the help from the separation theorem, we can obtain the optimal control directly from the current information, which is available at time k (current voice state information $x_i(k)$ and current video state $v_i(k)$). In the estimator part, we can first calculate $E[v_i(k+d_i)|v_i(k)]$ as an estimate of $v_i(k+d_i)$ given the current information $v_i(k)$. Then we could use our linear system with quadratic cost solution as an actuator part.

In conclusion, we can get an optimal control in our linear system with quadratic cost formulation as follows: We first calculate $E[v_i(k+d_i)|v_i(k)]$ as an estimate of $v_i(k+d_i)$ given the current information $v_i(k)$. Then, we can put this estimate instead of $v_i(k+d_i)$ in Equation (5.2); That is we get a stationary control vector U_k from

Equation (5.4) below,

$$U_k = -(B'_\gamma P B_\gamma + R_\gamma)^{-1} B'_\gamma P A_\gamma \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \\ E[v_1(k+d_1)|v_1(k)] \\ E[v_2(k+d_2)|v_2(k)] \\ \vdots \\ E[v_N(k+d_N)|v_N(k)] \end{bmatrix}, \qquad (5.4)$$

where $E[v_i(k+d_i)|v_i(k)]$ can be analytically calculated using Lemma 5.1.1 below.

Lemma 5.1.1. *Given a current video state $v_i(k)$ for i^{th} LSP, the expected value of the future video state $E[v_i(k+d_i)|v_i(k)]$ can be obtained from*

$$E[v_i(k+d_i)|v_i(k)] = v_i(k)e^{-(\alpha+\beta)d_i T},$$

where T is the time-step length. α and β are parameters in the video model.

Proof Lemma 5.1.1

Recall that $v_i(k) \stackrel{\triangle}{=} v_i^r(k) - \bar{v}_i$ where $v_i^r(k)$ is the real amount of video traffic at time k in i^{th} LSP (in unit of voice calls), and $\bar{v}_i$ is the mean of video traffic $v_i^r(k)$, which we assume to be fixed at all time k. Let $y(t)$ be a scaled continous-time version of $v_i^r(k)$, i.e., $v_i^r(k) = A \cdot y(kT)$, where T is the length of our time-step. Recall from Section 2.1, this $y(t)$ is a continuous-time Markov chain with a quantization step rate of A bps. Letting $M_y(t) = E[y(t)|y(0) = i]$, we will derive a differential equation

satisfied by $M_y(t)$. To begin, note that, given $y(t)$:

$$y(t+h) = \begin{cases} y(t)+1 & \text{with probability } [M - y(t)]\,\alpha h + o(h) \\ y(t)-1 & \text{with probability } y(t)\beta h + o(h) \\ y(t) & \text{with probability } 1 - [M\alpha + (\beta - \alpha)y(t)]h + o(h) \end{cases}$$

where, A, α, β and M are the parameters associated with our video model defined in Section 2.1.

Hence, taking expectations yields

$$E[y(t+h)/y(t)] = y(t) + [M - y(t)]\,\alpha h - y(t)\beta h + o(h).$$

Take expectations respected to $y(0)$ once again yields

$$M_y(t+h) = M_y(t) + M\alpha h - (\alpha + \beta)y(t)h + o(h),$$

or

$$\frac{M_y(t+h) - M_y(t)}{h} = M\alpha - (\alpha + \beta)y(t) + \frac{o(h)}{h}.$$

Letting $h \to 0$ gives

$$M_y'(t) = M\alpha - (\alpha + \beta)M_y(t).$$

As a result, we get

$$M_y(t) = \frac{M\alpha}{\alpha + \beta} + Ke^{-(\alpha+\beta)t}.$$

Since $M_y(0) = \frac{M\alpha}{\alpha+\beta} + K = i$, $K = i - \frac{M\alpha}{\alpha+\beta}$, we obtain

$$M_y(t) = \frac{M\alpha}{\alpha + \beta} + \left(i - \frac{M\alpha}{\alpha + \beta}\right) e^{-(\alpha+\beta)t}.$$

Recall that $v_i^r(k) = A \cdot y(kT)$. Hence,

$$E[v_i^r(k+d_i)|v_i^r(k)] \;\; = \;\; A \cdot \tfrac{M\alpha}{\alpha+\beta} + \left(v_i^r(k) - A \cdot \tfrac{M\alpha}{\alpha+\beta}\right) e^{-(\alpha+\beta)d_iT}.$$

Also $v_i(k) = v_i^r(k) - \bar{v}_i$. Hence,

$$E[v_i(k+d_i) + \bar{v}_i | v_i(k) + \bar{v}_i] \quad = \quad A \cdot \frac{M\alpha}{\alpha+\beta} + \left(v_i(k) + \bar{v}_i - A \cdot \frac{M\alpha}{\alpha+\beta} \right) e^{-(\alpha+\beta)d_i T}.$$

We could get a mean of video traffic[33] from $\bar{v}_i = A \cdot \frac{M\alpha}{\alpha+\beta}$. Then,

$$E[v_i(k+d_i) | v_i(k)] \quad = \quad v_i(k) e^{-(\alpha+\beta)d_i T}.$$

QED.

5.1.5 Randomized Rounding

Control actions obtained from Equation (5.4) can be any real number. However, the nature of our migration control is an integer number. The migration control operates on voice traffic by moving one or more calls from one LSP to other LSPs. The voice call cannot be moved fractionally because it can be moved only by an integer number of calls. So our control action needs to be integer also.

Srinivasan [36] explained an approach to obtain an approximation solution to Integer Programming. The approach started with relaxing an integer constraint. Then, he obtained a real number solution of that relaxed problem. Finally, he used Randomized Rounding to get an integer solution from the relaxed solution. This approach yields a nice bound to optimal solution.

The real number solution from a linear system with quadratic cost solver can be viewed as relaxation in the integer programming context. Then, we can obtain an integer solution from Randomized Rounding. In our problem, the procedure of Randomized Rounding can be illustrated with the following example. Suppose we get $u_5(k) = 3.4$ (control action to 5^{th} LSP) from the solver. We will round this to an

integer value $\hat{u}_5(k) = 3$ with probability 0.6 and to integer value $\hat{u}_5(k) = 4$ with probability 0.4. This rounding will get $E[\hat{u}_5(k)] = 3.4$, which is exactly the number $u_5(k)$ from the linear system with quadratic cost solver.

The complete algorithm of our linear system with quadratic cost approach for the migration problem is shown in Algorithm 2. The algorithm needs a state of the system, X_k, as an input and it will generate $\{\hat{u}_1, \ldots, \hat{u}_N\}$ as an output of the algorithm.

LQ(X_k)

$$U_k = -(B'_\gamma P B_\gamma + R_\gamma)^{-1} B'_\gamma P A_\gamma \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \\ E[v_1(k+d_1)|v_1(k)] \\ E[v_2(k+d_2)|v_2(k)] \\ \vdots \\ E[v_N(k+d_N)|v_N(k)] \end{bmatrix}$$

for $u_i \in U_k$

$$\hat{u}_i(k) = \begin{cases} \lceil u_i(k) \rceil & \text{with probability} \quad u_i(k) - \lfloor u_i(k) \rfloor \\ \lfloor u_i(k) \rfloor & \text{with probability} \quad \lceil u_i(k) \rceil - u_i(k) \end{cases}$$

$$\hat{u}_N(k) = -\sum_{i=1}^{N-1} \hat{u}_i(k)$$

*Note that $\lfloor x \rfloor$ is the largest integer that is less than x if $x \geq 0$, or the smallest integer that is greater than x if $x < 0$. $\lceil x \rceil$ is the smallest integer that is greater than x if $x \geq 0$, or the largest integer that is less than x if $x < 0$. P is pre-calculated off-line.

Algorithm 2: The algorithm of linear system with quadratic cost approach.

5.1.6 Conclusion on Linear System with Quadratic Cost Approach

We modify the model in our migration problem to suit for linear system with quadratic cost formulation. Then, we use that modified model, the LQ model, to approximate the optimal solution in our migration problem. There are four modifications from the original model to our LQ model. First, we make all *uncertainty* in the LQ model independent from its state variable. (In the original model, uncertainty depends on state variables, e.g., call departure depends on number of voice calls, which is one of the state variables.) Second, we use deviation of traffic from its mean as a way to quantify a chance of getting drops. By penalizing that deviation in our new cost function, we can also approximate a goal to minimize drops. Third, we omit the cost associated with the TCP traffic. Because of the relatively small cost compared to drops and control action in the original cost function, we still get a good approximation to the original model. Fourth, we let the solution from the LQ model to be a real number instead of strictly an integer number in the original model. With that real number solution, we use Randomized Rounding to convert it to a good integer solution later.

These modifications drive our LQ solution away from the optimal value. However, they provide a closed-form solution that simplifies online calculation and in turn scales well with large state space in our migration problem.

5.2 Simulated Lookahead Approach

Limited lookahead is a well-known technique in dynamic programming to approximate an optimal solution. Given good approximation of the optimal value function[4] $V^{\mu^*}(\cdot)$, the limited lookahead can achieve a nearly optimal solution [2]. In our migration problem, we use a value function from the linear system with quadratic cost approach as an approximated value function for the limited lookahead. This approximated value function has a big advantage from its closed form solution. The closed form solution can be calculated quickly with a small memory requirement. Hence, it reduces the computational complexity of the limited lookahead, especially an online version of the limited lookahead. We chose an online version because it scales better with a large state space. We also use a simulation to better cope with large state space. More precisely, we use simulation to approximate the expectation in the limited lookahead formula. We first explore a simulation approach that directly derives from the Monte-Carlo simulation. This combination of the limited lookahead and Monte-Carlo simulation, which we named Monte-Carlo Lookahead, have been proposed and extensively used in other applications. Then, we introduce a new approach that is better suited to the nature of our fast timescale problem. The small time and processing power in the fast timescale control limits the number of simulation samples generated. Our new approach, which we named Iterative Lookahead, uses additional information to reduce a need of large simulation samples.

Online vs. Off-line So far we have explored off-line approaches, i.e., linear system

[4]The optimal value function $V^{\mu^*}(\tilde{Y}(0))$ can be obtained from

$$V^{\mu^*}(\tilde{Y}(0)) = \min_{\mu} E^{[\tilde{Y}(0)]}\left[\sum_{k=0}^{\infty} \gamma^k R'(\tilde{Y}(k), \mu(\tilde{Y}(k)))\right],$$

where $R'(\tilde{Y}(k), \mu(\tilde{Y}(k)))$ is the stage cost in the MDP problem.

with quadratic cost approach. In those off-line approaches, the control policy is calculated off-line and can be used directly during the controller's operation. In this section, we will explore an online approach. In an online approach, the current state of a system is observed and the control action for that state is calculated on-the-fly. The most prominent advantage of the online approach is that we do not need to compute control policy for every states in state space in advance. So, it will scale well with large state space in our problem.

Limited lookahead

In the dynamic programming technique, we could obtain the optimal action a^* from Equation (5.5),

$$a^* = \text{argmin}_a E^{[s]} \left[R(s, a, s') + \gamma V^*(s') \right], \tag{5.5}$$

where we use s as a generalized state variable and a as a generalized control action. We also use s' as a generalized state variable that represents the next state after applying action a to current state s. From these generalized variables, our stage cost will be $R(s, a, s')$. We use these generalized state variable instead of the real state variable $\tilde{Y}(k)$ to draw a reader's attention to the algorithm and the explanation, instead of the variables themselves.[5]

In many cases, $V^*(s)$ is difficult to obtain. The limited lookahead is introduced to ease that difficulty. Instead of the optimal value function, $V^*(s)$, the limited lookahead

[5]Recall that the stage cost $R(s, a, s')$ in our MDP problem is

$$\begin{bmatrix} ||\vec{u}(k)||^2 \\ +D_1 \cdot \sum_{i=1}^{N} \left([X_i(k+1) - b_i(k+d_i+1)]^+ \right)^2 \\ +D_2 \cdot \sum_{i=1}^{N} f_{\text{TCP}} \left([b_i(k+d_i) - X_i(k)]^+, [b_i(k+d_i+1) - X_i(k+1)]^+ \right) \end{bmatrix}.$$

uses some approximation of true value function instead. In this migration problem, we use a value function from the linear system with quadratic cost approach, $V_{\mathrm{LQ}}(s)$, as an approximated value function. So, we get the limited lookahead policy, a_r, as shown in Equation (5.6). Given good approximation of value function, the limited lookahead can achieve a nearly optimal solution [2].

$$a_r = \mathrm{argmin}_a E^{[s]} \left[R(s, a, s') + \gamma V_{\mathrm{LQ}}(s') \right]. \tag{5.6}$$

From previous sections, the linear system with quadratic cost approach yields a closed-form solution for both the policy and value function. The closed-form solution makes $V_{\mathrm{LQ}}(s)$ calculation easy. Moreover, it does not require large memory to store pre-calculated values. Then, it makes on-the-fly calculations possible in a limited calculation time. So, the linear system with quadratic cost approach is an ideal choice for base policy, especially in our *online* version of the limited lookahead algorithm.[6] The detailed calculation of $V_{\mathrm{LQG}}(s)$ will be shown in Section 5.2.3.

Simulated Lookahead

Note that the online version of the limited lookahead reduces state explosion problem by considering only a *single* current state at a time. However, calculation of Equation (5.6) still suffers from state explosion. Let $P[s'|s, a]$ be the transition probability of the next state s', given the current state s and action a. To directly compute the expectation in Equation (5.6), we need $P[s'|s, a]$ for every s' in state space. Although we consider only a *single* current state s to reduce calculation complexity, we still need to consider *all* of the next state s' to obtain that expectation. As you could imagine, the calculation of the expectation will be tedious because our

[6] In the online version of the limited lookahead algorithm, we get a current state s and obtain the action a_r from Equation (5.6).

state space is large.

So we will use simulation to approximate that expectation. We will explore two simulation approaches. The first approach is Monte-Carlo Lookahead and the second approach is Iterative Lookahead.

5.2.1 Monte-Carlo Lookahead

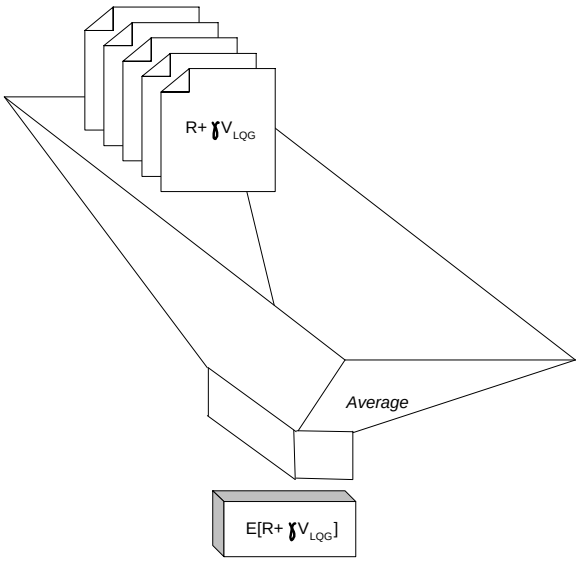
The Monte-Carlo simulation is a good alternative to get an approximate value to that expectation. Given current state s and action a, we can generate N_l samples of the next state, $\{s'_1, s'_2, \ldots, s'_{N_l}\}$. For each sample s'_i for $i = 1, \ldots, N_l$, we can calculate the associated value of $[R(s, a, s'_i) + \gamma \cdot V_{\mathrm{LQ}}(s'_i)]$. Then we can average them up as shown in Equation (5.7) to obtain the estimate of $E^{[s]}\left[R(s, a, s'_i) + \gamma \cdot V_{\mathrm{LQ}}(s'_i)\right]$. We can illustrate this description in Figure 5.1(a). Note that the Monte-Carlo simulation provides a good estimate if the number of samples N_l is large ($N_l \longrightarrow \infty$).

$$a_{\mathrm{MC}} = \mathrm{argmin}_a \frac{1}{N_l} \cdot \sum_{i=1}^{N_l} [R(s, a, s'_i) + \gamma \cdot V_{\mathrm{LQ}}(s'_i)] \tag{5.7}$$

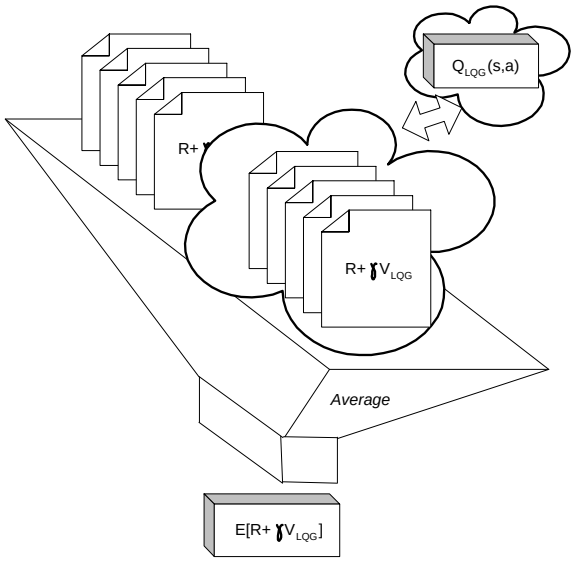
The application of the Monte-Carlo simulation to the limited lookahead has been proposed in a context of games as mentioned in [2]. This limited lookahead algorithm also uses the Monte-Carlo simulation. So, we called it the Monte-Carlo lookahead with its detail in Algorithm 3.

5.2.2 Iterative Lookahead

As mentioned before, the Monte-Carlo simulation provides a good estimate only if we have a large number of samples. In other words, we need to simulate the samples many times to obtain a nice controller. However, we cannot simulate that many samples in



(a) Using Monte-Carlo simulation



(b) Using Iterative simulation

Figure 5.1: The illustration of simulated lookahead algorithms

Monte(s)

$W(s,a) \leftarrow 0$, $a \in A(s)$

for $i \in 1, ..., N_l$

for $a \in A(s)$

- simulate s'_i from s and a

- $W(s,a) \leftarrow W(s,a) + [R(s,a,s'_i) + \gamma \cdot V_{\mathrm{LQ}}(s'_i)]$

$a_{\mathrm{MC}} \leftarrow \mathrm{argmin}_a \frac{W(s,a)}{N_l}$

Algorithm 3: Simulated lookahead algorithm using the Monte-Carlo simulation.

our fast timescale control. Time and processor capability limit a number of samples that we can generate. Hence, we might not get a good estimate of the expectation using the Monte-Carlo lookahead approach.

Fortunately, we can use a $Q_{\mathrm{LQ}}(s,a)$ (which is $E^{[s]}\left[R_{\mathrm{LQ}}(s,a,s') + \gamma V_{\mathrm{LQ}}(s')\right]$ from the linear system with quadratic cost approach[7]) to improve the accuracy of the estimate. Note that $Q_{\mathrm{LQ}}(s,a)$ is a weighted sum of $[R_{\mathrm{LQ}}(s,a,s'_i) + \gamma V_{\mathrm{LQ}}(s'_i)]$, i.e.,

$$\begin{aligned} Q_{\mathrm{LQ}}(s,a) &= E^{[s]}\left[R_{\mathrm{LQ}}(s,a,s') + \gamma V_{\mathrm{LQ}}(s')\right] \\ &= \lim_{N\to\infty} \tfrac{1}{N} \cdot \textstyle\sum_{i=1}^{N} \left[R_{\mathrm{LQ}}(s,a,s'_i) + \gamma \cdot V_{\mathrm{LQ}}(s'_i)\right]. \end{aligned}$$

Hence, having $Q_{\mathrm{LQ}}(s,a)$ is equivalent to having a large number of $[R_{\mathrm{LQ}}(s,a,s'_i) + \gamma V_{\mathrm{LQ}}(s'_i)]$. We can use these large number of available samples $[R_{\mathrm{LQ}}(s,a,s'_i) + \gamma V_{\mathrm{LQ}}(s'_i)]$ from $Q_{\mathrm{LQ}}(s,a)$ to approximate additional $[R(s,a,s'_i) + \gamma V_{\mathrm{LQ}}(s'_i)]$ samples that we want. This $[R_{\mathrm{LQ}}(s,a,s'_i) + \gamma V_{\mathrm{LQ}}(s'_i)]$ might

[7] $R_{\mathrm{LQ}}(s,a,s')$ is a stage cost when we formulate our migration problem in the linear system with quadratic cost context, i.e., $R_{\mathrm{LQ}}(X,U,X') = X^T Q X + U R U$. This is clearly the estimation of the real stage cost. Refer to Section 5.1 for further detail.

not be a good approximation but it is better than the case that we do not have any approximation at all and rely only on the small available samples.[8] Hence, we will use this $Q_{\mathrm{LQ}}(s,a)$ and the real simulation samples $[R(s,a,s'_i)+\gamma V_{\mathrm{LQ}}(s'_i)]$ to obtain better estimate of $E\left[R(s,a,s')+\gamma V_{\mathrm{LQ}}(s')\right]$. We can illustrate this idea in Figure 5.1(b). So, we could view the $Q_{\mathrm{LQ}}(s,a)$ usage as an artificial way to increase the number of samples.

From another point of view, we consider $Q_{\mathrm{LQ}}(s,a)$ as a good starting estimate so we do not need a large number of samples. Note that we will get a better estimate every update using additional samples. If we update using large enough samples, the estimate will eventually converge to $E^{[s]}\left[R(s,a,s')+\gamma V_{\mathrm{LQ}}(s')\right]$. However, we could update a fewer number of times if the initial value is good enough. This fact appears in many optimization techniques. For example;

- Newton's method: If the initial feasible point is near the optimal point, it takes only a few steps to reach the optimal point.
- Value iteration method in the dynamic programming technique: If the initial value function is good enough, it takes a few iterations to get an optimal solution.

The detailed calculation of $Q_{\mathrm{LQG}}(s,a)$ will be shown in Section 5.2.3.

Due to its nature of updating iteratively, we call this extension of the limited lookahead algorithm as Iterative Lookahead. The detail of the algorithm is shown in Algorithm 4.

[8] This case is the case using the Monte-Carlo lookahead algorithm.

Iterative(s)

$Q(s,a) \leftarrow Q_{\mathrm{LQ}}(s,a)$, $a \in A(s)$

for $i \in 1, ..., N_l$

for $a \in A(s)$

- simulate s'_i from s and a

- $Q(s,a) \leftarrow Q(s,a) + \alpha \cdot [R(s,a,s'_i) + \gamma \cdot V_{\mathrm{LQ}}(s'_i) - Q(s,a)]$

$a_{\mathrm{IT}} \leftarrow \mathrm{argmin}_a Q(s,a)$

Note that α is a step-size in updating $Q(\tilde{\mathbf{Y}}, u)$

Algorithm 4: Simulated lookahead algorithm using the iterative simulation.

5.2.3 Calculation of $V_{\mathrm{LQ}}(s)$ and $Q_{\mathrm{LQ}}(s,a)$

Recall that our LQ formulation minimizes the cost function below,

$$E_{W_k}\left[\sum_{k=0}^{\infty}\gamma^k(X'_kQX_k + \gamma U'_kRU_k)\right],$$

or

$$E_{W_k}\left[X'_0QX_0 + \gamma\sum_{k=0}^{\infty}\gamma^k(X'_{k+1}QX_{k+1} + U'_kRU_k)\right].$$

We could see that only the second term of the above equation follows the cost function in our migration problem.[9]

[9]Without TCP consideration, the cost function that we want to minimize in our migration problem is

$$E^{[\tilde{Y}(0)]}\left[\sum_{k=0}^{\infty}\gamma^k\left(||\vec{u}(k)||^2 + D_1\cdot\sum_{i=1}^{N}\left([X_i(k+1) - b_i(k+d_i+1)]^+\right)^2\right)\right],$$

in which the stage cost is the combination of current control cost $||\vec{u}(k)||^2$ and drops cost $\sum_{i=1}^{N}\left([X_i(k+1) - b_i(k+d_i+1)]^+\right)^2$, which depends on future state variables. This stage cost can be modeled by the second term of LQ cost function above.

From the above LQ cost function, we can write the associated value function $\hat{V}_{\mathrm{LQ}}(X_0)$ as

$$\begin{aligned}\hat{V}_{\mathrm{LQ}}(X_0) &= \min_{\mu} E_{W_k}\left[X_0' Q X_0 + \gamma \sum_{k=0}^{\infty} \gamma^k (X_{k+1}' Q X_{k+1} + \mu(X_k) R \mu(X_k))\right] \\ &= X_0' Q X_0 + \gamma \cdot \min_{\mu} E_{W_k}\left[\sum_{k=0}^{\infty} \gamma^k (X_{k+1}' Q X_{k+1} + \mu(X_k) R \mu(X_k))\right].\end{aligned}$$

We can take $X_0' Q X_0$ out of minimization terms because X_0 is given. As we discussed earlier, only the second term follows the migration problem. Hence, the desired value function $V_{\mathrm{LQ}}(X_0)$, which follows the migration problem, is

$$\begin{aligned}V_{\mathrm{LQ}}(X_0) &= \min_{\mu} E_{W_k}\left[\sum_{k=0}^{\infty} \gamma^k (X_{k+1}' Q X_{k+1} + \mu(X_k) R \mu(X_k))\right] \\ &= \tfrac{1}{\gamma}\hat{V}_{\mathrm{LQ}}(X_0) - \tfrac{1}{\gamma} X_0' Q X_0 \\ &= \tfrac{1}{\gamma} X_0' P X_0 + \tfrac{1}{1-\gamma} E[W_0' P W_0] - \tfrac{1}{\gamma} X_0' Q X_0 \\ &= \tfrac{1}{\gamma} X_0' (P - Q) X_0 + \tfrac{1}{1-\gamma} E[W_0' P W_0].\end{aligned}$$

As a result, we can analytically calculate $V_{\mathrm{LQ}}(X_k)$ from Equation (5.8).

$$V_{\mathrm{LQ}}(X_k) = \tfrac{1}{\gamma} X_k' (P - Q) X_k + \tfrac{1}{1-\gamma} E[W_k' P W_k] \tag{5.8}$$

Now, we get to the calculation of $Q_{\mathrm{LQ}}(X_0, U_0)$. Like the calculation of $V_{\mathrm{LQ}}(X_0)$, we will first get a relationship between $Q_{\mathrm{LQ}}(X_0, U_0)$ and $\hat{Q}_{\mathrm{LQ}}(X_0, U_0)$.

Similar to the discussion about $V_{\mathrm{LQ}}(X_0)$, we can write $Q_{\mathrm{LQ}}(X_0, U_0)$ associated with migration cost function as

$$\begin{aligned}Q_{\mathrm{LQ}}(X_0, U_0) &= U_0 R U_0 + E_{W_0}[X_1' Q X_1] \\ &\quad + \gamma \cdot \min_{\mu} E_{W_k}\left[\sum_{k=1}^{\infty} \gamma^k (X_{k+1}' Q X_{k+1} + \mu(X_k) R \mu(X_k))\right].\end{aligned}$$

Associated with the cost function of our LQ formulation, we can write $\hat{Q}_{\mathrm{LQ}}(X_0, U_0)$ as

$$\begin{aligned}\hat{Q}_{\mathrm{LQ}}(X_0, U_0) &= X_0'QX_0 + \gamma U_0 R U_0 \\ &\quad + \gamma \cdot \min_{\mu} E_{W_k}\left[X_1'QX_1 + \gamma \sum_{k=1}^{\infty} \gamma^k (X_{k+1}'QX_{k+1} + \mu(X_k)R\mu(X_k))\right] \\ &= X_0'QX_0 + U_0RU_0 + \gamma E_{W_0}[X_1'QX_1] \\ &\quad + \gamma \cdot \min_{\mu} E_{W_k}\left[\sum_{k=1}^{\infty} \gamma^k (X_{k+1}'QX_{k+1} + \mu(X_k)R\mu(X_k))\right].\end{aligned}$$

Hence, we get a relationship between $Q_{\mathrm{LQ}}(X_0, U_0)$ and $\hat{Q}_{\mathrm{LQ}}(X_0, U_0)$ as shown in Equation (5.9).

$$Q_{\mathrm{LQ}}(X_0, U_0) = \hat{Q}_{\mathrm{LQ}}(X_0, U_0) - X_0'QX_0 + (1-\gamma)E_{W_0}[X_1'QX_1] \tag{5.9}$$

From the intensive study on linear system with quadratic cost, we can get a closed-form solution of $\hat{Q}_{\mathrm{LQ}}(X_0, U_0)$ as follows: With the LQ formulation, we have

$$\begin{aligned}\hat{Q}_{\mathrm{LQ}}(X_0, U_0) &= E_{W_0}[X_0'QX_0 + \gamma U_0 R U_0 + \gamma \hat{V}_{\mathrm{LQ}}(AX_0 + BU_0 + W_0)] \\ &= X_0'QX_0 + \gamma U_0 R U_0 + \gamma \cdot E_{W_0}[\hat{V}_{\mathrm{LQ}}(AX_0 + BU_0 + W_0)].\end{aligned}$$

Davis and Vinter[8] give a nice closed-form of $E_{W_0}[\hat{V}_{\mathrm{LQ}}(AX_0 + BU_0 + W_0)]$ as

$$E_{W_0}[\hat{V}_{\mathrm{LQ}}(AX_0 + BU_0 + W_0)] = (AX_0 + BU_0)'P(AX_0 + BU_0) + \frac{1}{1-\gamma}E[W_0'PW_0].$$

Hence, we write $\hat{Q}_{\mathrm{LQ}}(X_0, U_0)$ analytically as shown in Equation (5.10).

$$\begin{aligned}\hat{Q}_{\mathrm{LQ}}(X_0, U_0) &= X_0'QX_0 + \gamma U_0 R U_0 \\ &\quad + \gamma(AX_0 + BU_0)'P(AX_0 + BU_0) + \tfrac{\gamma}{1-\gamma}E[W_0'PW_0]\end{aligned} \tag{5.10}$$

Using the relationship of $Q_{\mathrm{LQ}}(X_0, U_0)$ and $\hat{Q}_{\mathrm{LQ}}(X_0, U_0)$ in Equation (5.9) and the analytical solution of $\hat{Q}_{\mathrm{LQ}}(X_0, U_0)$ in Equation (5.10), we can get a desired

$Q_{\mathrm{LQ}}(X_0, U_0)$, which follows the migration problem. That is,

$$\begin{aligned}
Q_{\mathrm{LQ}}(X_0, U_0) &= \hat{Q}_{\mathrm{LQ}}(X_0, U_0) - X_0' Q X_0 + (1-\gamma) E_{W_0}[X_1' Q X_1] \\
&= (1-\gamma) E_{W_0}[X_1' Q X_1] + \gamma U_0 R U_0 \\
&\quad + \gamma (A X_0 + B U_0)' P (A X_0 + B U_0) + \tfrac{\gamma}{1-\gamma} E[W_0' P W_0] \\
&= (1-\gamma) E_{W_0}[(A X_0 + B U_0 + W_0)' Q (A X_0 + B U_0 + W_0)] \\
&\quad + \gamma U_0 R U_0 + \gamma (A X_0 + B U_0)' P (A X_0 + B U_0) + \tfrac{\gamma}{1-\gamma} E[W_0' P W_0] \\
&= \gamma U_0 R U_0 + E[W_0' ((1-\gamma) Q + \tfrac{\gamma}{1-\gamma} P) W_0] \\
&\quad + (A X_0 + B U_0)' ((1-\gamma) Q + \gamma P)(A X_0 + B U_0).
\end{aligned}$$

As a result, we can analytically calculate $Q_{\mathrm{LQ}}(X_k, U_k)$ from Equation (5.11).

$$\begin{aligned}
Q_{\mathrm{LQ}}(X_k, U_k) &= \gamma U_k R U_k + E[W_k' ((1-\gamma) Q + \tfrac{\gamma}{1-\gamma} P) W_k] \\
&\quad + (A X_k + B U_k)' ((1-\gamma) Q + \gamma P)(A X_k + B U_k)
\end{aligned} \tag{5.11}$$

5.3 Policy Evaluation and Comparison

So far, we have proposed a number of sub-optimal policies. In this section, we will evaluate and compare the performance of proposed policies. We will first evaluate them using TD(0), which is a learning method to estimate the value function of policies. Then we will use a packet-level simulation to strengthen the evaluation results.

