The Hong Kong Polytechnic University
Department of Electronic and Information Engineering

Proxy Servers for Internet Multimedia Streaming

Wai-Kong Cheuk

A thesis submitted in partial fulfillment of the requirements for the Degree of Doctor of Philosophy

December 2004

Certificate of Originality

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

_____ (Signed)

CHEUK WAI KONG (Name of student)

Abstract

In the topology of today's World Wide Web, proxy servers have been used extensively to reduce network traffic by cutting down the amount of repetitive information. However, traditional web proxy servers do not support multimedia streaming. One reason for this is that the general scheduling strategy adopted by most of these Web proxy servers does not provide any real-time support. Another reason concerns resource management. Such servers are required to handle hundreds of simultaneous connections between media servers and clients. Every video stream that arrives at the server and is exported from it follows a specific arrival and delivery schedule. While arrival schedules compete for incoming network bandwidth, delivery schedules also compete for outgoing network bandwidth. As a result, such servers have to provide sufficient buffer and disk cache space for storage in addition to the memory space, disk space, disk bandwidth, and computational resources they must provide. In order to optimize the throughput, it is necessary to govern the usage of these resources properly.

Furthermore, cache and trash decisions for a web proxy are no longer applicable to video streams. Decision-making for cache and trash in a web proxy is based on web objects such as text documents, images and files whose size rarely exceeds several hundred megabytes. However, when it comes to video streaming, the delivered video data can easily be in the range of gigabytes for MPEG-I, and even several gigabytes for MPEG-II. Even with today's computer power, it would be disastrous to have cached or trashed an incorrect video stream. In the light of this, the concept of video staging was proposed for the video streaming proxy service. This advances the cache-or-none concept in traditional proxy design by dividing a video stream into two parts. The first part of the video stream will be cached in the proxy while the other will remain in the video streaming server. They will be combined and sent to the client at their next cache-hit. This eases the cache and trash decision, since it can immediately be based on a smaller object.

Nevertheless, such a scheme also complicates resource management. In practice, an admission control scheme is responsible for the management of these resources. During the admission of a new request, it will evaluate the resource requirements of the requested stream and check whether the proxy server is able to support that requirement before accepting. Then, it will reserve those required resources so that a proper transmission of the stream can be guaranteed. A simple scheme can allot these reservations based on the maximum requirement, but it will not be able to achieve a satisfactory utilization rate, since the actual requirement varies from time to time

and underutilization is likely. In order to improve the utilization rate, more advanced admission control schemes become necessary. In short, multimedia proxy servers have a different design compared with traditional web proxies. In this research work, we deal with two aspects of design in such video streaming proxy servers. The first aspect is resource management, which includes processing power, memory, network and disk resources. The second aspect is admission control.

In regard to processing power management, we first describe our development work with a practical video streaming proxy server that supports real-time multimedia applications based on the concept of contractual scheduling. Moreover, we discuss the group scheduling mechanism which enables the transfer of processing power between tasks, something beyond the capabilities of traditional schedulers. By this mechanism we have achieved a substantially improved performance, particularly when both time-constrained and non-time-constrained processes coexist within the proxy server. For the management of the other resources, we begin by analyzing the properties of a traditional smoothing algorithm and a video staging algorithm. Then, based on the smoothing algorithm, we develop a pair of offline and online video staging smoothing algorithms for video streaming proxy servers. These two algorithms give rise to the proposed video staging arrival schedule based on the delivery schedule. Under the auspices of the arrival and delivery scheduling pair we have achieved a better resource utilization rate for the proxy server against different parameter sets. It is also interesting to note that the resource usage becomes transferable under these algorithms: they facilitate usage transfer between network bandwidth, disk bandwidth, and memory space.

Regarding admission control, scheduling schemes such as Constant-Data-Length (CDL) and Constant-Time-Length (CTL) were recently proposed for the purpose of improving the utilization though some two-level scheduling schemes. However, these approaches usually focus on disk bandwidth alone and ignore the memory constraint. In addition, these two-layer approaches can only reduce the waste. In order to improve utilization, we have analyzed the relationship between the sum of all schedules and their actual resource requirements. Based on this relationship, we have devised a grand scale scheduling approach by taking both the disk bandwidth and memory usage into account. In this way, we are able to re-allocate the unused resources reserved for a video stream to another stream that requires more resources, thus reducing their overall reservation requirement.

The resource availability-based admission control scheme can only provide a binary accept or reject decision to the video service request. However, our study shows that an acceptance of particular streams may jeopardize the capacity of a video streaming proxy server. That is, a server may no longer accept any more video streams after the acceptance of a particular stream – a black sheep. If we are able to detect such streams in advance, we can maintain the server capacity by rejecting them. However, this will lead to discrimination. When it comes to a multiple server platform, we may divert those particular streams to another server to avoid discrimination. Yet, there are numerous settings available. As a result, we have developed several schemes to facilitate such diversion and we have determined their strengths and weaknesses in different scenarios.

Statement of Originality

1 Contractual Function Scheduler (Chapter 3, Section 3.2.4)

Borrowing the idea of a Contractual Process Scheduler, we have developed a contractual function scheduler to help us schedule the execution of different primitive operations in MPS. By using a contractual function scheduler, we can provide a contractual platform for MPS without relying on the contractual process scheduler, which incurs many more process switches and causes performance degradation. We can improve the throughput of the MPS by adapting the contractual function scheduler for function scheduling.

2 Video Staging Schedule Γ (Chapter 4, Section 4.3)

We have constructed a novel video staging scheduling algorithm Γ . With this scheduling algorithm, we are able to provide a video staged schedule with a network bandwidth upper bound part and a local disk storage part. Our algorithm is capable of utilizing the network bandwidth in a more efficient way. As a result, the volume of data required to store in the local disk is smaller. And this will reduce the disk storage and bandwidth requirement which can increase the capacity in return.

3 Grand Scale Scheduling (Chapter 5, Section 5.2)

We have formulated a mathematical relation between the sum of all video delivery schedules and their corresponding resources requirements in a video streaming proxy server. It allows us to anticipate the capacity of a video streaming proxy server within a confined time period with implicit resource sharing.

4 Admission Control for Grand Scale Scheduling (Chapter 5, Section 5.3)

Through the mathematical relation obtained from Grand Scale Scheduling, we have developed a corresponding admission control scheme for a video streaming proxy server.

5 Classification of Black Sheep (Chapter 6, Section 6.2)

We have classified two types of “Black Sheep” occurrences in admission control of a video streaming proxy server. First type of them occurs when the video stream to admit exhibits a very large bit-rate variation when compared with the stream average bit-rate. The occurrence of the second type is dependent to the admission history of the server. However, both of their

occurrences will reduce the overall capacity of the server and have to be avoided. We have also devised a mechanism to identify a “Black Sheep” through the construction and comparison of a nominal loading curve.

6 Admission Schemes with Black Sheep Avoidance (Chapter 6, Section 6.2.1 and 6.2.2)

After we have identifies a “Black Sheep” during the admission, we have devised several strategies to avoid its occurrence. They include SSRC, MSRC-S1, MSRC-S2 and MSRC-P which deal with single server platform and multiple-server platform and the Simple Reject, Random and Lazy approaches.

Conference

1. Cheuk, W.K., Li, C.K., Lun, Daniel P.K., and To, T.P., "Design of a Process Scheduler with Guaranteed Resource Allocation", Proceedings of IASTED Networks, Parallel and Distributed Processing, and Applications Conference 2002, NPDPA 2002, pp.235-240, Tsukuba, Japan, Oct 2002.
2. Cheuk, W.K., and Lun, Daniel P.K., "Video Staging in Video Streaming Proxy Server", The IEEE International Conference on Multimedia & Expo, 2004, Taipei.
3. Cheuk, W.K., and Lun, D.P.K., "Grand Scale Scheduling for Admission Control", International Symposium on Intelligent Multimedia, Video and Speech Processing, ISIMP2004, 20-22 Oct 2004, Hong Kong.

Journal

1. Cheuk, W.K., Hsung T.C., and Lun, Daniel P.K., "Design and Implementation of Contractual Based Real-Time Scheduler for Multimedia Streaming Proxy Server", Multimedia, Tools and Applications, Netherlands. (Accepted)
2. Ho, K.H., Lun, Daniel P.K. and Cheuk, W.K., "Content-based Scalable H.263 Video Coding for Road Traffic Monitoring", IEEE Transaction on Multimedia. (Accepted)
3. Cheuk, W.K., and Lun, D.P.K., "Throughput Optimization for Video Streaming Proxy Servers Based on Video Staging", Multimedia, Tools and Applications, Netherlands. (Submitted)
4. Cheuk, W.K., Lun, D.P.K., and Li, C.K., "Grand Scale Scheduling for Admission Control in Video Streaming Proxy Server", IEEE Transactions on Multimedia. (Submitted)

Acknowledgements

With regard to the writing up of this report, I have to express my gratitude I owe so much to my chief supervisor Dr. Daniel Pak-Kong Lun for his patience, advice and help during the course of my study. In addition, the help from Dr. Chi-Kwong Li, Dr. Jimmy Tsun-Ping To, Dr. Wai-Kin Lam and Dr. Yui-Lam Chan are also immeasurable.

Secondly, I should also like to give thanks to my colleagues Dr. Tai-Chiu Hsung, Mr. Wai-Lam Hui, Dr. Chi-Lun Chan, Mr. Kai-Hong Ho, Dr. Wai-Chung Poon, Mr. Geoffery Ko-Cheung Hui, Mr. Yik-Hing Fung and Mr. Michael Kwok-Kong Yiu for their friendly assistance.

Last but not least, I should also give thanks to my parents for their unselfish love and support.

Table of Contents

ABSTRACT II

STATEMENT OF ORIGINALITY V

CONFERENCE VII

JOURNAL VII

ACKNOWLEDGEMENTS VIII

TABLE OF CONTENTS IX

LIST OF FIGURES XI

LIST OF TABLES XIV

LIST OF TABLES XIV

1 INTRODUCTION 1

1.1 Video Streaming Proxy Server 2

1.2 The Need 4

1.3 Summary 7

2 LITERATURE REVIEW 9

2.1 Contractual Computing 10

2.2 Video Smoothing 12

2.3 Video Staging 15

2.4 Disk Scheduling 16

3 COMPUTATIONAL RESOURCE 22

3.1 Video Streaming Proxy Server 22

3.2 Architecture of MPS 23

3.3 Experimental Results 35

3.4 Summary 44

4 MEMORY, NETWORK AND DISK BANDWIDTH 46

4.1 Quantifying Video Streaming 47

4.2 Video Staging Schedule 51
4.3 Video Staging Algorithm Γ 54
4.4 Experimental Results 64
4.5 Summary 74
5 GRAND SCALE SCHEDULING FOR ADMISSION CONTROL 76
5.1 Resource Usage in System Perspective 76
5.2 Grand Scale Scheduling 80
5.3 Admission Control Mechanism 87
5.4 Performance 89
5.5 Summary 95
6 ADMISSION CONTROL WITH EARLY "BLACK SHEEP" DETECTION 96
6.1 Admission Control 96
6.2 The Black Sheep 105
6.3 Simulation Results 113
6.4 Summary 120
7 CONCLUSION 122
7.1 Processing Power Management 122
7.2 Stream Level Resource Management 123
7.3 System Level Resource Management 123
7.4 Comprehensive Strategy 124
7.5 Future Work 124
REFERENCES 126

List of Figures

Figure 1: Network versus Memory 14
Figure 2: Network versus Disk....15
Figure 3: Individual Disk Schedules 17
Figure 4: Overall Disk Schedules....17
Figure 5: CTL Disk Schedule....18
Figure 6: CTL Memory Schedule 18
Figure 7: Overall CTL Disk Schedule....18
Figure 8: CDL Disk Schedule 19
Figure 9: CDL Memory Schedule....19
Figure 10: Overall CDL Disk Schedule 19
Figure 11: Uneven CDL Disk Access 20
Figure 12: Uneven CDL Memory Schedule....20
Figure 13: GCDL Memory Schedule 20
Figure 14: Perspective of the Video Streaming Proxy Server....25
Figure 15: MPS working as a gateway....26
Figure 16: MPS working as a reflector 26
Figure 17: MPS working as a mirror (cache in)....27
Figure 18: MPS working as a mirror (cache out)....27
Figure 19: Contractual Scheduling....32
Figure 20: Contractual Function Scheduler....33
Figure 21: Effective bandwidth against number of loadings for video title “F100” 38
Figure 22: Effective bandwidth against number of loadings for video title “BTC”....38
Figure 23: Effective bandwidth against number of loadings for video title “Patriot”....38
Figure 24: Average Packet Lost Rate against number of loadings for video title “F100” 39
Figure 25: Average Packet Lost Rate against number of loadings for video title “BTC”....39
Figure 26: Average Packet Lost Rate against number of loadings for video title “Patriot”....40
Figure 27: Maximum Packet Lost Rate against number of loadings for video title “F100” 40

Figure 28: Maximum Packet Lost Rate against number of loadings for video title “BTC” 41

Figure 29: Maximum Packet Lost Rate against number of loadings for video title “Patriot” 41

Figure 30: Bandwidth against number of loadings for video title “BTC” 42

Figure 31: Average Packet Loss Rate against number of loadings for video title “BTC” 43

Figure 32: Maximum Packet Loss Rate against number of loadings for video title “BTC” 43

Figure 33: Cumulative data transfer of a connection in a proxy server 48

Figure 34: Cut-off arrangement with video staging 49

Figure 35: A smoothed schedule 50

Figure 36: Schedule generated by algorithm Γ 58

Figure 37: Relationship between Incoming Bandwidth, Proxy Buffer Size and Disk Access for Video Staging Algorithm Γ 61

Figure 38: Relationship between Incoming Bandwidth, Proxy Buffer Size and Disk Access for the Traditional Video Staging Algorithm 63

Figure 39: Data delivery schedule of video title “Star Wars” 64

Figure 40: Disk access generated by traditional video staging algorithm (740Kbps 925KB) 65

Figure 41: Video staging schedules of the proposed and traditional algorithms (740Kbps 925KB) 66

Figure 42: Video staging schedule of the proposed and traditional algorithms (370Kbps 925KB) 68

Figure 43: Video staging schedule of the proposed and traditional algorithms (1110Kbps 925KB) 69

Figure 44: Video staging schedule of the proposed and traditional algorithms (740Kbps 462KB) 70

Figure 45: Video staging schedule of the proposed and traditional algorithms (740Kbps 1388KB) 71

Figure 46: Network Retrieval Comparison 72

Figure 47: Disk Storage Comparison (Logarithmic) 73

Figure 48: Disk Access Comparison (Logarithmic) 74

Figure 49: Proxy operating in forward mode 77

Figure 50: Proxy operating in cache-out mode 77

Figure 51: Proxy operating in cache-in mode 78

Figure 52: Grand Scale Schedule 83

Figure 53: Grand Scale Schedule and Individual Schedule 83

Figure 54: Cache-Out Grand Schedule 85
Figure 55: Acceptance variation with Disk Bandwidth 90
Figure 56: Acceptance variation with Memory Size 91
Figure 57: Acceptance variation with Memory Size and Disk Bandwidth 92
Figure 58: Acceptance variation with Time Span 93
Figure 59: Acceptance variation with Buffer Time 94
Figure 60: Effect of problematic streams to the heuristic 101
Figure 61: Heuristic vs. Delay 102
Figure 62: Admission Tree with Maximum Strategy 103
Figure 63: Heuristic from different Admission Strategy 104
Figure 64: Heuristic Ratio between different Admission Strategies 105
Figure 65: Type I Black Sheep 106
Figure 66: Type II Black Sheep 107
Figure 67: Construction of Optimal Heuristic Curve 109
Figure 68: SSRC Algorithm 110
Figure 69: MSRC-S1 Algorithm 111
Figure 70: MSRC-S2 Algorithm 112
Figure 71: MSRC-P Algorithm 113

List of Tables

Table 1: Average Processing Time (ms) 90

Table 2: SSRC (Normal) 115

Table 3: SSRC (BS1) 115

Table 4: SSRC (BS2) 115

Table 5: SSRC (BS3) 115

Table 6: MSRC-S1 (N) 116

Table 7: MSRC-S1 (BS1) 116

Table 8: MSRC-S1 (BS2) 117

Table 9: MSRC-S1 (BS3) 117

Table 10: MSRC-S2 (N) 118

Table 11: MSRC-S2 (BS1) 118

Table 12: MSRC-S2 (BS2) 118

Table 13: MSRC-S2 (BS3) 118

Table 14: MSRC-P (N) 119

Table 15: MSRC-P (BS1) 119

Table 16: MSRC-P (BS2) 119

Table 17: MSRC-P (BS3) 119

1 Introduction

Recently, flourishing developments in the fields of personal computer and broadband networks have given rise to the possibility of video streaming. In brief, video streaming enables digital video broadcasting from hosting servers to playback clients through a computer network. A video may be recorded, encoded and stored in the video streaming server prior to delivery or may be encoded on-the-fly. The whole video stream will be divided into many stand-alone packets that can be played back individually. In this way, clients can play the video immediately upon receiving it. Video streaming in MPEG-II quality has already become possible with current technology. Owing to the blooming developments in networking technology, video broadcasting is no longer available only to the few; anybody can set up their own stations in a garage. This situation has made the traditional television broadcasting approach seem inferior, which will allow video streaming to thrive.

Video streaming is simply a real-time transmission of a digitalized video signal between computers over a computer network. A video is a series of images rendered in an ordered and synchronous manner. In computers, these images are digitalized and stored, pixel by pixel. Depending on the color system – gray scale, RGB or YUV422 – each pixel may require from 8 bits to 32 bits to represent itself. For a video clip using RGB, a 32-bit color system with an image size of 320x240 playing at 30 fps (frames per second) for one second already has a volume of about 9.2MB. Due to this sheer size of data, compression must usually be applied. Spatially, some reversible 2-D transforms such as DCT [66] and Wavelet [67][70] can be applied to each image, allowing them to be stored in the frequency domain. As human vision is more sensitive to the lower frequency bands, we can quantize the transformed coefficients of these images with an unequal bit plan so that data of lower band frequencies are stored more accurately than the higher ones. A significant amount of space can be saved in this way. In addition, compression can also be conducted temporally. The information contained in consecutive images of a video is usually highly correlated, as an object appearing in one frame is not likely to disappear in the next. Usually it will only move around. Based on this, motion vectors are used to describe the motion of this object so that explicit coding of this object would not be required for the second frame. This reduces the size even further. Other attempts have been made to compress the video with 3-D transformations taking time as the third dimension. On playback, these images will be

decompressed and rendered one by one in a synchronous manner. These compression techniques are adopted into current mainstream digitalized video standards such as MPEG-I, II, IV and H.263 [71][72][73] [74] .

Although these compression techniques have already decreased the size of a video substantially, it is still too large for computer networks to convey efficiently using current technology. The size of a typical MPEG-I video of 100 minutes is about 1GB. However, the bandwidth of a general household broadband service seldom exceeds 10Mbps. Thus, it will take at least 800 seconds (~13 minutes) to retrieve the whole video before the playback can begin, which is not tolerable. Fortunately, there is another option – video streaming. As mentioned above, a video is a composition of ordered images played back in a synchronous manner. Therefore, each image will have a specific time at which it must be played back. As a result, we may begin the playback as long as we can guarantee that each of these images can be retrieved and displayed on time. This is the philosophy behind video streaming.

1.1 Video Streaming Proxy Server

Most infrastructures used to support Internet services can be shared with video streaming except for proxy servers. Web proxy servers are usually employed as bandwidth-saving devices deployed in the gateway between a LAN and a WAN. They make use of the tendency towards repetitive requests for the same object within a period of time to achieve the saving. Without a proxy server, the object must be sent all the way from the server to the client upon request. However, with a proxy server installed, the object can be stored in local storage the first time a client asks for that object. When the same object is requested the second time, the proxy server can retrieve the object directly from its local storage without bothering the actual server. Hence, the network bandwidth (WAN) between the proxy server and the actual server is saved. Many advanced algorithms and strategies have been developed to improve this caching and trashing mechanism. When the local storage of a proxy server approaches its capacity, it must trash some of its stored objects in order to make space for the new ones. The server has to anticipate the future usage patterns of these objects so that it can maximize its performance.

The idea of caching through proxy servers can also be applied to provide video streaming services. Video streaming proxy servers can be installed between clients and servers to save

bandwidth. However, the sheer size of a video stream makes it very difficult to decide when it should be sent to cache or trash. Web objects rarely exceed several megabytes in size and are much smaller than video streams, while the latter could easily reach a size measured in gigabytes. The local storage in the proxy can only hold a few video streams at a time currently. Therefore, caching and trashing become more frequent and the effect of an incorrect decision becomes more significant. The proxy server will end up caching new video streams and trashing just-cached video streams all the time. The lifespan of the local storage will be greatly reduced in this way.