We will compare five different migration controllers, which are NoMig, BAL, LQ, MONTE and IT controller. These controllers are the fast timescale controllers that acquire state information and make a migration decision every 50ms. Moreover, these five controllers implement five policies, in which three of them are the policies designed earlier in this chapter. The first two policies are extreme policies that we want to compare our three designed policies with. The two controllers that implement the two extreme policies are as follows: The NoMig controller implements one extreme

migration policy in such a way that we do not migrate any traffic at all. The BAL controller implements another extreme migration policy that always keeps the load in both LSPs balanced at all times. The BAL does not use the traffic models, instead it uses the *current* state information and performs load balancing so they are unable to take into account the delay in the network.

Unlike the first two controllers, the next three controllers implement our compromise policy that migrate flows intelligently. These policies have their frequency of migration between the least frequent policy, NoMig and the most frequent policy, BAL. The LQ controller implements LQ policy, the MONTE controller implements Monte-Carlo lookahead policy, and the IT controller implements Iterative lookahead policy. To get more detail on these policies, we will refer to Section 5.1 for LQ policy and Section 5.2 for Monte-Carlo lookahead policy and Iterative lookahead policy. We can summarize our controllers in Table 5.1

Note that all controllers are online controllers that build up on the Bernoulli Splitting (Bsplit) controller. As we described in Section 3.1, the BSplit controller implements an optimal slow timescale policy, which randomly dispatches new arriving calls among LSPs according to a split rule (probability distribution).[10] Given this long term optimality from the slow timescale BSplit controller, we hope to see the further improvement using the fast timescale migration.

5.3.1 Network Topology and Common Parameters

We evaluate each policy using our network topology as shown in Figure 5.2, which is the case of $N = 2$ in our general topology shown in Figure 4.1. In this network, we

[10]This BSplit controller is already included in our formulation in Section 4.2, which is modeled by the probability value $P^{(i)}$, for i^{th} LSP, $i = 1, 2, \ldots, N$.

Controller	Policy
NoMig	do not migrate
BAL	Always balance
LQ	LQ policy
MONTE	Monte-Carlo Lookahead Policy
IT	Iterative Lookahead Policy

Table 5.1: Controllers with their associated policies.

provision two LSPs for voice traffic. We choose $p_1 = 2$ and $p_2 = 1$, i.e., we setup the propagation delay from ingress LSR to the bottleneck link in the 1^{st} LSP and the 2^{nd} LSP to be 2 timeslots (100ms) and 1 timeslot (50ms), repectively. Moreover, we also choose $q_1 = q_2 = 0$, i.e., we assume that there is no information delay from bottleneck links to ingress LSR. The voice traffic arrival is governed by Poisson distribution with mean λ. The voice call duration is exponentially distributed with mean of $1/\mu$, which we fixed at 3 minutes. According to the optimal splitting rule, we assigned the probability value $\frac{1}{2}$ for both LSPs. With this voice traffic setup, we get an average voice traffic rate of $\frac{1}{2}\lambda * 180 * 0.064$ Mbps in each LSP, given a 64Kbps constant bit rate voice call, as discussed in Section 2.2. The bottleneck links in both LSPs have statistically identical video traffic. Both video traffic streams have the same statistical parameters $\alpha = 0.05$ and $\beta = 0.05$. All links including the bottleneck links have capacity of 90Mbps, which can fit (90Mbps/64Kbps)=1406 voice calls. Moreover, the TCP traffic also share the bottleneck link in both LSPs with video traffic and voice traffic.

Throughout this performance evaluation, we will fix all parameters in our cost function and TCP model as follows: $D_1 = 0.3$, $D_2 = 0.007$, discount factor (γ)$= 0.7$,

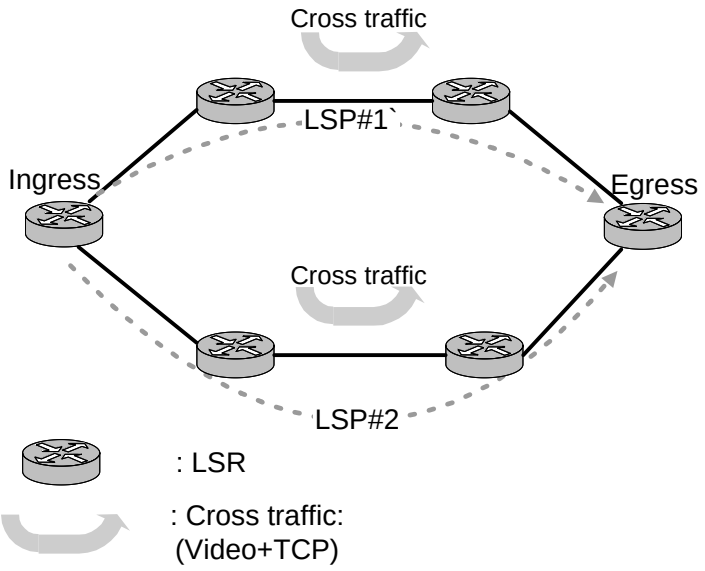


Figure 5.2: The network topology for the performance evaluation

$D_{\mathrm{tcp}} = 1$, and $\eta_{\mathrm{tcp}} = 100$.[11] Moreover, we will have a number of simulation samples $N_l = 10$ in both IT policy and MONTE policy.[12] We provide only a small number of samples in this evaluation ($N_l = 10$) to illustrate the limited time and processing power, which frequently occur in the fast timescale control.

5.3.2 Using TD(0)

TD(0) is a version of Temporal-Difference learning. This TD learning is a well-known prominent learning method for estimating value functions. It has great advantages in its deterministically fast convergence because it updates estimates based in part on other learned estimates (bootstrap). Moreover, it learns directly from raw experience without a model of the environment's dynamics, which in turn eases the estimation of the value function. The detail of TD learning can be found in many dynamic programming book

[11]The detail of the TCP model parameters is discussed in Section 2.3.1

[12]The detail of the N_l are discussed in Section 5.2

including [38]. Because of the above advantages, we will choose TD(0) to evaluate our designed sub-optimal policy.

We will start our TD(0) evaluation with a discussion about state aggregation. This state aggregation helps us cope with the large state space in our problem. Then, we will present an algorithm for our TD(0) with state aggregation. Finally, we will show the result of the comparison.

State Aggregation

As we mentioned earlier, we have the well-known curse dimensionality problem. With this problem, the number of states often grows exponentially with the number of state variables. With a large number of states, it is not only difficult to find an optimal control policy, but also difficult to evaluate a performance of control policy.

In this section, we will use state aggregation to combat with the explosion of state space. Generally, the state aggregation leads to a difficulty in the calculation of the transition probability, which is needed to calculate the value function $V^{\mu}(s)$ for a given control policy μ. This difficulty could be a problem in our performance evaluation, which compares the value function $V^{\mu}(s)$ of different policies μ.

However, TD learning can cope with this difficulty from the state aggregation. TD learning was introduced as an alternative method for cases where there is no explicit model and cost structure. It is also desirable in the cases where the simulated trace is easier to generate than the calculation of transition probability distribution. We could see that the latter cases represent the difficulty we got from the state aggregation. So the combination of the state aggregation and TD learning has the potential to yield a good approximation to the value function $V^{\mu}(s)$.

With the state aggregation, we will aggregate many states together and represent

them with a so-called superstate. So we could have a small number of superstates that can represent the whole range of our state space. Moreover each superstate $\hat{s}$ will have an associated value $\hat{V}(\hat{s})$. This $\hat{V}(\hat{s})$ gives an approximation of real value function $V^{\mu}(s)$ of the state s, for $\forall s \in \hat{s}$.

Now, we will determine how we should aggregate the states so that we can get a good approximation. Clearly, we should aggregate similar states together, which yield similar value function. Moreover, we should aggregate less aggressively if the state represents some value near link capacity. So we could get a better approximation of $V^{\mu}(s)$ near link capacity. The intuitive reason comes from the fact that migration should be done more carefully when the state is near link capacity.[13] Hence, a better approximation of $V^{\mu}(s)$ near link capacity is desirable in order to gain a better performance evaluation of different migration policies.

With the above state aggregation guideline, we will now select some proper state variables to aggregate. Then, we will show how we could use the guideline introduced previously to aggregate the chosen state variable. Recall that state space, $\mathbf{S}$, is composed of two components. The first component is the voice call traffic state, $X_i, i = 1, \ldots, N$. $[X_1, \ldots, X_N]$ takes value from $\mathbf{S_L}$, the state space of the voice component. The second component is the service rate information $b_i, i = 1, \ldots, N$. $[b_1, \ldots, b_N]$ takes value from $\mathbf{S_V}$, the state space of the service rate information component. So, our complete state space $\mathbf{S} = \mathbf{S_V} \times \mathbf{S_L}$. Throughout the rest of this state aggregation section, we will define the video traffic state, $v_i, i = 1, \ldots, N$. Where $v_i \stackrel{\triangle}{=} C_i - b_i, i = 1, \ldots, N$ and C_i is the capacity of the i^{th} LSP. In other words, we will

[13] We can compare this fact with a walking strategy near a cliff. We would rather walk with smaller steps than when we are further away from the cliff. We would execute more precision control where we have a higher chance of falling off the cliff.

use $[v_1, \ldots, v_N]$, which is directly the amount of video traffic, to represent the state variable $[b_1, \ldots, b_N]$.[14] These v_i variables will ease all explanations later in this state aggregation section.

Generally, $\mathbf{S_L}$ is large compared to $\mathbf{S_V}$. Note that each voice call consumes relatively small bandwidth. So a large number of voice calls can be filled in one LSP before reaching its capacity. Note that X_i represents the number of voice calls in i^{th} LSP. So $\mathbf{S_L}$, which is the state space of $[X_1, \ldots, X_N]$, is large. On the other hand, $\mathbf{S_V}$ is small. Even though $\mathbf{S_V}$ generally cannot be small, we will assume that we already applied a state reduction technique, which is available for video traffic, e.g. the technique in [6]. So we can further assume that $\mathbf{S_V}$ is small and do not need state aggregation.

Now, we know that we need to aggregate the state variables that are related to the voice component of our state space. If we write all state variables in one single vector, $[X_1, \ldots, X_N, v_1, \ldots, v_N]$, we could convert it into another vector, $[X_1 + v_1, \ldots, X_N + v_N, v_1, \ldots, v_N]$, without information loss. We define z_i as $X_i + v_i, i = 1, \ldots, N$, which simply is the utilization of high priority traffic in i^{th} LSP. Now, we can use our guideline to aggregate variable z_i to superstate variable $\hat{z}_i$. In other words, we will aggregate the state vector $[z_1, \ldots, z_N, v_1, \ldots, v_N]$ to the superstate vector $[\hat{z}_1, \ldots, \hat{z}_N, v_1, \ldots, v_N]$. The aggregation strategy to our new state variable z_i is shown in Figure 5.3.

We apply our state aggregation technique in a network with 90Mbps links. So the link capacity in a unit of voice calls is 90Mbps/64Kbps = 1406. (We can put at most 1406 voice calls in 90Mbps link.) We can divide the state variable z_i as shown in Table

[14] Recall from Chapter 4, we define the state variable b_i as a service rate information, which represent the amount of bandwidth left after the usage of video traffic.

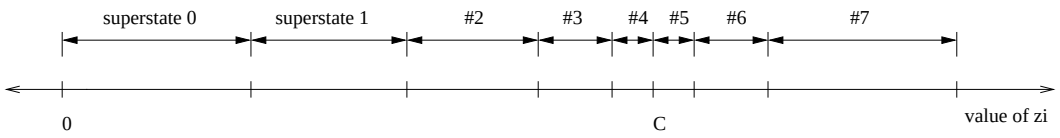


Figure 5.3: The division of state space with value of z_i

Minimum of z_i	Maximum of z_i	superstate
0	700	0
701	980	1
981	1204	2
1205	1297	3
1298	1368	4
1369	1406	5
1407	1450	6
1451	1550	7
1551	1700	8
1701	∞	9

Table 5.2: Example of state aggregation in our state space

5.2. As a result, this aggregation will be used in this TD(0) evaluation.

Algorithm

Algorithm 5 shows our off-line algorithm of our TD(0). For notation simplicity, we will use s instead of $\tilde{\mathbf{Y}}$ in this algorithm. The function $MigPolicy(\hat{s})$ represents the policies, in which we want to evaluate their performance including NoMig, BAL, LQ, Monte and IT policies.

TD(0)-mig[]

Initialize $V(.)$

Repeat [Outer Loop]

for every value of $\hat{s}$

$s \leftarrow$ uniform random on range of $\hat{s}$

Repeat [Inner Loop] {within episode}

$\vec{u} \leftarrow \mathrm{MigPolicy}(s)$

$s' \leftarrow$ simulate next state from s and $\vec{u}$

$\hat{s}' \leftarrow$ locate superstate from s'

$R_t \leftarrow$ get Reward from $s, \vec{u}, s'$

$V(\hat{s}) \leftarrow V(\hat{s}) + \alpha \left[R_t + \gamma^{k_s} V(\hat{s}') - V(\hat{s})\right]$ } Update $V(.)$

$\hat{s} \leftarrow \hat{s}'$

$s \leftarrow s'$ } replace next state with current state

*Note that $\hat{s}$ is current superstate s is current real state $\vec{u}$ is current action

$\hat{s}'$ is next superstate s' is next real state α is updating step-size

$\mathrm{MigPolicy}(s)$ is a sub-optimal policy that we want to evaluate

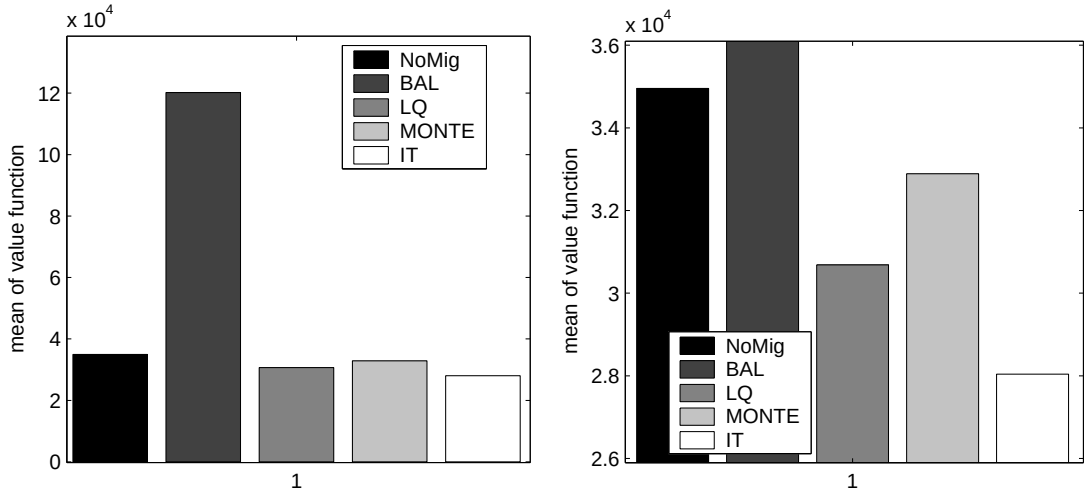
Algorithm 5: The TD(0) algorithm with the state aggregation for the migration problem.

Results and Discussion

To evaluate the performance of different controllers, we will compare the mean of the value function, which is generated from TD(0). Figure 5.4 shows an example result of this performance evaluation. Figure 5.4(b), which is a zoomed version of Figure 5.4(a), provides a better view to the gain in using our controllers. In this example, we fixed the average video traffic rate at 38.4Mbps and the average voice traffic rate at 42.5Mbps, which totally corresponds to 90% of the bottleneck-link bandwidth. We also fixed the number of steps (M) parameter in video traffic at 10, i.e., $M = 10$.

We can see that the BAL policy gives the worst performance, as shown in Figure 5.4(a). The reason comes from the fact that the BAL policy does not take into account the delay in the system and the fact that we chose $D_1 < 1$. Without the delay consideration, the received information might be out of date, which in turn reduces the performance of the BAL controller. Even though there is no clear restriction on the value of D_1, we chose $D_1 < 1$ in this evaluation. We believe that this choice will help emphasize the possibility that migration could interfere with a long-term optimality of the statistical splitting rule. As a result of $D_1 < 1$, we put more penalty toward our migration action. This in turn leads to the policies with small number of flow migration.

As shown in Figure 5.4(b), the LQ, MONTE and IT controllers provide a better performance over the NoMig controller. Moreover, this figure also illustrates an inaccuracy of the Monte-Carlo simulation technique under a small processing time. Recall that we have total number of simulation samples $N_l = 10$ for both MONTE and IT policies to illustrate the limited time and processing power, which frequently occur in the fast timescale control. With this small number of samples, the Monte-Carlo simulation technique could yield an inaccurate estimate of the expectation in Equation (5.6). Hence, MONTE controller, which depends on Monte-Carlo simulation, cannot



(a) Unzoomed version

(b) Zoomed version

Figure 5.4: Using TD(0) to compare the means of value function.

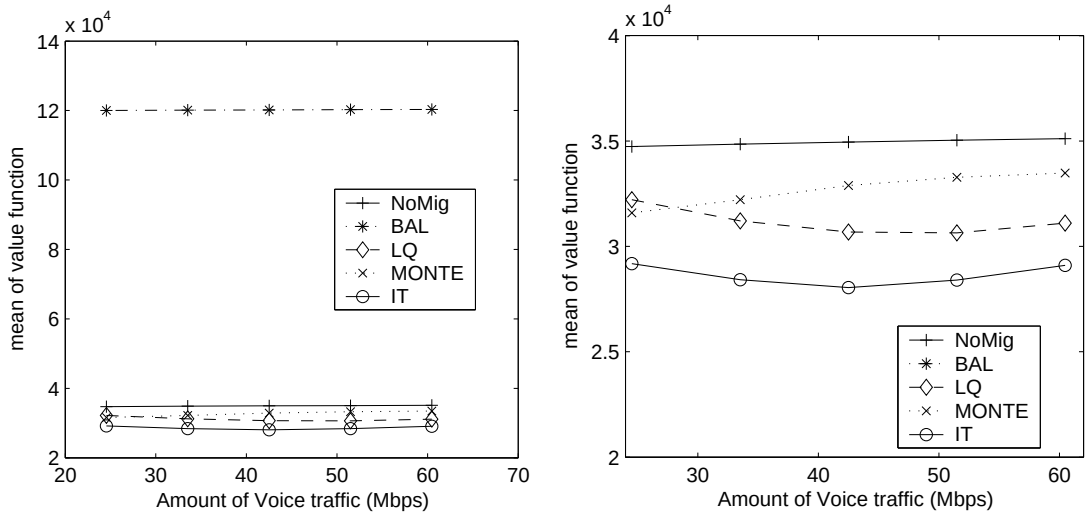
give a good performance.

We also see that the IT controller provides the best performance. This result can come from the fact that the IT policy is an improvement of the LQ policy, and it does not require a large number of simulation samples in its algorithm.

Now we can compare the performance of our controllers in various network environments. We use two experiments to do so.

The first experiment is an experiment designed to observe the performance of our controllers with different amounts of voice traffic. Hence, we will vary only an average voice traffic rate, while we keep others fixed. We can vary the voice rate by varying the arrival rate of the voice call, as discussed in Section 2.2.

Figure 5.5 shows the result of this experiment. In the experiment, we fixed the average video traffic rate at 38.4Mbps with number of steps ($M = 10$). We varied the



(a) Unzoomed version

(b) Zoomed version

Figure 5.5: Relationship of the mean of value function and the amount of voice traffic using TD(0).

average voice traffic rate from 24.5Mbps to 60Mbps, which corresponds to the varying of total traffic in the bottleneck link from 70 to 110 percent of the link capacity.

The result shows that all intelligent migration policies (the LQ, MONTE and IT policies) always give a good improvement over the NoMig policy. As the voice traffic increases, the improvement of the LQ policy over the NoMig policy increases at the beginning of the graph (at 24.5Mbps), but it decreases at the end (at 60Mbps). we can notice that 60Mbps correspond to 110% load, which is overloaded and the providers normally do not operate their network under this load condition. Hence, we could also say that the improvement of the LQ policy over the NoMig controller decreases when the traffic is overloaded.

There are two facts that govern this improvement pattern. The first fact governs the increasing improvement at the beginning (at 24.5Mbps), while the second fact governs

the decreasing improvement at the end (at 60Mbps).

As the first fact, the control action becomes more valuable when the load increases. Recall that the LQ controller keeps adjusting the utilization to its mean value (using migration). This adjustment will not be very helpful at the relative light load because this light load yields only small drops. Those small drops need only a small amount of migration, as opposed to the adjustment of utilization toward its mean value, which needs a large amount of migration. When the load increases, the same adjustment of the LQ controller can alleviate more drops with the same control cost. As a result, the LQ controller yields a greater improvement when the load increases.

As the second fact, the penalty of a wrong prediction increases when the load increases. In our stochastic environment, there is a possibility that our controller could give an incorrect prediction. The migration decision using this incorrect prediction could lead to the packet drops, especially in the receiving LSPs (LSPs that receive the migrated flows). With a relatively light load, the LSP will have more spare space for incoming (migrated) flows, including the mistaken ones. When the load increase, the spare space decreases, which leads larger drops (penalty). Although the controller still yields a good performance improvement using migration, the more penalties from incorrect prediction can continuously reduce that performance improvement when the load increases.

From the description of both facts above, we can summarize the information for better understanding as follows: The LQ controller's migration strategy is "to keep adjusting the utilization to the mean value". According to the first fact, this strategy will not effective at light to medium load because it is unnecessary to keep utilization near the mean. A small amount of migration will be enough. On the other hand, the second fact shows that the LQ's strategy will not be effective at a overloaded situation

from the risk of wrong prediction.

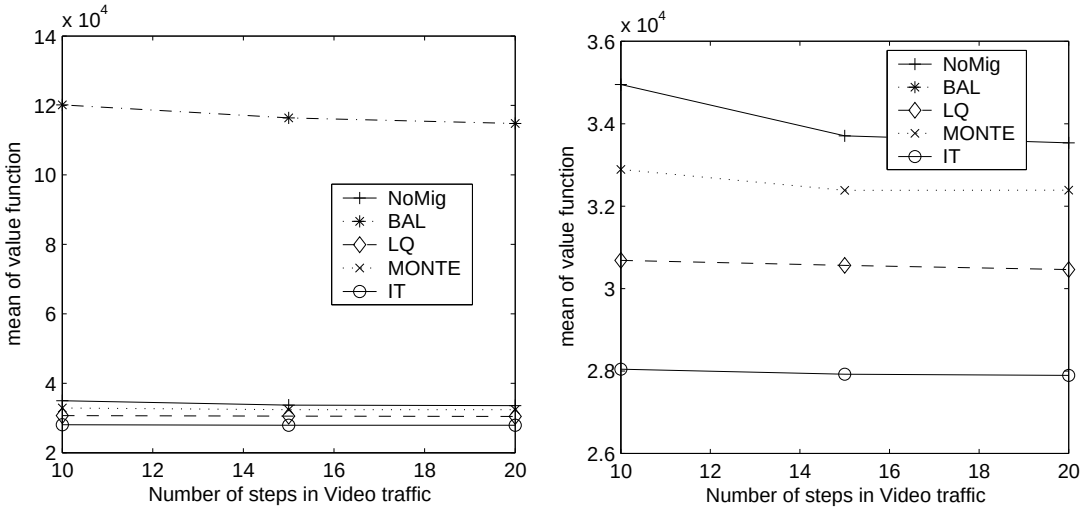
Hence, we can conclude that the LQ controller gives the greatest performance improvement over NoMig when the load are high but not overloaded.

In addition, we can also see that that the IT controller yields a similar performance pattern compared to the LQ controller. This result follows the fact that the IT policy is an improvement of the LQ policy. Hence, it should react to the voice load in the same way that the LQ policy does. However, the MONTE controller does not give any improvement over the LQ policy. As we discussed earlier, the Monte-Carlo simulation technique, which is the main ingredient of the MONTE controller, requires a large number of simulation samples. We also recall that we have a small number of samples in this evaluation ($N_l = 10$) to illustrate the limited time and processing power, which frequently occur in the fast timescale control. Hence, the MONTE controller will have a difficulty giving a good performance in this fast timescale environment.

Moreover, the figure also shows the superiority of the IT controller over the comparing controllers. This result strengthens our belief that our proposed Iterative lookahead policy is a good solution for this fast timescale problem.

The second experiment is an experiment designed to study the performance of our controllers under a various bursty level of video traffic. We can vary the burstiness of video traffic by changing the number of steps (M) defined in Section 2.1. We consider the video traffic to be bursty if the number of steps (M) is small. As we already described in Section 2.1, a small number of steps makes the stepsize (A) large, given a fixed average video traffic rate. The large stepsize (A) indicates a big jump between the traffic rate of the video traffic, which makes the video traffic more bursty.

Figure 5.6 shows the result of this experiment. In the experiment, we varied the



(a) Unzoomed version

(b) Zoomed version

Figure 5.6: Relationship of the mean of value function and the burstiness of video traffic using TD(0).

number of steps (M) parameter in video traffic from 10 to 20. We fixed the average video traffic rate at 38.4Mbps and the average voice traffic rate at 42.5Mbps, which totally corresponds to 90% of bottleneck link's capacity.

According to the result of this experiment, all intelligent migration policies (the LQ, MONTE, and IT polices) outperform the NoMig policy, which represents a case without any migration mechanism. Moreover, they give a greater improvement over the NoMig policy under bursty condition. Note that the migration mechanism will be most helpful in the situation when some LSPs are under-utilized while other LSPs are over-utilized. The bursty condition promotes a chance of getting that situation. Hence, the migration mechanism will have a higher chance to alleviate high priority drops. As a result, our migration mechanism provides a greater improvement under bursty condition. This result confirms that the migration mechanism is helpful for the QoS

improvement in the network, especially under bursty condition.

Among these three intelligent migration policies, we can see that the IT policy always performs best. As we discussed earlier, this result could come from the fact that the IT policy is an improvement of the LQ policy, and it does not require a large number of simulation samples in its algorithm (which the MONTE policy does). Along with the conclusion from the first experiment, this result further strengthens our belief that our proposed Iterative lookahead policy is a good solution for this fast timescale problem.

5.3.3 Using Packet-Level Simulation

The packet-level simulation can help us understand the behavior of our controller in the real network. Note that the packet-level simulation allows us to simulate the interaction of packets and routers, which are implemented our controller. Hence, it could give an evaluation that is close to the evaluation in an actual network.

As discussed earlier, we will compare the performance of our intelligent migration policies (the LQ, MONTE, and IT controllers) with two extreme policies (the NoMig and BAL controllers). These policies are computed online during our simulation. Moreover, this computation is done in a flow-level (we designed our policy based on the arrival and departure of flows),as opposed to a packet-level of our simulation itself. We will monitor the amount of packets and convert the number into a units of flows. The controller will use that number of flows to obtain the amount of migration based on their underline policy. The controller will calculate the amount of migration online upon the received information. For example, we monitored the link and found out that a total 32Mbps of voice packets were sent through the link. Because we send the voice flows at a constant bit rate at 64Kbps, we can interpret that the link has 500 voice calls.

Then the controller can use this information about number of flow and make a online decision on the migration amount, say 200 voice calls have to move from 2^{nd} LSP to 4^{th} LSP. From this example we can see that the calculation of the policy is done in the flow-level while the simulation proceed in a packet-level.

Simulator implementation

We use ns-2 simulator[39] for our simulations. As already discussed in Section 3.2, we add ability to migrate flows. Moreover, we add the receiver to keep track out-of-order packets, which is one of the consequences of the migration mechanism.