Worse yet, unlike other web objects, video streams have an inherent time constraint. A video sequence is merely a composition of images which will be shown to the client one by one over a fixed period of time. To avoid jitters, each of these images must be shown at the right time. This makes it necessary that each packet from the video stream arrives on time. Hence, this also imposes real-time constraints on these packets handled by the video streaming proxy server. The proxy server has to guarantee that it can immediately send back to the client any packet received from the server. As for cached video sequences, the proxy server has to pay close attention to disk access to make sure that it can deliver those packets on time. This has already made the design of a video streaming proxy server as difficult as that for a video streaming server. On top of all these, a video streaming proxy server is required to cache a video stream while the stream is going through it, making its design an even more challenging task.

As to any other kind of servers, capacity is one of the major concerns in video streaming proxy server design. The capacity of a proxy server is limited by the availability of resources. In fact, processing power has become a limited resource in a real-time environment, as only a certain number of instructions can be executed within a fixed period of time. To increase capacity, processing power has to be managed. Besides processing power, a video streaming proxy server has to provide enough ingress network bandwidth to receive data from a video streaming server. It must also provide enough memory space for temporary storage of these data. Finally, it has to provide enough egress network bandwidth to send those data to its client. In addition, if caching is required, ingress and egress disk bandwidths are also indispensable.

1.2 The Need

The cruelest truth about capacity and resource availability is that when any one kind of resources runs out, the server can no longer accept any more clients – even if it has abundant supply of the others. A good design has to be able to maintain a perfect balance over these resources so that none of them will run out prematurely and form a bottleneck in the capacity count. However, resource requirement varies from stream to stream; it is not an easy task to provide a good load balancing scheme.

Suppose we can provide a mechanism such that the usage of different resources can be interchanged, the management of these resources can be made much easier. Such interchange can operate on two levels: heterogeneous and homogeneous. For heterogeneous interchange, the usage of one resource can be replaced by another when it is running short. As a result, we may avoid overusing any resources. In homogeneous case, usage of a particular resource can be shared between different clients. Sometimes, a resource assigned to a client may not be used, thus going to waste. If we are able to transfer these wasted resources to other clients, we can utilize these resources more efficiently and improve the overall capacity.

Our objective in this project is to investigate and develop a set of resource usages interchanging mechanisms to answer this bottleneck challenge. When a resource is going to run out within the video streaming proxy server, its usage will be replaced by the usage of another resource in abundance. As a result, more video streams may be accepted before the capacity is reached. The system throughput is increased.

1.2.1 Processing Power

Sharing of processing power among processes is a kind of homogeneous interchange of resource usage. In an effort to achieve processing power sharing, different attempts have been made, and some of them target at the construction of video streaming servers. For instance, in the attempts of Rialto and SMART [58][59][60][61][62][63][64][65][97], real-time support is obtained from a low-cost general purpose platform so that it does not require an expensive and sophisticated real-time system. The contractual paradigm [13][14][15][16] is another attempt of this kind. It emphasizes an explicit respect for the ownership of resources such that the usage of them is specified in the form of a contract. The process that owns this contract retains the right of

using the resource according to the specification. Thus, transferring of resource is made possible by explicitly transferring the contract to another process.

A practical platform for the implementation of the contractual paradigm has been built in the form of a contractual process scheduler [18]. In this platform, processing power is specified by the trio of (start time, end time, percentage) . It is guaranteed that the holder of the contract will be granted usage of the processing unit in the specified percentage between the start time and the end time. Our solution to processing power transfer follows this development. Basically, there are four principle operations in a video streaming proxy server: receiving and transmitting the packets from the network as well as receiving and transmitting the packets from the disk. These are the tasks that require real-time support. To provide the support, a contractual function scheduler was developed to guarantee the operation time and to specify the target stream for which the processing power is assigned. In addition, we have also developed a direct processing power transfer mechanism to provide a solution to the priority inversion problem.

1.2.2 Memory, Network, and Disk Bandwidth

One of the objectives of this work is to develop the mechanism that facilitates the heterogeneous interchange of resources usage including memory, network and disk bandwidth. A video streaming proxy server cannot support a video stream with a peak network bandwidth of 5Mbps if it has only 4Mbps available bandwidth; this server would be considered as full with respect to this video stream. However, some algorithms such as video smoothing [25][32] and video staging [5][21][31] can make use of this nominally insufficient bandwidth to support the video stream. This is because the traffic of a video stream may not maintain its peak bandwidth all the time. At times it falls below the peak, the video smoothing algorithm can pre-fetch the data required in the peak by storing them in the memory first. Such mechanism has demonstrated the importance of the concept of resource interchangeability to a streaming proxy server. When the network bandwidth of a video streaming proxy server becomes insufficient, it can make use of its memory to compensate for this insufficiency in order to achieve a better utilization. Not only network bandwidth and memory, disk bandwidth can also come into play when we integrate the technique of video staging with the video smoothing algorithm. Network bandwidth can be supplemented by disk bandwidth with the aid of memory, adding a degree of freedom to the

system. Apart from this, video staging also eases the cache or trash dilemma through the introduction of partial caching. In this work, a new video staging smoothing algorithm was developed to achieve the abovementioned objective.

A video streaming proxy server is required to handle multiple video streams concurrently. It has to guarantee the resource required by these streams by reserving them in advance. A scheduler is responsible for reserving these resources during admission control upon the acceptance of a new request. A simple scheduling scheme can issue these reservations based on the maximum requirement of each stream hence it will not be able to achieve a satisfactory utilization of resources, since the requirement of different streams is different at any instant of time. In view of this, suggestions such as CDL-EDF and CTL-round [43] were made to allow the resource requirement of all streams to be considered collectively. However, these approaches usually only focus on disk bandwidth but ignore the memory constraint. Furthermore, a laborious search is required to evaluate the disk bandwidth requirement in different period of time. As we have discussed above that resource interchangeability is the key to success for a video streaming proxy server, a simple admission control algorithm that facilitates a more global consideration of all resource requirement of all streams at all times is required.

In this work, we proposed an integrated admission control algorithm called grand scale scheduling. With some mathematical manipulation, we have determined the relationship between the grand scale schedule of video streams and their resource requirements under any specific period of time. To be specific, given the aggregate data volume to be retrieved of all streams and the volume of total memory buffer available to the system, the grand scale schedule is able to predict the required disk bandwidth in a specific period of time. This in turn allows us to reduce the admission control mechanism of a streaming proxy server into a simple peak value check over the disk bandwidth limit of the server across each of these periods of time.

Experimental results show that the proposed grand scale scheduling scheme can significantly increase the capacity of the video streaming proxy server by reducing resource wastage, which is indeed achieved by an implicit interchange of resources between different video streams. The resource requirement of a video stream varies from time to time; a resource required by a video stream may not be required by another stream at the same time. In the system point of view, if the

system can support the aggregate of their requirement at all time, then it is possible to admit all of them. Grand Scale Schedule is able to provide information in this way. In a specific period of time, it is able to provide a CDL like guarantee to every admitted video stream. Yet, it admits stream according to the aggregate of their actual requirements. Excessive guarantee over a resource given to a stream can be retrieved back and used to support another stream in demand of such resource. Resource usage interchange has been conducted implicitly in the grand scale scheduling scheme.

The development of grand scale scheduling and other similar scheduling schemes has challenged ideas about loading level in a video streaming proxy server. Under these algorithms, the loading of the server can no longer be represented by a simple number. Yet, we are still able to deduce a heuristic for estimating the loading of a server based on the grand scale schedule. With this heuristic, we observed two phenomena that are important to video stream admission. First, introducing startup delay to the admitted stream can improve the capacity of a proxy server. However, in general, it is not necessary to have an exhaustive search of the optimal startup delay. A random search, which can greatly reduce the searching time, can perform similarly compared with the optimal search. Second, in some specific cases, the acceptance of some streams may dramatically reduce server capacity. Based on these observations, we have developed several admission control strategies applicable to single server and multiple server platforms. Simulation results indicate that the employment of these admission strategies can further improve the capacity of a streaming proxy. The capacity improvement in the multiple server platforms is actually an extension of homogeneous interchange of resource usage. In this case, it is an interchange of server capacity. As the capacity reduction due to the admission of a video stream is different for different servers, we can improve the overall capacity by choosing the server with the lowest reduction for admission. As a result, the capacity of the other servers can be used to accommodate more streams.

1.3 Summary

We believe that interchanging resource usage is the key to capacity improvement for a video streaming proxy server. In this study, we have investigated the possibility of processing power transfer, which is a kind of homogeneous interchange of resource usage; and joint resource allocation of network bandwidth, memory and disk bandwidth, which is a kind of heterogeneous interchange of resource usage. Based on the concept of interchanging resource usage, a new set of

admission control schemes was proposed. The improvement in server capacity as shown in the results has clearly justified our claim.

The content of this thesis is organized as follows. Chapter 2 details a literature review of recent research in video streaming proxy services and background information. A solution to the problem of processing power transfer can be found in Chapter 3. The approach for the realization of heterogeneous resource interchange is described in Chapter 4. A new set of admission control strategies based on grand scale scheduling is covered in Chapter 5 and 6. Finally, we draw the conclusions on this work in Chapter 7. Suggestions for further development can also be found in this chapter.

We have constructed a video streaming proxy server and implemented some mechanisms in a Linux based PC machine equipped with an Intel 600MHz CPU; 256MBytes RAM; and interconnected with a 100Mbps LAN.

2 Literature Review

Developments in digital video and Internet technology have fostered the development of video streaming technology, as digitalized video can provide a higher fidelity compared with its analog counterpart. Furthermore, the Internet has provided a readily available and massively interconnected communication channel. Video broadcasting is no longer the monopoly of a few franchised television stations. At present, any person can set up a private station with affordable equipment such as a PC; with the 3rd generation mobile communication system, to communicate via video is as simple as doing so via audio. In the future, multimedia and interoperable communication will continue to be the trend in development. Visual, audio and textual information will be combined to enable precise communication in different dimensions. In addition, interoperable operations such as Text-to-Speech and Speech-to-Text will also be equipped to allow rendering in different platforms. Among these, video streaming is always the most resource intensive, and thus one of the most challenging topics to investigate.

Consequently, numerous researches have already been carried out in an attempt to enhance the performance of video streaming. In this section, we will cover these developments as well as the results obtained in studies of video streaming servers and proxy servers. Firstly, we will discuss a real-time processing enabling platform which is an essential component to these servers. Then, we will cover topics on video smoothing, staging and disk scheduling, which are involved in resource management. We will focus our study in on-demand video streaming rather than live broadcasting video streaming. They differ, in the view of a streaming proxy server, in their predictability. Video streaming proxy server knows the resource requirements of an on-demand video before its admission and this provides a room for better management of its resource usages. Conversely, the resource requirements of a live video are unknown. Management of its resource usages is difficult if not impossible.

In addition to the areas covered below, there are various others techniques such as stream patching and merging [88][89][90][91] under rigorous development. They are essential tools in video streaming proxy server development; however, their features in resource usages interchange require additional investigation. Therefore, discussion on them will not be covered.

2.1 Contractual Computing

Processing power is a resource essential to any computer application. Its importance to real-time application is even greater. Most operations in a real-time application are time-dependent. On top of the causal relationship, the execution of these operations is also confined by a period contained by launch time and deadline. Undesirable effects may occur if they fail to meet these parameters. Video streaming is also a real-time application – a soft real-time application, to be precise – and some of its operations are time-dependent. Taking disk access as an example: if the access has been executed too late, data starving will occur at the client side; however, if it has been executed too soon, buffer overflow will occur at the server. Therefore, real-time support is required in video streaming. SMART [65] and Rialto [60] were two of the most successful platforms in this area. They have demonstrated that it is possible to provide real-time support for video streaming on an inexpensive, general purpose personal computer. They have already been adopted for commercial use.

Another attempt to provide a similar solution was based on the contractual computing paradigm. Lam and Li have proposed their contractual paradigm, emphasizing explicitness and guarantee over resource usage in a clustered computing environment [13][14][15][16]. Its objective is to improve efficiency and quality-of-service. It focuses on resource management among computers within a clustered environment and it dictates that any resource that is explicitly manageable, and has a sense of ownership and support for ownership transfer can be turned into a "contractible" resource. On the other hand, resource managers can request their subscriber to submit their demand for resource usage explicitly. As a result, the resource manager can match them up and allow different subscribers to share the resource in a guaranteed way by forming different contracts. In addition, this also promotes a sense of ownership that gives rise to the possibility of ownership transfer. With these two features, the contractual paradigm facilitates more efficient resource management in a clustered environment.

2.1.1 Contractual Process Scheduler

In order to construct the real-time platform, the processing power has to be transformed into a "contractible" resource. The core component of this transformation is the contractual process scheduler [18]. Under this scheduler, processing power required by a process is expressed in the

trio of (start time, end time, percentage) , which forms the terms of a processing power contract. The process has to submit a contract to the system and seek a guarantee before execution. Once guaranteed, the system is obligated to deliver the specified processing power to the process according to the contract, regardless of other process activities. The resource manager may adjust its policy in the granting of a guarantee, with a base line of not overbooking the resource.

Although the contract allows processes to specify their processing power requirements as a percentage within a contract, this is impractical; after all, there is always one instruction (process) executing at any time within the processing unit. The finest scale executing schedule is in the form of 100% delivery between the start time and end time. Therefore, the contractual process scheduler adopts a schedule table that consists of fixed length timeslots with 100% processing power to express the finest scale executing schedule. To bridge the gap between contracts and the schedule table, the resource manager assigns a cycle period so that the number of slots required by processes during the cycle period can be calculated and allocated. Each of these timeslots can be allocated to one process only. Once allocated, the system guarantees the execution of this process during the corresponding period. The sense of ownership of the usage of the processing unit is an extension of this one-to-one relation, whilst a transfer of ownership is merely a reallocation of the owner to the timeslot.

2.1.2 Priority Inversion

For various reasons, processes may have to communicate between themselves. In case a signal sender requires an immediate response or action from the receiver, the system should grant the receiver execution rights along with the signal. However, signal transfer bears no relation to execution right transfer in the traditional scheduling paradigm. Worse still, suppose there is a priority difference between these two processes, a phenomenon such as priority inversion may occur [19]. For example, suppose there are three processes, A , B and C , with a priority level $A > B > C$. Process A has been scheduled for execution between times 0 and 10, whereas processes B and C have the same deadline. Suppose process A sends a signal to process C at time 5 and blocks itself from the reply. Since process B has a higher priority than process C , process B will be executed first. Although process A has the highest priority, it cannot execute until process B finishes. Based on the contractual paradigm, the problem can be solved by

transferring the timeslot allocated to a process to another directly through group scheduling [75]. Group scheduling allows several processes to form a group. When one of the processes within this group changes to the yield state, indicating that it is waiting for a signal from other group members, the unused timeslots in this process will be scheduled only to its group members such that the soonest response can be assured. This mechanism allows not only a real-time signal transfer, but also direct processing power transfer between processes to ensure the shortest response time.

2.1.3 Summary

The importance of processing power management to a video streaming proxy server is obvious. To guarantee quality-of-service, the proxy server has to operate on a real-time platform. A wide range of choice is already available in today's state-of-the-art technology; even a low-end personal computer can fulfill the requirements easily.

2.2 Video Smoothing

Apart from processing power, there are other essential resources required to support video streaming. These include memory space, network and disk bandwidth. We will focus our discussion first on network bandwidth. Variable-Bit-Rate (VBR) video streaming has already become the mainstay of the video streaming service in recent years, since it can provide constant quality video playback, even for a motion intensive scene. However, it also creates difficulties in resource management, as those resources required to support it are no longer constant. Network bandwidth is one of these resources.

The worst and most trivial solution for handling network bandwidth management in this case is to assume that every video stream will consume its peak network bandwidth for its duration. In this way, network bandwidth management has been reduced to a simple accounting problem. But most of the network bandwidth will be wasted. To tackle this problem, solutions such as deterministic and statistical multiplexing and e-PCRTT [42] have been suggested. When there are multiple video streams, their peaks may not occur at the same time, so it is possible to multiplex their usage. Deterministic multiplexing attempts to seek a fit for every video stream; therefore, the result is always feasible but its operation will be time consuming. Conversely, statistical multiplexing relies on statistical parameters to decide whether multiplexing can be carried out. As a result, it may be infeasible.

On a broader view, multiplexing schemes have provided a means for homogeneous resource usage transfer. Network bandwidth which had been allocated for one video stream has been transferred to another. This transfer facilitates a better utilization of network bandwidth, therefore increasing the capacity of the proxy server. However, it has only provided a means for better utilization of wasted network bandwidth; the overall requirement in network bandwidth has not changed. The network could still become a bottleneck with incremental rises in capacity. Intuitively, we can ease this by reducing the overall requirement, which can be done with heterogeneous resource usage transfer by turning the requirement in network bandwidth into a requirement of other resources.

2.2.1 Bit-Rate Smoothing Algorithm

There are other means for network bandwidth management besides multiplexing schemes. Bit-rate smoothing is one of them [76][77][78][79]. Rexford et al. have proposed a bit-rate smoothing algorithm [32][33] for proxy servers. The structure of the Rexford model consists of a series of proxies in tandem. The idea is to transform an originally fluctuating variable-bit-rate video stream into a comparably smoother near constant-bit-rate video stream which is more favorable to the actual networking infrastructure. By doing this, network bandwidth management can again be reduced to a simple accounting problem.

However, this transform is not achieved without cost. Memory buffer is required to absorb the variation by means of pre-fetching. The greater the memory buffer employed, the greater variation can be absorbed and the smoother the video stream that can be obtained. Its operation is illustrated in Figure 1. It is obvious that the bandwidth requirement at slot 5 of the original network schedule is higher than our target network bandwidth. Suppose we smooth this schedule in the form as shown in the smoothed network schedule. We have effectively pre-fetched those extra data blocks in slot 5 to earlier blocks at slots 1, 2, and 4. Initially, this does not require any additional memory to support the operation. However, after we have smoothed the schedule, we have pre-fetched some data blocks in advance that will not be sent out immediately, so they must be stored in the proxy server. This requires extra memory to support the operation.

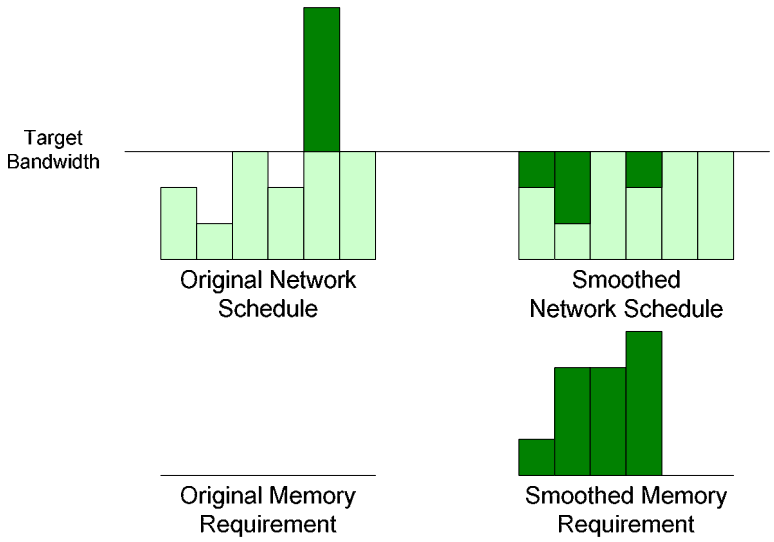


Figure 1: Network versus Memory

2.2.2 Scalable Video Coding

Bit-rate smoothing operates on the delivery schedule of a video stream alone; it will modify the delivery schedule, but the content and quality of the video will not be affected. However, recent technology has brought in a new method of network bandwidth management which can accomplish its task by modifying the video content or quality – scalable coding.

Traditional video coding encodes each frame in a video stream into one bitstream. When the data in this bitstream is corrupted, a part or even the whole frame will not be able to render itself correctly. Scalable coding encodes a frame into several streams of different quality. Unless the bitstream of the lowest quality is corrupted, any corruption can be remedied by rendering the bitstream with lower quality. In addition to the application of error resilience, scalable coding also provides an opportunity to network bandwidth management.

2.2.3 Summary

With Rexford's algorithm, a proxy server can utilize its memory and network much better to support more clients concurrently. Yet, memory buffer is also a limited resource in the proxy server. When memory buffer runs out, the server will also have reached its capacity. Therefore, we have actually performed a heterogeneous transform from the usage of network bandwidth to that of memory buffer. In addition, the network is the only transportation medium Rexford's algorithm considers. Incorporation of another essential resource, disk bandwidth, is still in need.

2.3 Video Staging

One of the most important differences between web proxy servers and video streaming proxy servers is in the size of their serving objects. Text and image files make up the major parts of objects served by web proxy servers, and their size is merely a few megabytes. However, objects in video streaming proxy servers are video streams; their size could easily reach several gigabytes – about a thousand times that of text and image files. To cache and trash such an enormous object makes a substantial demand on processing time as well as disk bandwidth. As a result, it becomes impossible to apply the original cache-all and trash-all strategies. Zhang et al. [5] propose an alternative solution – video staging.