Simulation Setup and Parameters

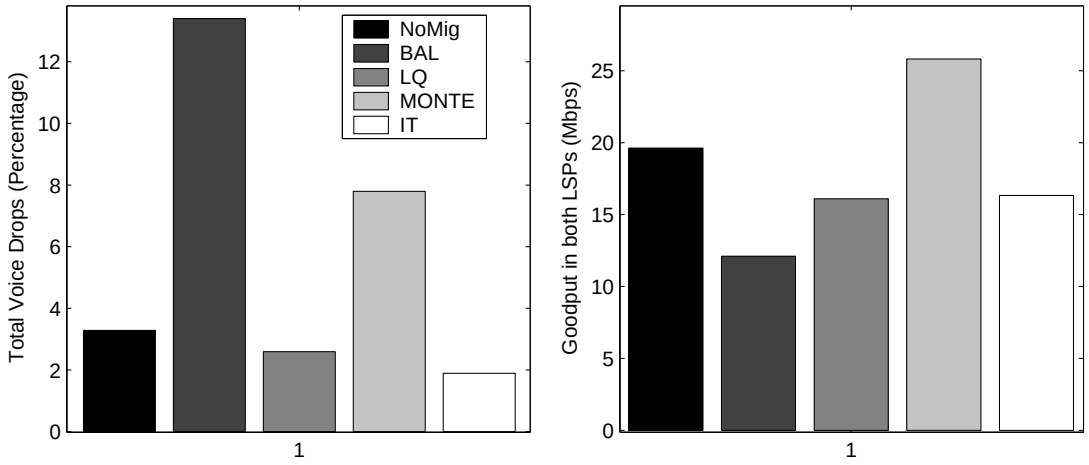
We use a network topology and some common parameters as we stated in Section 5.3.1. With ns-2, we use a Reno version of TCP, which is already in the ns-2 package. In our simulation, we setup short-lived TCP traffic flows in backup LSP. Each TCP flow sends small files with a uniformly distributed size of 20-40 Kbytes. Upon the end of each transmission, TCP sources will setup a new connection and begin to transmit a new file. Recall that the new connection setup will use a initial value for ITO. The small file transmission and re-connection of flows capture a characteristic of short-lived TCP flows. These flows have three distinct RTT value, 10ms, 30ms and 50ms. There are 15 TCP flows with each RTT value. Therefore, there are a total of 45 TCP flows. As discussed in Section 2.3, the different RTT values represent different locations of end-users, while a number of flows with the same RTT value indicates the fact that each location can have multiple users.

Result and Discussion

We can get the performance measurements from monitoring the total voice drops (include drops from out-of-order packets), the out-of-order drops and the TCP goodput. Beside above measurements, we will also collect a total number of voice flows that we decide to migrate, in which we will refer as "NumMig". This NumMig enables us to gain an access to the operation of our controllers.

Figure 5.7 shows the example of how our control affects each performance measurement. In this experiment, we fixed the average video traffic rate at 38.4Mbps and the average voice traffic rate at 42.6Mbps, which correspond to 90% of the bottleneck link's capacity. We also fixed the number of steps (M) parameter in video traffic at 10, i.e., $M = 10$. Note that the study in [17] points out that the losses greater than 2% affect end-to-end quality of VoIP calls. Hence, we assume that the quality of voice and video traffic is not good if their drops are greater than 2%. In the figure, voice traffic has drops around 3.3% and TCP goodput around 19.5Mbps without the migration mechanism (or the NoMig policy). If we naively migrate voice flows to keep the load balanced all the time, we would get unbearable 13.4% voice drops and a small 12.1Mbps goodput, as we can see in the figure with the BAL policy. We can see that the BAL policy gives the incredible large voice drop. As we discussed earlier, the reason could come from the fact that the BAL policy does not take into account the delay in the system. Without the delay consideration, the received information might be out of date, which in turn reduces the performance of the BAL controller.

We could carefully migrate flows and get smaller voice drops at 2.8% and 1.9% with the LQ controller and the IT controller, respectively. You can see that the network with the IT controller has acceptable voice drops under 2%. Moreover, the LQ controller and the IT controller yield around 16Mbps TCP goodput. This TCP goodput



(a) Total voice drops (include out-of-order) (b) Goodput

Figure 5.7: Using NS to compare the performance measurements.

is still higher (better) than that of the BAL controller even though it is smaller (worse) than that of the NoMig controller.

This example shows that we could use a migration mechanism to improve quality of high priority traffic up to an acceptable quality level. Moreover, it lets us see the benefit of applying the Iterative lookahead algorithm. For instance (as we can see in the figure), the voice drops when applying the IT policy is better than applying LQ policy while both policies produce the same TCP goodput.

Even though we see the high TCP goodput with the MONTE controller, it does not mean the MONTE controller try to maximize the TCP goodput. It might be because the MONT controller make a lot of high priority drops so that there is more rooms for TCP traffic. This might also be an unpredictable behavior of the MONT controller because of the inaccuracy in the Monte-Carlo simulation under small number of samples.

Like the setup of the TD(0) experiment in Section 5.3.2, we will compare the

performance of our controllers in various network environments. The first experiment is an experiment designed to see the performance of our controller under different amounts of voice traffic. The second experiment is an experiment designed to see the performance of our controller over changes in the burstiness level of video traffic.

Figure 5.8 shows a result of the first experiment. We varied the average voice traffic rate from 25Mbps to 52.3Mbps, which correspond to the varying total traffic in the bottleneck link from 70 to 100 percent of the link capacity. We also kept video traffic fixed at 38.4Mbps with the number of steps ($M = 10$).

Since this experiment is a performance comparison of our controller with different amounts of voice traffic, the experiment corresponds to the first experiment with TD(0) in Section 5.3.2. Like the experiment with TD(0), we also get the same conclusion that both the LQ policy and the IT policy always perform better than the NoMig policy. However, we have one exception at the voice load of 52.3Mbps, which the LQ policy does not outperform the NoMig. However, this average voice load corresponds to 100 percent of the link capacity, a condition in which networks typically do not operate under. Hence, we can refer to this condition as a overloaded condition.

Additionally, we also get the same performance improvement pattern, i.e., as the voice traffic increases, the improvement of the LQ policy over the NoMig increases at the beginning of the graph (at 25Mbps), but it decreases at the end (at 52.3Mbps). We can see this improvement pattern from Figure 5.8. As shown in Figure 5.8(a), the LQ policy gives only 0.49% performance improvement over the NoMig policy at a 25Mbps average voice traffic. At a 42.5Mbps average voice traffic, the LQ policy gives 0.84% improvement. In the same figure, the LQ policy gives -0.22% improvement (0.22% performance degradation) at a 52.3Mbps average voice traffic. On the other hand, the LQ policy gives approximately the same goodput degradation over the NoMig policy

with a different voice load as shown in Figure 5.8(c).

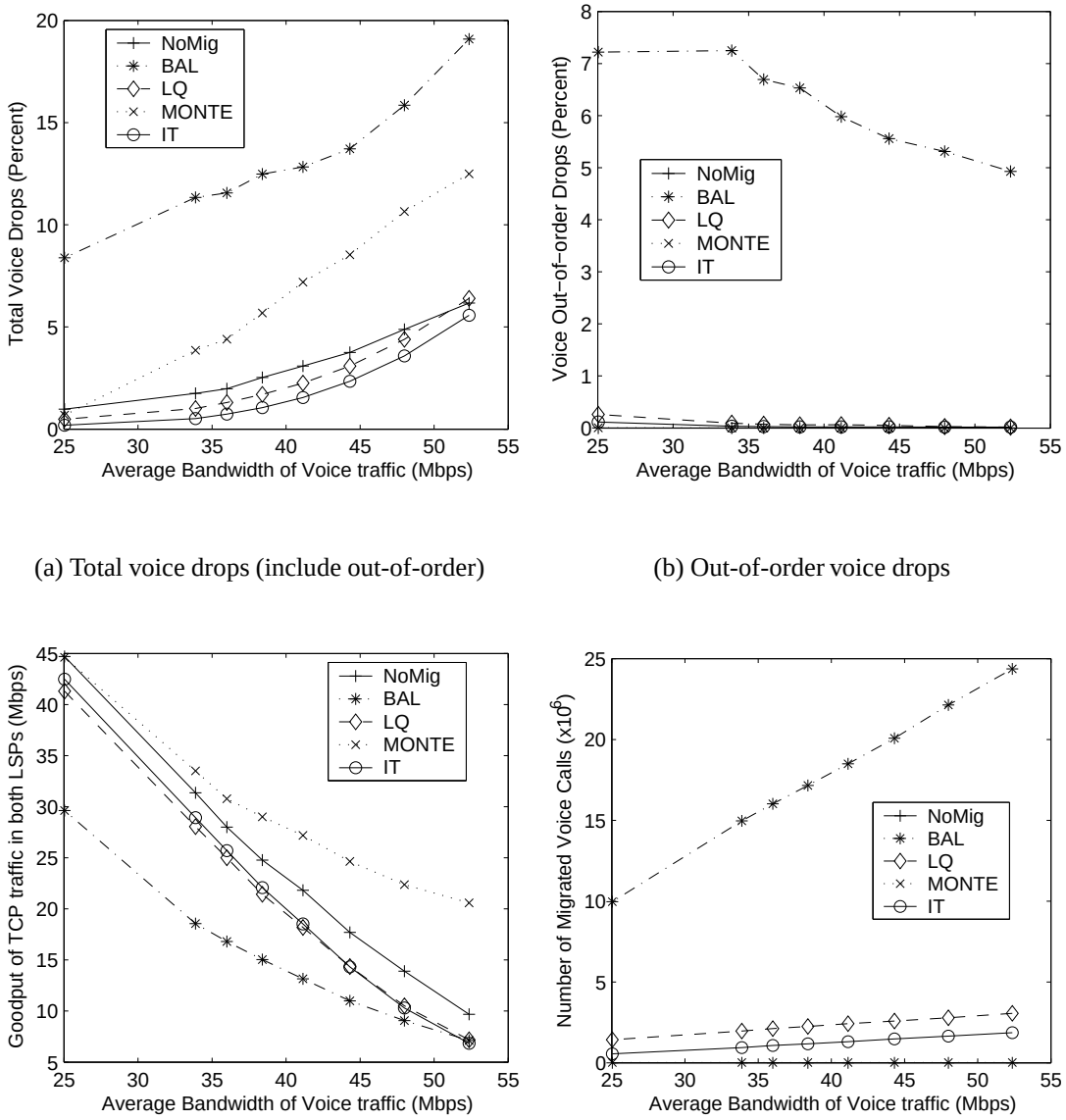
From the discussion in the last paragraph and the fact that we consider voice drops more important than the TCP performance, we can conclude that the LQ controller yields the greatest improvement over the NoMig policy when the load are high, but not overloaded.

Moreover, we can see that the IT controller, which is a policy that gives an improvement over the LQ policy, follows the same improvement pattern as the LQ controller over the NoMig controller. As a result, we could also conclude that IT controller yields the greatest improvement over NoMig policy when the load is high, but not overloaded. On the other hand, the MONTE controller gives an undesirable result coming from the small processing time as we discussed earlier.

To sum it all up, this result further strengthens our belief that the well-designed migration mechanism is helpful in improving the QoS, especially when the traffic load is high (but not overloaded).

Again, we assume that the quality of voice and video traffic is not good if their drops are greater than 2%. Using Figure 5.9, which is a zoomed version of Figure 5.8(a), we can quantify another advantage of the migration mechanism as follows: Without the migration mechanism (the NoMig policy), we can accept a voice traffic up to 36Mbps before its quality is unacceptable. On the other hand, we can accept up to 43Mbps with the migration mechanism using the IT policy. Hence, the existing network can accept 20% more voice traffic. In other words, the network can accept 20% more high priority traffic. With this capability to increase high priority traffic with an acceptable QoS, network providers gain a profit increase.

Figure 5.10 shows a result of the second experiment. As we mentioned before, this experiment aims to compare the performance of our controllers over changes in the



(a) Total voice drops (include out-of-order)

(b) Out-of-order voice drops

(c) Goodput

(d) Number of migrated flows

Figure 5.8: Relationship of the performance measurements and the amount of voice traffic using NS-2.

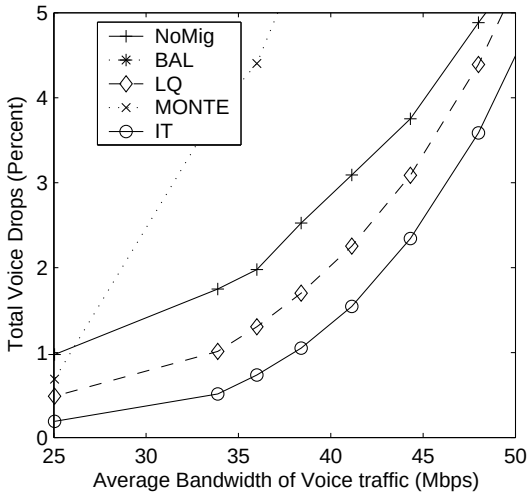


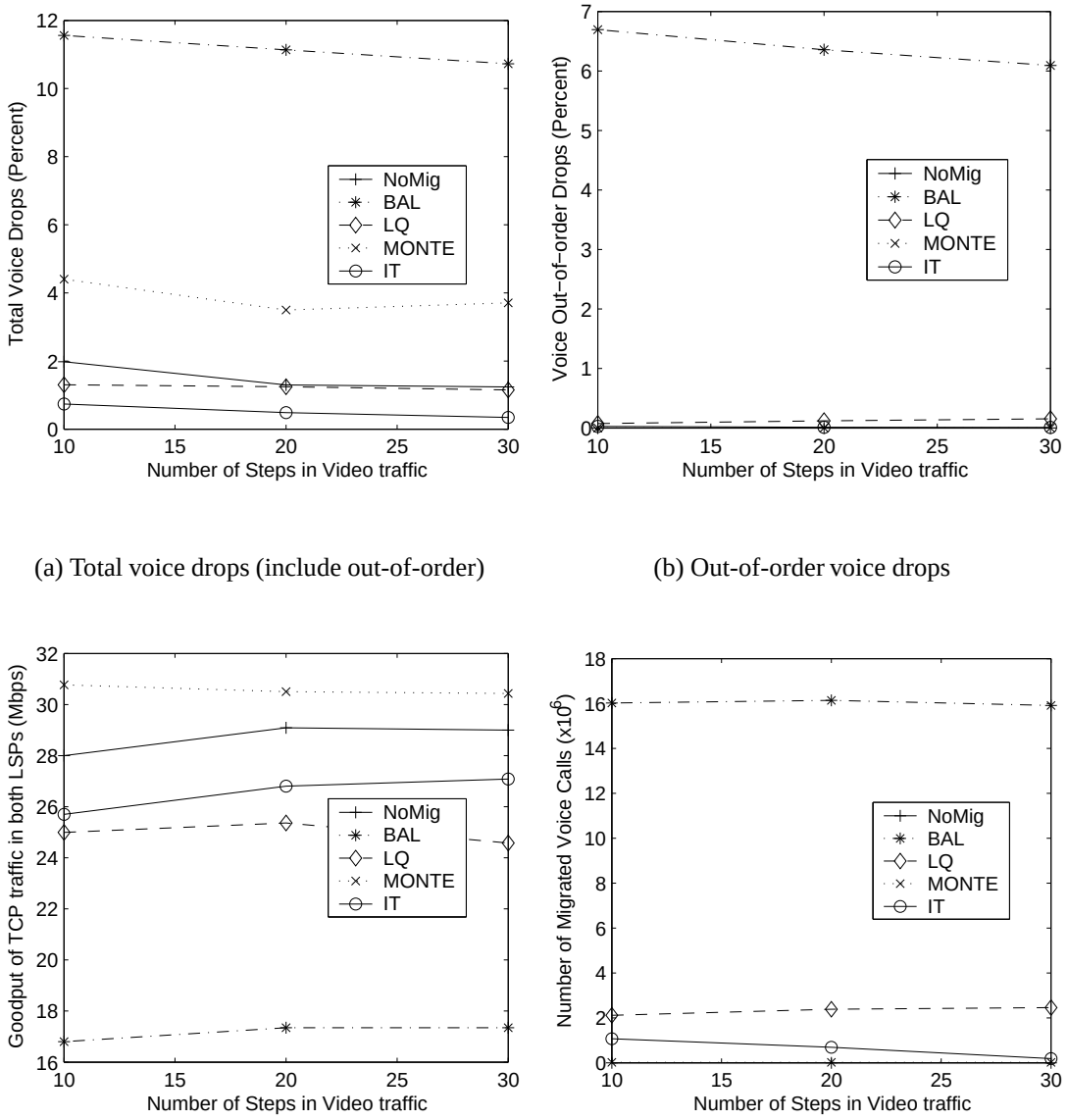
Figure 5.9: Relationship of the total voice drops and the amount of voice traffic(zoomed) using NS-2.

burstiness level of video traffic. As we discussed eariler, we consider the video traffic bursty if the number of steps (M) is small. We varied the number of steps (M) from 10 to 30. In addition, we fixed the average video traffic rate at 38.4Mbps and average voice traffic rate at 36Mbps, which correspond to a 83%load.[15]

We could see that there is an increasing gain of our IT controller over the NoMig controller (the case without any migration) when the network is more bursty. As shown in Figure 5.10(a), both the LQ controller and the IT controller gives an increasing gain of the total voice drops. Also, the IT controller also gives a slightly increasing gain of the goodput when the network is more bursty as shown in Figure 5.10(c). Hence, we could conclude that our IT controller performs better in a bursty network.[16] This conclusion on the IT controller complies with the conclusion of the second experiment

[15]In other words, the average traffic in the bottleneck link is 83% of the link capacity.

[16]However, the LQ controller gives an decreasing gain of goodput over the NoMig controller. As a result, we do not have a good conclusion for the LQ controller



(a) Total voice drops (include out-of-order)

(b) Out-of-order voice drops

(c) Goodput

(d) Number of migrated flows

Figure 5.10: Relationship of the performance measurements and the video traffic's burstiness using NS-2.

with TD(0) in Section 5.3.2.

The out-of-order voice drops are shown in Figure 5.8(b) and Figure 5.10(b). These graphs compared to the total voice drops graphs (Figure 5.8(a) and Figure 5.10(a)) show that the majority of drops are not out-of-order drops, especially at high load. This result let us to believe that the majority of drops, especially at high load, comes from the migration decision using the incorrect prediction. Moreover, The out-of-order drops with the BAL controller decrease when the voice load increases as shown in 5.8(b). This could come from the fact that the drops, which could have been out-of-order, could be dropped earlier in the network from the increasing congestion potential, resulting from increasing load.

In addition, These graphs show that the BAL controller gives a significant out-of-order drops compared to the other controllers. This could come from the fact that the BAL controller migrate more frequent than the other controllers as we can see in Figure 5.8(d) and Figure 5.10(d). These two figures (Figure 5.8(d) and Figure 5.10(d)) present the total number of migrated flows resulting from our migration policies.

5.4 Conclusion

We have introduced a fast timescale control traffic engineering, which is based on flow migration. This migration scheme exploits the same fact used in statistical multiplexing. With this fact, migration will migrate flows from over-utilized LSPs to under-utilized LSPs with an associated migration cost. Hence, we have a design goal using flow migration to improve the QoS of high priority traffic despite of the cost of the migration. Additionally, we also consider the interaction of migration and TCP

congestion control.

A controller, which resides in ingress LSR, will have to intelligently make a migration decision for every fixed time slot. We have designed three intelligent controllers named the LQ controller, the MONTE controller and the IT controller.

The LQ controller implements the LQ policy, which is a reformulation of our problem in a context of linear system with quadratic cost (LQ). This reformulation drives our LQ policy away from the optimal policy. However, it provides a closed-form solution that simplifies online calculation and in turn scales well with large state space in our migration problem.

The MONTE controller implements the Monte-Carlo lookahead policy, which is based on the limited lookahead algorithm. We use a value function of the LQ policy as an approximated value function for the limited lookahead algorithm. To be able to implement online, we use the Monte-Carlo simulation to get the expectation in the limited lookahead algorithm.

The IT controller implements the Iterative lookahead policy, which is again based on the limited lookahead algorithm. Like the Monte-Carlo lookahead policy, we use a value function of the LQ policy as an approximated value function and use simulation to get the expectation in the limited lookahead algorithm. However, this policy is designed for a fast timescale control, which has a small processing power and time to get simulation samples. Hence, this IT controller is an improvement of the MONTE controller in a fast timescale implementation.

With TD(0) and the packet-level simulation, we compared the above three controllers with two other extreme controllers, which are the NoMig controller and the BAL controller.

The NoMig controller stands for *No migration*. This controller will not migrate any flows. In other words, this controller presents the case of having no migration mechanism at all.

The BAL controller will migrate flows to make the load balanced. This controller presents the case that we naively migrate flows to load balance traffic without taking the delay into account.

All controllers are online controllers that build up on the Bernoulli Splitting (Bsplit) controller. This BSplit controller implements an optimal slow timescale policy, which randomly dispatchs new arriving calls among LSPs according to a split rule (probability distribution). Moreover, the BSplit controller is already included in our formulation. Hence, all of our migration policies (the LQ, MONTE and IT policies) already take this long term optimality into account.

Our empirical study demonstrates the superior of our intelligent migration controllers (the LQ, MONTE, IT controllers) over the case without migration control, especially in the bursty network environment. Moreover, they yielded a greater improvement over the case without migration when the traffic amount is high but not overloaded.

This empirical study also shows a clear advantage of the migration scheme, especially when we migrate intelligently. In addition, it points out the advantage of having a traffic prediction by reacting accordance to the prediction. However, this advantage greatly depends on the accuracy of the traffic model in both the voice and video traffic.

Chapter 6

Congestion Control with QoS guaranteed Backup Path (Duplication Problem)

In this work, we exploit an unused backup LSP to alleviate congestion that might occur in main LSPs. Figure 6.1 serves as an example of our idea. We provision one main LSP and its backup LSP for real-time high priority traffic from ingress LSR to egress LSR. There are cross traffics with the same priority class in bottleneck links in both the main LSP and backup LSP. We will call the cross traffic in the main LSP as the 1^{st} cross traffic, and the cross traffic in the backup LSP as the 2^{nd} cross traffic. Under normal operation (without congestion), the high priority traffic uses only the main LSP which shares the bottleneck link with the 1^{st} cross traffic. The backup LSP will be filled with lower priority traffic, which is mostly composed of TCP flows. It shares the bottleneck link with the 2^{nd} cross traffic. We assume that the high priority traffic uses UDP. Moreover it is served before the low priority traffic in every link following the link schedule in Section 1.2.

We introduce a duplication mechanism, which helps the QoS of high priority traffic as follows: *Upon the predicted congestion in the main LSP, the ingress router duplicates the high priority traffic and sends it in both main LSP and backup LSP*. Note

that, without the duplication mechanism above, this backup LSP will be used (for high priority traffic) only if a failure occurs in the main active LSP. That is, the backup path remains unused even thought there is congestion. With the duplication mechanism, the real-time traffic can choose protocol, like RTP, to take care the duplicated information at receiver end.[1] As a result, the QoS of real-time traffic is unlikely degrading even in the situation of congestion.[2]

However, the duplication mechanism could yield some drawbacks if we do not carefully design. As a first drawback, the high priority traffic in the backup LSP could experience a frequent congestion. The duplicated packets will increase an amount of high priority packets in the backup LSP. Hence, there is a higher chance of congestion. The second drawback is the fact that the duplication mechanism could work adversely with the congestion control of the TCP traffic. As a result of the decision to duplicate, the (lower priority) TCP traffic backs off and takes time to return back to original throughput. The TCP traffic might have no chance to transmit if we decide to duplicate packets again before the throughput of TCP sufficiently recovers (from their back-off mechanism in TCP).

We will design controllers to decide when we should duplicate the traffic. Our main goal is to improve the QoS of the real-time while we let TCP traffic to get a reasonable throughput. Definitely, we will consider the high priority traffic to be more important than the TCP traffic in our design. Moreover, we name this problem as "*duplication problem*", which is a result of its control nature.

[1]The Real-time Transport Protocol (RTP) simply discards the duplicated packets.[32]

[2]The cross traffic is also a high priority traffic. Hence, the providers might set up the backup path for it already. As a result, we could use the duplication control to improve the QoS of the cross-traffic as well (using its own backup path).

Beside the main design goal, we will also take into account the delay in the network. We consider two types of delay. The first type is the propagation delay. This delay could make our control decision, which is made at ingress node, obsolete when it reaches the bottleneck links. The second type is the information delay. This delay is the delay of information from the bottleneck link to the ingress node. This delay can make the information about the bottleneck link obsolete when it reaches the controller that is located at the ingress node. We will use traffic models as a tool to predict traffic characteristic. This prediction aims to compensate the delay that we mentioned above.

6.1 Related Work

Because the real-time traffic is sensitive to delay and we must provision a network to satisfy the Service Level Agreement (SLA), the path protection, which uses backup path in MPLS network, has been considered enormously in the research community. In MPLS network, providing a backup LSP can increase the reliability of Label Switched Paths (LSPs). All traffic can be switched to the backup LSP upon a failure in the active LSP. There are attempts to find routing protocols that find both active LSP and backup LSP at the same time according to a QoS-requirement[18, 19]. Such works are still open, and they represent the need of the backup path in the real network. In the MPLS recovery framework [34], there are two types of protection switching:

- In 1+1 (“one plus one”) protection, the resources (bandwidth, buffers, processing capacity) on the recovery path are fully reserved, and carry the same traffic as the working path. Selection between the traffic on the working and recovery paths is made at the path merge LSR (PML)

- In 1:1 (“one for one”) protection, the resources (if any) allocated on the recov-

ery path are fully available to preemptible low priority traffic except when the recovery path is in use due to a fault on the working path.

The 1+1 protection consumes twice the bandwidth of the network without any backups. So providers seem to be interested in the 1:1 protection, which uses the same bandwidth, given that they put low priority traffic in the backup LSP. Note that the backup LSP will not be used by the high priority traffic even though there is congestion in the active LSP. It will be used when failures occur in active LSP.

Clearly, our duplication scheme can be viewed as a hybrid solution between 1:1 and 1+1 protection. However, our goal is to improve both high throughput and Quality of Service, instead of improving the reliability in the above protection works.

To the best of our knowledge, we are the first one to take the advantage of the existing backup path for the congestion control management in this type of network.

In some cases, the backup path can be shared for different active paths, especially the active paths with different ingress nodes.[18, 19] Given a single point of failure, these shared backup paths are acceptable. We do not consider the above shared backup paths case. Instead, we consider only a backup path that associates with one and only one active path. However, our controller is implemented in ingress router. Hence, we do not eliminate the chance of using distributed algorithm to deal with the shared backup path case in the future.

6.2 System model

From Figure 6.1, the voice traffic is routed to main LSP from ingress LSR to egress LSR. The video cross traffic shares the same bottleneck link in both main LSP and backup LSP while the best-effort shares only the bottleneck link in the backup LSP.

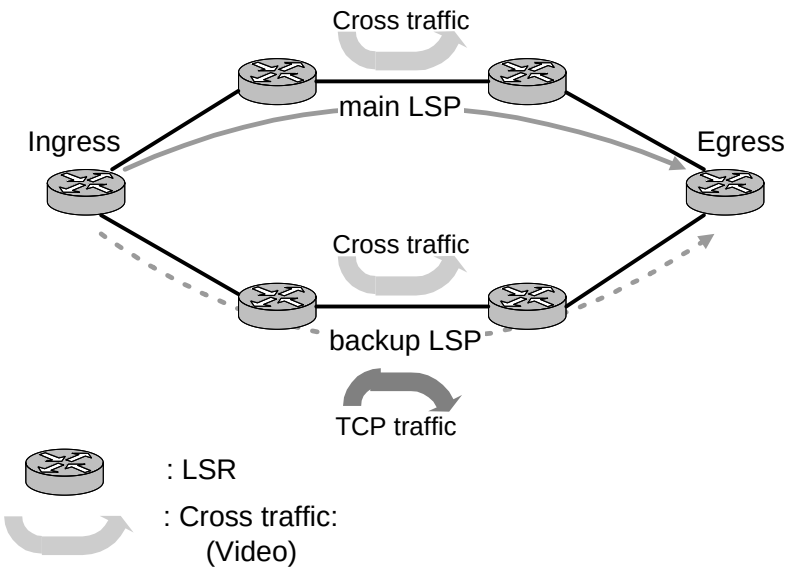


Figure 6.1: The main LSP with its backup LSP.

Both voice and video flows are in the same high priority class, and they are served in the link before lower priority traffic. Currently, we consider only the network in which each backup path is dedicated entirely to one and only one active path. In the other words, the shared backup path is not considered here. Also our controller is located in the ingress LSR.

There is a propagation delay from the ingress node to the bottleneck link in the main LSP,which will be divided into p_1 time slots. With the same time slot length, we will divide the propagation delay from the ingress node to the bottleneck link in the backup LSP into p_2 time slots. The bottleneck links of both LSPs periodically feed the video utilization back to the controller. Delay of the main LSP's and backup LSP's bottleneck information to be seen by the controller is q_1 and q_2 time slots, respectively. To simplify the notation, we define $d_1 = p_1 + q_1$ as a total delay in main LSP, and we also define $d_2 = p_2 + q_2$ as a total delay in backup LSP. Moreover, we let d_{max} as $\max(p_1 + q_1, p_2 + q_2)$

Below are the details of traffic models that we consider:

Voice Traffic We assume that each voice flow has a 64Kbps constant bit rate. The new voice calls arrive according to the Poisson process with rate λ_v. Each voice call has an exponentially distributed duration time with a mean of $\frac{1}{\mu_v}$ seconds. The number of voice calls in the main LSP is clearly a Markov chain with state space $\mathbf{S_{voice}} = \{0, \ldots, C\}$, where C is the capacity of links in the entire network.[3] (We represent C as in the unit of voice calls.) A transition probability matrix $M_v : (\mathbf{S}_{\text{voice}} \times \mathbf{S}_{\text{voice}}) \rightarrow [0, 1]$ can be calculated from the above voice traffic parameters.

Video Cross Traffic Since the video and voice traffic share the same queue, this cross traffic determines the service rate that voice traffic can be served at the bottleneck link before some of them get dropped. For convenience, instead of specifying the cross traffic distribution directly, we specify the distribution of "service process", which is the difference between the rate of cross traffic and the capacity of bottleneck link C. In i^{th} LSP, $i = 1, 2$, the service process is represented by a Markov chain with state space $\mathbf{S_i} = \{b_1, \ldots, b_{m_i}\}$, a transition probability matrix $M_i : (S_i \times S_i) \rightarrow [0, 1]$, and a set of distinct rate values, which are $b_1, \ldots, b_{m_i}$. Note that we refer to the 1^{st} LSP as main LSP and the 2^{nd} LSP as backup LSP.[4]

Low priority Traffic We do not attempt to monitor TCP packets and use them to estimate the goodput and fairness of TCP flows. Instead, we use the information,

[3] We assume that the entire network have same link capacity

[4] These two Markov Chains are assumed independent, e.g., the service rate is not used to do admission control on voice calls

which we got using high priority traffic model, to indirectly determine the TCP performance. We will discuss more details later in Section 6.3.