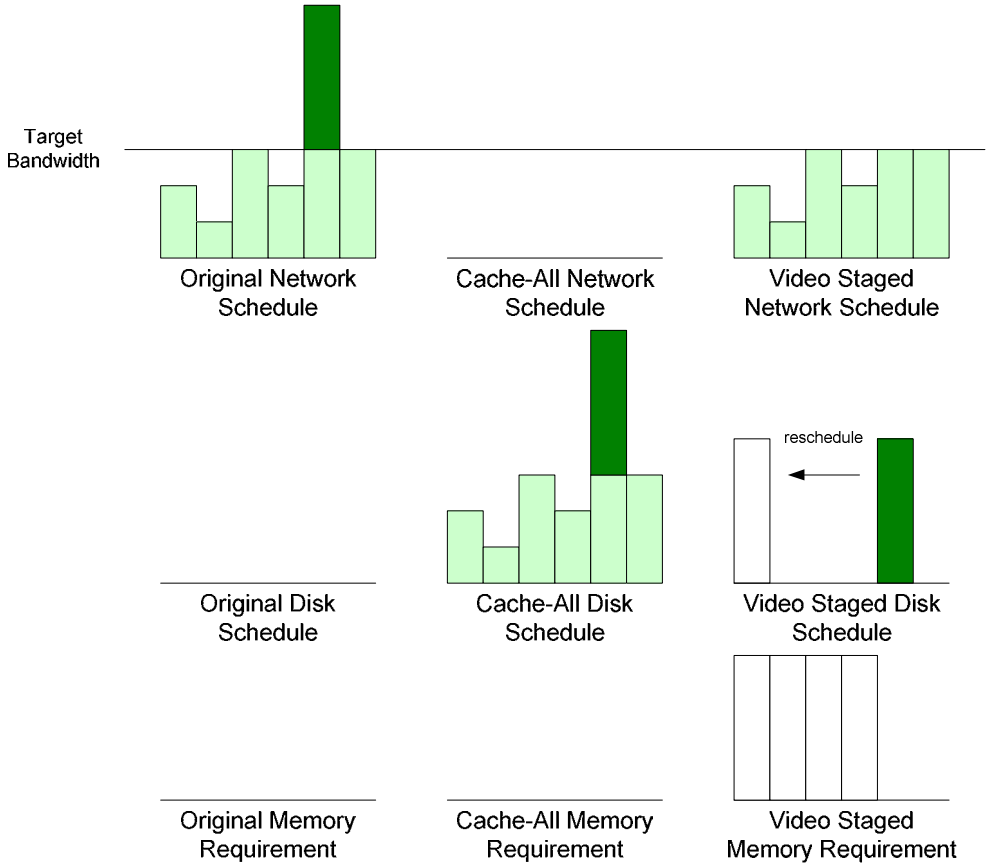


Figure 2: Network versus Disk

Instead of caching or trashing the whole video stream every time, video staging offers a different solution through partial caching or trashing. Its operation is illustrated in Figure 2. First, we have a target bandwidth over a network schedule. Without the assistance of video staging, we

have to apply the cache-all strategy, shown in the middle column. During cache out, network bandwidth is no longer required, as the whole video stream can be retrieved from the disk. Suppose we can construct a perfect disk schedule – then there is no additional memory requirement. However, with video staging, only a part of the video is stored in the disk cache of the proxy server. Therefore, during cache-out, it requires both data from the network and from the local disk to reconstruct the video stream. As a result, we have both a network schedule and a disk schedule. In proposal of Zhang et al. [5], we may select the target network bandwidth as a threshold bandwidth and divide the network schedule into two parts. In this way, the lower part is upper-bound by the threshold bandwidth and we have achieved a similar result as that provided by video smoothing, restricting the network bandwidth usage and reducing its management to a simple accounting problem. The upper part of the stream is treated in a different way. It will be retrieved from the disk as shown in the video staged disk schedule in the diagram. Similarly, we may construct a perfect disk schedule so that we do not require any additional memory. However, disk schedules obtained from different video streams may overlap, necessitating the application of a reschedule to offload the disk. This is similar to the pre-fetching operation in video smoothing. For example, as shown in the figure, if we have rescheduled the disk access from slot 5 to an earlier time slot, 1, we will require additional memory to hold these pre-fetched data from slot 1 to slot 4, as shown in the video staged memory requirement.

2.3.1 Summary

Video staging has been devised as a solution to the problematic cache-all or trash-all strategy in video streaming proxy servers. However, it has also provided a means for heterogeneous resource usage transfer between network and disk bandwidth. By means of a threshold bandwidth, it can divide a video schedule into a network bandwidth constrained schedule and a disk schedule. In practice, during second level scheduling, memory has to be used to retain those pre-fetched data due to rescheduling. This makes video staging a mechanism supports usage transfer between memory, network and disk bandwidth.

2.4 Disk Scheduling

The caching and subsequent retrieval of a video stream within a video streaming proxy server makes the disk scheduling problem no less simple than for a video streaming server. Unlike in a network, the operation of a disk still requires mechanical motion – the rotational motion of the disk

and the lateral motion of the head. When the server is accessing the disk in one region, it has to take some time before the head can move and access the other region. This makes multiplexing of disk access impossible.

2.4.1 Constant Data Length and Constant Time Length

To solve the problem, Chang et al. have devised a pair of solutions. These are CDL (Constant-Data-Length) and CTL (Constant-Time-Length) schemes. These schemes make use of memory buffer to relax the temporal relationship between the disk access and the delivery schedule. We will consider the operation of the CTL scheme first. Suppose we have an original disk schedule as shown in Figure 3 and there is yet another video stream coming, as shown below. If we try to serve these two video streams concurrently and without any preprocessing, we will obtain an overall schedule as shown in Figure 4, where disk usage is multiplexed between these two video streams. However, as these two video streams may reside on different tracks, it is necessary to find the track whenever the service target is changed. This will not only inflict performance degradation but also cause a reduction in the disk's operational life.

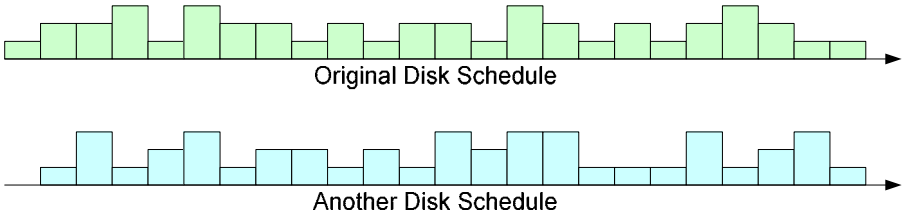


Figure 3: Individual Disk Schedules

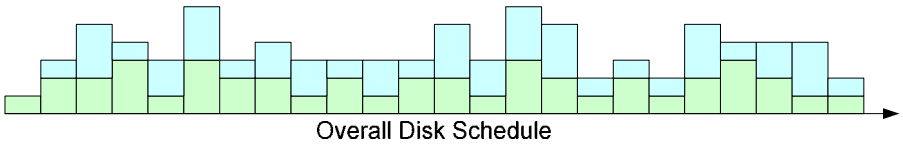


Figure 4: Overall Disk Schedules

Under the constant time length scheme, the original disk schedule will be rescheduled as shown in Figure 5. The original schedule will be divided into many regions with a constant period τ . Then, all data blocks that need to be read within this region will be rescheduled to the front of their corresponding region, as shown in the diagram. The time to retrieve a data block is no longer equal to its delivery deadline. Therefore, we have to employ some memory buffer to hold these data, which leads to a non-empty memory schedule, as shown in Figure 6.

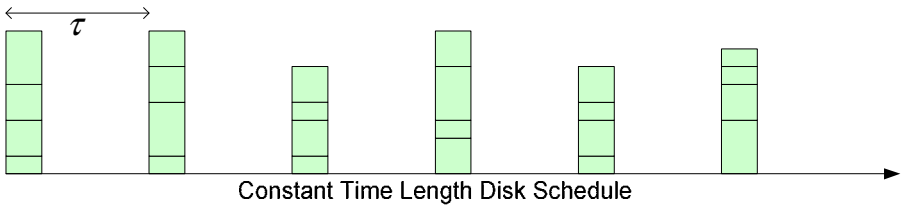


Figure 5: CTL Disk Schedule

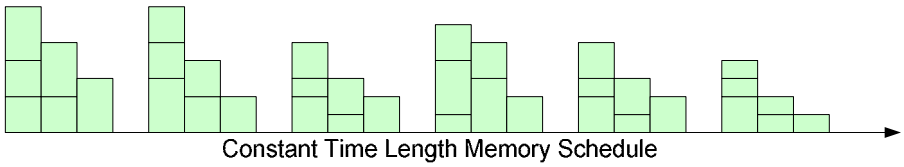


Figure 6: CTL Memory Schedule

Similarly, we can apply the CTL scheme to the other disk schedule and combine them, giving us a rescheduled overall disk schedule, as in Figure 7. After this rescheduling, all data blocks within the region are rescheduled to form a large block. In this way, we can first retrieve all the data blocks from one video stream, followed by the others, so that we can reduce the numbers engaged in track seeking. Admission control in the CTL scheme is rather simple, as the period is fixed; we only need to measure whether the total volume of data accessed in this region has exceeded the sustainable bandwidth of the disk.

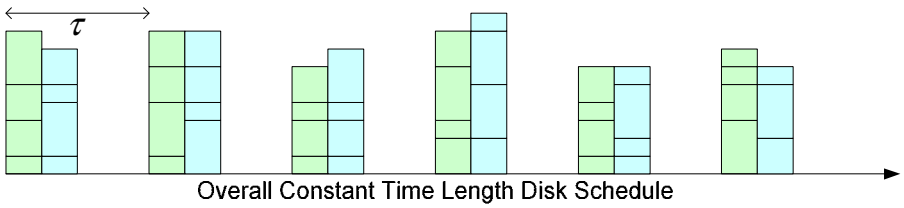


Figure 7: Overall CTL Disk Schedule

For the CDL scheme, data blocks from one video stream will be rescheduled to a larger block until the size of this larger box achieves a fixed size b . As a result, the time between these larger blocks are different. Similar to the CTL scheme, this also requires a memory buffer to hold those pre-fetched data, as shown in the memory schedule in Figure 9.

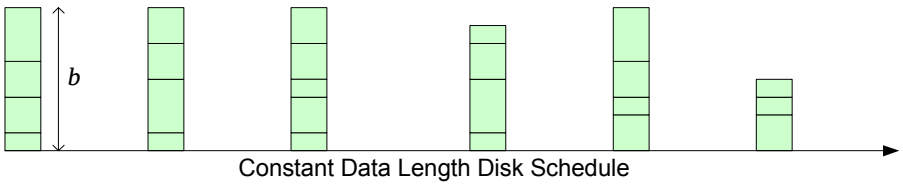


Figure 8: CDL Disk Schedule

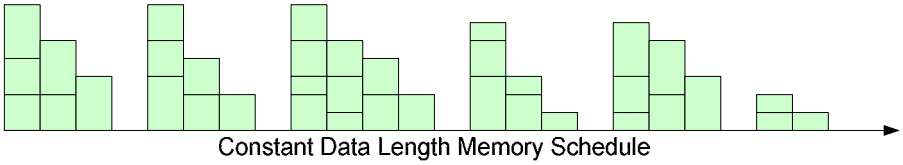


Figure 9: CDL Memory Schedule

If we combine these two schedules together, we will obtain an overall schedule, as in Figure 10. The number of seeks is reduced as in the CTL scheme. Admission control must be conducted in order to decide whether the disk bandwidth is capable of supporting the overall schedule.

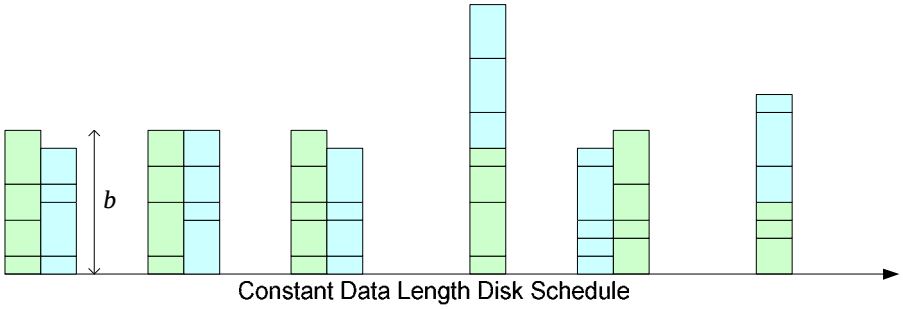


Figure 10: Overall CDL Disk Schedule

2.4.2 Generalized Constant Data Length

Both CTL and CDL schemes provide a practical solution to allow a disk to serve multiple video streams concurrently. However, they have also sacrificed a degree of freedom from the solution to the scheduling problem. Taking the CDL scheme for an example, as it has a constant data length, the buffer size required to serve the video stream is always constant. In a variable bit-rate video stream, the variability of bandwidth will therefore be reflected in the sparseness of disk access in the CDL disk schedule, as shown in Figure 11. In region (t_0, t_1) and (t_2, t_3), as the separation between disk accesses is low, the bit-rate of the video in these regions will therefore also be high; conversely, in region (t_1, t_2), the bit-rate will be comparatively low. The estimated

memory requirement as shown in Figure 9 and Figure 12 shows that the actual memory requirement will drop between disk accesses. As a result, when the distance between these disk accesses enlarges, the utilization rate of memory will drop. To reduce this loss, the GCDL (Generalized Constant Data Length) scheme is proposed [40]. Under GCDL, the memory buffer size allocated to a video stream will not remain constant. As shown in Figure 13, during the period (t_1, t_2), the allocated buffer size is reduced to $\frac{b}{2}$ by halving the distance between disk accesses. Less memory is then required to support the same number of clients. We are able to make use of this relinquished memory to support more video streams. A similar scheme, GCTL [40], operates on this principle.

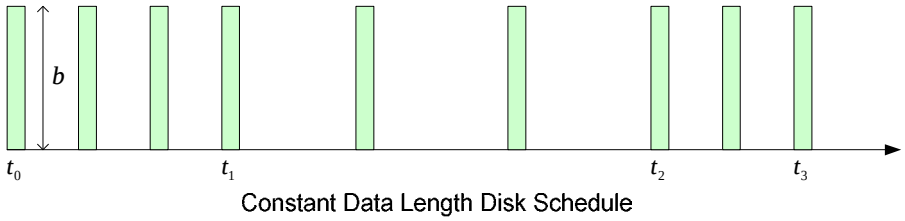


Figure 11: Uneven CDL Disk Access

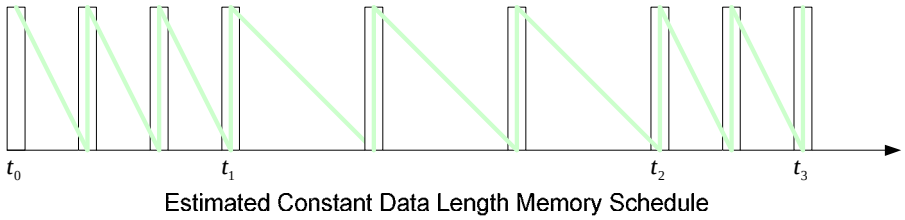


Figure 12: Uneven CDL Memory Schedule

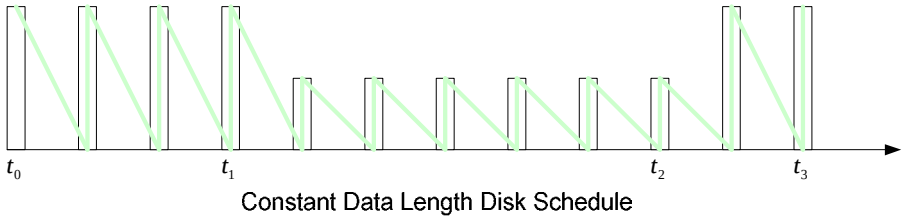


Figure 13: GCDL Memory Schedule

2.4.3 Summary

However, these scheduling schemes do not naturally lead to a boost in capacity, since the maximum required buffer size to support a video stream may remain unchanged. As the usage

reductions are scattered around, a comprehensive search is required to locate and utilize them. CDL and CTL schemes have provided a form of heterogeneous usage transfer between disk bandwidth and memory buffer. With the aid of the memory buffer, we can relax the temporal relationship between disk access and delivery schedules. This allows us to group some data blocks from the same video stream to form a larger one, so that we are not required to multiplex the disk usage with the other streams which consumes extra seek times.

3 Computational Resource

Any computer application requires computational power to execute. This computational power can be expressed in terms of processing time – the time the processing unit will take to execute instructions for this application. When there are multiple applications executing concurrently in a multiprocessing system, the processing unit will divide its time into many short periods, allocating each of these period to serve one application at a time. This is the philosophy behind a general time-sharing system. However, such division and allocation does not guarantee an application will be served in any specific time. When the execution of some operations in an application is time-related, such as in the control system in an airplane, then it should not be deployed in a general time-sharing platform. Another solution is required for these kinds of applications. Unfortunately, the video streaming proxy server is one such application.

3.1 Video Streaming Proxy Server

Video streaming [1][2][3] has become a hot topic in the Internet world recently. This technology could be employed to view stored or live television and radio programs via the Internet. Despite the great pleasure this technology can bring, it also produces a large amount of network traffic, especially when a vast audience views their favorite programs when they are stored in different media servers. A unique connection is usually required to support one client, but an Internet service and media content provider would require a more economical solution. In view of the possibility that some of these audiences might have similar requests over the course of the same program, the network traffic could be reduced by sharing video streams. This is the goal of the video streaming proxy server [4][5].

A video streaming proxy server usually resides between the Internet backbone and the subscriber leased line. It is responsible for the collection of all video streams coming from the Internet and their subsequent redirection to the corresponding subscribers. There are essentially two types of streaming proxy servers. The first type is proxy and the second type is reverse proxy [6]. Their classification depends upon client awareness. An Internet service provider usually installs a proxy server between the Internet backbone and its service subscribers. Clients are required to specify this proxy to their application explicitly in order to enjoy the provided functionalities. On the other hand, media content providers install reverse proxies to provide several different sites for their clients. Reverse proxies are transparent to clients. When a client

tries to connect to a media server, they might in fact be connecting to a reverse proxy, which will redirect requests to the actual server.

In designing a video streaming proxy server, one of the most challenging tasks is the provision of services upon request. Since user requests on video streaming services are usually sporadic, their occurrence and volume are unpredictable. This requires a dynamic scheduling scheme to handle the computational power management. Moreover, due to the soft real-time nature of video streaming services, a dynamic real-time resource scheduler is required to complete the task. However, a low-end computer cluster equipped with a traditional general-purpose operating system is incapable of this. As an example, Linux is an operating system employing a round-robin process queue for process scheduling. Previous studies have shown that this does not guarantee any explicitly specifiable form of timing required by a real-time process [11]. Recently, we have shown [12][18] that a contractual operating system designed through the adoption of the contractual paradigm is capable of providing the required real-time support using a low-end Linux cluster. To examine this possibility, we have designed and built a server under this paradigm.

3.2 Architecture of MPS

We have named the video streaming proxy server that we developed MPS. It is a reverse proxy. The target platform of MPS is an inexpensive personal computer cluster running on Linux and interconnected with a local area network. It supports international standard protocols such as Real-Time Streaming Protocol (RTSP) [7], Real-Time Transport Protocol (RTP) [8][9], Real-Time Control Protocol (RTCP) and Session Description Protocol (SDP) [10]. Users may configure MPS as a media stream gateway, reflector or mirror. Currently, it supports as many as 50 concurrent accesses.

We adopted the contractual paradigm to design the contractual function scheduler for the provision of real-time resource scheduling services. Each function is allocated a contract specifying its execution profile including its launch time and frequency. The contractual function scheduler is responsible for the granting of these contracts and the fulfillment of its guarantees. Moreover, we have developed a group scheduling mechanism to reduce the loss caused by under-utilization of a contract. This results in a substantially improved performance, particularly when both time-constrained and non-time-constrained processes coexist within the MPS.

The architecture of MPS is comprised of three major subsystems: the Control Flow Engine (CFE), the Data Flow Engine (DFE), and the Cache Engine (CE), as shown in Figure 14. The whole proxy server consists of one CFE and at least one DFE and CE. CFE has two main duties. Its first duty is to establish the RTSP communication channel between the media server and client. When a client requests a video service, an RTSP communication channel is set up between the client and the proxy server. The CFE will then redirect the RTSP commands made by the client to the media server. The second duty of CFE is to issue directives and conduct load balancing for DFE(s) and CE(s). Because neither the client nor the media server accesses DFE(s) or CE(s) directly, CFE is required to inform those DFE(s) regarding the settings of ports and their corresponding operation status. It also informs the corresponding CE(s) to look for the appropriate media cache for data storage and retrieval. Finally, it detects any overloading in DFE and avoids it through load balancing.

DFE deals with the actual media data transfer. As video streaming is real-time in nature, DFE must immediately send all the data in a media stream received from the server to its corresponding client. When it serves several client-server pairs concurrently, organizing the processing precedence among these pairs requires a form of real-time scheduling scheme. Moreover, it has to establish a policy of admission control in order to guarantee customers their services. Otherwise, some or even all these clients will not be able to receive their media stream in time, as they might not receive service when the loading of the proxy server reaches the limit. In such cases, the only solution for increasing the capacity is to share the loading among more machines, which involves software scalability. To achieve this in MPS, multiple machines can own their DFE(s) under the control of a single CFE. The CFE is responsible for assigning these client-server pairs to different machines according to the load-balancing strategy as well as their loading status. Lastly, CE is responsible for the caching of the media streams and their retrieval upon cache hit. It serves as a large data storage area that allows multiple accesses from different DFE(s).

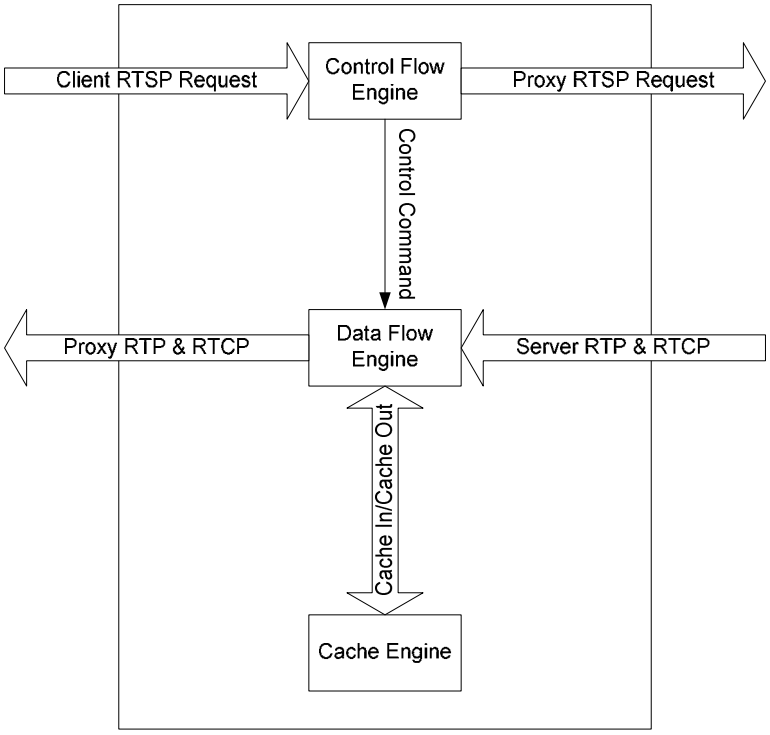


Figure 14: Perspective of the Video Streaming Proxy Server

3.2.1 Basic and Primitive Operations

In our design, DFE/CE is capable of three basic operations: gateway, reflecting, and mirroring. These are assembled by four primitive operations: *InetRx*, *InetTx*, *DiskRx* and *DiskTx*. Primitive *InetRx* is responsible for receiving the packets from the Internet and writing them to the buffer. Primitive *InetTx* is responsible for reading packets from the buffer and sending them to the Internet. Primitives *DiskRx* and *DiskTx* perform similar functions on disk. In MPS, we implement them as member functions of an object.

To perform as a gateway, MPS must serve as an interface between clients and servers only. It redirects every packet received from the server to the client. In this case, *InetRx* receives packets from the media server and *InetTx* sends the packets to the client. Communications between these two primitive functions through their corresponding buffers, S-Buffer and C-Buffer, are illustrated in Figure 15.

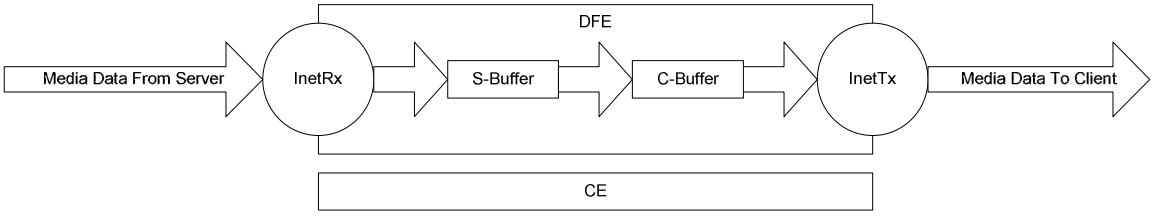


Figure 15: MPS working as a gateway

Performing as a reflector, MPS shares the media data received from the server among two or more clients in real-time. In this case, it keeps the data received by *InetRx* in the S-Buffer. For every client that requires data, data in the S-Buffer will be loaded to the C-Buffer. Then, *InetTx* will read them into the C-Buffer and send them to each client. Hence, different clients can independently obtain their video streaming services. Figure 16 gives an illustration of such an operation.

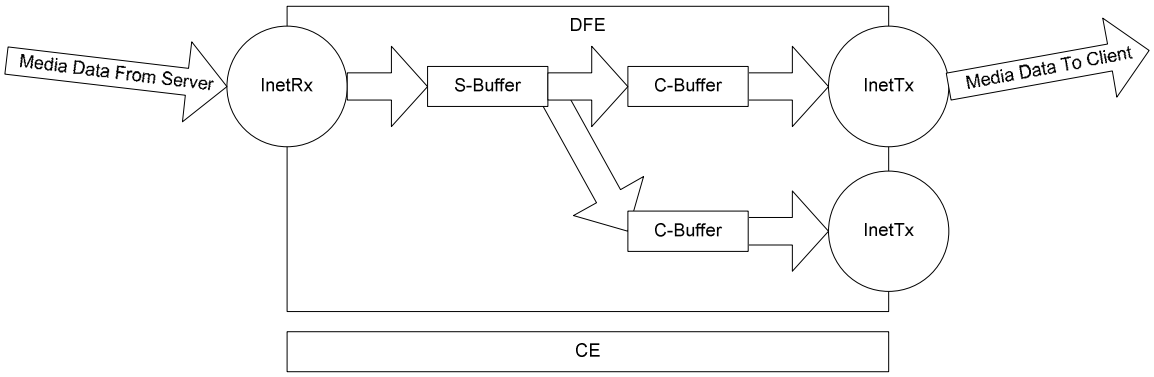


Figure 16: MPS working as a reflector

With mirroring, a copy of the media stream received from the server is stored in the cache at the same time it is sent to the client. MPS can then retrieve the cached copy later upon the request of other clients without consulting the original server. This is similar to the reflection operation, with one of the *InetTx(s)* replaced by *DiskRx*, such that those packets will be stored on the disk. When the client retrieves data, MPS conducts a modified gateway operation with *InetRx*, replaced by *DiskTx*. These operations are illustrated in Figure 17 and Figure 18 below.

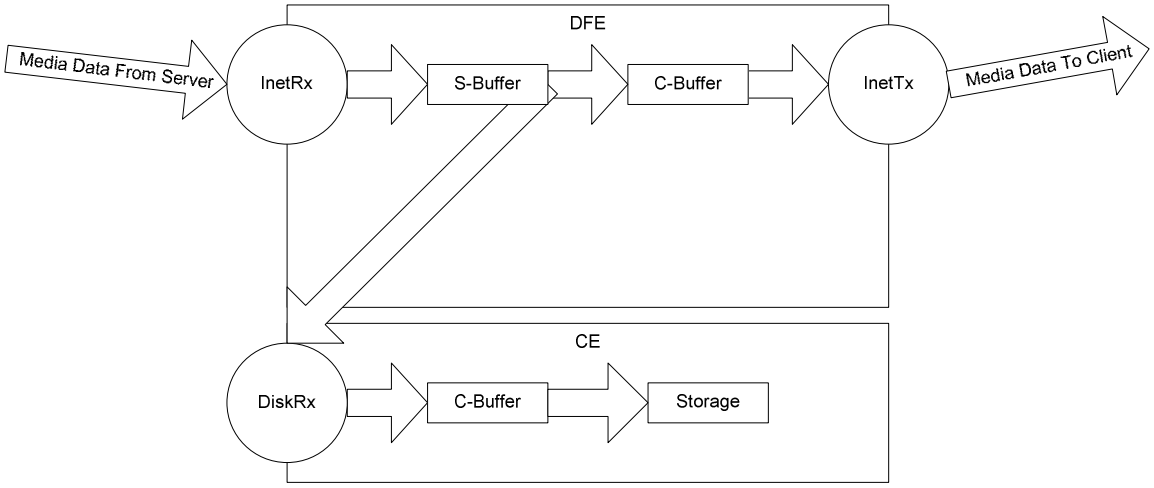


Figure 17: MPS working as a mirror (cache in)

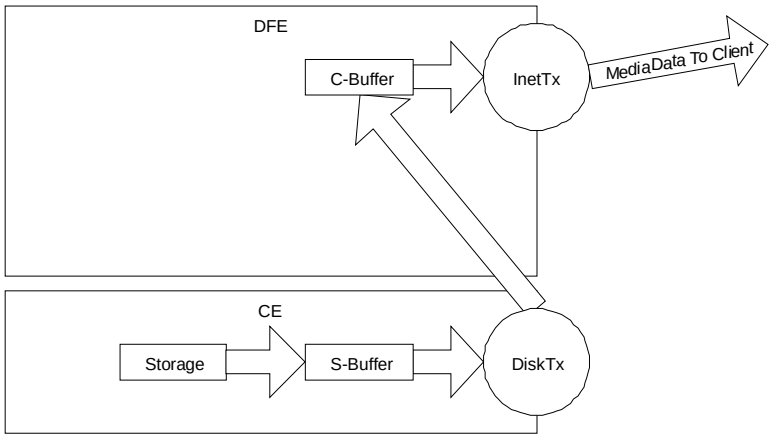


Figure 18: MPS working as a mirror (cache out)

Apart from those four primitive operations related to the reception and transmission of packets as mentioned above, we also implement an additional primitive operation, *Transcode*, in MPS. *Transcode* allows the modification of media data when they pass through MPS. This is very useful when, for instance, there is a need to change the media data format or to filter certain data during transmission.

3.2.2 Task Scheduling of MPS

Media streaming is a soft real-time task for which there are rigid timing requirements in processing the data. These timing constraints introduce different scheduling requirements to those of the abovementioned primitive operations in our system. First, let us consider the gateway

operation where two primitive operations, *InetTx* and *InetRx,* are involved. On one hand, in order to increase the throughput in the system, the best schedule to invoke *InetRx* is to match it with the incoming schedule of packets sent from the server. On the other hand, we want to invoke the corresponding *InetTx* immediately after a successful *InetRx* in order to minimize the time taken to retransmit the packet. For reflection, we do not only invoke the original *InetTx,* but also other *InetTx(s)* that require the sharing of streams. These invocations require a timely transfer of signal from *InetRx* to *InetTx* plus an immediate reaction from *InetTx* as well. To enable all this, we require a timely transfer of signal and execution right.

Situations become more complicated when *DiskTx* and *DiskRx* become involved. When the proxy server caches a video stream, it executes *DiskRx* and stores incoming packets in its local storage. However, unlike *InetRx,* the time taken by *DiskRx* is not unique all the time. The proxy does not need to store each packet in each *DiskRx* execution; it can wait until it has collected a bunch of packets and then write them to the storage device at one time. The time taken for packet collection and storage is different, thus they require different scheduling criteria to produce the correct schedule. During the caching out operation, *DiskTx* is responsible for the retrieval of packets from the local storage device and their delivery to *InetTx* for transmission. Unlike *InetTx, DiskTx* does not require any checking of the availability of packets upon every execution. Instead, it can retrieve the next packet first, examine its available time, and then suspend itself until the time arrives. To support these activities, our operation scheduler in the proxy has to provide the support of a multiple scheduling parameter set and a dynamic, non-delayed wakeup mechanism.

The *Transcode* operation requires real-time support in a less rigid way than *DiskRx* or *InetRx*. It has a certain deadline, but the system has the flexibility to perform the task earlier or later, as long as it satisfies that deadline. Nevertheless, a general task scheduler cannot fully exploit this kind of flexibility. In the worst case, since the *Transcode* operation usually takes a relatively longer computation time, it can ruin the timing of other primitive operations.

Most non-dedicated computer operating systems, including *Linux*, adopt the round-robin scheme or its variants to implement their task schedulers. Entities may leave or join the system dynamically by a removal or appending operation over the list. The scheme allocates each entity a fixed timeslot. In the simplest case, it schedules an entity again when it circulates the list for one

cycle. Therefore, the period between two consecutive executions of an entity is equal to the product of the number of existing entities and the size of a timeslot. This period increases as more entities join the system and remains unpredictable until run time.

However, the arrival of a data packet from the media server is relatively predictable and regular. For a data packet that contains a compressed picture frame to display within a certain time interval, the media server has to prepare and send the corresponding packet to the client within an appropriate time interval in order to avoid starvation in the client. Consequently, data packets become available to the client periodically. If we implement the proxy server with the round-robin scheduling scheme, its unpredictability will introduce much difficulty in ensuring reception of packets synchronously, in accord with their arrival time. In time, delay may occur and accumulate. Consequently, this incurs packet loss on the client side. Moreover, operations within a proxy server have different levels of importance; they can be prioritized and divided into three different levels. For MPS, the topmost level includes those packet reception operations such as *InetRx* and *DiskRx*. These two operations act as information intakes; they have the top priority, as all following operations rely on them. The second level contains operations *InetTx* and *DiskTx* and the other operations of the bottommost level. By making use of this relationship, we can implement a multiple-priority-queue in a round robin time-sharing system to improve its responsiveness to real-time service requests. In spite of this, the lack of notion in time remains its fatal drawback. The round robin time-sharing system will have problems in efficiently scheduling these operations, particularly when there is a combination of time-constrained (such as *InetRx* and *DiskRx*) and non-time-constrained operations (such as *Transcode*) to be performed.

In order to satisfy the real-time constraints associated with the arrival and delivery of packets, we need a dynamic soft real-time scheduler for our proxy server. Although it is possible for us to acquire a dynamic soft real-time scheduler from a dedicated real-time computing system, this would be a diversion from our goal, since the target platform of the proxy server is an inexpensive cluster of personal computers running *Linux*. Besides, it is inefficient to adopt a soft real-time process scheduler as the task scheduler in the proxy server due to the relatively short duration of the task. We shall elaborate further on this point later. In light of these considerations, we adopted the contractual computing paradigm to implement MPS.

3.2.3 Contractual Computing Paradigm

Lam and Li have proposed their contractual paradigm emphasizing explicitness and guarantee over resource usage in a clustered computing environment [13][14][15][16][17] to improve efficiency and quality-of-service. The paradigm focuses on resource management among computers within a clustered environment and it dictates that any resource that is explicitly manageable, respect a sense of ownership and enable ownership transfer can be turned into a "contractible" resource. On the other hand, the resource manager can request their subscriber to submit their demand for resource usage explicitly. As a result, the resource manager can match them up and allow different subscribers to share the resource in a guaranteed way by forming different contracts. This also promotes a sense of ownership that gives rise to the possibility of ownership transfer. With these two features, the contractual paradigm facilitates more efficient resource management in a clustered environment. This paradigm does not provide any feasibility test as compared with traditional real time system design [92][93][94][95][96]. Instead, it provides a underlying platform which these mechanisms can be built upon it to provide the necessary guarantee on processing power over a Linux platform.

3.2.4 Contractual Function Scheduler

Borrowing the idea of contractual scheduling for processes as discussed above, we have developed a contractual function scheduler to help us schedule the execution of different primitive operations in MPS: *InetRx*, *InetTx*, *DiskRx* and *DiskTx***.** The real-time nature of these primitive operations imposes a similar requirement on their scheduling as does the real-time process. Rather than directly scheduling these primitive operations using an operating system level contractual process scheduler [17], we have developed a user-level contractual function scheduler to reduce the overhead within the kernel for their execution. Each video stream, according to the service provided for it, has its own set of primitive operation. This contractual function scheduler is responsible to the scheduling of those stream specific primitive operations. This enables is to schedule operation *InetRx* of the first stream and the operation *InetRx* of another stream in a different way.

In fact, the time taken to complete functions such as *InetRx* and *InetTx* is no longer than 100 microseconds. This demonstrates that the proxy server must conduct function switching within a very short period of time. If we implemented these functions as process and scheduled them with the contractual process scheduler, the operating system would have to conduct context switching more frequently than normal. As a result, the overhead inflicted by such context switches would increase dramatically. By using a contractual function scheduler, the function switches are completed in a process such that the extra overhead can be very much reduced.

The first task to implement the contractual function scheduler is to construct a function schedule table. This table does not need to span the whole time period, but only a period of sufficient length; one second, for example. This is possible since all functions – *InetRx*, *InetTx*, *DiskRx* and *DiskTx* – are periodic. This table can always be expanded to span the whole period by repeating itself. We further divide this second into a cluster of fixed-length, continuous but non-overlapping timeslots. Then we can assign each of these timeslots to one primitive operation, and the scheduler will guarantee their execution during the timeslot. As this scheduler is non-preemptive, each of these operations has to be designed carefully so that it completes itself before intruding into the other timeslots. If an operation needs two timeslots for execution, we assign them with two consecutive timeslots. Figure 19 shows an example of scheduling a contractual function, A , together with some other round robin scheduling functions, B, C, D and E . Since function A has made a contract with the scheduler and has a record in the function schedule table in accord with the physical time, the execution of function A remains unchanged when a new function, *E,* launches, as shown in the figure.

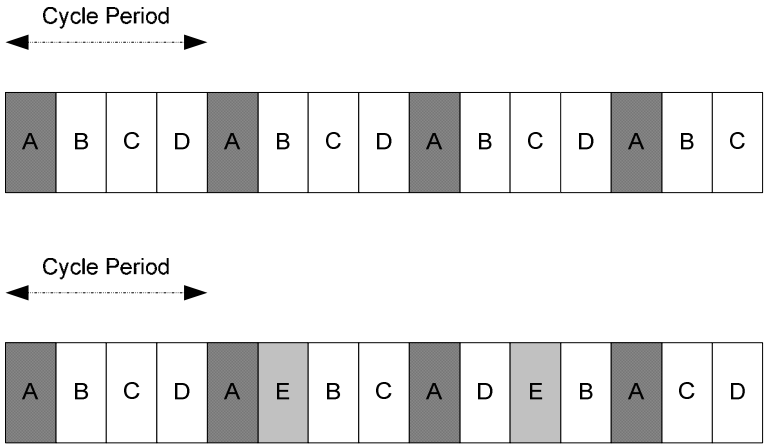


Figure 19: Contractual Scheduling

Figure 20 shows the flowchart of the contractual function scheduler. It takes up control upon the exit of a function. Firstly, it determines whether the system has reached the contract time of another function as specified by the contractual schedule table; if so, it will seek the next function in the table and select it for execution accordingly. Otherwise, it will transfer control to the round robin scheduler.

As a result, we can guarantee the computation power allocated to any individual primitive operation associated with the service to a certain video stream. This satisfies its real time requirement. Such real time requirement is evaluated in an adaptive manner. Initially, an operation acquired all its computation power through the round-robin scheduler. The requirement can be estimated according to this consumption and contract can be made to the contractual function scheduler and recorded in the contractual function schedule table. This table is designed to cover 1 second of time in our implementation. This period can be changed according to the actual requirement. And this table is also partitioned into many 100 microsecond long timeslots. Computation power is measured in terms of percentage. A 1% computation power contract will be interpreted as an occupation of timeslots in 0μs, 10000μs, 20000μs …while another 2% computation power contract will be interpreted as an occupation of timeslots in 100μs, 5100μs , 10100μs … . Each operation has to return from execution within its 100μs timeslots in order to allow the execution of the next operation.