6.3 MDP State Formulation and Notation

A Markov Decision Process (MDP) consists of an action space, a state space, a state-transition structure, and a cost structure. We describe each of these components in details as follows:

Action Space Let $u(k) \in \{0, 1\}$ denote the control action made at time k. $u(k) = 1$ represents the duplication made at time k while $u(k) = 0$ means there is no duplication. Hence, we have the action space $\mathbf{U} = \{0, 1\}$.

State Space The system state has three components. The first component is voice traffic variables $X(k)$. This $X(k) \in \Re$ is a number of voice calls that is assigned to the main LSP in our sampling interval (time slot) at ingress LSR.[5] Moreover, $X(k)$ is bounded by the link capacity C, i.e., $0 \leq X(k) \leq C$. From above definition, the voice traffic variable takes a value from $\mathbf{S}_{\mathrm{v}} = \{[0, C]\}$

The second component is the recently received service rate information, designated by $\vec{b}(k) = [b_1(k), b_2(k)]$ where b_1 and b_2 are the service rate information taken from $\mathbf{S_1}$ and $\mathbf{S_2}$, respectively. So $\vec{b}$ takes value from $\mathbf{S_c} = \mathbf{S_1} \times \mathbf{S_2}$. With the same reasoning as in Section 4.3, the state of service process is the recently received state information, instead of real state at the bottleneck links. We use

[5] We can obtain this information by counting a number of voice packets at ingress LSR in our sampling interval, e.g., from time $k-1$ to time k. Then we divide the number of voice packets with 64Kbps to get a number of voice calls in that sampling interval (time slot).

this recent information because of the information delay, which prevents our controller to gain an immediate access to a real state at the bottleneck link. Hence this $\vec{b}(k)$ is the information that controller has at time k.

To gain a better understanding, we can discuss the relationship of our service rate information and the actual state at the bottleneck link. We first refer to the service process in i^{th} LSP with the above state variable $b_i(k)$ as a service process B_i. Then we define an associated service process $\hat{B}_i$ to be a real service process, which actually occurs in bottleneck link. We can see that our service process B_i is essentially the actual service process $\hat{B}_i$ with the q_i delay. Note that both service processes are uncontrollable from our control action, i.e., they are unaffected by the control action made by our controller. Hence, we can use a simple equation $b_i(k) = \hat{b}_i(k - q_i)$ to represent the relationship between these two service processes, where $\hat{b}_i(k)$ is a state at time k of process $\hat{B}_i$.

The third component is the control history. Let a vector of d_{max} elements, $\vec{w}(k) = [w^{(1)}(k)\ w^{(2)}(k)\ \ldots\ w^{(3k_s)}(k)]$ represent the past control decisions. Each element $w^{(j)}(k)$ denotes the control decision at time $(k - j)$. The value k_s is associated to the initial time-out value (ITO), which we will describe when we talk about TCP cost function below.

Using the above three components, we can define our system state variable $Y(k)$

as

$$Y(k) = \begin{bmatrix} X(k) \\ \vec{b}(k) \\ \vec{b}(k+1) \\ \vdots \\ \vec{b}(k+d_{\max}) \\ \vec{w}(k) \end{bmatrix}$$

We could see that this state variable is a combination of past control history, current voice state, current service rate state and all of d_N future service rate states. Actually, we could view the state variable as an internal state, which some of them cannot be observed. These future service rate states are required in order to represent the cost function, which we will define later in this section.

So the complete state space, $\mathbf{S}$, is $\mathbf{S_v} \times \mathbf{S_c}^{(d_{\max}+1)} \times \{0,1\}^{3k_s}$

Observation As we mentioned before, we can partially observe the state variable $Y(k)$. We define observation $Z(k)$ as

$$Z(k) = \begin{bmatrix} X(k) \\ \vec{b}(k) \\ \vec{w}(k) \end{bmatrix}$$

In other words, we could observe only the past control history, the current voice state $\vec{X}(k)$ and the current service rate state $\vec{b}(k)$.

State transition From the current state $Y(k)$, we apply a control u so that the system makes a transition to the new state $Y(k+1)$. The transition of each component is as follows:

- The state of service rate information makes a transition from $\vec{b}(k)$ to $\vec{b}(k+1)$

with probability $P(b_i(k), b_i(k+1)), i = 1, 2$, given by the $(b_i(k), b_i(k+1))^{\text{th}}$ entry in the given matrix M_i.

- The voice traffic state $X(k)$ makes a transition from $X(k)$ to $X(k+1)$ with probability $P(X(k), X_i(k+1))$, specified by the $(X_i(k), X_i(k+1))^{\text{th}}$ entry in the matrix M_v.[6]
- The control history $\vec{w}(k)$ updates as $w^{(1)}(k+1) = u(k)$ and $w^{(j)}(k+1) = w^{(j-1)}(k)$, for $j = 2, \ldots, 3k_s$.

MDP Cost and Value Function We define the one-step cost function at state $Y(k)$ and control action $u(k)$ by

$$\begin{aligned} & R\left(Y(k), u(k), Y(k+1)\right) \\ & = D_1 \cdot R_a(Y(k), u(k), Y(k+1)) + R_b(Y(k), u(k), Y(k+1)) \\ & \quad + D_2 \cdot R_c(Y(k), u(k), Y(k+1)) + R_d(Y(k), u(k), Y(k+1)). \end{aligned} \tag{6.1}$$

This cost function is considered as a penalized cost, in which we would like to minimize. It depends on the current state $Y(k)$, the immediate future state $Y(k+1)$ and the current control decision $\vec{u}(k)$. Each individual term of the cost function $R(Y(k), \vec{u}(k), Y(k+1))$ can be described as follows:

a. Voice Drops We will focus on the drops of the voice traffic, which is normally sent in the main LSP. As you could imagine, The information loss from these drops can be alleviated if we duplicate the packets.[7] Note that $X(k)$

[6] Notice that both transition of $\vec{b}(k)$ and $X(k)$ does not depend on control action $u(k)$

[7] It is still possible that the duplicated packet also get dropped in the backup LSP. However, we will totally lose the information only when both the original packet and duplicated packet are dropped. The probability of this event is small. Hence, we assume that the duplicated packet always helps protecting the information loss. As a result, we will not get any information loss when the control decision is to duplicate.

will share the same bottleneck link with $\hat{b}_1(k+p_1)$ in main LSP. Hence, the competition of $X(k)$ and $\hat{b}_1(k+p_1)$ over the bottleneck link can cause packet drops, which gives our drops term in the cost function. Recall that $b_1(k) = \hat{b}_1(k-q_1)$. Hence, drops can be calculated from the value of $X(k)$ and $b_1(k+p_1+q_1)$ or $b_i(k+d_i)$ using a formula $[X(k) - b_1(k+d_1)]^+$. Moreover, we assume that drops in main LSP is treated fairly between voice and video traffic. This gives the cost as follows,

$$R_a(Y(k), u(k), Y(k+1))$$
$$= [1-u(k)] \cdot \frac{X(k+1)}{X(k+1)+(C-b_1(k+d_1+1))} \cdot [X(k+1) - b_1(k+d_1+1)]^+.$$

b. Video Drops in backup LSP This drops will not happen if we do not duplicate voice packets. Note that we ignore video drops in the main LSP because these drops still exist no matter we duplicate the packets or not. Similar to the calculation of drops in main LSP above, drops in backup LSP can be calculated from $X(k)$ and $b_2(k+p_2+q_2)$ or $b_2(k+d_2)$ using a formula $[X(k)-b_2(k+d_2)]^+$. Again, we assume that drops in backup LSP is treated fairly between voice and video traffic.

$$R_b(Y(k), u(k), Y(k+1))$$
$$= u(k) \cdot \frac{(C-b_2(k+d_2+1))}{X(k+1)+(C-b_2(k+d_2+1))} \cdot [X(k+1) - b_2(k+d_2+1)]^+.$$

c. Cost of Preempt TCP Traffic The TCP traffic will be preempted by the duplicated packets. Hence, this cost term represents the loss of the bandwidth of TCP traffic due to the duplication decision.

$$R_c(Y(k), u(k), Y(k+1))$$
$$= u(k) \cdot [b_2(k+d_2+1) - [b_2(k+d_2+1) - X(k+1)]^+].$$

d. TCP Congestion Control Cost There is a reduction of the TCP performance as a result of the congestion control in TCP. With the TCP model proposed

in Section 2.3.2, we can get the cost associated with TCP congestion control as

$$R_d(Y(k), u(k), Y(k+1))$$
$$= D_{\text{setup}} \cdot u(k) \cdot w^{(k_s)}(k) + D_{\text{setup2}} \cdot u(k) \cdot w^{(2k_s)}(k) \cdot w^{(3k_s)}(k),$$

where k_s is the time-out length of connection procedure (ITO) in the unit of time slots.

The first term and the second term represent the duplication behavior that lead to the first Initial Time-Out (ITO) and the second ITO, respectively. We chose this TCP model because it follows the nature of the network when we implemented the duplication mechanism. Note that our duplication mechanism can make channel for TCP traffic (and hence TCP service rate) to change abruptly between very low value and very high value. That is, the time slot with the duplicate decision will have a low TCP service rate while the time slot without the duplicate decision will have a high TCP service rate. Hence, we chose this TCP model, which is a model for the case of TCP service rate with a large fluctuation. The detail and the validation of this TCP model is described in Section 2.3.2.

Note that the duplication may not alleviate all drops in the main LSP but it can cause high priority drops in the backup LSP. Hence, we will consider the drops in the backup LSP (which are the result of duplication) to be more important than the drops in the main path. As a result, we will use $D_1 < 1$, which enables the controller to duplicate only if necessary.

In addition, we will also let $D_2 < D_1$ to show that the real-time traffic have higher priority than the TCP traffic.

Now we obtain a cost function for our H-stage MDP problem as,

$$E^{[Y(0)]}\left[\sum_{k=0}^{H-1}\gamma^k R(Y(k),u(k),Y(k+1))\right],$$

where $E^{[Y]}[X] \triangleq E[X|Y]$.

6.4 Optimal Policy

In this section, we will discuss an optimal policy in our partially observed MDP problem. In order to apply DP algorithm, we will use state augmentation to reformulate our problem to the one with perfect information. Then we will discuss the difficulty with the DP algorithm for finding the optimal policy.

We define information vector $I(k)$ as

$$I(k) = [Z(0), Z(1), \ldots, Z(k), u(0), u(1), \ldots, u(k-1)],\ I(0) = Z(0).$$

We consider the class of policies that consists of a sequence of functions

$$\pi = \{\mu_0, \mu_1, \ldots, \mu_{H-1}\},$$

where μ_k maps information vector $I(k)$ into control $u(k) = \mu_k(I(k))$ and is such that $\mu_k(I(k)) \in \mathbf{U}$ for all $I(k)$. Such policies will be called "admissible". We want to find an admissible policy π that minimizes the value function,

$$J_H^\pi(Z(0)) = J_H^\pi(I(0)) = E^{[I(0)]}\left[\sum_{k=0}^{H-1}\gamma^k R(Y(k),\mu_k(I(k)),Y(k+1))\right].$$

Hence, an optimal policy π^* is one that minimizes the finite horizon value function; that is,

$$J_H^{\pi^*}(Z(0)) = \min_{\pi\in\Pi} J_H^\pi(Z(0)),$$

where Π is the set of all admissible policies.

By using state augmentation, we can reformulate our imperfect information (partially observed) problem to the one with perfect information as follows: With the information vector, we get an evolution of a new system as

$$I(k+1) = (I(k), Z(k+1), u(k)),$$

and get the dynamic programming algorithm as

$$J_{H-1}(I(H-1)) = \min_{u(H-1)} E^{[I(H-1),u(H-1)]}\left[R(Y(H-1), u(H-1), Y(H))\right] \tag{6.2}$$

$$\begin{aligned} &J_k(I(k)) \\ &= \min_{u(k)} E^{[I(k),u(k)]}\left[R(Y(k), u(k), Y(k+1)) + \gamma \cdot J_{k+1}(I(k), Z(k+1), u(k))\right]. \end{aligned} \tag{6.3}$$

Hence, the optimal policy $\pi^* = \{\mu_0^*, \mu_1^*, \ldots, \mu_{H-1}^*\}$ can be obtained by minimizing the right-hand side of the DP Equation (6.2) and (6.3).

We could let H go to infinity and get a value function $J^{\mu^*}(Y(0))$ for a corresponding infinite time horizon problem; That is

$$J^{\mu^*}(Z(0)) = \lim_{H\to\infty} J_H^{\pi^*}(Z(0))$$

Then, the optimal policy π^* for this infinite-time horizon is $\{\mu^*, \mu^*, \ldots\}$.

6.5 State Reduction

The main difficulty with DP algorithm in Equation (6.2) and (6.3) is that it is carried out over a state space of expanding dimension. When a new observation is added at each stage k, the dimension of the state (the information vector $I(k)$) increases accordingly. This difficulty highlights the importance of state reduction in the this section. We will start our state reduction by first considering voice state reduction.

Then we will consider reducing control history state space. Finally we will conclude the result of our state reduction.

6.5.1 Voice State Reduction

We use the concept of *sufficient statistics* to reduce the voice state space. We start by writing a definition of sufficient statistic[2] as follows,

Sufficient statistic is a function $S_k(I(k))$ of Information vector, such that a minimizing control in Equation (6.2) and (6.3) depends on $I(k)$ via $S_k(I_k)$. By this we mean that the minimization in the right-hand side of DP algorithm (6.2) and (6.3) can be written in terms of some function G_k as

$$\min_{u(k)} G_k(S_k(I(k)), u(k)).$$

The salient feature of sufficient statistic is that an optimal policy obtained by the preceding minimization can be written as

$$\mu_k^*(I(k)) = \overline{\mu}(S_k(I(k))). \tag{6.4}$$

We will propose the observation $Z(k)$ to be the sufficient statistic for our problem. If we can prove that $Z(k)$ is the sufficient statistic following the above definition, we will be able to reformulate our MDP problem using $Z(k)$ as our state. This new formulation will indirectly give the optimal policy of the original problem using Equation (6.4)

Following above definition, we can prove that $Z(k)$ is a sufficient statistic by writing the minimization in the right hand side of DP algorithm (6.2) and (6.3) in terms of function G_k as

$$\min_{\vec{u}(k)} G_k(Z(k), \vec{u}(k))$$

.

Before the proof, we should simplify some of notation to ease the explanation in the proof. We start with our stage cost $R(Y(k), u(k), Y(k+1))$. From Equation (6.1), the stage cost can be written as

$$\begin{aligned}
&R(Y(k), u(k), Y(k+1)) \\
&= [1-u(k)] \cdot \tfrac{X(k+1)}{X(k+1)+(C-b_1(k+d_1+1))} \cdot [X(k+1) - b_1(k+d_1+1)]^+ \\
&+u(k) \cdot \tfrac{(C-b_2(k+d_2+1))}{X(k+1)+(C-b_2(k+d_2+1))} \cdot [X(k+1) - b_2(k+d_2+1)]^+ \\
&+u(k) \cdot [b_2(k+d_2+1) - [b_2(k+d_2+1) - X(k+1)]^+] \\
&+D_{\text{setup}} \cdot u(k) \cdot w^{(k_s)}(k) + D_{\text{setup2}} \cdot u(k) \cdot w^{(2k_s)}(k) \cdot w^{(3k_s)}(k)
\end{aligned}$$

We could alternately write the stage cost $R(Y(k), u(k), Y(k+1))$ as

$$\begin{aligned}
R(Y(k), u(k), W(k)) \quad &= (1-u(k)) \cdot h_1(X(k+1), b_1(k+d_1+1)) \\
&+u(k) \cdot h_2(X(k+1), b_2(k+d_2+1)) \\
&+R_d(\vec{w}(k), u(k))
\end{aligned}$$

where,

$$\begin{aligned}
h_1(X,b) \quad &= D_1 \cdot \tfrac{X}{X+(C-b)} \cdot [X-b]^+ \\
h_2(X,b) \quad &= \tfrac{C-b}{X+(C-b)} \cdot [X-b]^+ + D_2 \cdot [b-[b-X]^+] \\
R_d(\vec{w},u) \quad &= D_{\text{setup}} \cdot u \cdot w^{(k_s)} + D_{\text{setup1}} \cdot u \cdot w^{(2k_s)} \cdot w^{(3k_s)}
\end{aligned}$$

Now we shall start the proof from Equation (6.2):

$$
\begin{aligned}
& J_{H-1}(I(H-1)) \\
& = \min_{u(H-1)} E^{[I(H-1),u(H-1)]} \left[R(Y(H-1), u(H-1), Y(H)) \right] \\
& = \min_{u(H-1)} E^{[I(H-1),u(H-1)]} \left[\begin{array}{l} [1-u(H-1)] \cdot h_1(X(H), b_1(H+d_1)) \\ +u(H-1) \cdot h_2(X(H), b_2(H+d_2)) \\ +R_d(\vec{w}(H-1), u(H-1)) \end{array} \right] \\
& = \min_{u(H-1)} E^{[Z(H-1),Z(H-2),\ldots,Z(0),u(H-2),\ldots,u(0),u(H-1)]} \\
& \qquad \left[\begin{array}{l} [1-u(H-1)] \cdot h_1(X(H), b_1(H+d_1)) \\ +u(H-1) \cdot h_2(X(H), b_2(H+d_2)) \\ +R_d(\vec{w}(H-1), u(H-1)) \end{array} \right]
\end{aligned}
$$

Now we will analyze each term above as follows:

$$
\begin{aligned}
& E^{[Z(H-1),Z(H-2),\ldots,Z(0),u(H-2),\ldots,u(0),u(H-1)]} \left[[1-u(H-1)] \cdot h_1(X(H), b_1(H+d_1)) \right] \\
& = E^{[X(H-1),X(H-2),\ldots,X(0),b_1(H-1),b_1(H-2),\ldots,b_1(0),u(H-1),u(H-2),\ldots,u(0)]} \\
& \qquad \left[[1-u(H-1)] \cdot h_1(X(H), b_1(H+d_1)) \right] \\
& = E^{[X(H-1),b_1(H-1),u(H-1),u(H-2),\ldots,u(0)]} \left[[1-u(H-1)] \cdot h_1(X(H), b_1(H+d_1)) \right] \\
& = E^{[X(H-1),b_1(H-1),u(H-1)]} \left[[1-u(H-1)] \cdot h_1(X(H), b_1(H+d_1)) \right]
\end{aligned}
$$

The third equality comes from Markov Property of our Markov Decision Process (MDP) problem. The last equality comes from the fact that $X(\cdot)$ and $b(\cdot)$ do not depend on $u(\cdot)$

Similar to the first term above, we can analyze and get the other two terms,

$$
\begin{aligned}
& E^{[Z(H-1),Z(H-2),\ldots,Z(0),u(H-2),\ldots,u(0),u(H-1)]} \left[u(H-1) \cdot h_2(X(H), b_2(H+d_2)) \right] \\
& = E^{[X(H-1),b_2(H-1),u(H-1)]} \left[u(H-1) \cdot h_2(X(H), b_2(H+d_2)) \right]
\end{aligned}
$$

and

$$
\begin{aligned}
& E^{[Z(H-1),Z(H-2),\ldots,Z(0),u(H-2),\ldots,u(0),u(H-1)]}\left[R_d(\vec{w}(H-1),u(H-1))\right] \\
&= E^{[\vec{w}(H-1),\vec{w}(H-2),\ldots,\vec{w}(0),u(H-2),\ldots,u(0),u(H-1)]}\left[R_d(\vec{w}(H-1),u(H-1))\right] \\
&= E^{[\vec{w}(H-1),u(H-1)]}\left[R_d(\vec{w}(H-1),u(H-1))\right].
\end{aligned}
$$

Last equality comes from the fact that we need only $\vec{w}(H-1)$ and $u(H-1)$ to calculate $R_d(\vec{w}(H-1))$.

Next, we can go back to our DP algorithm; That is

$$
\begin{aligned}
& J_{H-1}(I(H-1)) \\
&= \min_{u(H-1)} \\
&\qquad \left[\begin{array}{l}
E^{[X(H-1),b_1(H-1),u(H-1)]}\left[[1-u(H-1)]\cdot h_1(X(H),b_1(H+d_1))\right] \\
+E^{[X(H-1),b_2(H-1),u(H-1)]}\left[u(H-1)\cdot h_2(X(H),b_2(H+d_2))\right] \\
+E^{[\vec{w}(H-1),u(H-1)]}\left[R_d(\vec{w}(H-1),u(H-1))\right]
\end{array}\right] \\
&= \min_{u(H-1)} E^{[Z(H-1),u(H-1)]}\left[\begin{array}{l}
[1-u(H-1)]\cdot h_1(X(H),b_1(H+d_1)) \\
+u(H-1)\cdot h_2(X(H),b_2(H+d_2)) \\
+R_d(\vec{w}(H-1),u(H-1))
\end{array}\right] \\
&= \overline{J}_{H-1}(Z(H-1)).
\end{aligned}
$$

Therefore, we can write the DP algorithm (6.2) in term of a function G_{H-1} as

$$\min_{u(H-1)} G_{H-1}(Z(H-1),\vec{u}(H-1)).$$

Now we continue our work on the DP algorithm (6.3). Using induction, we assume

$$J_k(I(k+1)) = \overline{J}_k(Z(k+1))$$

Then

$$
\begin{aligned}
& J_k(I(k)) \\
& = \min_{u(k)} E^{[I(k),\vec{u}(k)]} \left[R(Y(k), u(k), Y(k+1)) + \gamma \cdot J_{k+1}(I(k), Z(k+1), u(k)) \right] \\
& = \min_{u(k)} E^{[I(k),u(k)]} \left[\begin{array}{l} [1-u(k)] \cdot h_1(X(k+1), b_1(k+d_1+1)) \\ +u(k) \cdot h_2(X(k+1), b_2(k+d_2+1)) \\ +R_d(\vec{w}(k), u(k)) \\ +\gamma \cdot J_{k+1}(I(k), Z(k+1), u(k)) \end{array} \right] \\
& = \min_{u(H-1)} E^{[Z(k),Z(k-1),\ldots,Z(0),u(k-1),\ldots,u(0),u(k)]} \\
& \qquad \left[\begin{array}{l} [1-u(k)] \cdot h_1(X(k+1), b_1(k+d_1+1)) \\ +u(k) \cdot h_2(X(k+1), b_2(k+d_2+1)) \\ +R_d(\vec{w}(k), u(k)) \\ +\gamma \cdot \overline{J}_{k+1}(Z(k+1)) \end{array} \right]
\end{aligned}
$$

Again, we will analyze each terms. First, the next stage cost becomes

$$
\begin{aligned}
& E^{[Z(k),Z(k-1),\ldots,Z(0),u(k-1),\ldots,u(0),u(k)]} \left[\overline{J}_{k+1}(Z(k+1)) \right] \\
& = E^{[Z(k),u(k)]} \left[\overline{J}_{k+1}(Z(k+1)) \right].
\end{aligned}
$$

The equality comes from the fact that $\vec{b}(k+1)$, $\vec{X}(k+1)$ and $\vec{w}(k+1)$ in $Z(k+1)$ are conditionally independent of the other variables given $Z(k)$ and $u(k)$.

For the other terms, we could analyze them the same way we did in $\overline{J}_{H-1}(Z(H-1))$. So we complete the analysis.

Next, we can go back to the DP algorithm,

$$
\begin{aligned}
& J_k(I(k)) \\
& = \min_{u(k)} \\
& \qquad \left[\begin{array}{l}
E^{[X(k),b_1(k),u(k)]} \left[[1-u(k)] \cdot h_1(X(k+1), b_1(k+d_1+1)) \right] \\
+E^{[X(k),b_2(k),u(k)]} \left[u(k) \cdot h_2(X(k+1), b_2(k+d_2+1)) \right] \\
+E^{[\vec{w}(k),u(k)]} \left[R_d(\vec{w}(k), u(k)) \right] \\
+\gamma \cdot E^{[Z(k),u(k)]} \left[\overline{J}_{k+1}(Z(k+1)) \right]
\end{array} \right] \\
& = \min_{u(k)} E^{[Z(k),u(k)]} \\
& \qquad \left[\begin{array}{l}
[1-u(k)] \cdot h_1(X(k+1), b_1(k+d_1+1)) \\
+u(k) \cdot h_2(X(k+1), b_2(k+d_2+1)) \\
+R_d(\vec{w}(k), u(k)) \\
+\gamma \cdot \overline{J}_{k+1}(Z(k+1))
\end{array} \right] \\
& = \overline{J}_k(Z(k)).
\end{aligned}
$$

Therefore we can write the DP algorithm (6.3) in term of function G_k as

$$\min_{u(k)} G_k(Z(k), \vec{u}(k)).$$

QED.

So $Z(k)$ is a sufficient statistic of $I(k)$, and we have the new dynamic programming Algorithm as,

$$
\begin{aligned}
& \overline{J}_{H-1}(Z(H-1)) \\
& = \min_{u(H-1)} E^{[Z(H-1),u(H-1)]} \left[\begin{array}{l}
[1-u(H-1)] \cdot h_1(X(H), b_1(H+d_1)) \\
+u(H-1) \cdot h_2(X(H), b_2(H+d_2)) \\
+R_d(\vec{w}(H-1), u(H-1))
\end{array} \right]
\end{aligned}
$$

and

$$\overline{J}_k(Z(k))$$

$$= \min_{u(k)} E^{[Z(k),u(k)]} \left[\begin{array}{l} [1-u(k)] \cdot h_1(X(k+1), b_1(k+d_1+1)) \\ +u(k) \cdot h_2(X(k+1), b_2(k+d_2+1)) \\ +R_d(\vec{w}(k), u(k)) \\ +\gamma \cdot \overline{J}_{k+1}(Z(k+1)) \end{array} \right]$$

Now we can reformulate the problem as follows: We will define a new state variable[8]

$$\Psi(k) = Z(k) = \left[\begin{array}{c} X(k) \\ \vec{b}(k) \\ \vec{w}(k) \end{array} \right],$$

and we have a new system with an infinite horizon cost,

$$\lim_{H \to \infty} E^{[\Psi(0)]} \left[\sum_{k=0}^{H-1} \gamma^k R'(\Psi(k), u(k)) \right],$$

where

$$R'(\Psi(k), u(k)) = R_{\text{abc}}(\Psi(k), u(k)) + R_d(\Psi(k), u(k)),$$

given $R_{\text{abc}}(\Psi(k), u(k))$ is the cost of high priority drops in main LSP , backup LSP and preempt TCP traffic, respectively. $R_d(\Psi(k), u(k))$ is a cost of TCP congestion control

[8]We replace observation variable $Z(k)$ with a new state variable $\Psi(k)$ to prevent a confusion with z_k in subsequence sections.

mechanism, i.e.,

$$
\begin{aligned}
&R_{\text{abc}}(\Psi(k), u(k)) \\
&= E^{[\Psi(k),u(k)]} \left[\begin{array}{l}
D_1 \cdot [1 - u(k)] \cdot \frac{X(k+1)}{X(k+1)+(C-b_1(k+d_1+1))} \cdot [X(k+1) - b_1(k+d_1+1)]^+ \\
+u(k) \cdot \frac{(C-b_2(k+d_2+1))}{X(k+1)+(C-b_2(k+d_2+1))} \cdot [X(k+1) - b_2(k+d_2+1)]^+ \\
+D_2 \cdot u(k) \cdot [b_2(k+d_2+1) - [b_2(k+d_2+1) - X(k+1)]^+]
\end{array} \right]
\end{aligned}
$$

and

$$R_d(\Psi(k), u(k)) = D_{\text{setup}} \cdot u(k) \cdot w^{(k_s)}(k) + D_{\text{setup2}} \cdot u(k) \cdot w^{(2k_s)}(k) \cdot w^{(3k_s)}(k).$$

6.5.2 Control History State Reduction

Following the last section on voice state reduction, we now mathematically prove that we need only three control history state variables instead of all $3k_s$ control history state variables $\vec{w}$.

So far, we have a MDP problem with associated cost function

$$E^{[\Psi(0)]} \left[\sum_{k=0}^{H-1} \gamma^k \left[R'(\Psi(k), u(k)) \right] \right],$$

where our state variable is $\Psi(k) = \left[\vec{b}(k), X(k), w^{(1)}(k), \ldots, w^{(3k_s)}(k) \right]$.