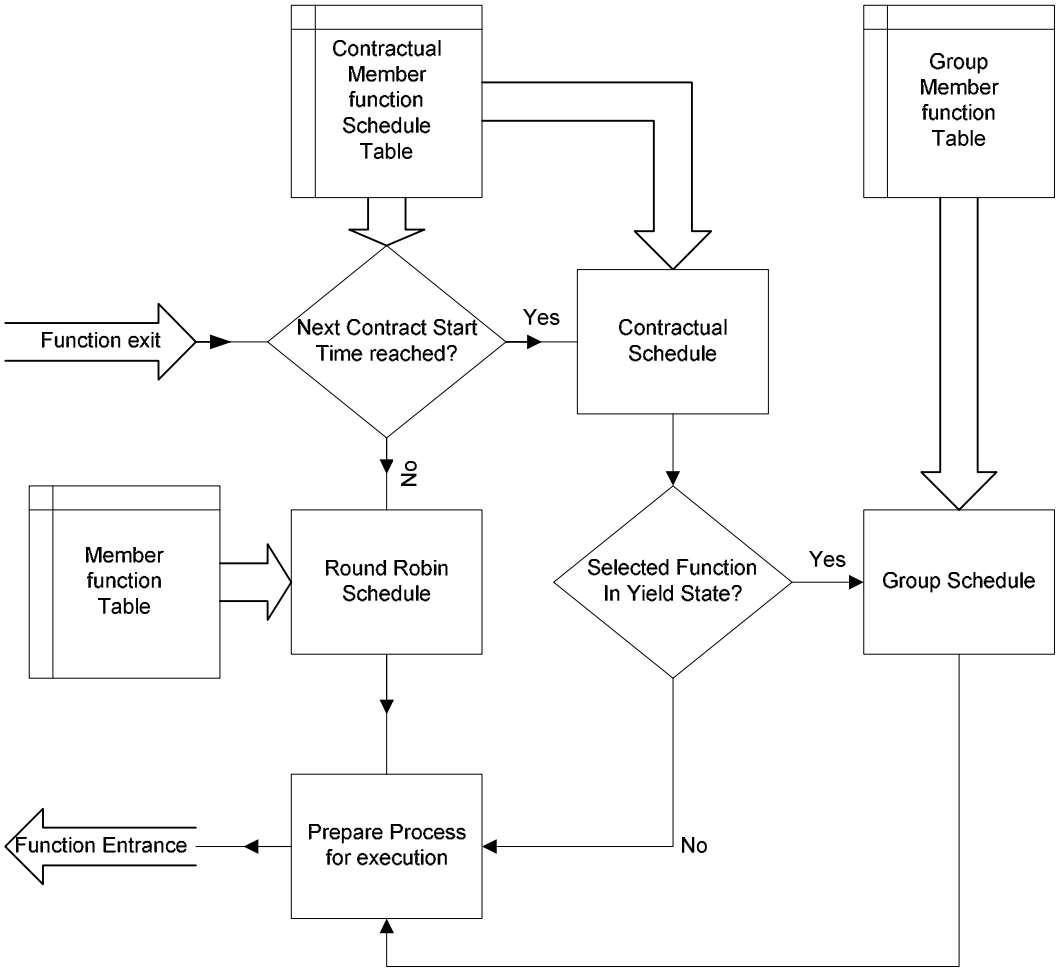


Figure 20: Contractual Function Scheduler

Earlier, we illustrated the problem of scheduling a group of processes with different priorities. In fact, similar situations might occur in scheduling functions within the proxy server. Let us take *InetRx* as an example. The easiest way to schedule *InetRx* is to match it with the servicing video stream's peak bandwidth. This is the minimum requirement that will guarantee *InetRx* receives all packets. In fact, most *InetRx* attempts do not receive any packets, as the video stream does not attain its peak bandwidth at all times. Moreover, video servers usually send out packets in bursts, followed by a period of silence. Therefore, instead of relinquishing back to the system the reservation made by *InetRx*, we can transfer the ownership of these reservations to its corresponding *InetTx* or *DiskTx* to ensure the quickest response. Such transfer is impossible in traditional priority-based scheduling schemes, since they do not possess any sense of ownership.

The implementation of group scheduling in our contractual function scheduler is a simple task, as the schedule has already been set according to the slot ownership in its schedule table. In Figure 20, it is seen that a table is implemented to record different groups of member functions. When the selected function yields its execution right within the period specified by its own timeslots, the scheduler will select for execution a member function from its group to allow for a rapid execution right transfer between them. In the proxy, primitive functions with an explicit execution order form a group, such as the *InetTx* and *InetRx* in the gateway operation. When the proxy conducts a gateway operation, it will only form a contract with the kernel for the *InetRx* operation. This contract must guarantee the packet incoming rate from the video server. Whenever the *InetRx* receives an incoming packet, it will yield its execution right to its corresponding *InetTx* in order to allow immediate retransmission.

In order to illustrate the importance of group scheduling to MPS, a test was carried out to evaluate the performance of the MPS service over a video stream with and without group scheduling. In this test, a 1200Kbps stream was sent through MPS to the client, and the amount of packet loss, if any, was recorded. The testing environment was specially designed such that the packet loss, if any, would be caused mainly by the delay introduced by MPS. With group scheduling in effect, we first determined the best setting to serve this stream by gradually increasing the amount of contracted computational power until we barely recorded packet loss on the client side. Then, we turned off the group scheduling mechanism and recorded a 21.7% packet loss. In other words, it is necessary to increase the computational power requirement in order to serve this stream without group scheduling. This indicates that group scheduling does improve the utilization rate of a contract.

In some rare situations, an operation might not follow the execution schedule exactly if the contractual function scheduler is not the only executable process within the system. In this case, the system might preempt the contractual function scheduler during the timeslot when an operation is due to execute. The *Linux* process scheduler is then free to schedule any other processes for execution. To solve this problem, we can either ignore the other process completely or adopt the contractual process scheduler to work together with the contractual function scheduler. In the former case, we can implement the system in a standalone machine such that it serves alone as a streaming proxy. In the latter case, we could allow the contractual function scheduler to acts as a

contractor to the system. For example, it may ask the contractual process scheduler to guarantee its right of execution repeatedly in the first half of each second. In this way, the contractual function scheduler can sub-contract the processing time to different primitive operations during that guaranteed period of time.

3.3 Experimental Results

In practice, we implement each of the primitive operations as functionoid object derived from a unique base class. The operation scheduler invokes a specific, named member function of the base class according to the schedule. Corresponding functionoid provides the actual service through polymorphism. According to the service demand of the client and the caching policy of the proxy, different functionoid would be instantiated, making a request on computational power.

We have tested three different scheduling schemes in our study. In the first case, we have adopted the priority round robin scheme (PRR), wherein all operations obtain their computational power through this scheduler. We have adopted the contractual operation scheduler in the second case (CT), wherein each operation could apply either the contractual scheme for soft real-time support or the priority round robin scheme for general scheduling support. Under this scheme, the contractual scheduler has priority to determine the exact operation for execution, while the priority round robin scheduler will take effect only when the contractual scheduler does not make any decision. For the last scheme, a hybrid approach has been adopted (PRR+CT). Functionoid could request computational power from both schedulers at the same time. In this case, the contractual scheduler delivers a guaranteed baseline of computational power with additional power support from the priority round robin scheduler through the reserve of the system.

Our test platform was set up as follows: a ***Linux***-based "QuickTime Darwin Streaming Server" [20] was set up as the video server; the proxy server MPS was placed in between the video server and client. The client, video server and MPS were equipped with an Intel 600MHz CPU; 256MBytes RAM; and interconnected with a 100Mbps LAN.

A contract in our contractual function scheduler has two parameters: period and duration, specifying the launching frequency of the function and the length of its execution in the unit of 100μs. In our experiments, the period and duration were 1000 units and 1 unit respectively. Experimental results showed that this setting was applicable in most practicable conditions. As a

proxy server merely redirects all received packets from the video server to the client, it does not generate any extra packets itself; therefore, the outgoing bandwidth of a proxy server cannot be greater than its incoming bandwidth. In other words, the highest bandwidth sustained by the video server is the best performance that a proxy server can ever achieve. We denote this bandwidth as B_{best} . We used this parameter as a reference to compare the performances achieved by different scheduling methods.

The objective of the first experiment was to determine the performance of MPS using different scheduling schemes. We adopted a quantitative index of effective bandwidth to measure the performance of MPS under different stressed conditions during the delivery of a video stream. We defined the effective bandwidth as the average quantity of useful data received by the client within a period. It also indicates the number of packets lost indirectly during transmission, i.e. the larger the number of lost packets the client detects, the lower is the effective bandwidth. Packet loss can occur for several reasons. First, it can be an actual situation of a missing packet. In this case, the client can never receive the packets, which may have been lost during the transmission between the server and the client. The second possible reason is transmission delay – the packet arrives after the correct playtime of the corresponding frame. The reasons for missing packet or transmission delay again can be two-fold: network bandwidth insufficiency or improper internal operations within the proxy server.

The setup of our experiment is as follows. We tested three different video streams with different bandwidth requirements varying from low to high in our platform. In addition, we introduced different amounts of loading into the proxy through the addition of some self-looping functionoid, which were used to simulate the non-time-constraint operations in the system such as *Transcode*. Figure 21 to Figure 23 show the effective bandwidth sustained by MPS in the delivery of three different video streams with three different scheduling schemes (PRR, CT and PRR+CT) under different loading conditions. The three video streams are “F100” (Final Fantasy in 100Kbps), “BTC” (Bless the Child) and “Patriot”. Their average bandwidth requirements are 100Kbps, 500~600Kbps and greater than 1500Kbps respectively. They represent three different classes of bandwidth requirement that we regard as low, medium and high. Figure 21 and Figure 22 show that when we adopted the PRR scheduling scheme in MPS, the effective bandwidth sustainable by MPS dropped substantially when we introduced a larger loading. Systems that have applied CT

and PRR+CT performed much better, even when the system was heavily loaded. Figure 23 illustrates the situation for the delivery of a video stream with high bandwidth requirement, the "Patriot". It shows that none of these three scheduling strategies is capable of sustaining the required amount of bandwidth. The major reason is that the bandwidth requirement of the video stream is too high for the platform to support. While "Patriot" has a bandwidth requirement larger than 1500 Kbps, we measured the bandwidth sustained by the video server while streaming "Patriot" to be only 1233.03 Kbps (B_{best}), with average percentage of packet lost to be 8.75% and maximum percentage of packet lost to be 34.58%. Considering the packet loss and delay caused by the proxy server, we expect that the client will detect a substantial amount of lost packets, and the effective bandwidth will be very low. In fact, the use of RTCP further consolidates this result. The reporting mechanism of RTCP allows the video server to understand the current connection status during playback and correct the connection bandwidth by adjusting the frame rate. When streaming "Patriot", RTCP sent the information on packet loss back to the server, which reduced the frame rate to adapt to the network condition. Consequently, our record of effective bandwidth consistently went down. Although Figure 23 shows that even the CT and PRR+CT scheduling strategies are not capable of sustaining B_{best} , their performances are still better than those in the PRR strategy. Apart from this, when the system was free from other loads and the sustained effective bandwidth was 1161.86 Kbps and 1150.46 Kbps for the CT and CT+PRR strategies respectively, both of them approached B_{best} .

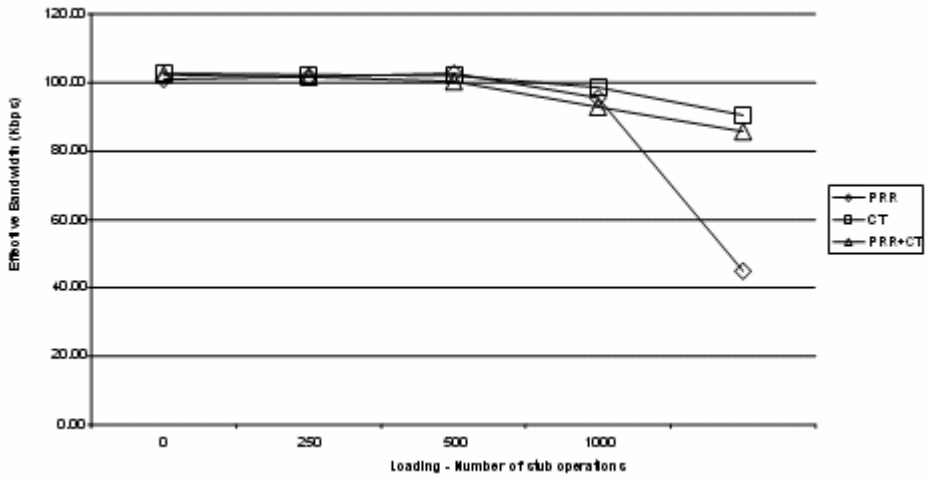


Figure 21: Effective bandwidth against number of loadings for video title "F100"

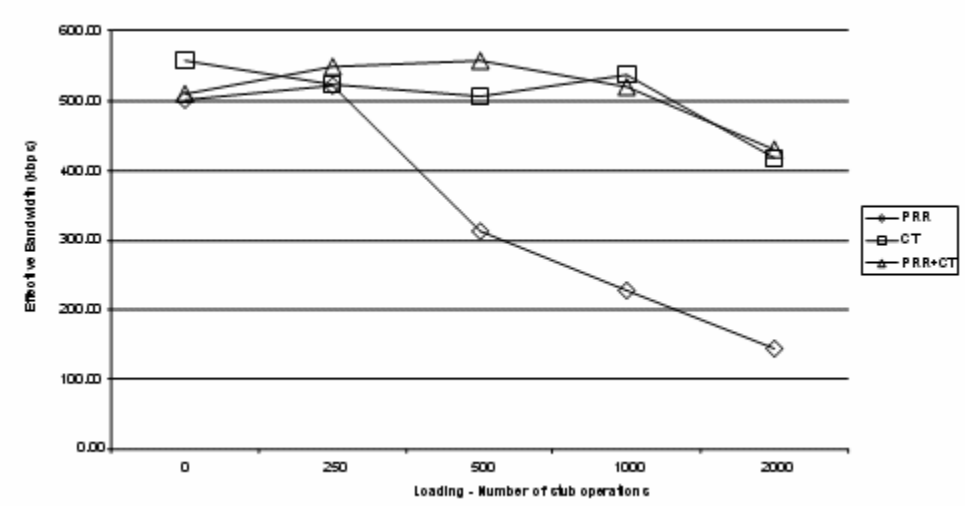


Figure 22: Effective bandwidth against number of loadings for video title "BTC"

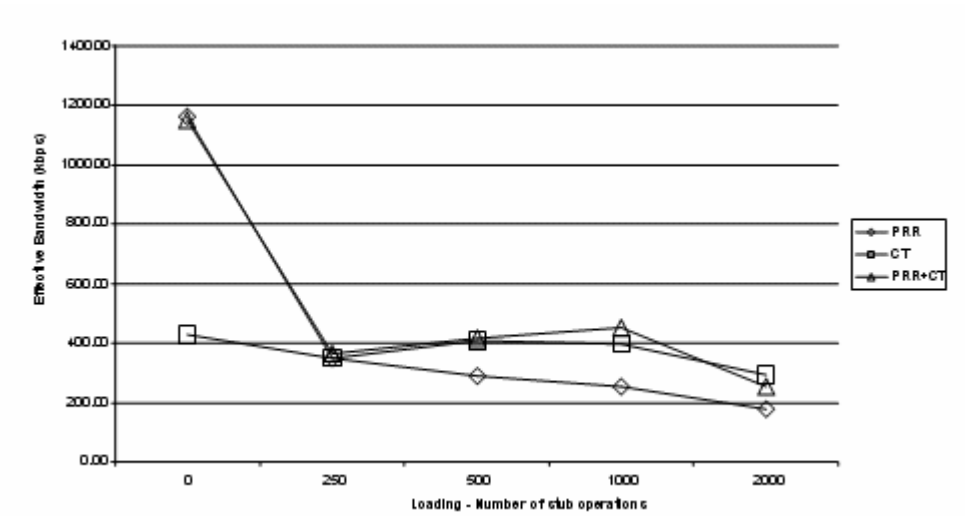


Figure 23: Effective bandwidth against number of loadings for video title "Patriot"

Figure 24 to Figure 26 shows the average percentage of lost frames experienced by the client for the three streams when the system was under different amounts of loading. Figure 27 to Figure 29 shows the maximum percentage of lost frames. As these figures show, CT and PRR+CT outperformed PRR in all video streams, giving a lower average rate of packets lost and a lower maximum packet lost rate.

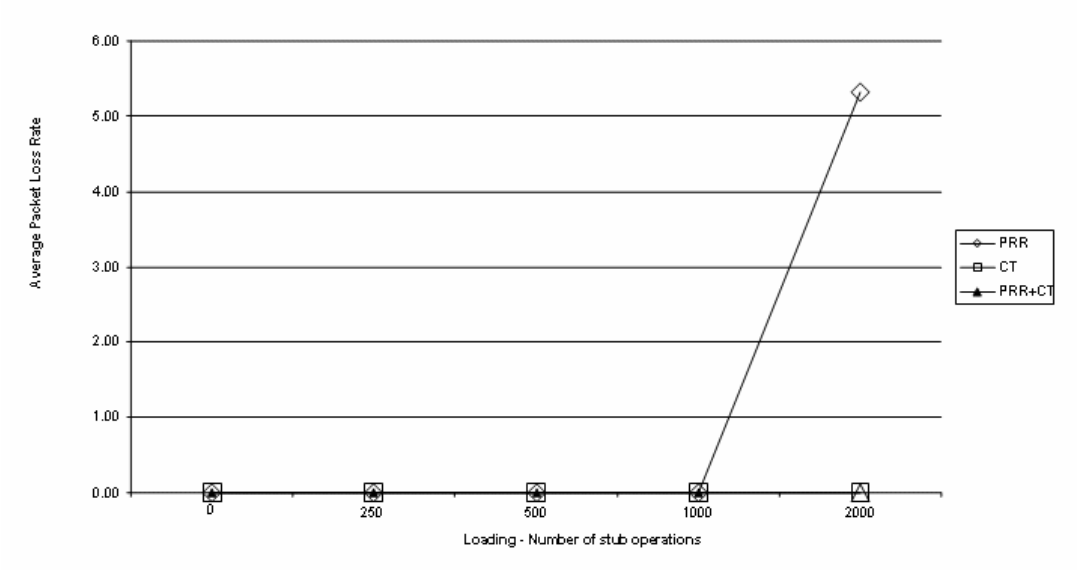


Figure 24: Average Packet Lost Rate against number of loadings for video title "F100"

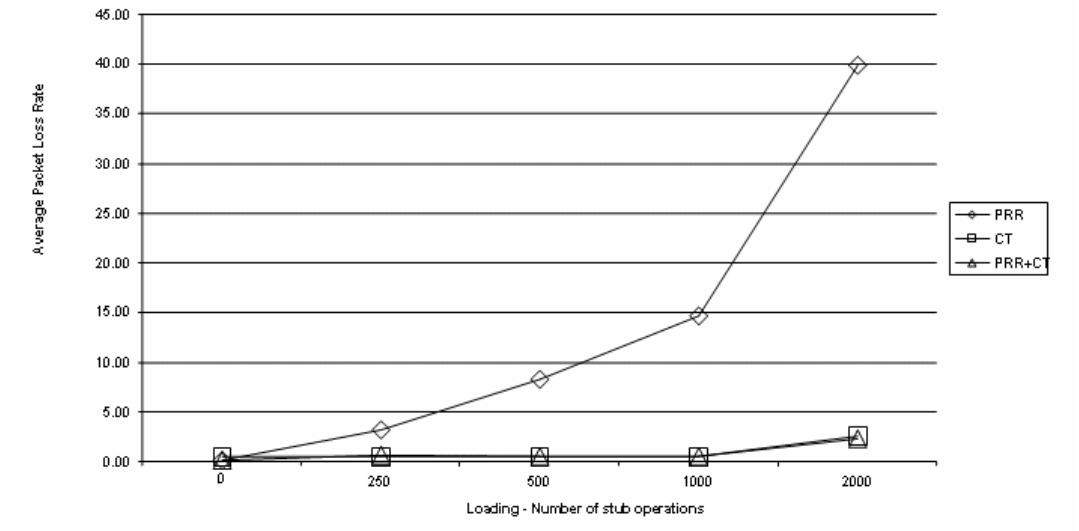


Figure 25: Average Packet Lost Rate against number of loadings for video title "BTC"

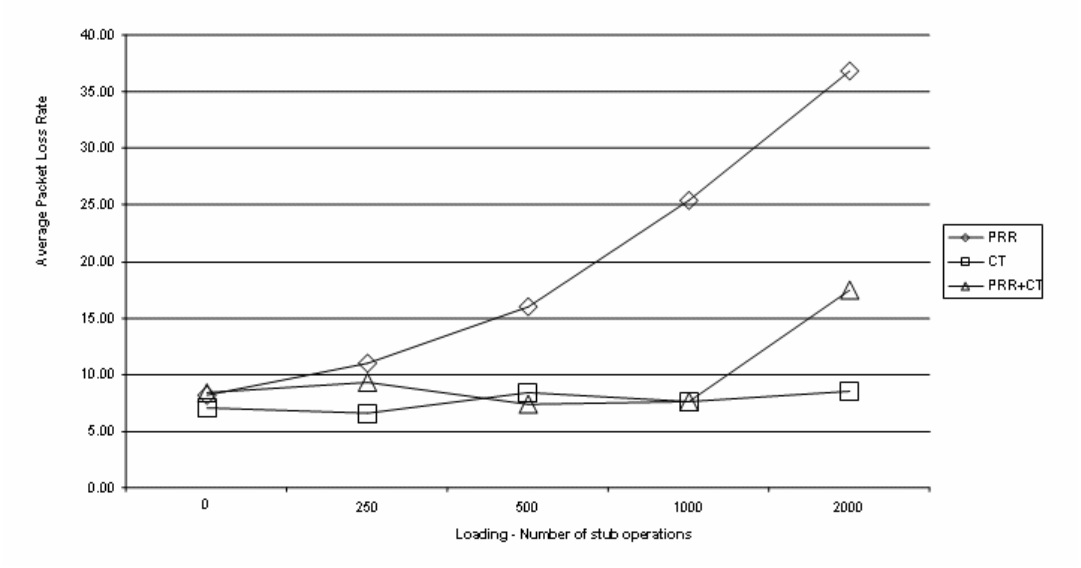


Figure 26: Average Packet Lost Rate against number of loadings for video title “Patriot”