In this section, we will define a reduced order MDP problem, which is sampled every k_s time slot from original problem. More specifically, we will focus on a MDP problem with a new cost function

$$E^{[\Psi(0)]} \left[\sum_{n=0}^{N-1} \gamma^{nk_s} \left[R'(\Psi(nk_s), u(n)) \right] \right].$$

We can easily see that the cost function in this N-stage problem comes from sampling original cost function for every k_s time slot, i.e. cost sampled at $0, k_s, 2k_s, \ldots, (N-1)k_s$. In this reduced order problem, we will define a new state

variable as $\Upsilon(k) \triangleq \left[X(k), \vec{b}(k), w^{(k_s)}(k), w^{(2k_s)}(k), w^{(3k_s)}(k)\right]$. Actually, we could view $\Upsilon(k)$ as a mapping function from our state variable $\Psi(k)$ to the new state variable such that

$$\Upsilon(k) = \Upsilon(\Psi(k)) = \left[X(k), \vec{b}(k), w^{(k_s)}(k), w^{(2k_s)}(k), w^{(3k_s)}(k)\right].$$

We now have a reduced system that progress k_s time-steps instead of only single time-step in original system. The system equation or state transition of this reduced system can be described as follows:

The state transitions of $X(k)$ and $\vec{b}(k)$ are still governed by a discrete-time Markov chain. The transitions of $w^{(i)}(k)$ are changed because the reduced system has only three control history variables instead of all $3k_s$ variables in the original system. In the reduced system, $u(k)$ becomes $w^{(k_s)}(k+k_s)$ in the next k_s time-steps, $w^{(k_s)}(k)$ becomes $w^{(2k_s)}(k+k_s)$ in the next k_s time-steps, and $w^{(2k_s)}(k)$ becomes $w^{(3k_s)}(k+k_s)$ in the next k_s time-steps. This state transition can be summarized to

$$\begin{aligned}
X(k) &\rightarrow X(k+k_s),\\
\vec{b}(k) &\rightarrow \vec{b}(k+k_s),\\
u(k) &\rightarrow w^{(k_s)}(k+k_s),\\
w^{(k_s)}(k) &\rightarrow w^{(2k_s)}(k+k_s),\\
w^{(2k_s)}(k) &\rightarrow w^{(3k_s)}(k+k_s).
\end{aligned}$$

Considering N-stage problem for this reduced system, we can use the dynamic programming algorithm to obtain its optimal policy $\widehat{\mu}_0, \widehat{\mu}_1, \ldots, \widehat{\mu}_{N-1}$

Now we go back to $\{Nk_s\}$-stage (or H-stage) problem[9] for original system. We

[9]Through the rest of this section, we will assume that H is divisible by k_s. Hence we can get $H = Nk_s$. One can easily relax this assumption and get the same claim.

would like to claim that the optimal policy for original system is

$$\pi^* = \left[\underbrace{\widehat{\mu}_0, \ldots, \widehat{\mu}_0}_{k_s}, \underbrace{\widehat{\mu}_1, \ldots, \widehat{\mu}_1}_{k_s}, \cdots, \underbrace{\widehat{\mu}_{N-1}, \ldots, \widehat{\mu}_{N-1}}_{k_s} \right].$$

In other words, we would like to claim that the optimal policy $\pi^* = [\mu_0^*, \mu_1^*, \ldots, \mu_{H-1}^*]$ for original system is

$$\mu_k^*(\Psi(k)) = \widehat{\mu}_n(\Upsilon(k)),$$

where $n = \lfloor \frac{k}{k_s} \rfloor$, for $k = 0, 1, \ldots, H-1$.

With this claim, we can obtain the optimal policy of original problem via optimal policy of reduced order problem. Hence, we can work on reduced order system, which has much smaller state space.

The intuitive reason behind this claim can be explained as follows: We start by taking a closer look at $R_d(\Psi(k)(k), u(k))$ term in our stage cost $R'(\Psi(k)(k), u(k))$,

$$R_d(\Psi(k)(k), u(k)) = \quad D_{\text{setup}} \cdot u(k) \cdot w^{(k_s)}(k) + D_{\text{setup2}} \cdot u(k) \cdot w^{(2k_s)}(k) \cdot w^{(3k_s)}(k).$$

We could see that $R_d(\Psi(k)(k), u(k))$ uses only three variables out of all $\vec{w}$ variables. These three variables are evenly set apart from each other, i.e., every k_s time-steps. Moreover, they are the only variables from $\vec{w}$ that are used to calculate the optimal control because there is no $\vec{w}$ variables in $R_{\text{abc}}(\Psi(k)^d(k), u(k))$ at all. Subsequently, there is a strong potential that we can keep only these three variables and eliminate the rest from our state variable vector to reduce the state space.

Now, we will mathematically prove our claim in detail as follows: we will first state the dynamic programming algorithm for this original problem. Then we will state the dynamic programming algorithm for the reduced order problem. Following these two dynamic programming algorithms, we will propose our proposition as an accurate version of our claim above. Then we will mathematically prove the proposition.

Dynamic Programming Algorithm of $\tilde{J}_{H,k}(\Psi(k))$

In this new finite time horizon problem, we could restate the dynamic programming algorithm as follows:

For every initial state $\Psi(0)$, the optimal cost $\tilde{J}_H^{\pi^*}(\Psi(0))$ is equal to $\tilde{J}_{H,0}(\Psi(0))$, given by the last step of the following algorithm, which proceeds backward in time from period $H-1$ to period 0:

$$\tilde{J}_{H,H}(\Psi(H)) = 0$$

$$\tilde{J}_{H,k}(\Psi(k)) = \min_{u(k)} E^{[\Psi(k)]}\left[R'(\Psi(k), u(k)) + \gamma \cdot \tilde{J}_{H,k+1}(\Psi(k+1))\right]$$

for $k = 0, 1, \ldots, H-1$

Moreover, the optimal policy at time k, $\mu_k^*(\Psi(k))$, can be obtained from $u(k)$ that minimize the right hand side of $\tilde{J}_{H,k}(\Psi(k))$ in its k^{th} dynamic programming algorithm step.

Dynamic Programming Algorithm of $V_{N,n}(\Upsilon(nk_s))$

As mentioned before, we will setup reduced order MDP Problem by Sampling every k_s. So we look at the problem with cost function

$$E^{[\Psi(0)]}\left[\sum_{n=0}^{N-1} \gamma^{nk_s}\left[R'(\Psi(nk_s), u(n))\right]\right].$$

As define earlier, we have new state variable $\Upsilon(k) \triangleq \left[X(k), \vec{b}(k), w^{(k_s)}(k), w^{(2k_s)}(k), w^{(3k_s)}(k)\right]$. Recall that we could view $\Upsilon(k)$ as a mapping function from our state variable $\Psi(k)$ to the new state variable, i.e.,

$$\Upsilon(k) = \Upsilon(\Psi(k)) = \left[X(k), \vec{b}(k), w^{(k_s)}(k), w^{(2k_s)}(k), w^{(3k_s)}(k)\right].$$

We will now define new cost-to-go function $V_{N,n}(\Upsilon(nk_s))$ and our new dynamic programming Algorithm as follows:

$$V_{N,N}(\Upsilon(Nk_s)) = 0$$

$$V_{N,n}(\Upsilon(nk_s)) = \min_{u(n)} E^{[\Upsilon(nk_s)]}\left[R'(\Upsilon(nk_s), u(n)) + \gamma^{k_s} \cdot V_{N,n+1}(\Upsilon(nk_s + k_s))\right]$$

for $n = 0, 1, \ldots, N-1$.

With this new cost-to-go function, we define a function, $\widehat{\mu}_n(\Upsilon(nk_s))$, associated to $u(n)$ that minimizes the right hand side of $V_{H,n}(\Upsilon(nk_s))$ in its n^{th} dynamic programming algorithm step; That is

$$\widehat{\mu}_n(\Upsilon(nk_s)) = \operatorname{argmin}_{u(n)} E^{[\Upsilon(nk_s)]}\left[R'(\Upsilon(nk_s), u(n)) + \gamma^{k_s} \cdot V_{N,n+1}(\Upsilon(nk_s + k_s))\right],$$

for $n = 0, 1, \ldots, N-1$.

Proposition 6.5.1. *The optimal policy $\pi^* = [\mu_0^*, \mu_0^*, \ldots, \mu_{H-1}^*]$ of the original system can be written in term of the optimal policy of reduced order problem as*

$$\mu_k^*(\Psi(k)) = \widehat{\mu}_n(\Upsilon(k)),$$

where $n = \lfloor \frac{k}{k_s} \rfloor$, for $k = 0, 1, \ldots, H-1$.

Moreover, the value function $\tilde{J}_{H,k}(\Psi(k))$ of original problem can be written in term of the value function $V_{N,n}(\Upsilon(nk_s))$ of reduced order problem as

$$\tilde{J}_{H,k}(\Psi(k)) = V_{N,\lfloor \frac{k}{k_s} \rfloor}(\Upsilon(k)) + E^{[\Psi(k)]}\left[\sum_{i=1}^{k_s-1} \gamma^i \cdot V_{N,\lfloor \frac{k+i}{k_s} \rfloor}(\Upsilon(k+i))\right],$$

for $k = 0, 1, \ldots, H - k_s - 1$,

$$\tilde{J}_{H,H-l}(\Psi(H-l)) = V_{N,N-1}(\Upsilon(H-l)) + E^{[\Psi(H-l)]}\left[\sum_{i=1}^{l-1} \gamma^{l-i} \cdot V_{N,N-1}(\Upsilon(H-i))\right],$$

for $l = 2, 3, \ldots, k_s$, and

$$\tilde{J}_{H,H-1}(\Psi(H-1)) = V_{N,N-1}(\Upsilon(H-1)).$$

Proof of Proposition 6.5.1

We will use induction to prove this proposition.

Initial step: at $H-1$

$$\begin{aligned}
&\tilde{J}_{H,H-1}(\Psi(H-1)) \\
&= \min_{u(H-1)} E^{[\Psi(H-1)]}\left[R'(\Psi(H-1), u(H-1))\right] \\
&= \min_{u(H-1)} E^{[\Upsilon(H-1)]}\left[R'(\Upsilon(H-1), u(H-1))\right] \\
&= V_{N,N-1}(\Upsilon(H-1))
\end{aligned}$$

The second equality comes from the fact that we need only some parts of state variable $\Psi(H-1)$ to calculate $R'(\Psi(H-1), u(H-1))$. Moreover, we could see that

$$\mu^*_{H-1}(\Psi(H-1)) = \widehat{\mu}_{N-1}(\Upsilon(H-1)).$$

Instead of presenting the rest of boundary condition, i.e., steps of $k = H-2, H-3, \ldots, H-k_s$, we will now present induction step, where $k = 0, 1, \ldots, H-k_s-1$. The reason to rearrange the proof is their similarity; as the readers understand the induction step, the steps of $k = H-2, H-3, \ldots, H-k_s$ is almost trivial.

Induction step: we will first assume

$$\begin{aligned}
&\tilde{J}_{H,k+1}(\Psi(k+1)) \\
&= V_{N,\lfloor\frac{k+1}{k_s}\rfloor}(\Upsilon(k+1)) + E^{[\Psi(k+1)]}\left[\sum_{i=1}^{k_s-1} \gamma^i \cdot V_{N,\lfloor\frac{k+i+1}{k_s}\rfloor}(\Upsilon(k+i+1))\right].
\end{aligned}$$

Then,

$$
\begin{aligned}
&\tilde{J}_{H,k}(\Psi(k)) \\
&= \min_{u(k)} E^{[\Psi(k)]}\left[R'(\Psi(k),u(k)) + \gamma \cdot \tilde{J}_{H,k+1}(\Psi(k+1))\right] \\
&= \min_{u(k)} E^{[\Psi(k)]}\left[\begin{array}{l} R'(\Psi(k),u(k)) + \gamma \cdot V_{N,\lfloor\frac{k+1}{k_s}\rfloor}(\Upsilon(k+1)) \\ +\gamma \cdot E^{[\Psi(k+1)]}\left[\sum_{i=1}^{k_s-1} \gamma^i \cdot V_{N,\lfloor\frac{k+i+1}{k_s}\rfloor}(\Upsilon(k+i+1))\right] \end{array}\right] \\
&= \min_{u(k)}\left[\begin{array}{l} E^{[\Psi(k)]}[R'(\Psi(k),u(k))] \\ +\gamma \cdot E^{[\Psi(k)]}[V_{N,\lfloor\frac{k+1}{k_s}\rfloor}(\Upsilon(k+1))] \\ +\gamma \cdot E^{[\Psi(k)]}E^{[\Psi(k+1)]}\left[\sum_{i=1}^{k_s-1} \gamma^i \cdot V_{N,\lfloor\frac{k+i+1}{k_s}\rfloor}(\Upsilon(k+i+1))\right] \end{array}\right] \\
&= \min_{u(k)}\left[\begin{array}{l} E^{[\Psi(k)]}[R'(\Psi(k),u(k))] \\ +\gamma \cdot E^{[\Psi(k)]}[V_{N,\lfloor\frac{k+1}{k_s}\rfloor}(\Upsilon(k+1))] \\ +\gamma \cdot E^{[\Psi(k)]}\left[\sum_{i=1}^{k_s-1} \gamma^i \cdot V_{N,\lfloor\frac{k+i+1}{k_s}\rfloor}(\Upsilon(k+i+1))\right] \end{array}\right] \\
&= \min_{u(k)}\left[\begin{array}{l} E^{[\Psi(k)]}[R'(\Psi(k),u(k))] \\ +E^{[\Psi(k)]}\left[\sum_{i=1}^{k_s-1} \gamma^i \cdot V_{N,\lfloor\frac{k+i}{k_s}\rfloor}(\Upsilon(k+i))\right] \\ +\gamma^{k_s} \cdot E^{[\Psi(k)]}[V_{N,\lfloor\frac{k+k_s}{k_s}\rfloor}(\Upsilon(k+k_s))] \end{array}\right].
\end{aligned}
$$

We could write out all state variables of $\Upsilon(k+i)$ and get,

$$
\begin{aligned}
&\tilde{J}_{H,k}(\Psi(k)) \\
&= \min_{u(k)} \\
&\left[\begin{array}{l} E^{[\Psi(k)]}[R'(\Psi(k),u(k))] \\ +E^{[\Psi(k)]}\left[\sum_{i=1}^{k_s-1} \gamma^i \cdot V_{N,\lfloor\frac{k+i}{k_s}\rfloor}\left(\begin{array}{l} X(k+i), \vec{b}(k+i), \\ w^{(k_s)}(k+i), w^{(2k_s)}(k+i), w^{(3k_s)}(k+i) \end{array}\right)\right] \\ +\gamma^{k_s} \cdot E^{[\Psi(k)]}[V_{N,\lfloor\frac{k+k_s}{k_s}\rfloor}(\Upsilon(k+k_s))] \end{array}\right].
\end{aligned}
$$

From state transition: $w^{(m)}(k+i) = w^{(m-1)}(k+i-1) = \cdots = w^{(m-i)}(k)$, for

$0 < i < k_s$ and $k_s \leq m \leq 3k_s$,

$$\tilde{J}_{H,k}(\Psi(k))$$
$$= \min_{u(k)} \left[\begin{array}{l} E^{[\Psi(k)]}[R'(\Psi(k), u(k))] \\ +\gamma^{k_s} \cdot E^{[\Psi(k)]}[V_{N,\lfloor \frac{k+k_s}{k_s} \rfloor}(\Upsilon(k+k_s))] \\ +E^{[\Psi(k)]} \left[\sum_{i=1}^{k_s-1} \gamma^i \cdot V_{N,\lfloor \frac{k+i}{k_s} \rfloor} \left(\begin{array}{l} X(k+i), \vec{b}(k+i), \\ w^{(k_s-i)}(k), w^{(2k_s-i)}(k), w^{(3k_s-i)}(k) \end{array} \right) \right] \end{array} \right].$$

We can see that all $X(k+i), \vec{b}(k+i), w^{(k_s-i)}(k), w^{(2k_s-i)}(k)$ and $w^{(3k_s-i)}(k))$ are independent of $u(k)$. The reason for such independence is as follows: $X(k+i)$ and $\vec{b}(k+i)$ are uncontrollable variables as we discussed earlier. Hence, they will not depend on $u(k)$. The control history state variables, $w^{(k_s-i)}(k), w^{(2k_s-i)}(k)$ and $w^{(3k_s-i)}(k))$ are given in $\Psi(k)$. Hence, they are independent of $u(k)$. As a result of the above dependences, $V_{N,\lfloor \frac{k+i}{k_s} \rfloor}(\cdot)$, which is a function of those independent state variables, would be independent of $u(k)$ as well. Hence, we can take all $V_{N,\lfloor \frac{k+i}{k_s} \rfloor}(\cdot)$ terms out of our minimization,

$$\begin{aligned} &\tilde{J}_{H,k}(\Psi(k)) \\ &= \min_{u(k)} \left[E^{[\Psi(k)]}[R'(\Psi(k), u(k))] + \gamma^{k_s} \cdot E^{[\Psi(k)]}[V_{N,\lfloor \frac{k}{k_s} \rfloor+1}(\Upsilon(k+k_s))] \right] \\ &\quad + \sum_{i=1}^{k_s-1} \gamma^i \cdot E^{[\Psi(k)]} \\ &\qquad \left[V_{N,\lfloor \frac{k+i}{k_s} \rfloor}(X(k+i), \vec{b}(k+i), w^{(k_s-i)}(k), w^{(2k_s-i)}(k), w^{(3k_s-i)}(k)) \right] \\ &= \min_{u(k)} \left[E^{[\Psi(k)]}[R'(\Psi(k), u(k))] + \gamma^{k_s} \cdot E^{[\Psi(k)]}[V_{N,\lfloor \frac{k}{k_s} \rfloor+1}(\Upsilon(k+k_s))] \right] \\ &\quad + \sum_{i=1}^{k_s-1} \gamma^i \cdot E^{[\Psi(k)]} \left[V_{N,\lfloor \frac{k+i}{k_s} \rfloor}(\Upsilon(k+i)) \right]. \end{aligned} \tag{6.5}$$

Recall that the optimal policy at time k, $\mu_k^*(\Psi(k))$, can be obtained from $u(k)$ that minimize the right hand side of $\tilde{J}_{H,k}(\Psi(k))$ in its k^{th} dynamic programming algorithm

step. Hence, we can obtain $\mu_k^*(\Psi(k))$ from

$$\begin{aligned}
&\mu_k^*(\Psi(k)) \\
&= \text{argmin}_{u(k)} \left[\; E^{[\Psi(k)]}[R'(\Psi(k), u(k))] + \gamma^{k_s} \cdot E^{[\Psi(k)]}[V_{N,\lfloor \frac{k}{k_s} \rfloor + 1}(\Upsilon(k+k_s))] \; \right] \\
&\quad + \sum_{i=1}^{k_s-1} \gamma^i \cdot E^{[\Psi(k)]} \left[\; V_{N,\lfloor \frac{k+i}{k_s} \rfloor}(\Upsilon(k+i)) \; \right] \\
&= \text{argmin}_{u(k)} \left[\; E^{[\Psi(k)]}[R'(\Psi(k), u(k))] + \gamma^{k_s} \cdot E^{[\Psi(k)]}[V_{N,\lfloor \frac{k}{k_s} \rfloor + 1}(\Upsilon(k+k_s))] \; \right].
\end{aligned}$$

We actually can see that this $\mu_k^*(\Psi(k))$ is the same as if we get a $u(k)$ that minimizes the right hand side of $V_{N,n}(\Upsilon(nk_s))$ in its n^{th} dynamic programming Algorithm step; That is

$$\mu_k^*(\Psi(k)) = \widehat{\mu}_n(\Upsilon(k)),$$

where $n = \lfloor \frac{k}{k_s} \rfloor$.

Also from Equation (6.5), we can write out the minimizing term and get

$$\tilde{J}_{H,k}(\Psi(k)) = V_{N,\lfloor \frac{k}{k_s} \rfloor}(\Upsilon(k)) + \sum_{i=1}^{k_s-1} \gamma^i \cdot E^{[\Psi(k)]} \left[\; V_{N,\lfloor \frac{k+i}{k_s} \rfloor}(\Upsilon(k+i)) \; \right].$$

Now we will go over *the rest of boundary condition*, i.e., steps of

$k = H-2, H-3, \ldots, H-k_s$,

$$\tilde{J}_{H,H-l}(\Psi(H-l)) = V_{N,N-1}(\Upsilon(H-l)) + E^{[\Psi(H-l)]} \left[\sum_{i=1}^{l-1} \gamma^{l-i} \cdot V_{N,N-1}(\Upsilon(H-i)) \right],$$

for $l = 2, 3, \ldots, k_s$.

We start at step of $l = 2$ such that we can prove

$$\begin{aligned}
&\tilde{J}_{H,H-2}(\Psi(H-2)) \\
&= \min_{u(H-2)} E^{[\Psi(H-2)]} \left[R'(\Psi(H-2), u(H-2)) + \gamma \cdot \tilde{J}_{H,H-1}(\Psi(H-1)) \right] \\
&= \min_{u(H-2)} E^{[\Psi(H-2)]} \left[\; R'(\Psi(H-2), u(H-2)) + \gamma \cdot V_{N,N-1}(\Upsilon(H-1)) \; \right].
\end{aligned}$$

We could write out all state variables of $\Upsilon(H-1)$ and get

$$\tilde{J}_{H,H-2}(\Psi(H-2))$$
$$= \min_{u(H-2)}$$
$$\left[\begin{array}{l} E^{[\Psi(H-2)]}[R'(\Psi(H-2),u(H-2))] \\ +E^{[\Psi(H-2)]}\left[\gamma \cdot V_{N,N-1}\left(\begin{array}{l} X(H-1),\vec{b}(H-1), \\ w^{(k_s)}(H-1),w^{(2k_s)}(H-1),w^{(3k_s)}(H-1) \end{array} \right)\right] \end{array} \right].$$

From state transition, $w^{(m)}(H-1) = w^{(m-1)}(H-2)$, for $m = k_s, 2k_s$ and $3k_s$,

$$\tilde{J}_{H,H-2}(\Psi(H-2))$$
$$= \min_{u(H-2)}$$
$$\left[\begin{array}{l} E^{[\Psi(H-2)]}[R'(\Psi(H-2),u(H-2))] \\ +E^{[\Psi(H-2)]} \\ \quad \left[\gamma \cdot V_{N,N-1}\left(\begin{array}{l} X(H-1),\vec{b}(H-1), \\ w^{(k_s-1)}(H-2),w^{(2k_s-1)}(H-2),w^{(3k_s-1)}(H-2) \end{array} \right)\right] \end{array} \right].$$

Similar to the discussion in induction step, we can see that all $X(H-1), \vec{b}(H-1), w^{(k_s-1)}(H-2), w^{(2k_s-1)}(H-2)$ and $w^{(3k_s-1)}(H-2)$ are independent of $u(H-2)$. The reason for such independence is as follows: $X(H-1)$ and $\vec{b}(H-1)$ are uncontrollable variables as we discussed earlier. Hence, they will not depend on $u(H-2)$. The control history state variables, $w^{(k_s-1)}(H-2), w^{(2k_s-1)}(H-2)$ and $w^{(3k_s-1)}(H-2)$ are given in $\Psi(H-2)$. Hence, they are independent of $u(H-2)$. As a result of the above dependences, $V_{N,N-1}(H-1)$, which is a function of those independent state variables, would be independent of $u(H-2)$ as well. Hence, we can take $V_{N,N-1}(H-1)$ term out of our

minimization,

$$
\begin{aligned}
&\tilde{J}_{H,H-2}(\Psi(H-2)) \\
&= \min_{u(H-2)} \left[\; E^{[\Psi(H-2)]}[R'(\Psi(H-2),u(H-2))] \; \right] \\
&\quad + E^{[\Psi(H-2)]} \left[\gamma \cdot V_{N,N-1} \left(\begin{array}{l} X(H-1), \vec{b}(H-1), \\ w^{(k_s-1)}(H-2), w^{(2k_s-1)}(H-2), w^{(3k_s-1)}(H-2) \end{array} \right) \right] \\
&= \min_{u(H-2)} \left[\; E^{[\Psi(H-2)]}[R'(\Psi(H-2),u(H-2))] \; \right] \\
&\quad + E^{[\Psi(H-2)]} \left[\gamma \cdot V_{N,N-1}(\Upsilon(H-1)) \right].
\end{aligned}
$$

We can easily see that our minimization term is exactly the same as the equation of $V_{N,N-1}(\Upsilon)$; That is,

$$
\begin{aligned}
&\tilde{J}_{H,H-2}(\Psi(H-2)) \\
&= V_{N,N-1}(\Upsilon(H-2)) + E^{[\Psi(H-2)]} \left[\gamma \cdot V_{N,N-1}(\Upsilon(H-1)) \right].
\end{aligned}
$$

We get an optimal policy $\mu^*_{H-2}(\Psi(H-2))$ from

$$
\mu^*_{H-2}(\Psi(H-2)) = \widehat{\mu}_{N-1}(\Upsilon(H-2))
$$

Now we can get to step of $l = 3, 4, \ldots, k_s$, such that we can prove

$$
\tilde{J}_{H,H-l}(\Psi(H-l)) = V_{N,N-1}(\Upsilon(H-l)) + E^{[\Psi(H-l)]} \left[\sum_{i=1}^{l-1} \gamma^{l-i} \cdot V_{N,N-1}(\Upsilon(H-i)) \right],
$$

for $l = 3, 4, \ldots, k_s$.

To use an induction to prove this, we first assume

$$
\begin{aligned}
&\tilde{J}_{H,H-(l-1)}(\Psi(H-(l-1))) \\
&= V_{N,N-1}(\Upsilon(H-(l-1))) + E^{[\Psi(H-(l-1))]} \left[\sum_{i=1}^{l-2} \gamma^{(l-1)-i} \cdot V_{N,N-1}(\Upsilon(H-i)) \right].
\end{aligned}
$$

Then,

$$
\begin{aligned}
&\tilde{J}_{H,H-l}(\Psi(H-l)) \\
&= \min_{u(H-l)} E^{[\Psi(H-l)]}\left[R'(\Psi(H-l),u(H-l)) + \gamma \cdot \tilde{J}_{H,H-l+1}(\Psi(H-l+1))\right] \\
&= \min_{u(H-l)} E^{[\Psi(H-l)]}\left[\begin{array}{l} R'(\Psi(H-l),u(H-l)) + \gamma \cdot V_{N,N-1}(\Upsilon(H-l+1)) \\ +\gamma \cdot E^{[\Psi(H-l+1)]}\left[\sum_{i=1}^{l-2} \gamma^{l-1-i} \cdot V_{N,N-1}(\Upsilon(H-i))\right] \end{array}\right] \\
&= \min_{u(H-l)} \left[\begin{array}{l} E^{[\Psi(H-l)]}[R'(\Psi(H-l),u(H-l))] \\ +\gamma \cdot E^{[\Psi(H-l)]}[V_{N,N-1}(\Upsilon(H-l+1))] \\ +\gamma \cdot E^{[\Psi(H-l)]} E^{[\Psi(H-l+1)]}\left[\sum_{i=1}^{l-2} \gamma^{(l-1)-i} \cdot V_{N,N-1}(\Upsilon(H-i))\right] \end{array}\right] \\
&= \min_{u(H-l)} \left[\begin{array}{l} E^{[\Psi(H-l)]}[R'(\Psi(H-l),u(H-l))] \\ +\gamma \cdot E^{[\Psi(H-l)]}[V_{N,N-1}(\Upsilon(H-l+1))] \\ +\gamma \cdot E^{[\Psi(H-l)]}\left[\sum_{i=1}^{l-2} \gamma^{l-1-i} \cdot V_{N,N-1}(\Upsilon(H-i))\right] \end{array}\right] \\
&= \min_{u(H-l)} \left[\begin{array}{l} E^{[\Psi(H-l)]}[R'(\Psi(H-l),u(H-l))] \\ +\gamma \cdot E^{[\Psi(H-l)]}\left[\sum_{i=1}^{l-1} \gamma^{l-i} \cdot V_{N,N-1}(\Upsilon(H-i))\right] \end{array}\right].
\end{aligned}
$$

We could write out all state variables of $\Upsilon(H-i)$ and get

$$
\begin{aligned}
&\tilde{J}_{H,H-l}(\Psi(H-l)) \\
&= \min_{u(H-l)} \\
&\left[\begin{array}{l} E^{[\Psi(H-l)]}[R'(\Psi(H-l),u(H-l))] \\ +\gamma \cdot E^{[\Psi(H-l)]} \\ \qquad \left[\sum_{i=1}^{l-1} \gamma^{l-i} \cdot V_{N,N-1}\left(\begin{array}{l} X(H-i), \vec{b}(H-i), \\ w^{(k_s)}(H-i), w^{(2k_s)}(H-i), w^{(3k_s)}(H-i) \end{array}\right)\right] \end{array}\right].
\end{aligned}
$$

From state transition, $w^{(m)}(H-i) = w^{(m+1)}(H-i+1) = \cdots = w^{(m+i-l)}(H-l)$,

for $0 < i < l$ and $k_s \leq m \leq 3k_s$,

$$
\begin{aligned}
&\tilde{J}_{H,H-l}(\Psi(H-l)) \\
&= \min_{u(H-l)} \\
&\left[\begin{array}{l}
E^{[\Psi(H-l)]}[R'(\Psi(H-l), u(H-l))] \\
+\gamma \cdot E^{[\Psi(H-l)]} \\
\left[\begin{array}{l}
\sum_{i=1}^{l-1} \gamma^{l-i} \\
\qquad \cdot V_{N,N-1} \left(\begin{array}{l} X(H-i), \vec{b}(H-i), \\ w^{(k_s+i-l)}(H-l), w^{(2k_s+i-l)}(H-l), w^{(3k_s+i-l)}(H-l) \end{array} \right)
\end{array} \right]
\end{array} \right].
\end{aligned}
$$

Similar to the discussion in the induction step, all $X(H-i), \vec{b}(H-i), w^{(k_s+i-l)}(H-l), w^{(2k_s+i-l)}(H-l)$ and $w^{(3k_s+i-l)}(H-l)$ are independent of $u(H-l)$ (for $i < l$ or $H-i > H-l$). The reason for such independence is as follows: As we discussed earlier, $X(H-i)$ and $\vec{b}(H-i)$ are uncontrollable variables. Hence, they will not depend on $u(H-l)$. Moreover, the control history state variables, $w^{(k_s+i-l)}(H-l), w^{(2k_s+i-l)}(H-l)$ and $w^{(3k_s+i-l)}(H-l)$ are given in $\Psi(H-l)$. Hence, they are also independent of $u(H-l)$. As a result of the above dependencies, $V_{N,N-1}(H-i)$, which is a function of those independent state variables, would be independent of $u(H-l)$ as well. Then,

we can take all $V_{N,N-1}(H-i)$ terms out of our minimization,

$$\begin{aligned}
&\tilde{J}_{H,H-l}(\Psi(H-l)) \\
&= \min_{u(H-l)} \left[\; E^{[\Psi(H-l)]}[R'(\Psi(H-l),u(H-l))] \; \right] \\
&\quad +\gamma \cdot E^{[\Psi(H-l)]} \\
&\quad \left[\sum_{i=1}^{l-1} \gamma^{l-i} \cdot V_{N,N-1} \left(\begin{array}{l} X(H-i), \vec{b}(H-i), \\ w^{(k_s+i-l)}(H-l), w^{(2k_s+i-l)}(H-l), w^{(3k_s+i-l)}(H-l) \end{array} \right) \right] \\
&= \min_{u(H-l)} \left[\; E^{[\Psi(H-l)]}[R'(\Psi(H-l),u(H-l))] \; \right] \\
&\quad +\gamma \cdot E^{[\Psi(H-l)]} \left[\sum_{i=1}^{l-1} \gamma^{l-i} \cdot V_{N,N-1}(\Upsilon(H-i)) \right].
\end{aligned}$$

We can see that our minimization term is exactly the same as the equation of $V_{N,N-1}(\Upsilon)$. Hence, we get

$$\begin{aligned}
&\tilde{J}_{H,H-l}(\Psi(H-l)) \\
&= V_{N,N-1}(\Upsilon(H-l)) \quad + \gamma \cdot E^{[\Psi(H-l)]} \left[\sum_{i=1}^{l-1} \gamma^{l-i} \cdot V_{N,N-1}(\Upsilon(H-i)) \right],
\end{aligned}$$

and we get an optimal policy $\mu^*_{H-l}(\Psi(H-l))$ from

$$\mu^*_{H-l}(\Psi(H-l)) = \widehat{\mu}_{N-1}(\Upsilon(H-l)),$$

for $l = 3, 4, \ldots, k_s$.

This completed the proof of the Proposition.

QED.

Now we can reformulate the problem as follows: We will define new state variable,

$$\begin{aligned}
\tilde{Y}(k) &\stackrel{\triangle}{=} \Upsilon(kk_s) \\
&= \left[X(kk_s), \vec{b}(kk_s), w^{(k_s)}(kk_s), w^{(2k_s)}(kk_s), w^{(3k_s)}(kk_s) \right].
\end{aligned}$$

We have a new system with an infinite horizon cost,

$$\lim_{H\to\infty} E^{[\tilde{Y}(0)]}\left[\sum_{k=0}^{H-1}\gamma^k R'(\tilde{Y}(k),u(k))\right],$$

where

$$R'(\tilde{Y}(k),u(k)) = R_{\text{abc}}(\tilde{Y}(k),u(k)) + R_d(\tilde{Y}(k),u(k)),$$

$R_{\text{abc}}(\tilde{Y}(k),u(k))$ is the cost of high priority drops in main LSP , backup LSP and Preempt TCP traffic, respectively. $R_d(\tilde{Y}(k),u(k))$ is a cost of TCP congestion control mechanism, i.e.,

$$\begin{aligned}
&R_{\text{abc}}(\tilde{Y}(k),u(k))\\
&= E^{[\tilde{Y}(k),u(k)]}\\
&\left[\begin{array}{l}
D_1\cdot[1-u(k)]\cdot\frac{X(kk_s+1)}{X(kk_s+1)+(C-b_1(kk_s+d_1+1))}\cdot[X(kk_s+1)-b_1(kk_s+d_1+1)]^+\\
+u(k)\cdot\frac{(C-b_2(kk_s+d_2+1))}{X(kk_s+1)+(C-b_2(kk_s+d_2+1))}\cdot[X(kk_s+1)-b_2(kk_s+d_2+1)]^+\\
+D_2\cdot u(k)\cdot[b_2(kk_s+d_2+1)-[b_2(kk_s+d_2+1)-X(kk_s+1)]^+]
\end{array}\right]
\end{aligned}$$

and

$$R_d(\tilde{Y}(k),u(k)) = D_{\text{setup}}\cdot u(k)\cdot w^{(k_s)}(kk_s) + D_{\text{setup2}}\cdot u(k)\cdot w^{(2k_s)}(kk_s)\cdot w^{(3k_s)}(kk_s).$$

6.5.3 Conclusion on State Reduction

With our state reduction technique from previous sections, we successfully get a new value function $V^{\mu_0^*}(\tilde{Y}(0))$,

$$V^{\mu^*}(\tilde{Y}(0)) = \min_{\mu} E^{[\tilde{Y}(0)]}\left[\sum_{k=0}^{\infty}(\gamma^{k_s})^k \cdot \left[R'(\tilde{Y}(k), u(k))\right]\right],$$

with a new state variable vector $\tilde{\mathbf{Y}}(k)$,

$$\tilde{Y}(k) \;\; = \left[X(kk_s), \vec{b}(kk_s), w^{(k_s)}(kk_s), w^{(2k_s)}(kk_s), w^{(3k_s)}(kk_s)\right],$$

where

$$R'(\tilde{Y}(k), u(k)) = R_{\text{abc}}(\tilde{Y}(k), u(k)) + R_d(\tilde{Y}(k), u(k)),$$

$R_{\text{abc}}(\tilde{Y}(k), u(k))$ is the cost of high priority drops in main LSP , backup LSP and preempt TCP traffic, respectively. $R_d(\tilde{Y}(k), u(k))$ is a cost of TCP congestion control mechanism, i.e.,

$$\begin{aligned}
&R_{\text{abc}}(\tilde{Y}(k), u(k)) \\
&= E^{[\tilde{Y}(k), u(k)]} \\
&\quad \left[\begin{array}{l}
D_1 \cdot [1 - u(k)] \cdot \frac{X(kk_s+1)}{X(kk_s+1)+(C-b_1(kk_s+d_1+1))} \cdot [X(kk_s+1) - b_1(kk_s+d_1+1)]^+ \\
+u(k) \cdot \frac{(C-b_2(kk_s+d_2+1))}{X(kk_s+1)+(C-b_2(kk_s+d_2+1))} \cdot [X(kk_s+1) - b_2(kk_s+d_2+1)]^+ \\
+D_2 \cdot u(k) \cdot [b_2(kk_s+d_2+1) - [b_2(kk_s+d_2+1) - X(kk_s+1)]^+]
\end{array}\right]
\end{aligned}$$

and

$$R_d(\tilde{Y}(k), u(k)) = D_{\text{setup}} \cdot u(k) \cdot w^{(k_s)}(kk_s) + D_{\text{setup2}} \cdot u(k) \cdot w^{(2k_s)}(kk_s) \cdot w^{(3k_s)}(kk_s).$$

Chapter 7

Subobtimal Policy: Duplication Problem

We achieved a large reduction in state space by converting the partially observed MDP problem into the equvalent perfect information MDP problem in Chapter 6. In the chapter, we showed the duplication problem, which we originally formulated as a partially observed MDP problem, can be reduced to the perfect information MDP problem as far as control is concerned. That is, we will automatically get the optimal control policy for the partially observed MDP problem if we solve that perfect information MDP problem. Since the perfect information MDP has a smaller state space, it is wise to work on the perfect information MDP problem in this chapter, instead of working on the original partially observe MDP problem.

In spite of the large state reduction from the original partially observe MDP problem, we still encounter a well-known curse of dimensionality problem. With this problem, the number of states often grows exponentially with the number of state variables. This exponential growth can easily induce large state space. The resulting large state space could adversely create a difficulty in finding an optimal policy (even after we apply state reduction technique). Hence, it is nice to design policies that yield a good approximation to the optimal policy and can be easily calculated. In this chapter, we will propose some sub-optimal policies for the duplication problem. Then

we will evaluate and compare their performance using the Temporal Difference (TD) learning and the packet-level simulation.

7.1 Without TCP Consideration Approach

Recall from Section 6.5, the optimal policy $V^{\mu^*}(\tilde{Y}(0))$ for duplication problem (after our state reduction) can be obtained from minimizing the cost below:

$$E^{[\tilde{Y}(0)]}\left[\sum_{k=0}^{\infty}(\gamma^{k_s})^k \cdot \left[\begin{array}{l} R_{abc}(\tilde{Y}(k),u(k)) \\ +R_d(\tilde{Y}(k),u(k)) \end{array}\right]\right], \tag{7.1}$$

where $R_{\text{abc}}(\tilde{Y}(k),u(k))$ is the cost of the voice drops in the main LSP, the video drops the backup LSP, and the preempt TCP traffic, respectively. $R_d(\tilde{Y}(k),u(k))$ is the cost of the TCP congestion control mechanism, i.e.,

$$\begin{aligned} & R_{\text{abc}}(\tilde{Y}(k),u(k)) \\ & = E^{[\tilde{Y}(k),u(k)]} \\ & \left[\begin{array}{l} D_1\cdot[1-u(k)]\cdot\frac{X(kk_s+1)}{X(kk_s+1)+(C-b_1(kk_s+d_1+1))}\cdot[X(kk_s+1)-b_1(kk_s+d_1+1)]^+ \\ +u(k)\cdot\frac{(C-b_2(kk_s+d_2+1))}{X(kk_s+1)+(C-b_2(kk_s+d_2+1))}\cdot[X(kk_s+1)-b_2(kk_s+d_2+1)]^+ \\ +D_2\cdot u(k)\cdot[b_2(kk_s+d_2+1)-[b_2(kk_s+d_2+1)-X(kk_s+1)]^+] \end{array}\right] \end{aligned}$$

and

$$R_d(\tilde{Y}(k),u(k)) = D_{\text{setup}}\cdot u(k)\cdot w^{(k_s)}(kk_s) + D_{\text{setup2}}\cdot u(k)\cdot w^{(2k_s)}(kk_s)\cdot w^{(3k_s)}(kk_s),$$

with a state variable,

$$\tilde{Y}(k) \;\; = \left[X(kk_s),\vec{b}(kk_s),w^{(k_s)}(kk_s),w^{(2k_s)}(kk_s),w^{(3k_s)}(kk_s)\right].$$

To simplify our equation, we let

$$\begin{aligned} h_1(X,b) &= D_1\cdot\tfrac{X}{X+(C-b)}\cdot[X-b]^+, \\ h_2(X,b) &= \tfrac{C-b}{X+(C-b)}\cdot[X-b]^+ + D_2\cdot[b-[b-X]^+]. \end{aligned}$$

Hence, we can rewrite $R_{\rm abc}(\tilde{Y}(k), u(k))$ as

$$\begin{aligned} & R_{\rm abc}(\tilde{Y}(k), u(k)) \\ & = E^{[\tilde{Y}(k), u(k)]} \left[\begin{array}{l} [1-u(k)] \cdot h_1(X(kk_s+1), b_1(kk_s+d_1+1)) \\ +u(k) \cdot h_2(X(kk_s+1), b_2(kk_s+d_2+1)) \end{array} \right]. \end{aligned}$$

In this section, we propose a suboptimal policy based on Certainty Equivalent Control (CEC). The CEC approach often performs well in practice and yields near-optimal policies[2]. Hence, the CEC approach could work well in our problem also.

Using the concept of the Certainty Equivalent Control, we fix all uncertain quantities (noise) and substitute them with their means. Hence, our CEC control can be obtained by minimizing the cost function

$$\left[\sum_{k=0}^{\infty} (\gamma^{k_s})^k \cdot \left[\begin{array}{l} R_{abc}^{\rm CEC}(\tilde{Y}(k), u(k)) \\ +R_d(\tilde{Y}(k), u(k)) \end{array} \right] \right],$$

where

$$\begin{aligned} & R_{\rm abc}^{\rm CEC}(\tilde{Y}(k), u(k)) \\ & = [1-u(k)] \cdot h_1(E^{[X(kk_s)]}[X(kk_s+1)], E^{[b_1(kk_s)]}[b_1(kk_s+d_1+1)]) \\ & \quad +u(k) \cdot h_2(E^{[X(kk_s)]}[X(kk_s+1)], E^{[b_2(kk_s)]}[b_2(kk_s+d_2+1)]). \end{aligned}$$

We perform a further approximation by ignoring the cost associated with the TCP congestion control mechanism $R_d(\tilde{Y}(k), u(k))$. We call this control policy as a NoTCP control. This NoTCP control minimize the cost below

$$\sum_{k=0}^{\infty} (\gamma^{k_s})^k \cdot \left[\; R_{\rm abc}^{\rm CEC}(\tilde{Y}(k), u(k)) \; \right].$$

Remind you that the $R_d(\cdot, \cdot)$ terms, in which we ignored, are less importance than the other terms in the cost function. Hence, this new cost function of NoTCP control is

likely to be a good approximation of the cost function of the previous CEC control. As we already discuss, CEC often performs well in practice and yields near-optimal policies. Hence, the NoTCP control, which is close to the CEC control, is likely to perform well also.

So far, we have a new dynamic programming problem with the cost function in Equation (7.2).

$$V^{\mu^{\text{notcp}}}(\tilde{Y}(0)) = \min_{\mu} \sum_{k=0}^{\infty} (\gamma^{k_s})^k \cdot \left[\; R_{\text{abc}}^{\text{CEC}}(\tilde{Y}(k), \mu(\tilde{Y}(k))) \; \right] \tag{7.2}$$

Next, we find the optimal policy μ^{notcp} to this new problem. This policy μ^{notcp} will serve as an approximation to the optimal policy μ^* of our original duplication problem. To find the policy μ^{notcp}, we first claim that this new DP problem can be simplified to a simple step-by-step minimization problem. The step-by-step minimization problem is a problem in which its optimal policy depends only on the current state, $\tilde{Y}(0)$. Note that an optimal policy of a normal DP problem depends on both current state and future states, $\tilde{Y}(k), k > 0$. This claim eases the calculation of optimal policy. As a result, we can show a closed-form solution of this optimal policy after the claim.

Lemma 7.1.1. *The MDP problem with the cost function $V^{\mu^{\text{notcp}}}(\tilde{Y}(0))$ is a step-by-step minimization problem, i.e., a problem in which its optimal policy depends only on a current state. Hence, the policy $\mu^{\text{notcp}}(\tilde{Y}(k))$ can be obtained from*

$$\begin{aligned}
&\mu^{\text{notcp}}(\tilde{Y}(k)) \\
&= \text{argmin}_{u(k)\in\{0,1\}} \left[R_{\text{abc}}^{CEC}(\tilde{Y}(k), u(k)) \right] \\
&= \text{argmin}_{u(k)\in\{0,1\}} \\
&\quad \left[\begin{array}{c} [1-u(k)] \cdot h_1(E^{[X(kk_s)]}[X(kk_s+1)], E^{[b_1(kk_s)]}[b_1(kk_s+d_1+1)]) \\ +u(k) \cdot h_2(E^{[X(kk_s)]}[X(kk_s+1)], E^{[b_2(kk_s)]}[b_2(kk_s+d_2+1)]) \end{array} \right].
\end{aligned}$$

In order to easily understand the behavior of our control, the control policy $\mu^{\text{notcp}}(\tilde{Y}(k))$ in Lemma 7.1.1 can be alternatively written as follows:

$$\mu^{\text{notcp}}(\tilde{Y}(k)) = \begin{cases} 0 & \text{if } C_{\text{nodup}}(k) < C_{\text{dup}}(k) \\ 1 & \text{if } C_{\text{nodup}}(k) > C_{\text{dup}}(k), \end{cases}$$

where

$$\begin{aligned} C_{\text{nodup}}(k) = &\quad h_1(E^{[X(kk_s)]}[X(kk_s+1)], E^{[b_1(kk_s)]}[b_1(kk_s+d_1+1)]), \\ C_{\text{dup}}(k) = &\quad h_2(E^{[X(kk_s)]}[X(kk_s+1)], E^{[b_2(kk_s)]}[b_2(kk_s+d_2+1)]). \end{aligned}$$

From the above equation, $C_{\text{nodup}}(k)$ is the occurring cost when there is no duplication in time slot k, while $C_{\text{dup}}(k)$ is an occurring cost when there is a duplication in time slot k. From another point of view,, $C_{\text{nodup}}(k)$ is the cost that the voice traffic will suffer from the drops in the main LSP. On the other hand, $C_{\text{nodup}}(k)$ is the cost that the video traffic in the backup LSP will suffer from the drops and the TCP traffic will suffer from the loss of their bandwidth.

We could see that a closed-form solution for μ^{notcp} is computable if we can analytically calculate $E^{[X(kk_s)]}[X(kk_s+1)]$, $E^{[b_2(kk_s)]}[b_2(kk_s+d_2+1)]$ and $E^{[b_1(kk_s)]}[b_1(kk_s+d_1+1)]$.

Lemma 7.1.2. *Given a current voice state $X(k)$, the expected value of the future voice state $E^{[X(k)]}[X(k+\Delta k)]$ can be obtained from*

$$E^{[X(k)]}[X(k+\Delta k)] \;=\; \tfrac{\lambda}{\mu} + \left(X(k) - \tfrac{\lambda}{\mu}\right) e^{-\mu \Delta k T},$$

where T is the time-step length. λ is the arrival rate and $1/\mu$ is the mean duration time of the voice calls.

Lemma 7.1.3. *Given a current video state $b(k)$, the expected value of the future video state $E^{[b(k)]}[b(k+\Delta k)]$ can be obtained from*

$$E^{[b(k)]}[b(k+\Delta k)] \;=\; C - A \cdot \tfrac{M\alpha}{\alpha+\beta} + \left(b(k) - (C - A \cdot \tfrac{M\alpha}{\alpha+\beta})\right) e^{-(\alpha+\beta)\Delta k T},$$

where T is the time-step length and C is the link capacity. A, α, β and M are the parameters in the video model.

With Lemma 7.1.2 and Lemma 7.1.3, we get the *closed-form solution* for our sub-optimal control. We called this policy as "NoTCP" policy to indicate the absence of the terms associated with the TCP congestion control.

Proof Lemma 7.1.1

The intuition behind Lemma 7.1.1 is that X and $\vec{b}$ are uncontrollable. In other words, the amount of voice traffic X and the service rate (from video traffic) $\vec{b}$ do not depend on our duplication decision.[1] Moreover, $R_{abc}^{CEC}(\tilde{Y}(k), u(k))$, depends only on $X(k)$, $\vec{b}(k)$ and $u(k)$. (It does not depend on $w^{(k_s)}(k)$, $w^{(2k_s)}(k)$ and $w^{(3k_s)}(k)$ in our state variable $\tilde{Y}(k)$.) Hence, $R_{abc}^{CEC}(\tilde{Y}(k), u(k))$ is independent of any other control decisions except the control decision $u(k)$.

Recall that we have our value function $V^{\mu^{\text{notcp}}}(\tilde{Y}(0))$ of our NoTCP control as shown in Equation (7.2). From that value function, we can get the optimal policy μ^{notcp} from

$$\mu^{\text{notcp}} = \text{argmin}_{u(k)} \left[\; R_{\text{abc}}^{CEC}(\tilde{Y}(k), u(k)) \; + (\gamma^{k_s}) \cdot V^{\mu^{\text{notcp}}}(\tilde{Y}(k+1)) \right], \qquad (7.3)$$

where

$$V^{\mu^{\text{notcp}}}(\tilde{Y}(k+1)) = \min_{u(k+1), u(k+2), \ldots} \sum_{j=k+1}^{\infty} (\gamma^{k_s})^j \cdot \left[\; R_{\text{abc}}^{\text{CEC}}(\tilde{Y}(j), u(j)) \; \right].$$

Note that $V^{\mu^{\text{notcp}}}(\tilde{Y}(k+1))$, which is in the second minimizing term of Equation (7.3), consists of $R_{\text{abc}}^{CEC}(\tilde{Y}(j), u(j))$,for $j > k$. From the independence that we discussed above, this $V^{\mu^{\text{notcp}}}(\tilde{Y}(k+1))$ will not depend on $u(k)$. Hence, we can easily see that the policy μ^{notcp} can be obtained by minimizing only $R_{\text{abc}}^{CEC}(\tilde{Y}(k), u(k))$.

[1] Recall that the duplication decision is our control decision.

Now we can prove the lemma in detail as follows: From Equation (7.2), $V^{\mu^{\text{notcp}}}(\tilde{Y}(0))$ can be obtained from minimizing the following cost function

$$\sum_{k=0}^{\infty}(\gamma^{k_s})^k \cdot \left[\; R_{abc}^{CEC}(\tilde{Y}(k), u(k)) \; \right]$$
$$= \sum_{k=0}^{\infty}(\gamma^{k_s})^k \cdot \left[\begin{array}{l} [1-u(k)] \cdot h_1(E^{[X(kk_s)]}[X(kk_s+1)], E^{[b_1(kk_s)]}[b_1(kk_s+d_1+1)]) \\ +u(k) \cdot h_2(E^{[X(kk_s)]}[X(kk_s+1)], E^{[b_2(kk_s)]}[b_2(kk_s+d_2+1)]) \end{array} \right].$$

Notice that all the terms above do not depend on $w^{(k_s)}(k), w^{(2k_s)}(k), w^{(3k_s)}(k)$ components in $\tilde{Y}(k)$. Hence, $V^{\mu^{\text{notcp}}}(\tilde{Y}(0))$ will become $V^{\mu^{\text{notcp}}}(X(0), \vec{b}(0))$. Moreover, we can write $V^{\mu^{\text{notcp}}}(X(0), \vec{b}(0))$ in a recursive form as

$$V^{\mu^{\text{notcp}}}(X(kk_s), \vec{b}(kk_s))$$
$$= \min_{u(k)\in\{0,1\}} \left[\begin{array}{l} [1-u(k)] \cdot h_1(E^{[X(kk_s)]}[X(kk_s+1)], E^{[b_1(kk_s)]}[b_1(kk_s+d_1+1)]) \\ +u(k) \cdot h_2(E^{[X(kk_s)]}[X(kk_s+1)], E^{[b_2(kk_s)]}[b_2(kk_s+d_2+1)]) \\ +(\gamma^{k_s}) \cdot E^{[X(kk_s),\vec{b}(kk_s))]}\left[V^{\mu^{\text{notcp}}}(X((k+1)k_s), \vec{b}((k+1)k_s))\right] \end{array} \right].$$

Since $X(k)$ and $\vec{b}(k)$ are the uncontrollable variables, the term $E^{[X(k),\vec{b}(k))]}\left[V^{\mu^{\text{notcp}}}(X(k+1), \vec{b}(k+1))\right]$ does not depend on the control decision $u(k)$. As a result, we can take it out of the minimization terms

$$V^{\mu^{\text{notcp}}}(X(kk_s), \vec{b}(kk_s))$$
$$= \min_{u(k)\in\{0,1\}} \left[\begin{array}{l} [1-u(k)] \cdot h_1(E^{[X(kk_s)]}[X(kk_s+1)], E^{[b_1(kk_s)]}[b_1(kk_s+d_1+1)]) \\ +u(k) \cdot h_2(E^{[X(kk_s)]}[X(kk_s+1)], E^{[b_2(kk_s)]}[b_2(kk_s+d_2+1)]) \end{array} \right]$$
$$+(\gamma^{k_s}) \cdot E^{[X(kk_s),\vec{b}(kk_s))]}\left[V^{\mu^{\text{notcp}}}(X((k+1)k_s), \vec{b}((k+1)k_s))\right]$$

Similar to the above derivation, we can get the policy μ^{notcp} from

$$\begin{aligned}
&\mu^{\text{notcp}}(\tilde{Y}(k)) \\
&= \text{argmin}_{u(k)\in\{0,1\}} \\
&\qquad \left[\begin{array}{l} [1-u(k)] \cdot h_1(E^{[X(kk_s)]}[X(kk_s+1)], E^{[b_1(kk_s)]}[b_1(kk_s+d_1+1)]) \\ +u(k) \cdot h_2(E^{[X(kk_s)]}[X(kk_s+1)], E^{[b_2(kk_s)]}[b_2(kk_s+d_2+1)]) \\ +(\gamma^{k_s}) \cdot E^{[X(kk_s),\vec{b}(kk_s))]}\left[V^{\mu^{\text{notcp}}}(X((k+1)k_s), \vec{b}((k+1)k_s))\right] \end{array} \right].
\end{aligned}$$

Again, we could see that the last term does not depend on $u(k)$. Hence,

$$\begin{aligned}
&\mu^{\text{notcp}}(\tilde{Y}(k)) \\
&= \text{argmin}_{u(k)\in\{0,1\}} \\
&\qquad \left[\begin{array}{l} [1-u(k)] \cdot h_1(E^{[X(kk_s)]}[X(kk_s+1)], E^{[b_1(kk_s)]}[b_1(kk_s+d_1+1)]) \\ +u(k) \cdot h_2(E^{[X(kk_s)]}[X(kk_s+1)], E^{[b_2(kk_s)]}[b_2(kk_s+d_2+1)]) \end{array} \right].
\end{aligned}$$

Or,

$$\begin{aligned}
&\mu^{\text{notcp}}(\tilde{Y}(k)) \\
&= \text{argmin}_{u(k)\in\{0,1\}} \left[R^{\text{CEC}}_{\text{abc}}(\tilde{Y}(k), u(k)) \right].
\end{aligned}$$

This is clearly an minimization problem that depends only on the current state variable $\tilde{Y}(k)$.

QED.

Proof Lemma 7.1.2

Let $x(t)$ be a continuous-time version of $X(k)$, i.e., $X(k) = x(kT)$, where T is the length of our time-step. From Section 2.2, this $x(t)$ is the continuous-time Markov chain. By letting $M_x(t) = E[x(t)|x(0) = i]$, we can derive a differential equation satisfied by $M_x(t)$.

To begin, for a given $x(t)$,

$$x(t+h) = \begin{cases} x(t)+1 & \text{with probability } \lambda h + o(h) \\ x(t)-1 & \text{with probability } x(t)\mu h + o(h) \\ x(t) & \text{with probability } 1-[\lambda - \mu x(t)]h + o(h) \end{cases}$$

Their expectations yields

$$E[x(t+h)|x(t)] = x(t) + [\lambda - \mu x(t)]\, h + o(h).$$

Taking expectations once again (with respected to $x(0)$) yields

$$M_x(t+h) = M_x(t) + [\lambda - \mu M_x(t)]\, h + o(h).$$

or,

$$\frac{M_x(t+h) - M_x(t)}{h} = \lambda - \mu M_x(t) + \frac{o(h)}{h}.$$

Letting $h \to 0$ gives

$$M_x'(t) = \lambda - \mu M_x(t).$$

We can solve this differential equation and obtain,

$$M_x(t) = \frac{\lambda}{\mu} + K \cdot e^{-\mu t}.$$

Since $M_x(0) = \frac{\lambda}{\mu} + K = i$ and $K = i - \frac{\lambda}{\mu}$, we have that

$$M_x(t) = \frac{\lambda}{\mu} + \left(i - \frac{\lambda}{\mu}\right) e^{-\mu t}.$$

Recall that $X(k) = x(kT)$. Hence,

$$E[X(k+\Delta k)|X(k)] \;\; = \;\; \tfrac{\lambda}{\mu} + \left(X(k) - \tfrac{\lambda}{\mu}\right) e^{-\mu \Delta k T} \;\; .$$

QED.

Proof Lemma 7.1.3

Let define $v(k)$ be an amount of video traffic, where $v(k) \triangleq C - b(k)$ and C is the link capacity. Recall that $b(k)$ is the service rate from video traffic, i.e., the available bandwidth after we subtract the video traffic utilization.[2] Like the proof of Lemma 5.1.1, we also let $y(t)$ be a scaled continous-time version of $v(k)$, i.e., $v(k) = A \cdot y(kT)$, where T is the length of our time-step. Recall from Section 2.1, this $y(t)$ is a continuous-time Markov chain with a quantization step rate of A bps. Letting $M_y(t) = E[y(t)|y(0) = i]$, we will derive a differential equation satisfied by $M_y(t)$. To begin, note that, given $y(t)$:

$$y(t+h) = \begin{cases} y(t)+1 & \text{with probability } [M - y(t)]\,\alpha h + o(h) \\ y(t)-1 & \text{with probability } y(t)\beta h + o(h) \\ y(t) & \text{with probability } 1 - [M\alpha + (\beta - \alpha)y(t)]h + o(h) \end{cases}$$

Their expectations yields

$$E[y(t+h)|y(t)] = y(t) + [M - y(t)]\,\alpha h - y(t)\beta h + o(h).$$

Taking expectations once again (with respected to $y(0)$ yields

$$M_y(t+h) = M_y(t) + M\alpha h - (\alpha + \beta)y(t)h + o(h).$$

or,

$$\frac{M_y(t+h) - M_y(t)}{h} = M\alpha - (\alpha + \beta)y(t) + \frac{o(h)}{h}.$$

Letting $h \to 0$ gives

$$M_y'(t) = M\alpha - (\alpha + \beta)M_y(t).$$

[2]Instead of working directly on $b(k)$, we define $v(k)$ to be the complementary part of $b(k)$. Since v(k) comes from the original video model that we describe in Section 2.1, it is easier to work on compared to $b(k)$.

We can solve this differential equation and get,

$$M_y(t) = \frac{M\alpha}{\alpha+\beta} + Ke^{-(\alpha+\beta)t}.$$

Since $M_y(0) = \frac{M\alpha}{\alpha+\beta} + K = i$, $K = i - \frac{M\alpha}{\alpha+\beta}$, we obtain that

$$M_y(t) = \frac{M\alpha}{\alpha+\beta} + \left(i - \frac{M\alpha}{\alpha+\beta}\right) e^{-(\alpha+\beta)t}.$$

Recall that $v(k) = A \cdot y(kT)$. Hence,

$$E[v(k+\Delta k)|v(k)] \;=\; A \cdot \tfrac{M\alpha}{\alpha+\beta} + \left(v(k) - A \cdot \tfrac{M\alpha}{\alpha+\beta}\right) e^{-(\alpha+\beta)\Delta kT}.$$

Also $b(k) = C - v(k)$. Hence,

$$E[C - b(k+\Delta k)|C - b(k)] \;=\; A \cdot \tfrac{M\alpha}{\alpha+\beta} + \left(C - b(k) - A \cdot \tfrac{M\alpha}{\alpha+\beta}\right) e^{-(\alpha+\beta)\Delta kT}.$$

Then,

$$\begin{aligned} E[b(k+\Delta k)|b(k)] &= C - A \cdot \tfrac{M\alpha}{\alpha+\beta} - \left(C - b(k) - A \cdot \tfrac{M\alpha}{\alpha+\beta}\right) e^{-(\alpha+\beta)\Delta kT} \\ &= C - A \cdot \tfrac{M\alpha}{\alpha+\beta} + \left(b(k) - (C - A \cdot \tfrac{M\alpha}{\alpha+\beta})\right) e^{-(\alpha+\beta)\Delta kT}. \end{aligned}$$

QED.