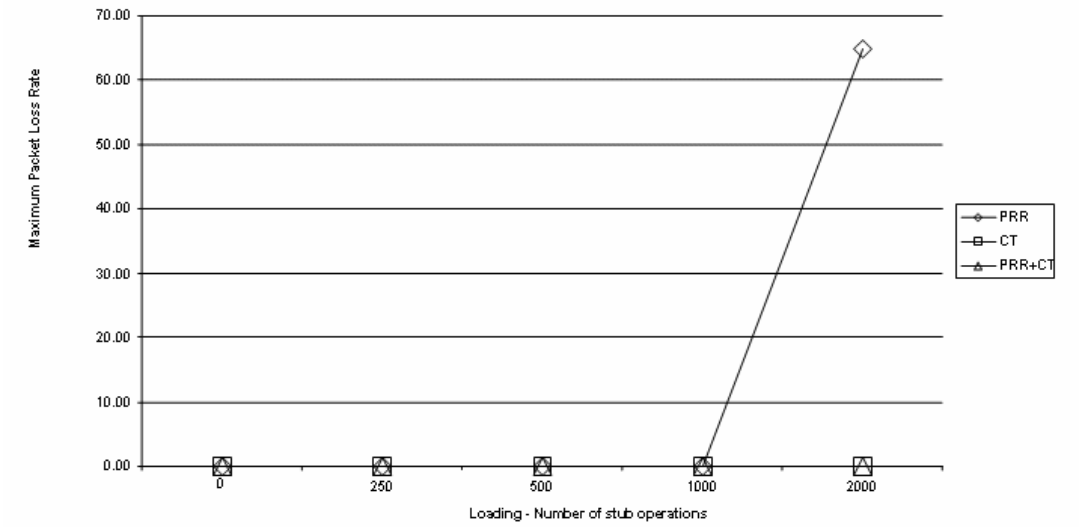


Figure 27: Maximum Packet Lost Rate against number of loadings for video title “F100”

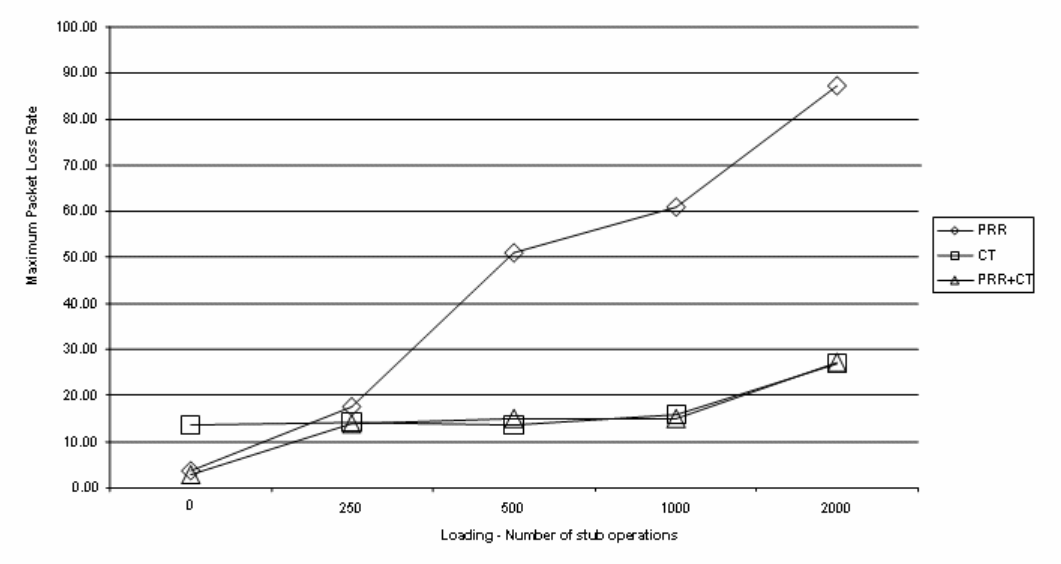


Figure 28: Maximum Packet Lost Rate against number of loadings for video title "BTC"

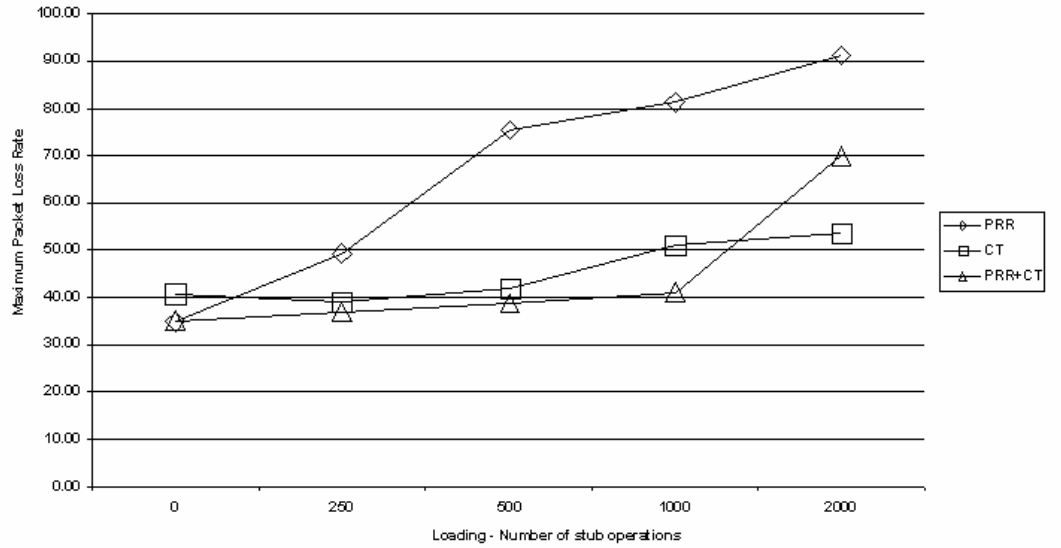


Figure 29: Maximum Packet Lost Rate against number of loadings for video title "Patriot"

Figure 21 to Figure 23 shows that the bandwidth dropped when the system was heavily loaded, even if we applied the CT or PRR+CT scheduling strategies. Note that for those experiments above, we have adopted the CT or PRR+CT strategies for RTP connection only. The scheduling scheme for RTSP and RTCP connections was still PRR. The loading in the system affected both of them in two ways. First, a delay in RTSP caused a drag between connection and playback. Second, a delay in RTCP confused the server, which concluded a connection problem

and triggered the frame skipping mechanism. To correct this, we had scheduled such connections by the scheduler with real-time support. Figure 30 to Figure 32 illustrate the playback result of the video title “BTC” under the modified scheduling scheme of CT. They show the effective bandwidth, average percentage of packet lost, and maximum percentage of packet lost. It was found that the bandwidth no longer dropped, even when the system was heavily loaded. This result illustrates that if we schedule any connection of a video stream with CT strategy, we should also schedule all the other connections associated with this video stream with CT.

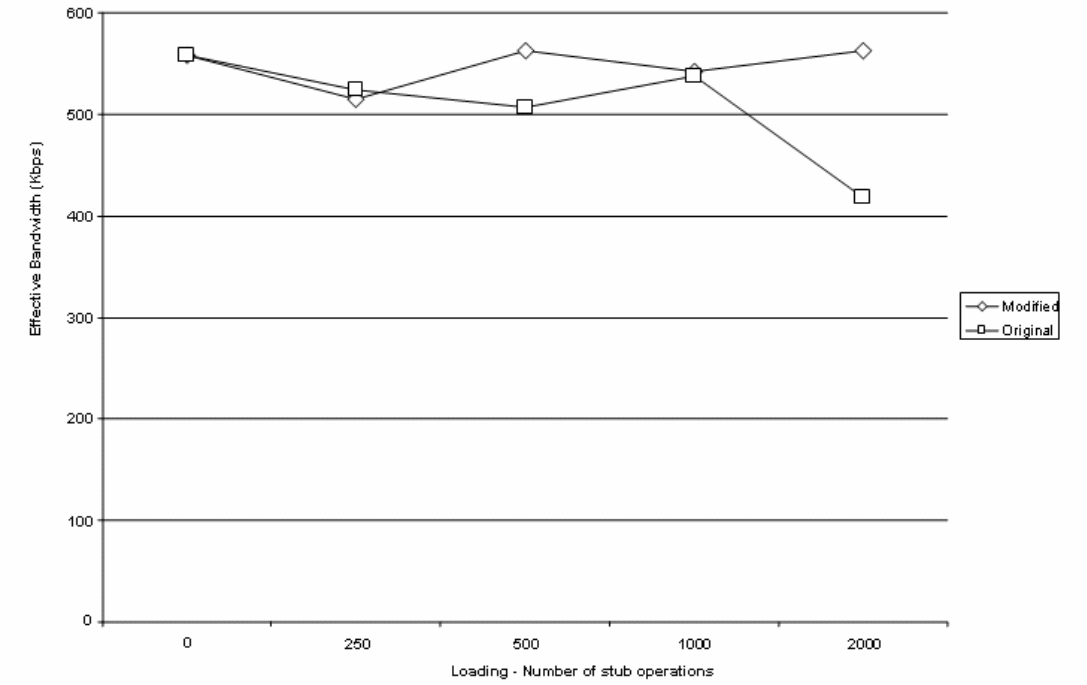


Figure 30: Bandwidth against number of loadings for video title “BTC”

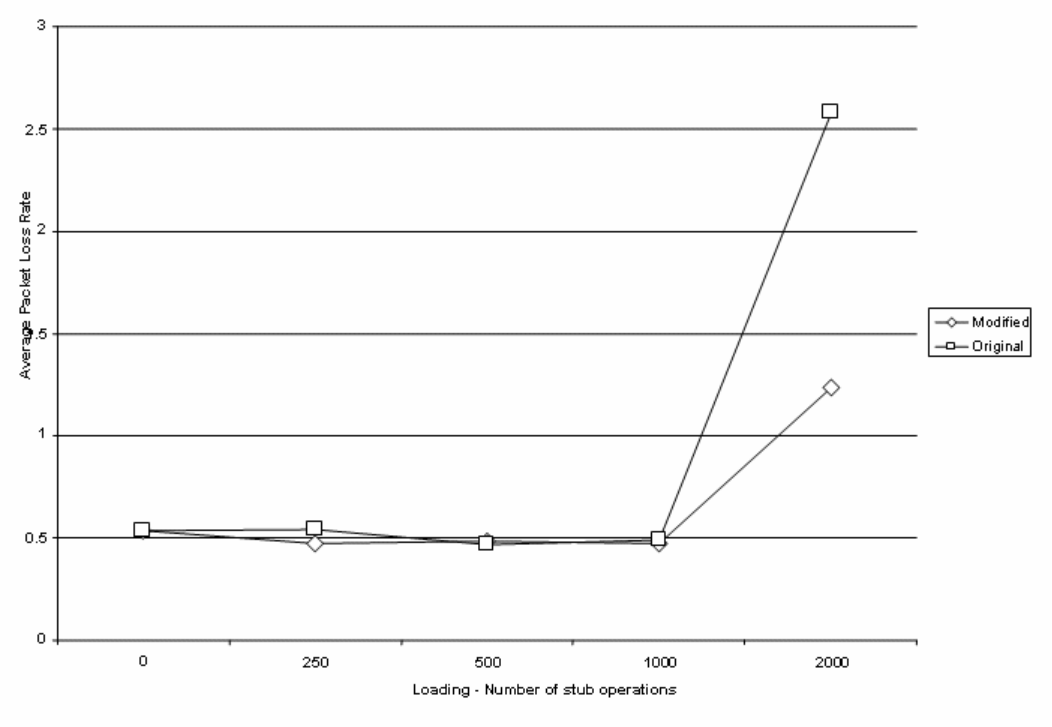


Figure 31: Average Packet Loss Rate against number of loadings for video title "BTC"

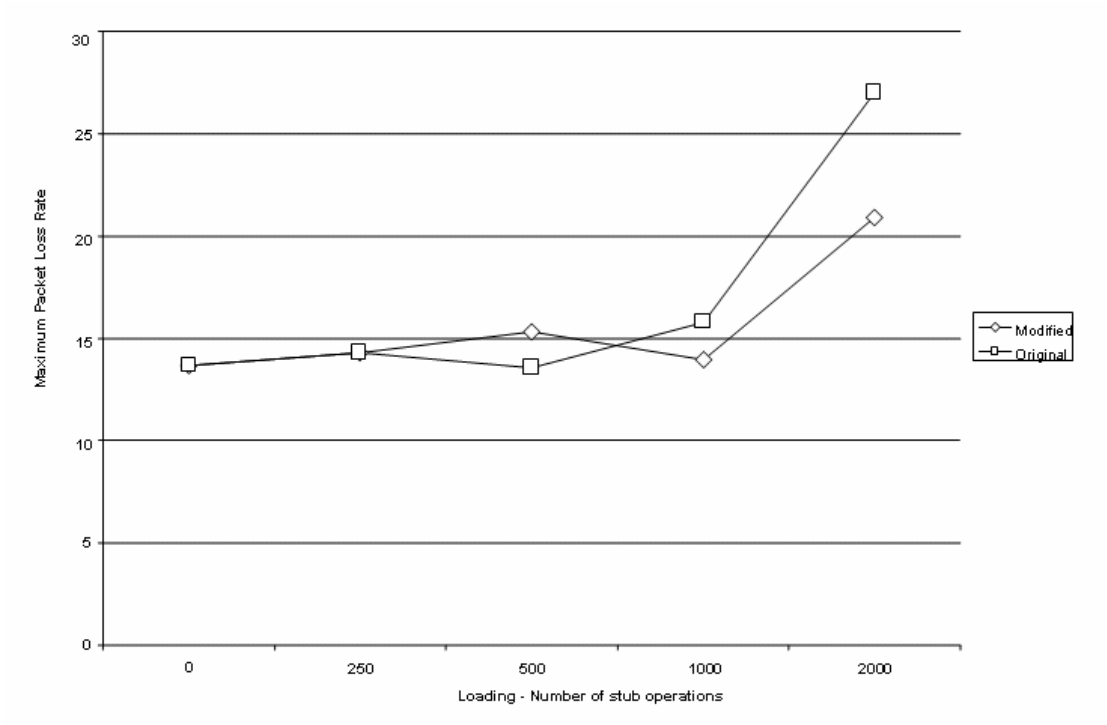


Figure 32: Maximum Packet Loss Rate against number of loadings for video title "BTC"

3.4 Summary

In this section, we have detailed the design and implementation of a video streaming proxy server – MPS. We have shown that MPS consists of three major components: Control Flow Engine, Data Flow Engine, and Cache Engine. As the video streaming is soft real-time, the proxy server must also send the packets received from the video server to its client in real-time. This implies that some of its tasks must work in a real-time manner also. From the empirical results above, we found that the priority round robin scheduling strategy was incapable of providing the real-time schedule for our proxy server when the system is under load. To cope with this, we adopted the contractual paradigm and developed a contractual function scheduler to provide real-time support. Experimental results show that the contractual approach performs much better than the priority round robin scheme, even when the system is under load. The contract provides a form of guarantee from the scheduler against upcoming resource subscribers. In addition, we have also found that it is very difficult to issue an exact contract for a task within MPS. As a result, it is necessary to book more than enough computational power for a task, as some will be wasted. The introduction of group scheduling can greatly increase this utilization rate by transferring some of the wasted processing power to the other related tasks.

From another point of view, the contractual paradigm provides us with a means to manage computational power explicitly. A specific amount of computational power is allocated to support each video stream according to its real-time requirement. The provision of these computational powers is guaranteed by the schedule table and the contractual function scheduler. As a result, it can fulfill its role even in a worst case environment; yet, it also implies wastage. Since it does not always require computational power for a worst case scenario all the time, in our implementation, this overbooked computational power can be transferred to other specific operations through group scheduling in the form of a homogeneous resource usage transfer.

Further improvements in the utilization of computational resources within a video streaming proxy server can be classified into two areas. The first asks how to determine the computational power requirement to support a video stream. Currently, an estimation of the worst case requirement can be achieved. However, the actual requirement will vary with time. A better utilization of computational resource and a capacity boost can be expected if we are able to

schedule these requirements properly. The support of varying computational resources can easily be adopted into our contractual function scheduler. The second area is heterogeneous resource usage transfer with computational power. Many operations require computational power that can be adopted in a video streaming proxy server. These include: transcoding [80][27][81], providing fine-granularity-scalability [83][84][85], region-of-interest [82][86], etc. Apart from computational power, memory, network and disk bandwidth are also required in such operations. An investigation into the possibility of providing usage transfer between these resources in these applications should lead us to a more efficient and flexible platform.

In short, in this section, we have,

1. Design and implemented a video streaming proxy server
2. Developed the contractual function scheduler to provide real time computational power support
3. Developed the group scheduling mechanism to allow computation power interchange

4 Memory, Network and Disk Bandwidth

The primary role of a video streaming proxy server is to exploit the temporal redundancy in web streaming object requests in order to save precious Internet bandwidth. Whenever a client makes the request for a video stream, the proxy server will determine the corresponding video streaming server for request redirection. Then the proxy server will begin to receive video data packets. These packets will be stored temporarily in the proxy server for packet re-ordering, transcoding [27][28], or traffic re-shaping [29]. If the proxy server decides to cache this video stream, it will make a copy of these packets to its cache storage. In contrast, if the proxy finds a copy in its cache storage, it will obtain the media directly from there rather than retrieving the stream from the actual media server. In addition to this all-or-nothing caching mechanism, recent research suggests that a proxy server may also implement the video staging mechanism [5][33], of which it will only cache part of the video stream, leaving the rest in the media server. During playback, it will combine both packets retrieved from the local disk and from the remote server and forward them to the client. Finally, when the client wants to terminate the operation, it will request a disconnection from the media server and the proxy server can release the resource allocated for this connection.

When there are concurrent connections, the actions described above will impose usage constraints on different resources in the proxy server. For example, the incoming and outgoing bandwidths of these concurrent connections are limited by the total incoming and outgoing data rate of the server, while their buffer size is limited by the volume of available memory space on the server. Moreover, when the proxy server considers caching a video in its disk cache, its storage will occupy disk space and consume disk bandwidth.

In order to increase the throughput of a proxy server, the usage of such resources has to be scheduled. Since different resources have different favorable usage patterns, a proxy server should consider this during the scheduling. Besides, a better schedule can usually be obtained from some tradeoff of usage between different resources. For example, incoming bandwidth requirements can be reduced by increasing the memory buffer size. Therefore, in order to optimize the throughput, a proxy server can schedule its connections so that their resource usage can be compromised to allow for a larger volume of sustainable concurrent connections.

To achieve this, Rexford et al. have proposed a smoothing algorithm [25][32] for proxy servers to allow interchange between memory usage and network bandwidth. With this algorithm, a proxy server can utilize its memory and network much more effectively in order to support a greater number of clients concurrently. However, the network is the only transportation medium considered by Rexford's algorithm. It does not consider bringing in proxy server local storage for smoothing the backbone network bandwidth. As mentioned above, the video staging technique [5] separates a video stream into two parts. One part is obtained from the media server while the other is obtained directly from the local storage of the proxy server to allow saving of network bandwidth. It is desirable if we can incorporate the video staging technique with Rexford's algorithm so that a better video smoothing mechanism with staging feature can be developed [35].

4.1 Quantifying Video Streaming

As shown by Rexford [25][32], data transfer of a connection within a proxy server can be illustrated by a graph plotting its cumulative transferred data against time. As shown in Figure 33, line A illustrates the arrival schedule that addresses the maximum possible data cumulated from the media server at the proxy. Line D shows the client data deadline schedule, illustrating the least amount of data the client needs to receive in order to provide a correct play back. Between these lines, line S illustrates the actual delivery schedule for this connection in the proxy. The value $\max\{A-S\}$ is the buffer requirement of the proxy server required to support this connection with schedule S without causing buffer overflow. In addition, the incoming bandwidth requirement of the connection is given by $\max\{dA/dt\}$, while the outgoing bandwidth requirement is $\max\{dS/dt\}$. The amount of information delivered in a specific time is denoted as S_k , where $k=0,\ldots,N$ in a video with total length $N+1$; hence the total size of the video is denoted as S_N .