7.2 Simulated Lookahead Approach

As we mentioned in Section 5.2, the limited lookahead is a well-known technique in the dynamic programming to approximate an optimal solution. We can briefly review the limited lookahead as follows: The optimal action u^* in our duplication problem can be found using Equation (7.4),

$$u^* = \text{argmin}_u E^{[s]}\left[R(s,u,s') + \gamma_{k_s} \cdot V^{\mu^*}(s')\right], \tag{7.4}$$

where s is a generalized state variable and u is a control action. s' is a generalized state variable that represents the next state after applying the action u to the current state s. We use this generalized state variable s instead of the real state variable $\tilde{Y}(k)$ to draw a reader's attention to the algorithm and the explanation. With these generalized variables, our stage cost will be $R(s, u, s')$. [3]

In many cases, $V^{\mu^*}(s)$ is difficult to obtain. The limited lookahead is introduced to ease that difficulty. The limited lookahead uses some approximated function $\tilde{V}(s)$ to approximate the optimal value function, $V^{\mu^*}(s)$ in its algorithm. So we get the limited lookahead policy, u_l, as shown in Equation (7.5). Given a good approximation of the optimal value function, the Limited lookahead can achieve a nearly optimal solution [2].

$$u_r = \text{argmin}_u E^{[s]}\left[R(s, u, s') + \gamma_{k_s} \cdot \tilde{V}(s')\right]. \tag{7.5}$$

We will use $V^{\mu^{\text{notcp}}}(s')$ developed in Section 7.1 as our approximated value function $\tilde{V}(s')$. As we discussed earlier, the NoTCP policy could provide a good approximation to optimal policy. Moreover, we could get a closed-form solution of its value function. Hence, the value function $V^{\mu^{\text{notcp}}}(s')$ seem to be a good choice as our approximated value function.

As we already discussed in Section 5.2, the simulated lookahead is introduced to

[3] Recall that The stage cost $R(s, u, s')$ in our MDP problem is

$$\left[\begin{array}{l} D_1 \cdot [1 - u(k)] \cdot \frac{X(kk_s+1)}{X(kk_s+1)+(C-b_1(kk_s+d_1+1))} \cdot [X(kk_s+1) - b_1(kk_s+d_1+1)]^+ \\ +u(k) \cdot \frac{(C-b_2(kk_s+d_2+1))}{X(kk_s+1)+(C-b_2(kk_s+d_2+1))} \cdot [X(kk_s+1) - b_2(kk_s+d_2+1)]^+ \\ +D_2 \cdot u(k) \cdot [b_2(kk_s+d_2+1) - [b_2(kk_s+d_2+1) - X(kk_s+1)]^+] \\ +D_{\text{setup}} \cdot u(k) \cdot w^{(k_s)}(kk_s) + D_{\text{setup2}} \cdot u(k) \cdot w^{(2k_s)}(kk_s) \cdot w^{(3k_s)}(kk_s) \end{array}\right].$$

cope with state explosion problem. It will use a simulation to approximate the expectation $E^{[s]}[\cdot]$ in Equation (7.5). Note that the direct calculation of that expectation uses $P[s'|s,u]$ (the transition probability of the next state s' given the current state s and the action u), which we have to explicitly calculate $P[s'|s,u]$ for every s' in the state space. If the state space is large, the calculation can be tedious. Hence, we will use the simulation to approximate the expectation $E^{[s]}[\cdot]$.

We have two simulation approaches. The first approach is Monte-Carlo lookahead and the second approach is Iterative lookahead.

7.2.1 Monte-Carlo Lookahead

In this approach, we use the Monte-Carlo simulation to estimate the expectation in Equation (7.5). In our Monte-Carlo simulation, we will simulate totally N_l of the next state s'_i, $i \in \{1, ..., N_l\}$, given the current state s. Then we will get the Monte-Carlo lookahead policy u_{MC} from Equation (7.6).

$$u_{\mathrm{MC}} = \mathrm{argmin}_u \frac{1}{N_l} \cdot \sum_{i=1}^{N_l} \Big[R(s,u,s') + \gamma_{k_s} \cdot V^{\mu^{\mathrm{NoTCP}}}(s'_i) \Big] \tag{7.6}$$

As mentioned before, the Monte-Carlo simulation provides a good estimate if the number of samples N_l is large. ($N_l \longrightarrow \infty$). We show the algorithm of our Monte-Carlo lookahead approach in Algorithm 6.

7.2.2 Iterative Lookahead

As already discuss in Section 5.2, the Monte-Carlo simulation provides a good estimate only if we can have a large number of samples. In other words, we need to simulate the samples many times to obtain a nice controller. However, we cannot simulate that

Monte(s)

$W(s,u) \leftarrow 0$, $u \in \{0,1\}$

for $i \in 1, ..., N_l$

for $u \in \{0,1\}$

- simulate s_i' from s and u

- $W(s,u) \leftarrow W(s,u) + \left[R(s,u,s_i') + \gamma^{k_s} \cdot V^{\mu^{\text{notcp}}}(s_i')\right]$

$u_{\text{MC}} \leftarrow \text{argmin}_u \frac{W(s,u)}{N_l}$

Algorithm 6: Simulated lookahead algorithm using Monte-Carlo simulation.

many samples in our fast timescale control. Time and processors capability limit a number of samples that we can generate. Hence, we might not achieve a good expectation estimate using Monte-Carlo lookahead approach.

With the same objective in Section 5.2, we introduced the Iterative lookahead approach to cope with the above problem of a small sample amount. Recall that we want to estimate $E^{[s]}\left[R(s,u,s') + \gamma_{k_s} V^{\mu^{\text{notcp}}}(s')\right]$. In this approach, we try to use a good starting estimate (estimation point). From that starting point, we keep update with simulation samples to get better estimate. Even with small number of simulation samples, we could obtain better policy than Monte-Carlo approach if that starting point is good enough. In other words, we use a good starting point instead of any random point to reduce convergence time. With a short convergence time, only a small number of samples is needed in order to achieve a good estimate. The same idea appears in many optimization technique. For example, a good starting point in the Newton's method and a good initial value function in the value iteration method in the dynamic programming technique.

We use $Q_{\mathrm{NoTCP}}(s,u)$ as our good starting point. In Q-learning terminology, this $Q_{\mathrm{NoTCP}}(s,u)$ is a Q-function such that we could get $V^{\mu^{\mathrm{notcp}}}(s)$ from, i.e.,

$$V^{\mu^{\mathrm{notcp}}}(s) = \min_u Q_{\mathrm{NoTCP}}(s,u),$$

and vice versa, we can get $Q_{\mathrm{NoTCP}}(s,u)$ from $V^{\mu^{\mathrm{notcp}}}(s)$,

$$Q_{\mathrm{NoTCP}}(s,u) = \ R^{\mathrm{CEC}}_{\mathrm{abc}}(s,u) + \gamma^{k_s} \cdot V^{\mu^{\mathrm{notcp}}}(s'),$$

where s' is obtained from applying u to current state variable s.

The algorithm of this Iterative lookahead approach is shown in Algorithm 7.

Iterative(s)

$Q(s,u) \leftarrow Q_{\mathrm{NoTCP}}(s,u)$, $u \in \{0,1\}$

for $i \in 1, ..., N_l$

for $u \in \{0,1\}$

- simulate s'_i from s and u

- $Q(s,u) \leftarrow Q(s,u) + \alpha \cdot \left[R(s,u,s'_i) + \gamma^{k_s} \cdot V^{\mu^{\mathrm{notcp}}}(s'_i) - Q(s,u)\right]$

$u_{\mathrm{IT}} \leftarrow \mathrm{argmin}_u Q(s,u)$

Note that α is a step-size in updating $Q(s,u)$

Algorithm 7: Simulated lookahead algorithm using Iterative simulation.

7.3 Policy Evaluation and Comparison

So far, we have proposed a number of sub-optimal policies. In this section, we will evaluate and compare the performance of the proposed policies. We will first evaluate them using TD(0), which is a learning method to estimate the value function of

policies. Then we will use a packet-level simulation to strengthen the evaluation results.

We will compare five different duplication controllers, which are the Nodup, Alldup, NoTCP, MONTE and IT controllers. Like the controllers in the migration problem, these controllers are the fast timescale controllers that acquire state information and make a migration decision every 50ms. These five controllers implement five policies, in which three of them are the policies designed earlier in this chapter. The first two policies are extreme policies that we want to compare our three designed policies with. The two controllers that implement the two extreme policies are as follows: The Nodup controller implements one extreme duplication policy that we do not send any duplicated packets at all. The Alldup controller implements another extreme duplication policy that we always send duplicated packets to the backup LSP.

Unlike the first two controllers, the next three controllers implement our compromise policy that send the duplicated packet intelligently. These policies have a frequency of duplication between the least frequent policy Nodup, and the most frequent policy Alldup. The NoTCP controller implements the NoTCP policy, the MONTE controller implements the Monte-Carlo lookahead policy, and the IT controller implements the Iterative lookahead policy. To get more detail on these policies, we will refer to Section 7.1 for NoTCP policy and Section 7.2 for Monte-Carlo Lookahead policy and Iterative Lookahead policy. We can summarize our controller in Table 7.1

7.3.1 Network Topology and Common Parameters

We evaluate each policy using a simple network topology as already shown in Figure 6.1. In our network, we provision one main LSP and one backup LSP for voice traffic.

Controller	Policy
Nodup	do not duplicate
Alldup	Always duplicate
NoTCP	NoTCP policy
MONTE	Monte-Carlo Lookahead Policy
IT	Iterative Lookahead Policy

Table 7.1: Controllers and their associated policies.

We choose $p_1 = 2$ and $p_2 = 1$, i.e., we setup the propagation delay from the ingress LSR to the bottleneck link in the main LSP and the backup LSP to be 2 timeslots (100ms) and 1 timeslot (50ms), respectively. Moreover, we also choose $q_1 = q_2 = 0$, i.e., we assume that there is no information delay from bottleneck links to ingress LSR. The voice traffic arrival is governed by Poisson distribution with mean λ. The voice call duration is exponentially distributed with the mean of $1/\mu$, which we fixed at 3 minutes. With this voice traffic setup, we get an average voice traffic rate at $\lambda * 180 * 0.064$ Mbps, given a 64Kbps constant bit rate voice call, as discussed in Section 2.2. The bottleneck links in both LSPs have statistically identical video traffic. Both video traffic streams have the same statistical parameters $\alpha = 0.05$ and $\beta = 0.05$. All links including the bottleneck links have capacity of 90Mbps, which can fit (90Mbps/64Kbps)=1406 voice calls. Moreover, the TCP traffic also shares the bottleneck link with video traffic and duplicated voice traffic in the backup LSP.

7.3.2 Using TD(0)

As we discussed in Section 5.3.2, TD(0) is a version of Temporal-Difference learning. We choose this TD(0) to evaluate our designed sub-optimal policy because it has a fast

convergence from the fact that it updates estimates based in part on other learned estimates (bootstrap). Moreover, it learns directly from raw experience without a model of the environment's dynamics, which in turn eases the estimation of the value function.

We will start our TD(0) evaluation with a discussion about the state aggregation. This state aggregation helps us cope with the large state space in our problem. Then, we will present an algorithm for our TD(0) with the state aggregation. Finally, we will show the result of the comparison.

State Aggregation

As described in Section 5.3.2, we need state aggregation to make our problem manageable. We will start discussing how to select proper state variables to be aggregated. Then we will specify an actual number of state aggregation in our TD(0) evaluation.

We will aggregate only voice component in our state space.[4] In general, the state space of voice component ($[0, C]$) is large, especially when we compare it with the state space of video component ($\mathbf{S_V}$). Recall that the voice component X represents a number of voice calls in the main LSP. It follows the fact that each voice call consumes relatively small bandwidth. Hence, a large number of voice calls can be filled in one LSP before reaching its capacity. In other words, C, which represents the total maximum number of voice calls in LSP, is large. So $[0, C]$, which is the state space of

[4]Recall that the state space, $\mathbf{S}$, is composed of three components. First is video traffic state, b_1 and b_2. $[b_1, b_2]$ takes value from $\mathbf{S_V}$, state space of video component. Second component is voice call traffic state, X. X take value from $[0, C]$, state space of voice component, where C is link capacity in unit of voice calls. The last component is control history variable $[w^{(k_s)}, w^{(2k_s)}, w^{(3k_s)}]$. $[w^{(k_s)}, w^{(2k_s)}, w^{(3k_s)}]$ takes value from $\{0, 1\}^3$. So our complete state space $\mathbf{S} = \mathbf{S_V} \times [0, C] \times \{0, 1\}^3$.

X, becomes large as well.

On the other hand, we would have a small state space of video component ($\mathbf{S_V}$). As discussed earlier, we can apply a state reduction technique, which is available for video traffic, e.g. the technique in [6] in the case that the state space is large. Hence, we will always assume that $\mathbf{S_V}$ is small and do not need state aggregation.

For simplicity, we will evenly aggregate the voice component of our state space. Let define $\hat{X}$ be an aggregated superstate variable. In this experiment, we have $C = 1406$. Hence, we will have 10 superstate, in which each superstates has around 140 states.

Algorithm

Algorithm 8 shows our off-line algorithm of our TD(0). To simplify our notation, we will use s instead of $\tilde{\mathbf{Y}}$ in this algorithm. The function $DupPolicy(s)$ represents the policies, in which we want to evaluate their performance including the Alldup, Nodup, NoTCP, MONTE and IT policies.

Results and Discussion

In the first experiment, we compare the performance of different policies when changing discount factor in the cost function but keeping the network setup fixed.

Figure 7.1 shows a mean of our value function from different duplication policies. Figure 7.2, which is a zoomed version of Figure 7.1, provides a better view to the gain for using our intelligent controllers. In the experiment, we fixed the average video traffic rate at 32Mbps and the average voice traffic rate at 43.8Mbps. We also fixed the number of steps (M) parameter in the video traffic at 10, i.e., $M = 10$. We chose other parameters in our cost function as follows: $D_1 = 0.5, D_2 = 0.07, D_{\text{setup}} = 0.3$ and $D_{\text{setup2}} = 1.0$. The left graph in the Figure 7.2 and Figure 7.1 represents the

TD(0)-dup[]

Initialize $V(.)$

Repeat [Outer Loop]

for every value of $\hat{s}$

$s \leftarrow$ uniform random on range of $\hat{s}$

Repeat [Inner Loop] {within episode}

$u \leftarrow \mathrm{DupPolicy}(s)$

$s' \leftarrow$ simulate next state from s and u

$\hat{s}' \leftarrow$ locate superstate from s'

$R_t \leftarrow$ get Reward from s, u, s'

$V(\hat{s}) \leftarrow V(\hat{s}) + \alpha \left[R_t + \gamma^{k_s} V(\hat{s}') - V(\hat{s})\right]$ } Update $V(.)$

$\hat{s} \leftarrow \hat{s}'$

$s \leftarrow s'$ } replace next state with current state

*Note that $\hat{s}$ is current superstate s is current real state u is current action

$\hat{s}'$ is next superstate s' is next real state α is updating step-size

$\mathrm{DupPolicy}(s)$ is a sub-optimal policy that we want to evaluate

Algorithm 8: The TD(0) algorithm with the state aggregation for the duplication problem.

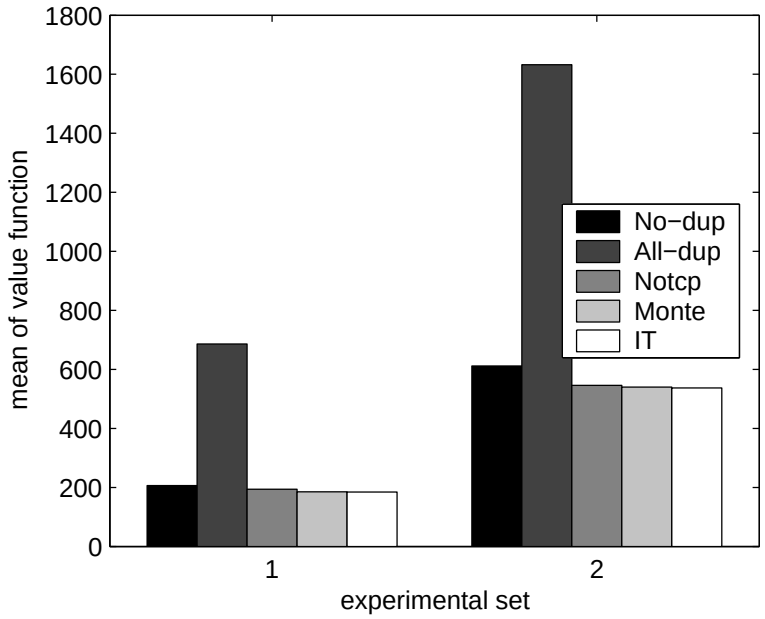


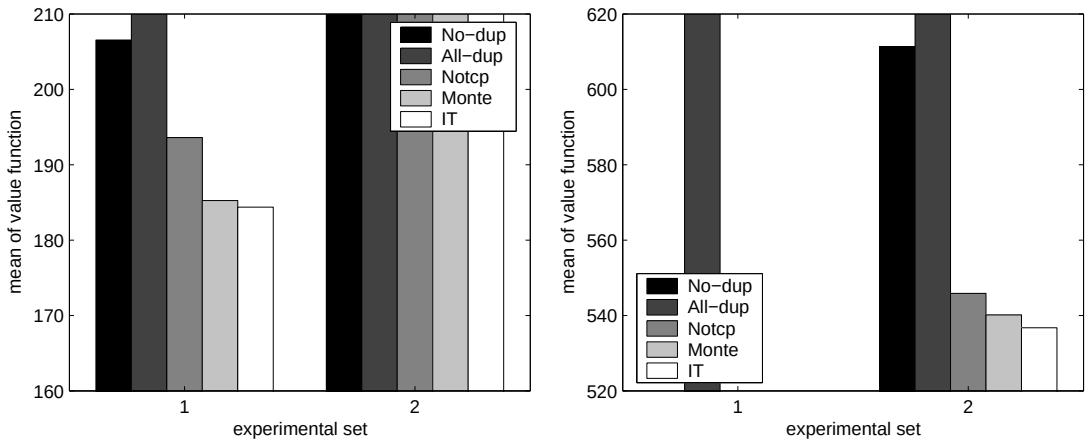
Figure 7.1: Using TD(0) to compare the means of the value functions.

performance comparison with the discount factor $\gamma^{k_s} = 0.7$. On the other hand, the left graph in both figures represents the performance comparison with the discount factor $\gamma^{k_s} = 0.9$.

We can see that the Alldup policy gives the worst performance. The reason comes from the fact that we let $D_1 < 1$, i.e. we consider drops of voice traffic (in main LSP) less important than drops of the video traffic in backup LSP. As we already mentioned before, we setup the D_1 parameter this way because we want to prevent a large number of video packet drops in the backup LSP. Without duplication mechanism, there is no significant drops in this video traffic.

We also see that the IT controller provides the best performance in both discount factor parameters.

The second experiment is an experiment designed to observe the performance of



(a) zoomed at set1

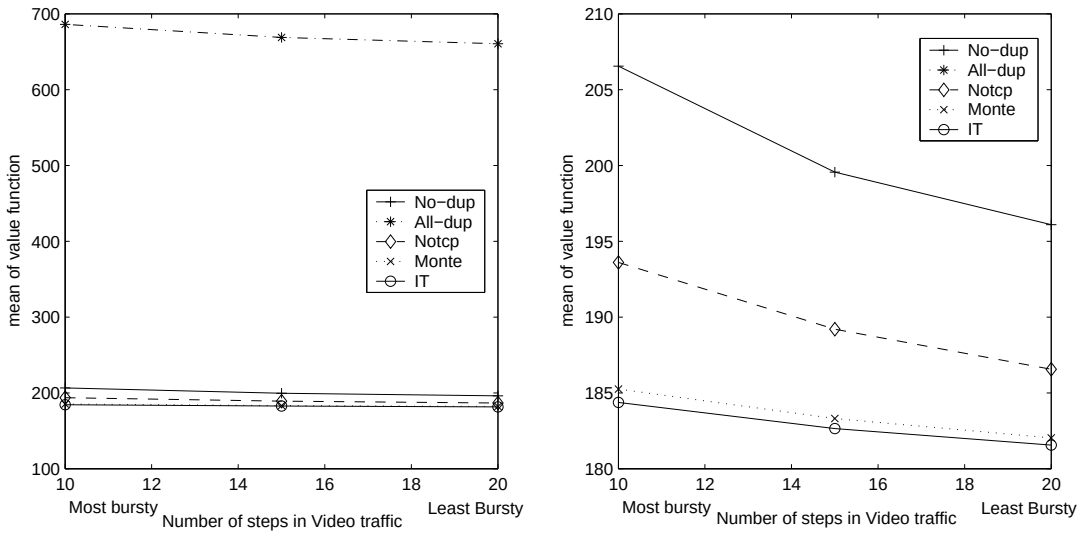
(b) zoomed at set2

Figure 7.2: Using TD(0) to compare the means of the value functions.

our controllers under a various bursty levels of video traffic. As we discussed eariler, we can vary the burstiness of video traffic by changing the number of steps (M). Furthermore, we consider the video traffic to be bursty if the number of steps (M) is small.

Figure 7.3 shows the result of this experiment. In the experiment, We varied the number of steps (M) parameter in the video traffic from 10 to 20. We fixed the average video traffic rate at 32Mbps and the average voice traffic rate at 43.8Mbps, which totally corresponds to the 84% of bottleneck link bandwidth. In addition, we chose all parameters in our cost function as follows;
$D_1 = 0.5, D_2 = 0.07, D_{\text{setup}} = 0.3, D_{\text{setup2}} = 1.0$ and the discount factor (γ^{k_s})$= 0.7$.

The result shows that all intelligent duplication policies (the NoTCP, MONTE and IT policies) always outperform the Nodup policy, which represents a case without any duplication mechanism. Moreover, they give greater improvement over Nodup policy with the increasing level of bursiness. Note that the duplication mechanism will be

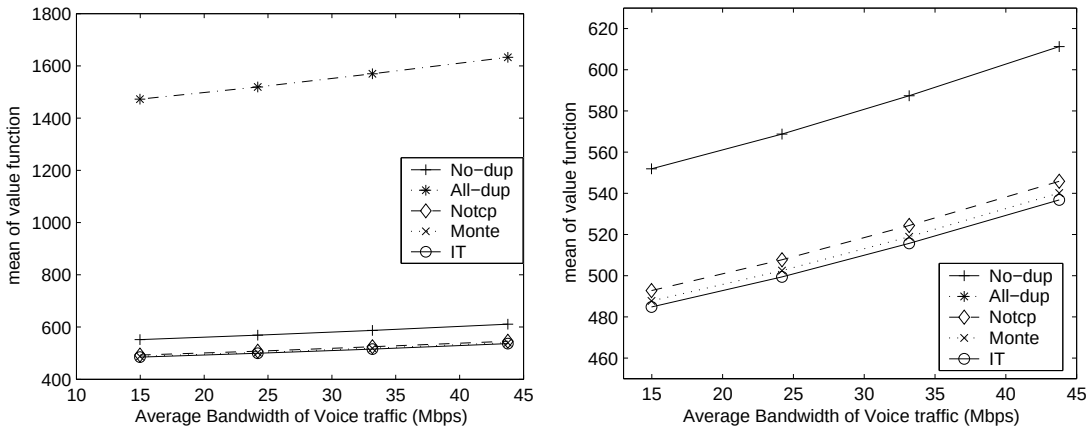


(a) unzoomed version (b) zoomed version

Figure 7.3: Relationship of the mean of the value function and the burstiness of the video traffic using TD(0).

most helpful in the situation when video amount is high in the main LSP while it is low in the backup LSP. The bursty condition promotes a chance of getting that situation. Hence, the duplication mechanism will have a higher chance to alleviate voice drops without significant video drops in backup LSP. As a result, our duplication mechanism provide a greater improvement under bursty condition. This result confirms that duplication mechanism is helpful for QoS improvement, especially under bursty condition.

In addition, the figure shows that the IT policy always performs best. For example, the IT policy gives 10.7% improvement over the Nodup policy in the most bursty network, while the NoTCP policy and the MONTE policy only give 6.2% and 10.2% improvement. Follow our discussion in Section7.2, both the MONTE and IT policies are the improvement of the NoTCP policy. Hence, they provide better results than the



(a) unzoomed version (b) zoomed version

Figure 7.4: Relationship of the mean of value function and the voice traffic amount using TD(0).

NoTCP policy. Compared to the MONTE policy, the IT policy gives a better result because of its good starting point as we already discussed in Section 7.2.2. This result strengthens our belief that our proposed Iterative lookahead Policy is a good solution for this fast timescale problem.

The third experiment is an experiment designed to study the performance of our controllers with different amounts of voice traffic. Hence, we will vary only an average voice traffic rate, while we keep others fixed. As you know, we can vary the voice rate by varying the arrival rate of the voice call.

Figure 7.4 shows the result of this experiment. In this experiment, we fixed the average video traffic rate at 32Mbps with the number of steps ($M = 10$). We varied average voice traffic rate from 15Mbps to 44Mbps, which corresponds to varying total traffic in the bottleneck link from 52 to 85 percents of the link capacity. Similar to the

second experiment, we chose all parameters in our cost function as follows: $D_1 = 0.5, D_2 = 0.07, D_{\text{setup}} = 0.3, D_{\text{setup2}} = 1.0$ and the discount factor (γ^{k_s})= 0.9.[5]

Again, the result shows that all intelligent duplication policies (the NoTCP, MONTE and IT policies) always give a superior performance compared to the Nodup policy. Moreover, they give slightly greater improvement over the Nodup policy when there is more voice traffic, i.e., there is a slightly larger gain when the average voice traffic rate is larger. As discussed earlier, the duplication mechanism will be most helpful in the situation when the video amount is high in the main LSP while it is low in the backup LSP. If the controller is good, it can identify the occurrence of this situation correctly and, in advance, duplicate the traffic to compensate the voice drops in the main LSP. When the average voice load increases, the successful compensation of the voice drops increases. Note that the compensation amount indicates the gain of the duplication control over the case without it (the case with the NoDup controller). Hence, the gain of the duplication control increases when the average voice load increases. As a result, our intelligent duplication policies give a increasing gain over Nodup policy when there is more voice traffic. This result further confirms our conclusion in the second experiment that duplication mechanism is helpful for QoS improvement under various network condition.

Among these three intelligent duplication policies, we again see that the IT policy always performs best. For example, the IT policy gives a 12.2% improvement over the Nodup policy under the heaviest traffic, while the NoTCP policy and the MONTE policy only give 10.6% and 11.6% improvement, respectively.[6] Along with the

[5]Note that the experiment result of $\gamma^{k_s} = 0.7$ and $\gamma^{k_s} = 0.9$ are similar. Hence, we decided to show the case of $\gamma^{k_s} = 0.7$ in the second experiment and the case of $\gamma^{k_s} = 0.9$ in the third experiment. With this decision, the reader could get a benefit of seeing a wide range of experiment results.

[6]Notice that the relative gain in this experiment, which have $\gamma^{k_s} = 0.9$, are different from the

conclusion in the second experiment, this result further strengthens our belief that our proposed Iterative lookahead policy is a good solution for this fast timescale problem.

7.3.3 Using Packet-Level Simulation

The packet-level simulation can help us understand the behavior of our controller in the real network. Note that the packet-level simulation allows us to simulate the interaction of packets and routers, which implemented our controller. Hence, it could give a meaningful evaluation, which is close to the evaluation in an actual network.

As we discussed in Section 5.3.3, we compute all policies online during the simulation. The computation is done in a flow-level (we designed our policy based on the arrival and departure of flows),as opposed to a packet-level of our simulation itself. We will monitor the amount of packets and convert the number into a units of flows. The controller will use that number of flows to obtain the amount of migration based on their underline policy. The controller will calculate the amount of migration online upon the received information.

Simulator implementation

We used ns-2 simulator[39] for our simulations. In addition to our modification done in Section 3.2 and Section 5.3.3, we added some new features to meet our control action in this duplication problem. We added an ability to duplicate the packets, which allows

relative gain in the second experiment, which have $\gamma^{k_s} = 0.7$. For example, the IT policy gives a 12.2% improvement in this experiment while give a 10.7% improvement in the second experiment. The main reason comes from the fact that the $V^{\mu}(\tilde{Y(0)})$ is calculated from $V^{\mu}(\tilde{Y(0)}) = E^{[\tilde{Y}(0)]}\left[\sum_{k=0}^{\infty}(\gamma^{k_s})^k \cdot \left[R'(\tilde{Y}(k), \mu(\tilde{Y}(k)))\right]\right]$. Hence $V^{\mu}(Y(0))$ will change when γ^{k_s} changes and so does the relative gain.

us to send the same information to the same destination with different paths. Moreover, we added a new packet receiver to be able to keep track the packet loss. This receiver can buffer packets up to 100 packets. Hence, it can cope with the out-of-order packets, which normally happen when the duplicated packets is sent on the LSPs with different propagation delay.

Simulation Setup and Parameters

We use a network topology and some common parameters as we stated in Section 7.3.1. With ns-2, we use a Reno version of TCP, which is already in the ns-2 package. In our simulation, we setup short-lived TCP traffic flows in the backup LSP. Each TCP flow sends small files with a uniformly distributed size of 20-40 Kbytes. Upon the end of each transmission, the TCP source will setup a new connection and begin to transmit a new file. Recall that the new connection setup will use a initial value for ITO. This small file transmission and the re-connection of flows capture a characteristic of short-lived TCP. These TCP flows has four distinct RTT value, 10ms, 30ms, 50ms and 70ms. There are 25 TCP flows that have the same RTT value each. Therefore, there are totally 100 TCP flows. As discussed in Section 2.3, the different value of RTT represents different locations of end-users, while a number of flows with same RTT value indicate the fact that each location can have multiple users.

Results and Discussion

We can get the performance measurements from monitoring the voice drops, the video drops in backup LSP, the TCP goodput and the total number of second ITO in TCP traffic.

Figure 7.5 shows the example how our control affects each of performance

measurements. In this experiment, we fixed the average video traffic rate at 38.4Mbps and the average voice traffic rate at 46Mbps. We also fixed the number of steps (M) parameter in video traffic at 10, i.e., $M = 10$. Moreover, we chose the parameters in our cost function as follow; $D_1 = 0.5, D_2 = 0.07, D_{\text{setup}} = 0.3, D_{\text{setup2}} = 1.0$ and the discount factor (γ^{k_s})$= 0.7$.