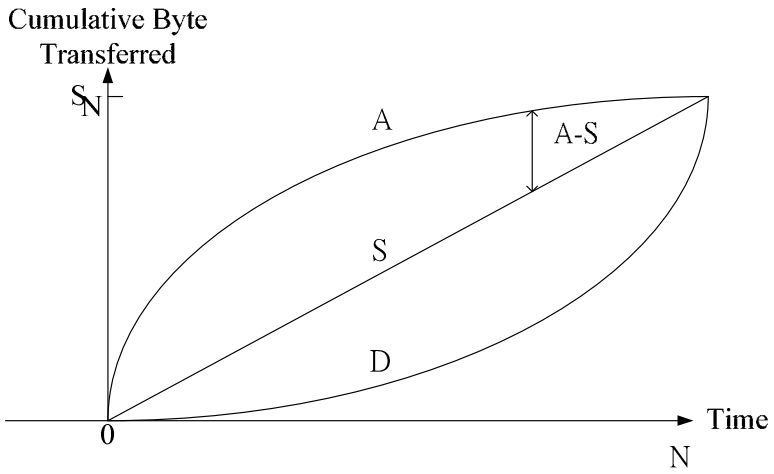


Figure 33: Cumulative data transfer of a connection in a proxy server

Now we consider the case of caching. Under a traditional whole-or-none caching strategy, both the storage and retrieval bandwidth requirements of the disk are equal to $\max\{dA/dt\}$. For video staging [5], a video stream is divided into two parts, as shown in Figure 34. The first part is obtained from the media server through the network, deciding the bandwidth requirement. The other part is obtained from the disk and decides the storage and retrieval bandwidth requirements. It can operate in two modes: CAS (Cut-off after Smoothing) and CBS (Cut-off before Smoothing). Both of them rely on a cutoff bandwidth dC/dt for separation of the upper and lower streams. The upper part will be retrieved from the disk, while the lower part has to be obtained from the network. In CAS, as the video has been smoothed before being cut off, the incoming bandwidth requirement and the disk retrieval bandwidth will be dC/dt and $\max\{d(A'-C)/dt\}$, where A' represents the smoothed version of schedule A . As for CBS, the video will be cut off before smoothing, the incoming bandwidth will become dC/dt , and the retrieval bandwidth will become $\max\left\{d(A-C)'/dt\right\}$, where $(A-C)'$ represents the smoothed version of schedule $A-C$. The staging mechanism essentially divides a variable-bit-rate video stream into a bounded-bit-rate video stream and another smaller variable-bit-rate video stream. As the network is responsible for the delivery of the bounded-bit-rate stream only, this relieves the complexity in its bandwidth allocation. The local disk within the proxy server will handle the rest of the unsmoothed stream.

From an operational point of view, a video staging schedule must accept a video delivery schedule as its parameter and propose an arrival schedule for the proxy server. This arrival schedule shall consist of two parts: the proxy shall obtain the first part from the server through the network and the other part from its local disk. Consequently, we can break this proposed arrival schedule down into a disk schedule and a network schedule. The design of a video staging schedule is therefore a design of these two schedules.

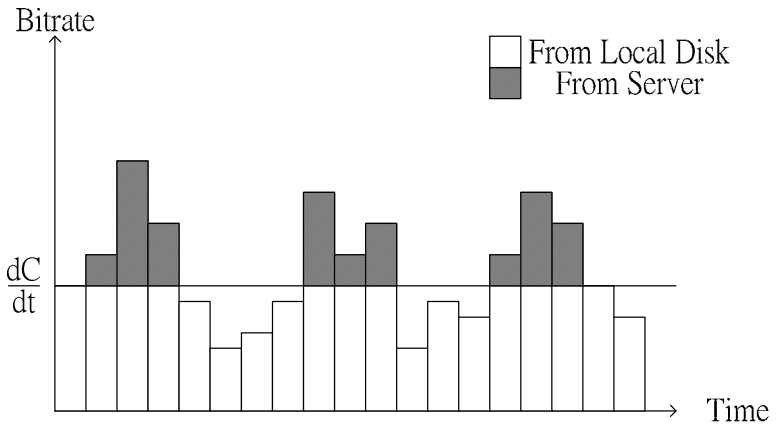
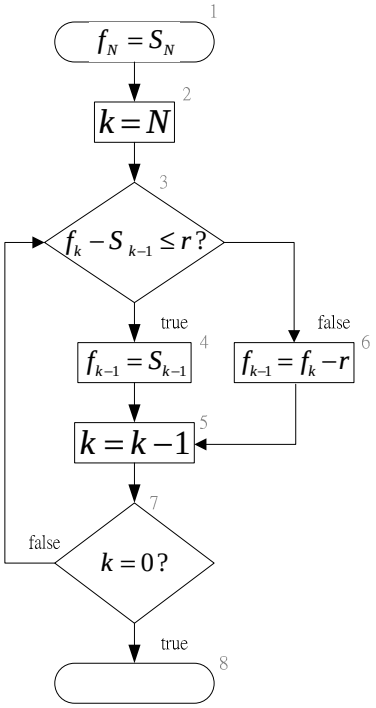


Figure 34: Cut-off arrangement with video staging

4.1.1 Smoothing Algorithm

Rexford et al. have suggested a smoothing mechanism to obtain a delivery schedule for a variable-bit-rate video stream in a proxy server [25]. Since our interest is in the arrival schedule rather than the delivery schedule, we will apply this algorithm in a reversed scenario and call it algorithm Φ in the following context. It will work under a limited incoming bandwidth r and a delivery schedule $S = S_k : k = 0,\ldots,N$ and generate a proposed arrival schedule $f = f_k : k = 0,\ldots,N$ for the server to deliver the video stream to the proxy. We state the algorithm with a flowchart as follows:



1. Assign $f_N = S_N$
2. Begin at the end of the schedule with $k = N$
3. If $f_k - S_{k-1} \le r$ then 4 else 6
4. $f_{k-1} = S_{k-1}$
5. Iterate backward by $k = k - 1$; goto 7
6. $f_{k-1} = f_k - r$
7. If scheduling completed $k = 0$ then 8 else 3
8. End

Flowchart 1: Algorithm Φ

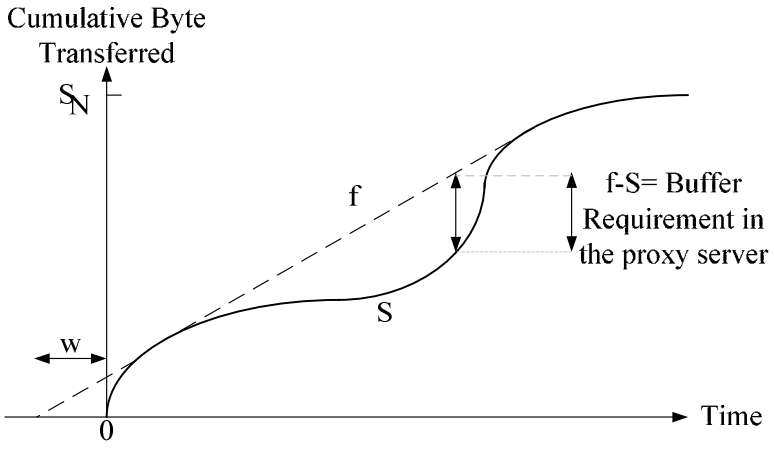


Figure 35: A smoothed schedule

Without loss of generality, both schedules f and S are indexed to 1 second. Therefore, the difference between two consecutive points in a schedule, such as $f_k - f_{k-1}$, gives the bandwidth requirement of schedule f during time interval $[k, k-1)$. Applying algorithm Φ to a delivery schedule S as shown in Figure 35 will generate a proposed arrival schedule f with an incoming bandwidth constraint r and delivery delay w. The buffer requirement of this schedule imposed

on the proxy is given by $b_p = \max\{ f_k - S_k \} : k = 0, \ldots, N$. Occasionally, a large delay is required to avoid f having any part of it go below S , which implies starvation in the buffer of the proxy server. In this case, the buffer requirement of the proxy has to be obtained by $b_p = \max\{ f_k - S_{k+w} \} : k = 0, \ldots, N$.

4.1.2 Resource Usage Interchangeability

Algorithm Φ provides a link between the delivery schedule, incoming bandwidth constraint, delivery delay, and its buffer requirement. The relationships between them give the upper limits in incoming bandwidth and buffer requirements for each connection. A proxy server can adjust the usage of these two resources to achieve the maximum number of sustainable clients.

However, the network is the only transportation medium handled by Φ . When we take video staging into consideration, disk access becomes another transportation medium. The timing of disk access, the amount of buffer required for temporary storage, and the relationship with the incoming network bandwidth should be further investigated. In the following sections, we construct an algorithm based on algorithm Φ that incorporates disk access into its scheduling consideration. It maximizes the throughput of the proxy server by optimally scheduling the usage of different resources with video staging applied.

4.2 Video Staging Schedule

Before we describe the new algorithm, let us illustrate some properties of the smoothing algorithm Φ . We first consider the relationship between the proxy buffer size and the allowable incoming bandwidth. We shall show below that the proxy buffer size requirement decreases monotonically when the allowable incoming bandwidth increases. This is reasonable, since the purpose of the buffer is to smooth out the abrupt increase in bandwidth demand from the video stream. As every video stream has its own maximum bandwidth, less buffer is required if the allowable incoming bandwidth increases and approaches that maximum value. This observation can be proven as follows. Let $f = \Phi(S, r)$ and $f' = \Phi(S, r')$; the statement we have to prove becomes $\max\{ f - S \} \le \max\{ f' - S \}$ if $r \ge r'$. To prove this statement, we first show that f_k can be written as $S_k + c_k$.

Lemma 4-1:

If $f = \Phi(\mathrm{S},\mathrm{r})$ *where* $S = S_k : k = 0,\ldots,N$ *and* $f = f_k : k = 0,\ldots,N$ *are the delivery and arrival schedules respectively and* r *is the incoming bandwidth, then* f_k *can be written as* $S_k + c_k$ *, where* c_k *is a constant and* $c_k \geq 0$ *with* $k = 0,\ldots,N$ *.*

Proof:

Assuming there exists an integer $k = 0,\ldots,N$ *such that* $f_k = S_k + c_k$ *with* $c_k \geq 0$ *.*

If $f_k - S_{k-1} \leq r$ *then*

$$f_{k-1} = S_{k-1}$$

Else $f_k - S_{k-1} > r$ *and we have* $S_k + c_k - S_{k-1} > r$ *then*

$$f_{k-1} = f_k - r$$

$$f_{k-1} > S_{k-1} + r - r$$

$$f_{k-1} > S_{k-1}$$

Therefore, we have $f_{k-1} = S_{k-1} + c_{k-1}$ *and* $c_{k-1} > 0$ *.*

As $f_N = S_N$ *, we have proved the statement by induction.*

Based on lemma 4-1, we can prove the abovementioned relationship between incoming bandwidth and proxy buffer size under Φ .

Theorem 4-2:

If there are two schedules $f = \Phi(S,r)$ *and* $f' = \Phi(S,r')$ *with two different incoming bandwidths such that* $r' > r$ *, then* $\max\{f_k - S_k\} \geq \max\{f'_k - S_k\}$ *.*

Proof:

First, we assume there exists an integer $k = 0,\ldots,N$ *such that* $f_k = f'_k$ *.*

If $f'_k - S_{k-1} \leq r < r'$ *then*

$$f'_{k-1} = S_{k-1}$$

It also implies $f_k - S_{k-1} \leq r$

$$\Rightarrow f_{k-1} = S_{k-1}$$

Therefore, we have $f_{k-1} \geq f'_{k-1}$ *,*

$$\Rightarrow f_{k-1} - S_{k-1} > f'_{k-1} - S_{k-1}$$

If $r' \geq f'_k - S_{k-1} \geq r$ *then*

$f'_{k-1} = S_{k-1}$ and $f_{k-1} = S_{k-1} + c_{k-1}$

As $c_{k-1} \geq 0$ *, therefore*

$f_{k-1} \geq f'_{k-1}$

$\Rightarrow f_{k-1} - S_{k-1} > f'_{k-1} - S_{k-1}$

If $f_k - S_{k-1} \geq r' > r$ *then*

$\Rightarrow f_{k-1} = f_k - r$ and $f'_{k-1} = f'_k - r'$

As we have assumed that $f_k = f'_k$ *, therefore we have* $f_k \geq f'_k$ *as well as* $r' > r$ *. Combining them together, they give*

$f_k - r > f'_k - r'$

$\Rightarrow f_{k-1} > f'_{k-1}$

$\Rightarrow f_{k-1} - S_{k-1} > f'_{k-1} - S_{k-1}$

Since we must have $f_N = f'_N$ *, by induction we have proved that* $f_k - S_k \geq f'_k - S_k$ *.*

$\Rightarrow \max\{f_k - S_k\} \geq \max\{f'_k - S_k\}$

The second property of Φ is that it always minimizes the maximum required buffer size of the proxy server.

Theorem 4-3:

If there is a schedule $f = \Phi(S, r)$ *and any arbitrary smoothing function* f' *with the same incoming bandwidth* r *then we shall have* $\max\{f_k - S_k\} \leq \max\{f'_k - S_k\}$ *for all* $k = 0, \ldots, N$ *.*

Proof:

Suppose we have two feasible schedules f *and* f' *and there exists an integer* $k = 0, \ldots, N$ *such that* $f'_k < f_k$ *. The value* $f_k - S_k = \max\{f - S\}$ *determines the required proxy buffer size of schedule* f *. Therefore, we have* $\max\{f - S\} = f_k - S_k$ *. These two schedules do not touch each other at point* k *. Suppose they touch each other again* m *points after point* k *and the incoming rate constraint is* r *, then,*

$f_k + mr = S_{k+m}$

$f'_k + mr < S_{k+m}$

Therefore, f_k' goes below S at point $k+m$. It cannot be a feasible schedule. No schedule may give $f_k' < f_k$. As a result, $\max\{f - S\} = f_k - S_k$ gives the minimum of the maximum proxy buffer requirement.

From the properties above, we conclude that algorithm Φ is the optimal schedule in terms of proxy buffer requirement. It yields a smaller buffer requirement when a higher incoming bandwidth is allowed.

4.2.1 Property of Staging and Resource Usage Scheduling

For a proxy server, the staging mechanism divides the source of an incoming video data flow between the network and hard disk. These flows exhibit distinct preferences in their usage. This is because the effective bandwidth supported by most wide area networks is not yet comparable with disk bandwidth nowadays. However, due to a comparatively lower switching cost, network communications can be multiplexed to serve several clients at a time without being noticed, even in the sub-second time scale. While the seek time of a disk removes the possibility of making it a multiplexing device in this time scale, it is also impractical for a disk to provide a long period of service to only one client, ignoring the others. Based on these observations, we can conclude that the network prefers a long-term, small-scale service, while the disk prefers a short-term, large-scale service.

As mentioned previously, the lower part of a video staging schedule accounts for data retrieved from the network and behaves like a constant-bit-rate schedule. This behavior is favorable to the usage pattern of network resources; mechanisms such as IntServ [30] can reserve such a requirement in the network. The upper part of the schedule accounts for data retrieved from a local disk. This part of the schedule absorbs most of the bit rate variations from the arrival schedule. Although we can reserve disk bandwidth in the same way as network bandwidth, this will introduce many small volume disk retrievals and lower its throughput. Instead of this, a desirable schedule should allocate fewer but larger volume disk retrievals. The construction of such a schedule will be analyzed in the next section.

4.3 Video Staging Algorithm Γ

From algorithm Φ, we know that we can compensate for the insufficiency of incoming bandwidth by means of a larger memory buffer to retrieve data in advance. On the other hand,

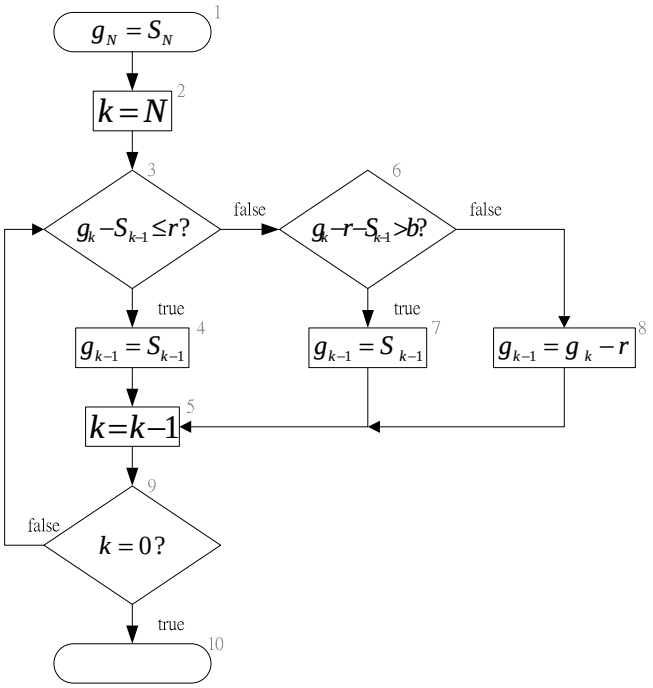
from the staging mechanism we know that we can separate a video stream into two parts; one part is more favorable for network retrieval and the other is more favorable for disk retrieval. Now, we begin to construct our own video staging algorithm Γ . In this algorithm, we will take the incoming bandwidth constraint, memory buffer usage, and disk access into consideration so that we can interchange them for optimization.

Similarly to Φ , algorithm Γ is also built under an incoming bandwidth constraint r . As shown in Figure 35, the minimum buffer requirement for a proxy server working under the smoothing algorithm Φ is the maximum vertical distance between the proposed arrival schedule obtained by Φ and the delivery schedule. Moreover, we recognize that the separation of these two schedules stems from the failure to satisfy the condition $f_k - S_{k-1} \le r$, which implies that the incoming network bandwidth is unable to cover the consumption required in the delivery schedule. However, we can compensate for it by having a disk access of data volume $b_k = f_k - S_{k-1} - r$.

Both the staging algorithms stated above – CAS and CBS – will introduce continuous disk access that is undesirable for multiple-client services. In these cases, we have to multiplex disk services in much the same way as the network services in order to retrieve the data required by different clients. Since the operating cost is much higher for multiplexing disk accesses, the throughput of a disk will drop dramatically. Worse still, this will increase the wearing rate on the disk and reduce its operational lifetime. Hence, in addition to those constraints imposed on algorithm Φ , we should also prevent algorithm Γ from generating continuous disk access.

During the development of the proposed algorithm, we have made the following assumptions without loss of generality. We assume that the employed memory buffer b will be much larger than the network bandwidth r, so that the memory buffer is able to hold several seconds of data. Besides, we assume that the maximum variation in the delivery schedule $\max\{dS/dt\}$ is not too much larger than r, which is true for general video title. The values of b, r and $\max\{dS/dt\}$ can be described as $b >> \max\{dS/dt\} > r$.

Under a limited incoming bandwidth r, maximum proxy buffer size b, delivery schedule $S = S_k : k = 0,\ldots,N$ and the suggested arrival schedule $g = g_k : k = 0,\ldots,N$, we state the proposed algorithm Γ with $g = \Gamma(S, r, b)$ as follows:



1. Assign $g_N = S_N$
2. Begin at the end of the schedule with $k = N$
3. If $g_k - S_{k-1} \le r$ then 4 else 6
4. $g_{k-1} = S_{k-1}$
5. Iterate backward by $k = k - 1$; goto 9
6. If $g_k - r - S_{k-1} > b$ then 7 else 8
7. $g_{k-1} = S_{k-1}$; goto 5
8. $g_{k-1} = g_k - r$; goto 5
9. If scheduling completed $k = 0$ then 3 else 10
10. End

Flowchart 2: Algorithm Γ

Algorithm Γ is similar to algorithm Φ, except in blocks 6 and 7, which show that whenever the vertical distance between the proposed arrival schedule g and the delivery schedule S accumulates to a volume exceeding a threshold $b + r$, the algorithm will create a rapid drop to close up the gap. However, if we traverse Figure 36 in a reverse direction from S_0 to S_N, the rapid drop is actually a rapid rise. We consider this rapid rise in cumulated data to be the result of disk access. As shown in Figure 36, the rise lifts up the schedule in the period of $\lfloor m, n)$ to avoid starvation during this period, even if the proxy relies on a network to obtain data only for the rest of the schedule.