As we mention before, the study in [17] shows that the losses greater than 2% affect end-to-end quality of VoIP calls. Hence, we assumed that the quality of voice and video traffic is not good if their drops are greater than 2%. In the figure, the voice traffic has drops around 4% without the duplication mechanism (or the Nodup policy). At the same time, there is negligible video traffic drops in the backup LSP (, which we refer in the figure as "VDO#2 Drops"). If we naively duplicate voice packets all the time, we would have around 1.48% voice drops while we have unbearable 8.83% video drops in the backup LSP with the Alldup policy. However, we could intelligently duplicate voice packets as needed and get both the voice drops and video drops under 2% with either the MONTE policy or the IT policy. This example shows that we could use the duplication mechanism, which is a very simple control action, to improve quality of the voice traffic up to the acceptable quality level without causing the video traffic to be in the unacceptable quality level.

Moreover, the number of second ITO with the Alldup controller is significantly high compared to that with the other controllers. This indicates that we could get a long waiting time for internet users when we naively duplicate all the time. Recall that the approximation from the CEC control itself and the ignorance of the TCP congestion control cost drive the NoTCP policy away from the optimal solution. We can see that the MONTE and IT policies help improve the sub-optimality of the NoTCP policy. As shown the figure, the MONTE and IT policies can improve the NoTCP by gaining a

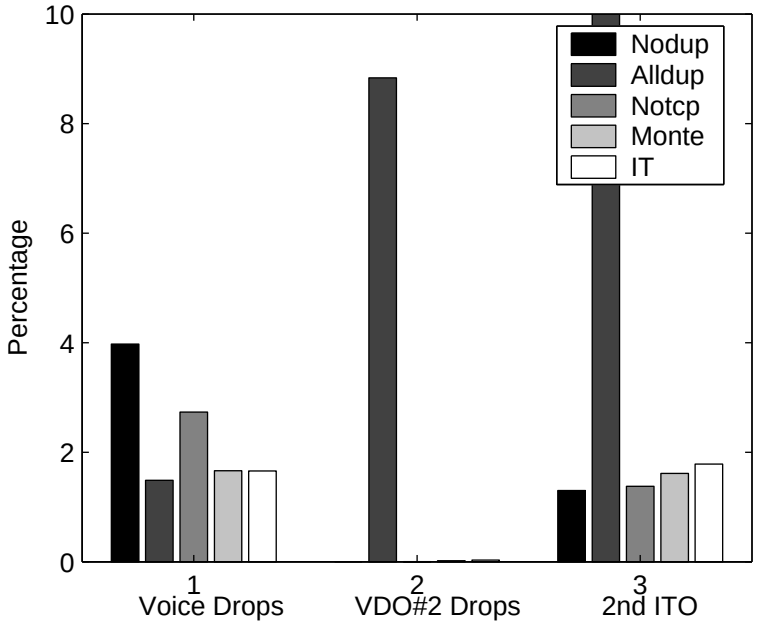


Figure 7.5: Using NS to compare the performance measurements.

significant reduction of voice drops with a small increment of the second ITO as its trade-off. Ones might expect a lower number of the second ITO with both the MONT and IT controllers compared to that with the NoTCP controller. However, the mistake of the NoTCP in voice drops is more important than its mistake in TCP control cost.

Like the TD(0) experiment setup in Section 7.3.2, we also compare our controllers' performance in various network environments. The first experiment is the experiment designed to observe the performance of our controllers under different amounts of voice traffic. The second experiment is the experiment designed to study the performance of our controllers over changes in the burstiness level of video traffic.

Figure 7.6 shows results in the first experiment. We varied the average voice traffic rate from 25Mbps to 52.3Mbps, which corresponds to the varing of the total traffic in the bottleneck link from 70 to 100 percent of the link capacity. We also kept the video

traffic fixed at 38.4Mbps with the number of steps $M = 10$. Again, We chose all parameters in our cost function as follow;
$D_1 = 0.5, D_2 = 0.07, D_{\text{setup}} = 0.3, D_{\text{setup2}} = 1.0$ and the discount factor (γ^{k_s})$= 0.7$.

Since this experiment is the performance comparison of our controller with different amount of voice traffic, it corresponds to the third experiment with TD(0) in Section 7.3.2. Like the experiment with TD(0), we also get the same conclusion that all intelligent duplication policies (the NoTCP, MONTE and IT policies) give a greater improvement over the Nodup policy when there is more voice traffic, i.e., there is a larger gain when average voice traffic rate is larger. We could see this conclusion from all graphs in Figure 7.6 as follows: As shown in Figure 7.6(a), the IT policy and the MONTE policy give an increasing improvement of voice drops over the Nodup policy. For example, they give only a 0.35% improvement of voice drops over the Nodup policy at a 25Mbps average voice traffic, while they give as high as a 2.8% improvement of voice drops over the Nodup policy at 52.3Mbps average voice traffic. On the other hand, our intelligent controller and Nodup controller give approximately the same performance using other performance measurements including video drops, TCP goodput and number of second ITO, shown in Figure 7.6(b),7.6(c) and 7.6(d), respectively. Recall that we consider the voice drops and the video drops more important than the TCP performance. Hence, we can conclude that all intelligent duplication policies (the NoTCP, MONTE and IT policies) give a greater improvement over the Nodup policy when there is more voice traffic. Again, this result further strengthen our belief that duplication mechanism is helpful to improve QoS, especially under high voice traffic load.

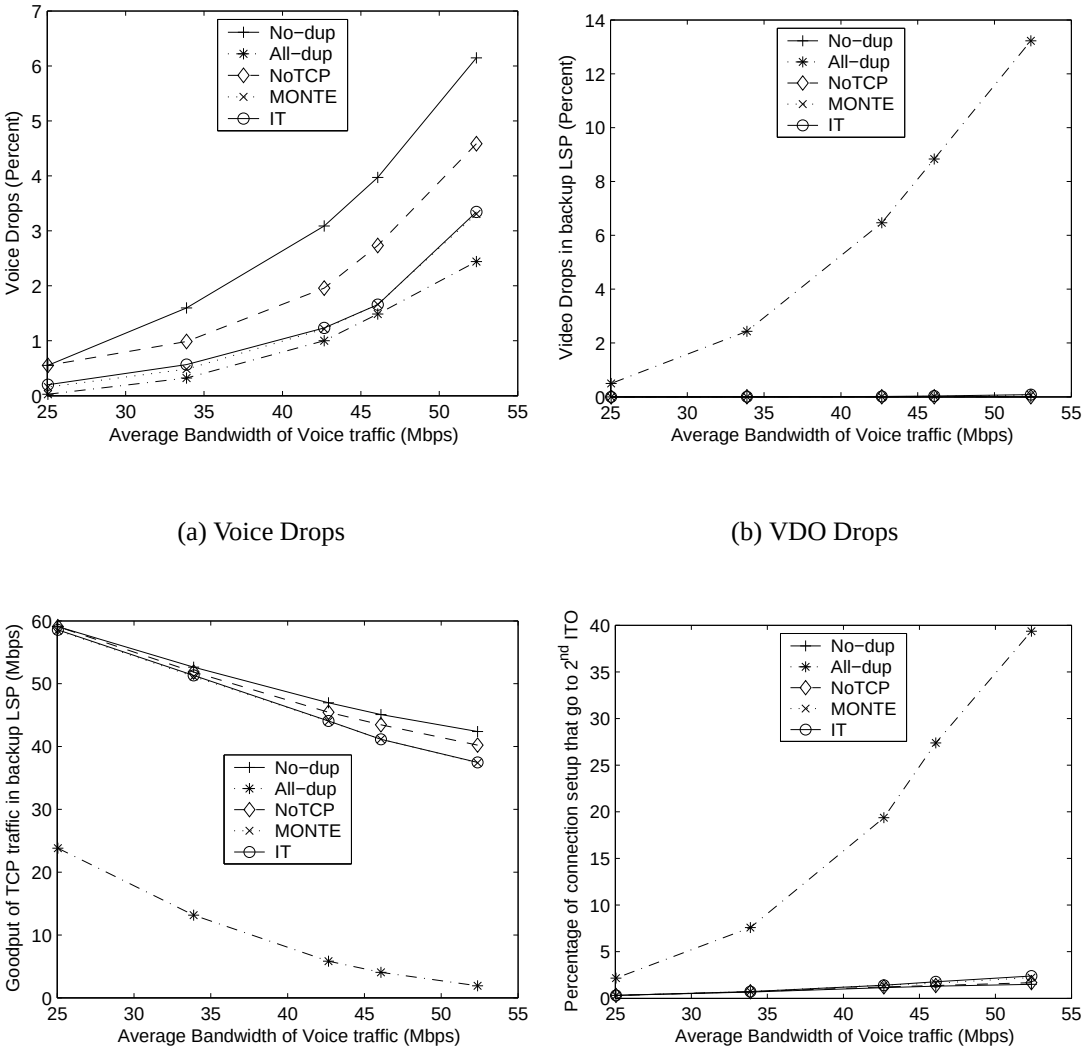
Again, we assume that the quality of voice and video traffic is not good if their drops are greater than 2%. We can quantify another advantage of duplication

mechanism from Figure 7.6(a). Without the duplication mechanism (the Nodup policy), we can accept the voice traffic up to 36Mbps before its quality is unacceptable. On the other hand, we can accept up to 46Mbps with the duplication mechanism using the IT policy. Hence, the existing network can accept 27% more voice traffic. In other words, we have an ability to increase the high priority traffic using the duplication mechanism. This ability could lead to a profit increase of the network provider.

Figure 7.7 shows the results of the second experiment. As we mentioned before, this experiment aim to compare the performance of our controllers over changes in the burstiness level of video traffic. We varied the number of steps (M) from 10 to 40. In addition, we fixed the average video traffic rate at 38.4Mbps and the average voice traffic rate at 42.7Mbps, which corresponds to 90%load.[7] Moreover, we fixed all parameters in our cost function as follows:
$D_1 = 0.5, D_2 = 0.07, D_{\text{setup}} = 0.3, D_{\text{setup2}} = 1.0$ and the discount factor (γ^{k_s})= 0.7.

We can see the increasing gain of our intelligent controllers (the NoTCP, MONTE and IT controllers) over the Nodup controller (case without any duplication) in all Figure 7.7(a), 7.7(b),7.7(c) and 7.7(d). Recall that our cost function is a linear combination of the voice drops, the video drops in the backup LSP, the TCP goodput and the second ITO cost. Hence, the increasing gain in all figures leads to the increasing gain over the cost function. As a result, we can conclude that our intelligent controllers (the NoTCP, MONTE and IT controllers) perform better than the Nodup controller, especially in a bursty network. This conclusion complies with the conclusion from the second TD(0) experiment in Section 7.3.2.

[7]The average traffic in the bottleneck link is 90% of link capacity.



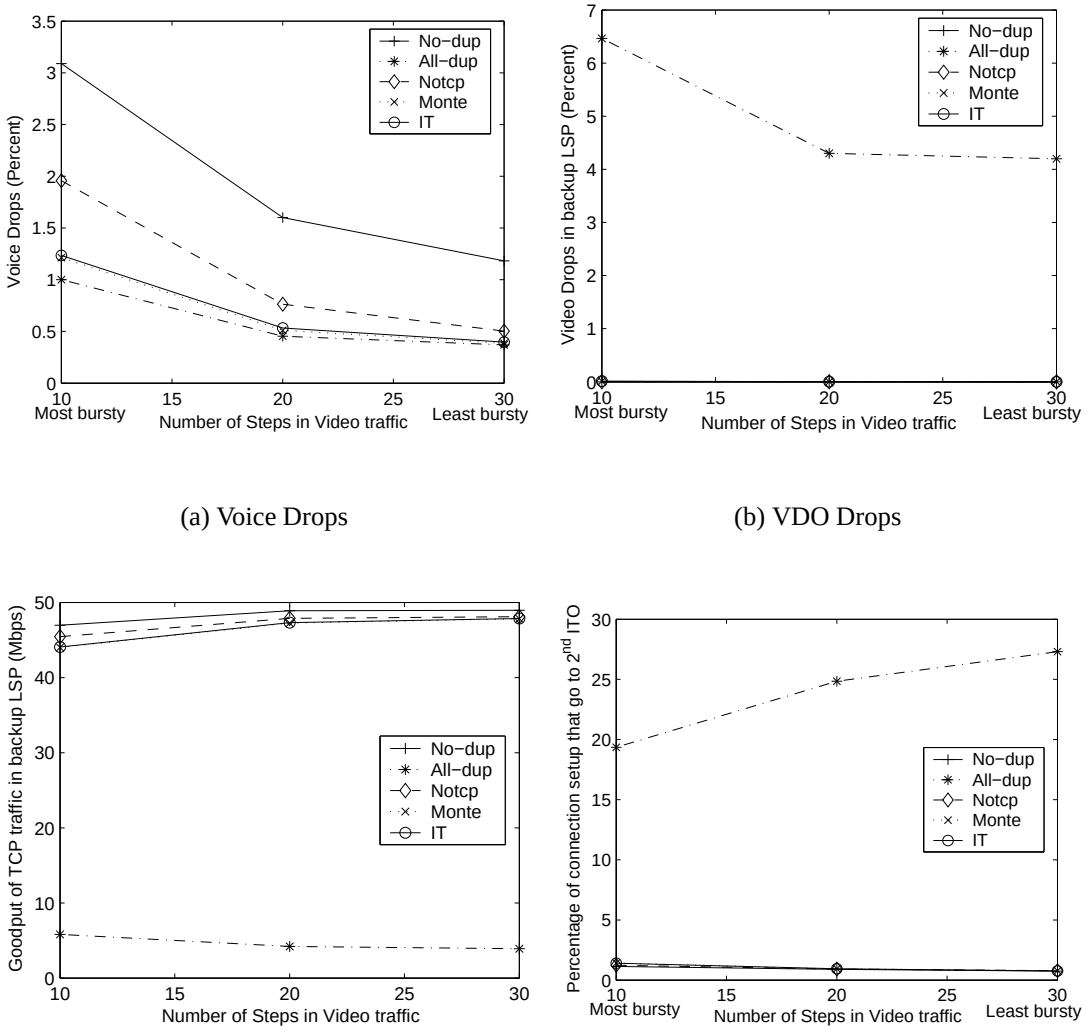
(a) Voice Drops

(b) VDO Drops

(c) Goodput

(d) Second ITO

Figure 7.6: Relationship of the performance measurements and the voice traffic amount using NS-2.



(a) Voice Drops

(b) VDO Drops

(c) Goodput

(d) Second ITO

Figure 7.7: Relationship of the performance measurements and the video traffic's burstiness using NS-2.

7.4 Conclusion

We introduced the fast timescale control traffic engineering, which based on the duplication scheme. This duplication scheme exploited the unused backup path in order to increase a quality of service of the high priority traffic. Upon the predicted congestion in the main LSP, our duplication control would duplicate packets and send them in the backup LSP, in addition to their normal transmission over the main LSP. This duplicated packets served as redundancies to the packets in the main LSP. These redundancies reduced a chance of getting information loss from packet drops and hence increased the QoS of our high priority traffic. Moreover, we assumed that the above duplication decision is made every fixed time slot from the controller, which resides in the ingress LSR.

We designed three intelligent controllers, which are the NoTCP controller, the MONTE controller and the IT controller.

The NoTCP controller implements the NoTCP policy, which is based on the Certainty Equivalence Control (CEC). Moreover, we ignore the TCP congestion control term, which relatively less important than other terms in the cost function. The small ignorance of the relatively less important term and a practically proven CEC could make this NoTCP policy performs well.

The MONTE controller implements the Monte-Carlo lookahead policy, which is based on the limited lookahead algorithm. In the limited lookahead algorithm, we choose a value function from NoTCP policy to approximate the optimal value function. We also approximate the expectation in the algorithm using the Monte-Carlo simulation.

The IT controller implements the Iterative lookahead policy, which is again based on

the limited lookahead algorithm. Like Monte-Carlo lookahead policy, we use a value function of NoTCP policy as an approximated value function and use simulation to get an expectation in Limited lookahead algorithm. However, this policy is better designed for the fast timescale control, which has a small processing power and time to calculate the simulation samples.

With TD(0) and the packet-level simulation, we compared the above three controllers with two other extreme controllers, which are the Nodup controller and the Alldup controller.

The Nodup controller stands for "No Duplication". This controller will not duplicate any packets. In other words, this controller presents the case that we do not have duplication mechanism at all.

The Alldup controller will duplicate packets all the time. This controller presents that case that we naively send duplicated packets all the times.

The above two controllers gave two extreme results. The Nodup controller, which does not duplicate any packets, gave the highest information loss in the main LSP while it gave the smallest drops in the backup LSP. On the other hand, the Alldup controller, which duplicates all the time, gave the smallest information loss in the main LSP while it gave the largest drops in the backup LSP.

Our empirical study exhibited the superior of our intelligent duplication controllers (the NoTCP, MONTE, IT controllers) compared to the case without the duplication mechanism (the Nodup controller), especially under the bursty network environment. Moreover it also showed the greater improvement of our intelligent controllers over the Nodup controller when the traffic amount is higher.

This empirical study also confirmed our belief that the duplication mechanism is effective for the QoS improvement. We demonstrated the benefit of proactive control using the traffic prediction. Nevertheless, this benefit relies on the precision of the traffic model.

Chapter 8

Conclusion and Future Direction

8.1 Summary of the Dissertation

In this dissertation, we presented the design and evaluation of two fast timescale controls to do traffic engineering. The first fast timescale control traffic engineering is based on the migration mechanism. In the migration mechanism, we moved high priority traffic around to preserve the QoS. We designed the migration control by taking into account both the propagation delay of the packets and the information delay to the controller. Moreover, the designed migration control also considered its interaction with the TCP congestion control in the low priority traffic.

We evaluated our migration design using both Temporal-Difference (TD) learning and packet-level simulation. The evaluation result displayed the superiority of the designed controllers over the case without migration control, especially in the bursty network environment. Moreover, they yielded a greater improvement over the case without migration when the traffic amount is high but not overloaded.

The second fast timescale control traffic engineering is based on the duplication control. We exploited the unused backup path, in which a provider normally set it up to increase the network reliability. This backup path is normally unused even when there

is congestion in its working path. Upon the predicted congestion in the working path, we would duplicate the packets and send them in both the working path and the backup path. The duplicated packets increased the QoS of the traffic that they duplicated from. However, the duplicated packets could degrade both the low priority and the high priority traffic that was normally sent through the backup path. We designed the duplication control, which determines the appropriate duplication period. Like the migration work, we also took into account both the propagation delay of the packets and the information delay.

We also evaluated the designed duplication control using both Temporal-Difference (TD) learning and packet-level simulation. The designed duplication controllers yielded their superiority over the case without duplication control, especially in the bursty network environment. Additionally, they achieved a greater improvement over the case without duplication when the amount traffic is high.

Overall, our empirical study showed a clear advantage of the fast timescale traffic engineering. It also pointed out the advantage of having a traffic prediction, by reacting in accordance to the prediction. However, this advantage greatly depends on the accuracy of the traffic model.

8.2 Future Direction

We could extend the works from this dissertation as follows:

Extension of Migration Control

We can explore a distributed migration control with shared bottleneck links. The migration problem in the dissertation can be extended in such a way that we allow each

bottleneck links to carry the traffic from the different ingress-egress node pairs. Then, we could explore a distributed algorithm to allow the controllers (which are located in the different ingress nodes) to make their separate migration decision. In this new problem, we could face the following two technical issues.

The first issue is the available information issued for each controller. It is unrealistic for all controllers to have the information about the traffic in every link. As one possible solution, we could define a set of the *reachable links*, from which each ingress can get information. For example, the controller could only have the information about the traffic in the links that have a path passing through those links.

The second issue is the coordination issue among controllers that share the same bottleneck link. With the independent decision made from each controller, there is a need to coordinate two or more controllers that shares the same bottleneck links. We can use an example to illustrate this issue. Suppose we have two ingress nodes that have paths sharing the same bottleneck link. They could simultaneously receive the information indicated that the bottleneck link is under-utilized. Without any coordination, they might simultaneously migrate flows in order to exploit that under-utilized link. As a result, that link could suddenly be over-utilized, which in turn could lead to a serious QoS violation.

As one possible solution for this issue, we could identify a set of the controllers that has a need to communicate to one another. More specifically, we could have a constraint that the communication among controllers is required if those controllers share the same bottleneck link. The communication between the two controllers can be direct or can be indirect through the other controllers. Moreover, the communication content could be only the controllers' decision, or it could be all information about the traffic in their ingress node.

Extension of Duplication Control

We can allow a shared backup path for two or more working paths. Many providers assume that there is single link or node failure at a time. Hence, they setup two or more working paths to have some part of their backup path shared. This setup certainly is not desirable for our duplication controller in this dissertation because we assumed that there is a dedicated backup path for each working path. In this new problem, we could explore a distributed algorithm to allow the duplication controllers (which are located in different ingress node) to make the independent decision. This extended duplication problem can be faced with the similar technical issues as the extended migration problem.

The first issue is also the available information issue for each controller. This is the same technical issue as in the extended migration problem. Hence, we can use the same possible solution by setting up a set of reachable links, from which each controller can get information.

Again, the second issue is the coordination issue among controllers that share the same bottleneck link. We will use another example to illustrate the need of this coordination. Suppose we have two working paths and a part of their backup paths share the same link. It is possible that both the working paths could simultaneously have congestions. Without any coordination, the controllers, which are in the ingress of both working paths, could independently duplicate their traffic. As a result, it could lead to undesirable drops in the shared links. Because of the similarity with this problem, we could use the same possible solution from the extended migration problem. That is, we could define a set of the controllers that need to communicate to one another.

Combination of Migration and Duplication Control

We can consider combining the migration control and duplication control into a single optimization problem. We can explore the following two potential combinations.

The first combination considers the multiple working paths with their own separate (non-shared) backup paths. In this combination, we let each parallel path to have their own backup paths. Now, we can choose to either migrate flows or duplicate packets if there is a congestion in a LSP.[1]

The duplication control will help improve the traffic's QoS in that LSP. However, the high priority cross traffic still suffer from a degraded QoS. It is because the duplication control does not alleviate the congestion in the working path. On the other hand, the migration control will relieve the congestion in that LSP by migrating flows away. As a result, both the traffic in that LSP and the high priority cross traffic can get a better QoS together. However, there is a risk of getting out-of-order packets and a risk of making other parallel LSPs congested. We will make a choice of either using migration or duplication. At some point of time migration would be better because it will also help the drops in cross traffic. However, duplication would be great in situations when all paths are congested.

In general, the migration control will not be a good choice when all link are overloaded. As we can see from the empirical result of this dissertation, the benefit of having migration control is diminished in an overloaded situation. Hence, the duplication control can be a good alternative in this situation. As a result, the controller

[1]We will not consider migrating flows to the backup path even though it has a room for high priority traffic. The main reason comes from the fact that we still have to reserve the backup path in case of failure in the working path. If failure does occur, we will have to shift all traffic into that backup path.

will have to choose either a migration or duplication control based on the situation at hand.

The second combination considers the multiple working paths with a single shared backup path. In this combination, we let all parallel working paths share one and only one backup path. With the shared backup path, we will have to carefully duplicate the traffic. There could be significant drops if two or more working paths send duplicated packets to the shared backup path at the same time. This situation should be prevented because the cross traffic in the backup path could risk causing a significant QoS violation.

Bibliography

[1] Abdelnaser Adas. Traffic models in broadband networks. *IEEE Communications Magazine*, 35(7):–, July 1997.

[2] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*, volume I. Athena Scientific, Belmont, MA, 2 edition, 2000.

[3] T. Bogovic, J.E. Burns, T. Carpenter, K.R. Krishnan, T. Ott, and D . Shallcross. Path selection and bandwidth assignment in MPLS. In *Proceedings of MPLS2000 International Conference*, Octobor 2000.

[4] Stephan Bohacek, Joao P. Hespanha, Junsoo Lee, and Katia Obraczka. Analysis of a tcp hybrid model.

[5] Neal Cardwell, Stefan Savage, and Thomas Anderson. Modeling TCP latency. In *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, Tel Aviv, Israel, March 2000.

[6] Costas Courcoubetis, Antonis Dimakis, and George Stamoulis. Traffic equivalence and substitution in a multiplexer. In *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, New York, March 1999.

[7] Bruce Davie and Yakov Rekhter. *MPLS: Technology and Applications*. Morgan Kaufmann, CA, 2000.

[8] M. Davis and R. Vinter. *Stochastic Modelling and Control*. Chapman and Hall, London, 1985.

[9] Francois Baccelli Dohy. The AIMD model for TCP sessions sharing a common router.

[10] Anwar Elwalid, Cheng Jin, Steven Low, and Indra Widjaja. MATE: MPLS adaptive traffic engineering. In *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, Anchorage, Alaska, April 2001.

[11] V. Firoiu, I. Yeom, and X. Zhang. A framework for practical performance evaluation and traffic engineering in ip networks, 2000.

[12] Victor Firoiu and Marty Borden. A study of active queue management for congestion control. In *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, Tel Aviv, Israel, March 2000.

[13] Sally Floyd and Kevin Fall. Promoting the use of end-to-end congestion control in the internet. *IEEE/ACM Transactions on Networking*, 7(4):458–472, August 1999.

[14] Liang Guo and Ibrahim Matta. The War between Mice and Elephants. In *Proceedings of ICNP'2001: The 9th IEEE International Conference on Network Protocols*, Riverside, CA, November 2001.

[15] Go Hasegawa and Masayuki Murata. Analysis of dynamic behaviors of many TCP connections sharing tail-drop/RED routers. In *Proceedings of the IEEE Conference on Global Communications (GLOBECOM)*, November 2001.

[16] D. P. Heyman and T. V. Lakshman. What are the implications of long-range dependence for VBR-Video traffic engineering. *IEEE/ACM Transactions on Networking*, 4(3):301–317, 1996.

[17] Keith Kim, Petros Mouchtaris, Suil Samtani, Rajesh Talpade, and Larry Wong. Qos provisioning for voip in bandwidth broker architecture: A simulation approach. In *In SCS WMC01*, January 2001.

[18] S. Kini, M. Kodialam, T. Lakshman, and C. Villamizar. Shared backup label switched path restoration. Internet Draft, Internet Engineering Task Force, November 2000. Work in progress.

[19] Murali Kodialam and T. V. Lakshman. Dynamic routing of bandwidth guaranteed tunnels with restoration. In *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, Tel Aviv, Israel, March 2000.

[20] Murali Kodialam and T. V. Lakshman. Minimum interference routing with applications to MPLS traffic engineering. In *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, Tel Aviv, Israel, March 2000.

[21] K-T Kuo, S. Phuvoravan, T. Guven, L. Sudarsan, H.S. Chang, S. Bhattacharjee, and M.A. Shayman. Fast Timescale Control for MPLS Traffic Engineering, Submitted to Globecom 2002.

[22] San-Qi Li, Song Chong, and Chia-Lin Hwang. Link capacity allocation and network control by filtered input rate in high-speed networks. *IEEE/ACM Transactions on Networking*, 3(1):10–25, February 1995.

[23] San-qi Li and Chia-Ling Hwang. On input state space reduction and buffer non-effective region. In *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, 1994.

[24] Basil Maglaris, Dimitris Anastassiou, Prodip Sen, Gunnar Karlsson, and John D. Robbins. Performance models of statistical multiplexing in packet video communications. *IEEE Transactions on Communications*, 36(7):834–844, July 1988.

[25] Marco Mellia Member. Tcp model for short lived flows.

[26] Archan Misra, John Baras, and Teunis Ott. The window distribution of multiple TCPs with random loss queues. Technical report, Institute for Systems Research, University of Maryland, 1999.

[27] Vishal Misra, Wei-Bo Gong, and Donald F. Towsley. Fluid-based analysis of a network of AQM routers supporting TCP flows with an application to RED. In *SIGCOMM*, pages 151–160, 2000.

[28] Jitendra Padhye, Victor Firoiu, Donald F. Towsley, and James F. Kurose. Modeling TCP reno performance: a simple model and its empirical validation. *IEEE/ACM Transactions on Networking*, 8(2):133–145, April 2000.

[29] Kihong Park and Walter Willinger. *Self-similar network traffic and performance evaluation*. John Wiley & Sons, NY, 2000.

[30] S. Phuvoravan, K-T Kuo, T. Guven, L. Sudarsan, H.S. Chang, S. Bhattacharjee, and M.A. Shayman. Fast Timescale Control for MPLS Traffic Engineering. Technical Report CS-TR-4351 and UMIACS-TR-2002-31, University of Maryland, College Park, 2002.

[31] Bong K. Ryu and Anwar Elwalid. The importance of long-range dependence of VBR video traffic in ATM traffic engineering: Myths and realities. In *SIGCOMM*, pages 3–14, 1996.

[32] Schulzrinne, Casner, Frederick, and Jacobson. RTP: A transport protocol for real-time applications. *RFC 1889*, Jan 1996.

[33] Mischa Schwartz. *Broadband integrated networks*. Prentice Hall, NJ, 1996.

[34] V. Sharma et al. Framework for MPLS-based recovery. Internet Draft, Internet Engineering Task Force, July 2001. Work in progress.

[35] B. Sikdar, S. Kalyanaraman, and K. Vastola. Analytic models for the latency and steady-state throughput of tcp tahoe, reno and sack, 2001.

[36] Aravind Srinivasan. Approximation algorithms via randomized rounding: a survey. In M. Karonski and H. J. Promel, editors, *Lectures on Approximation and Randomized Algorithms*, Series in Advanced Topics in Mathematics, pages 9–71. Polish Scientific Publishers PWN, 1999.

[37] Subhash Suri, Marcel Waldvogel, and Priyank Ramesh Warkhede. Profile-based routing: A new framework for MPLS traffic engineering. In Fernando Boavida, editor, *Quality of future Internet Services*, number 2156 in Lecture Notes in Computer Science, pages 138–157, Berlin, September 2001. Springer Verlag. An earlier version is available as Washington University Computer Science Technical Report WUCS-00-21, July 2000.

[38] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: an introduction*. The MIT press, 1998.

[39] UCB/LBNL/VINT. Network simulator - ns (version 2).