The volume of data retrieved from the disk access is in fact an upper boundary to the buffer requirement of the proxy server. Firstly, it is interesting to point out that although we define in Flowchart 2 the condition for having a disk access as $g_k > S_{k-1} + r + b$, in practice g_k can only be

slightly greater than $S_{k-1}+r+b$ when the disk access occurs. This is because the vertical distance between g and S accumulates following the rate of $S_k - S_{k-1} - (g_k - g_{k-1})$, whereas $\lceil S_k - S_{k-1} - (g_k - g_{k-1}) \rceil = \max\left\{\frac{dS}{dt}\right\} - r$, which is much smaller than b, following the assumptions stated above. Hence, the algorithm will not allow g_k going too much greater than $S_{k-1}+r+b$ before its detection through the introduction of a disk access. In fact, it is safe to rewrite the condition for disk access as $g_k \cong S_{k-1}+r+b$. Now, suppose schedules g and S touch each other at time k; that is, $g_k = S_k$, and, according to algorithm Γ, the value of g_{k+1} is constrained by either $g_{k+1} \le r + S_k$ or $g_{k+1} \cong S_k + r + b$ depending on the need of a disk access at time k. We define the buffer requirement at time $k+1$ as $g_{k+1} - S_{k+1}$. For the case without disk access, the buffer requirement will be bounded to $r - (S_{k+1} - S_k)$, which is much smaller than b (note that $S_{k+1} - S_k < r$ in this case). In addition, for the case $g_k = g_{k+1} - r$, the value of g_{k+1} is constrained by $g_{k+1} - r - S_k < b$ and $g_{k+1} - S_k > r$. After rearrangement, we have the following inequality $S_{k+r} < g_{k+1} < b + r + S_k$, and after subtracting S_{k+1} from both terms, we arrive at $r - (S_{k+1} - S_k) < g_{k+1} - S_{k+1} < b + r - (S_{k+1} - S_k)$. For the case with disk access, that is $g_{k+1} - S_{k+1} \cong b + r - (S_{k+1} - S_k)$. As $S_{k+1} - S_k > r$ in this case, it is safe to approximate the upper bound of $g_{k+1} - S_{k+1}$ to be b. It should be noted that between schedule g and S, there should not be a gap larger than b otherwise a disk access will be introduced by algorithm Γ to close up the gap to zero. Hence, we have shown that the buffer requirement of the proxy server with algorithm Γ is bounded to b.

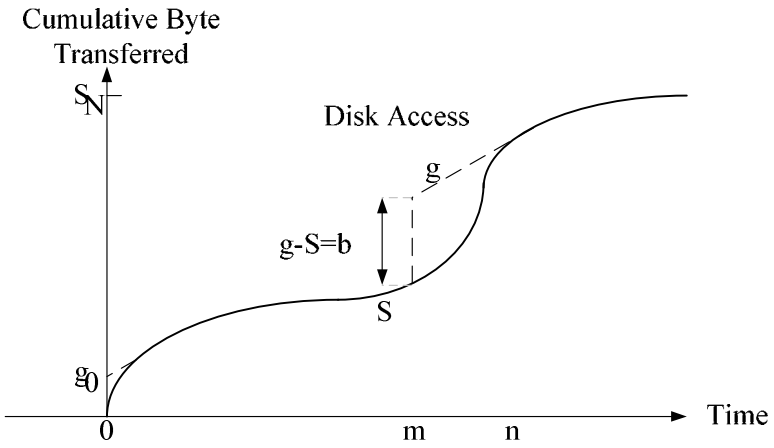


Figure 36: Schedule generated by algorithm Γ

4.3.1 Minimum Buffer Size

We can adjust the volume of these disk accesses by means of variable b , as mentioned above. At any time disk access is required, only data of size b will be retrieved, so that the data remaining in the buffer can cover, with replenishment from the network, the consumption as indicated by the delivery schedule until another disk access. Suppose the first disk access appears at time m and the next at n , and g does not touch the delivery schedule in between; the relationship of these two disk accesses will be $S_m + b + r \cdot (n-m) = S_n$. To avoid starvation between the two disk accesses, the choice of b cannot be arbitrary. We must satisfy a necessary and sufficient condition $b \geq \max\{dS/dt\} - r$.

Theorem 4-4:

Condition $b \geq \max\{dS/dt\} - r$ is a necessary and sufficient condition to provide a feasible schedule for any schedule $g = \Gamma(S, r, b)$ accounting on the buffer size.

Proof:

Assume that at any instant m , we have

$$\max\left\{\frac{dS}{dt}\right\} = \frac{dS_m}{dt}$$

Also, assume that $b < \max\left\{\frac{dS}{dt}\right\} - r$,

$$\Rightarrow b < S_m - S_{m-1} - r$$

As $g_m \geq S_m$ *,*

$$\Rightarrow g_m - S_{m-1} > b + r$$

According to algorithm Γ *, we have*

$$\Rightarrow g_{m-1} = S_{m-1}$$

We need a disk access at instant $m-1$ *, since we must have*

$$S_{m-1} + b + r \cdot (m - m + 1) = S_m$$

$$\Rightarrow S_{m-1} + b + r = S_m$$

$$\Rightarrow S_m - S_{m-1} = b + r$$

This violates our assumption; therefore, it is necessary to have $b \geq \max\{dS / dr\} - r$ *in order to satisfy the condition. Now, we assume* $b \geq \max\{dS / dr\} - r$ *and an arbitrary time* n *such that* $0 < n \leq N$ *,*

$$\Rightarrow b \geq S_m - S_{m-1} - r \quad \forall m = 1, \ldots, n$$

Take the sum of both sides for all m *,*

$$\Rightarrow nb \geq S_n - nr$$

$$\Rightarrow n(b + r) \geq S_n$$

When both disk and network accesses are at their full strength during period $[0, n]$ *, their cumulated workload must overcome the scheduled delivery at that time. Therefore, as it is possible to allocate disk and network access to their extreme at all times, it is a sufficient condition.*

Theoretically, disk accesses can be performed at any time during the streaming of the video data; however, this will introduce continuous disk access, reducing the disk throughput as mentioned above. If disk accesses are allowed at a frequency of $1/T$ then the following condition has to be met in order to guarantee a feasible schedule:

$$b \geq \max\{S_{i+T} - S_i\} - r \cdot T$$

The buffer size has to be larger than the maximum size of the data cumulated in a period of T minus the volume of data contributed by the network during this period. Once the necessary buffer size is satisfied; unlike algorithm Φ , it will be independent of the incoming bandwidth

constraint. Regardless of the change to this rate, the buffer requirement remains below or equal to size b .

4.3.2 Disk Access

The invariance of the proxy buffer requirement does not come free. Although the buffer size remains unchanged even if a lower incoming bandwidth is used, the proposed arrival schedule will introduce more disk accesses to replenish the data insufficiency incurred.

Theorem 4-5:

For two different incoming bandwidths $r' > r$, if a' and a denote the corresponding numbers of disk accesses required when applying algorithm Γ with delivery schedule S and proxy buffer size b , then the corresponding numbers of disk accesses will have a relation of $a > a'$.

Proof:

Consider the first two disk accesses, $g_m = S_m$ and $g'_{m'} = S_{m'}$ of schedule g and g' obtained by $\Gamma(S, r)$ and $\Gamma(S, r')$ respectively. Prior to any disk access, algorithm Γ behaves the same way as Φ ; therefore, $g_k > g'_{k'}$ for all $k > m$ and $k > m'$. As a result, $g - S$ must reach the buffer limit before g' . A disk access has to be made by g before the one made by g' , if any.

Now, consider the interval between two consecutive disk accesses made by g , say $g_m = S_m$ and $g_n = S_n$ where $n > m$. For g , suppose there are also two disk accesses within the region and the first access is made on $n+1 \geq k \geq m$ so that $g_k \geq g'_k = S_k$. As we have $r' > r$, according to algorithm Γ , the largest possible value of g' at time $m+1$ can only develop to attain $b > g_{m+1} \geq g'_{m+1} > S_m$. Therefore, g' can make its second disk access at time m , but it will never have more than two disk accesses during this interval.

Therefore, g' will make its first disk access later than g and for every two consecutive disk accesses of g there can only be at most two disk accesses of g' , so we can conclude that the number of disk accesses of g' will be smaller than g if their corresponding incoming rate constraints r' and r have a relation of $r' > r$.

Similarly, a change in proxy buffer size would also affect the number of disk accesses required.

Theorem 4-6:

Suppose there are two different proxies applying algorithm Γ with the same delivery schedule S and an incoming bandwidth constraint r . If their buffer sizes are b and b' such that $b' > b$, and the corresponding numbers of disk accesses are a and a' respectively, then they will have a relation $a > a'$.

Proof:

Suppose g and g' are the corresponding schedules of the two proxies with buffer size b and b'. As $b' > b$, g will make its first disk access later than g'. When we consider the interval between two consecutive disk accesses of g, suppose g' has also made a disk access within the interval. It will only be able to make another at the beginning of the interval. It will not be possible to have more than two disk accesses within this interval. Therefore, the number of disk accesses in g' will be smaller than in g.

Hence, we have determined the behavior of algorithm Γ in terms of the disk access requirement against the incoming bandwidth as well as the proxy buffer size. When the incoming bandwidth or proxy buffer size increases, the required number of disk accesses decreases, and vice versa. Figure 37 illustrates the relationship between the number of disk accesses, incoming bandwidth, and memory buffer size. A darker color is used to indicate a larger number of disk accesses.

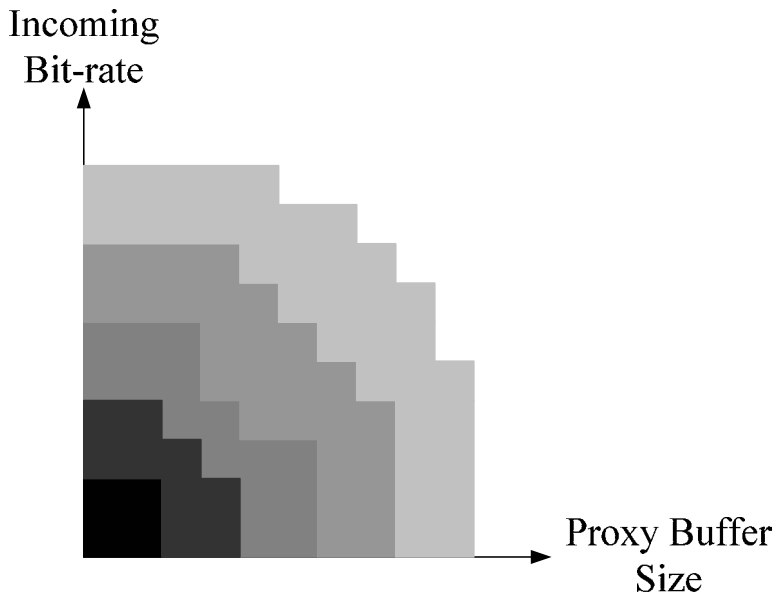


Figure 37: Relationship between Incoming Bandwidth, Proxy Buffer Size and Disk Access for Video Staging Algorithm Γ

4.3.3 Comparison with the Traditional Algorithm

Suppose we have a schedule $g = \Gamma(S, r, b)$. We know from the algorithm that $g_{i+1} - g_i < r$ when and only when $S_{i+1} - S_i < r$ and $g_{i+1} = S_{i+1}$ for a particular time i. In

addition, when this condition is satisfied we shall have $g_i = S_i$ as well; therefore, we shall have $g_{i+1} - g_i = S_{i+1} - S_i$. Conversely, when $S_{i+1} - S_i \geq r$ or $g_{i+1} \neq S_{i+1}$ then we shall have $g_{i+1} - g_i \geq r$, which implies a full utilization to the network bandwidth. Now, we consider the incoming network bandwidth utilization for the traditional video staging algorithm. The cutoff bandwidth, C in this case, should be equal to r . When $S_{i+1} - S_i \geq C = r$ the incoming network bandwidth will be fully utilized to be r , and when $S_{i+1} - S_i < C = r$ the utilized incoming network bandwidth will be $S_{i+1} - S_i$, which will be less than r. Let us consider again algorithm Γ . When we have $S_{i+1} - S_i < r$, it is not necessary to have $g_{i+1} - g_i = S_{i+1} - S_i < r$ as well, since it may not satisfy condition $g_{i+1} = S_{i+1}$. This means that at the time the traditional CBS algorithm does not fully utilize the network bandwidth algorithm Γ may still fully utilize the network bandwidth for data retrieval. As the traditional algorithm does not fully utilize the network bandwidth, it has to supplement that left over by the disk, leading to a situation whereby the disk access requirement of the traditional algorithm becomes higher than or in its best case the same as algorithm Γ .

Now, we can consider the traditional CAS algorithm. For an arbitrary point g_k on $f = \Phi(S, r, b)$ where $f_k > S_k$, we know that f will touch S again at points $f_k - m_0 r = S_{k-m_0}$ and $f_k + m_1 r = S_{k+m_1}$. Similarly, we can determine another detached point f_l for the next detached region so that $f_l - m_2 r = S_{l-m_2}$ and $f_l + m_3 r = S_{l+m_3}$ where $m_3 > m_2 > m_1 > m_0$. Therefore, we will have regions $[m_{2n}, m_{2n+1})$ as detached regions where the schedule will utilize the network bandwidth completely and $[m_{2n+1}, m_{2n+2})$ as connected regions where the schedule will not use up the network bandwidth. Consider point f_k' for any smoothing function f' applied on S . Suppose $f_k' > f_k$. It must be a detached region and it will touch S before m_0 and after m_1 since $f_k' - m_0 r > S_{k-m_0}$ and $f_k' + m_1 r > S_{k+m_1}$. It will have a better network utilization, as the detached region is larger; however, it is infeasible since it requires a larger memory buffer. Conversely, for $f_k' \leq f_k$, it may lie in a connected region or a detached region. In both cases, we have $f_k' - m_0 r \leq S_{k-m_0}$ and $f_k' + m_1 r \leq S_{k+m_1}$, which represent a smaller or

equally detached region than $[m_0, m_1)$. Therefore, it can only have smaller or in its best case the same network bandwidth utilization as algorithm Γ .

In other words, algorithm Γ always provides higher network bandwidth utilization. As it retrieves more data from the network, it can offload the disk requirement by reducing the need of data access and storage size. This gain in network utilization over that from traditional algorithms is extremely high when the incoming network bandwidth is high as they will waste much of the bandwidth in this case. Besides, the traditional algorithm will not benefit from supplying additional memory at its disposal, since the partitioning relies solely on the cutoff bandwidth. There is no relation between the memory buffer size and the incoming network bandwidth or disk access. We illustrate this in Figure 38.

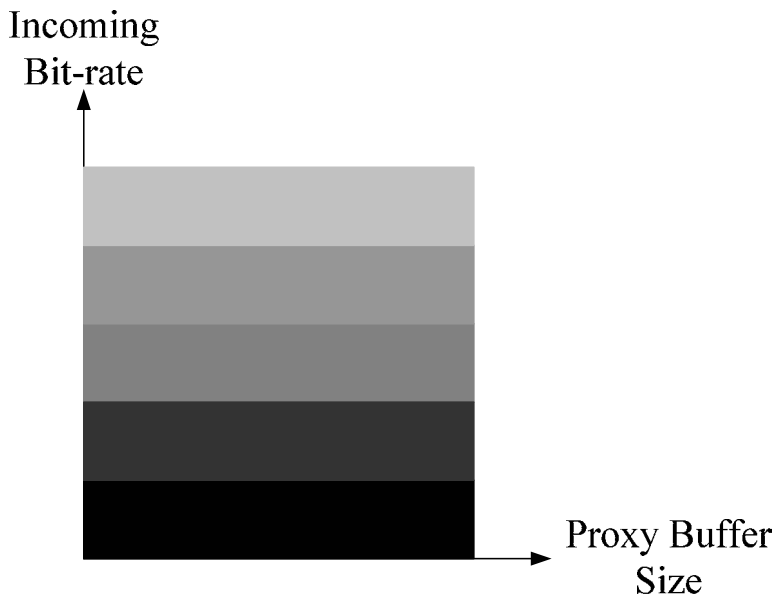


Figure 38: Relationship between Incoming Bandwidth, Proxy Buffer Size and Disk Access for the Traditional Video Staging Algorithm

To summarize, algorithm Γ is built based on algorithm Φ so that it can give an optimal schedule under a constrained proxy buffer size and network incoming bandwidth. These constraints are favorable to the scheduling of memory and network usages. The number of disk accesses will usually be smaller and will never be greater than the traditional video staging algorithm. It also provides us with the relationship between incoming network bandwidth,

memory buffer size, and disk access so that we can adjust the number of disk accesses to suit to any buffer size and incoming network bandwidth.

4.4 Experimental Results

Both the original video staging mechanism and the proposed video staging mechanisms require disk accesses and network transfers to achieve information retrieval. In order to ensure the quality of transfer, related resources such as network bandwidth, disk bandwidth and memory are usually reserved to guarantee the service. However, such reservations also limit the number of clients supportable by the proxy server, as no reservation can be made to exceed these resource capacities. Therefore, the ability to achieve better utilization of reserved resource becomes a figure of merit for a video staging algorithm.

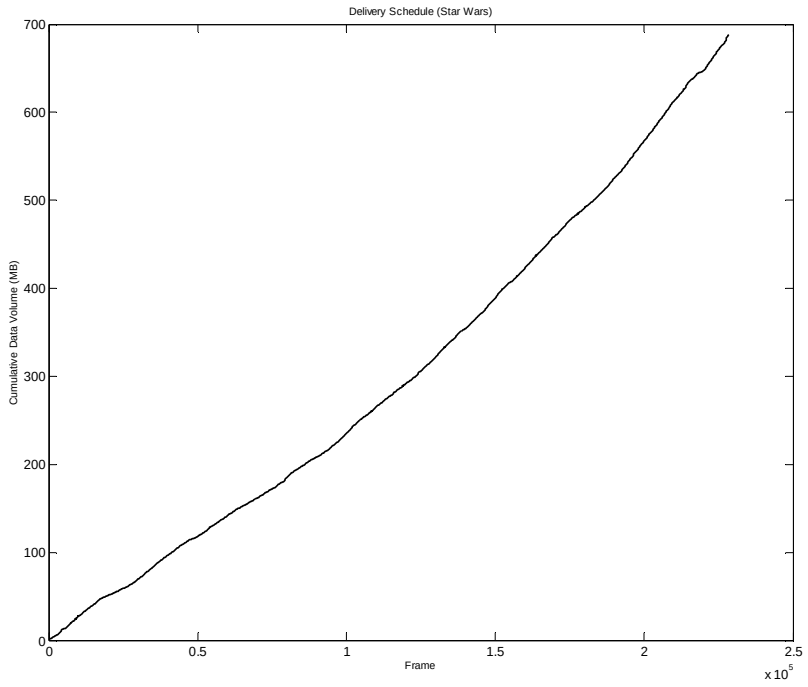


Figure 39: Data delivery schedule of video title "Star Wars"

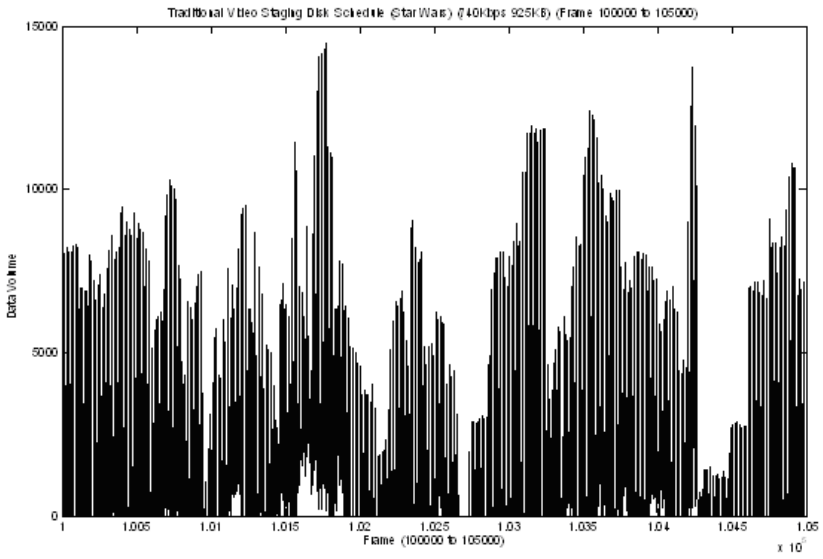


Figure 40: Disk access generated by traditional video staging algorithm (740Kbps 925KB)

Figure 39 shows the data delivery schedule of a video title "Star Wars". It is recorded at 30fps with an average bandwidth of 740Kbps. Figure 40 shows the disk schedule obtained by the traditional video staging algorithm. Figure 41 shows the traditional and the proposed video staging schedule Γ for such video with 740Kbps network bandwidth and 925KB memory buffer (equivalent to 10s of storage) within an arbitrarily chosen period (frame 100000 to 105000). This schedule requires small volume continuous disk accesses and is therefore unfavorable for practical implementation. However, it is understood that such dispersed disk accesses can be grouped together to form a few disk accesses. The volume is limited by the memory buffer constraint and the accesses should be initiated before starvation of the memory buffer. Therefore, we modified the traditional video staging algorithm according to these constraints to group the continuous disk accesses into a few separated ones for a fair comparison with the proposed algorithm. For the rest of this paper, we refer the traditional video staging schedule to this modified one and refer the gamma video staging schedule to the one generated by the proposed algorithm Γ . This same set of experiments has also been done on several other titles. As the results obtained are very similar and generate no new observation, we have excluded them.

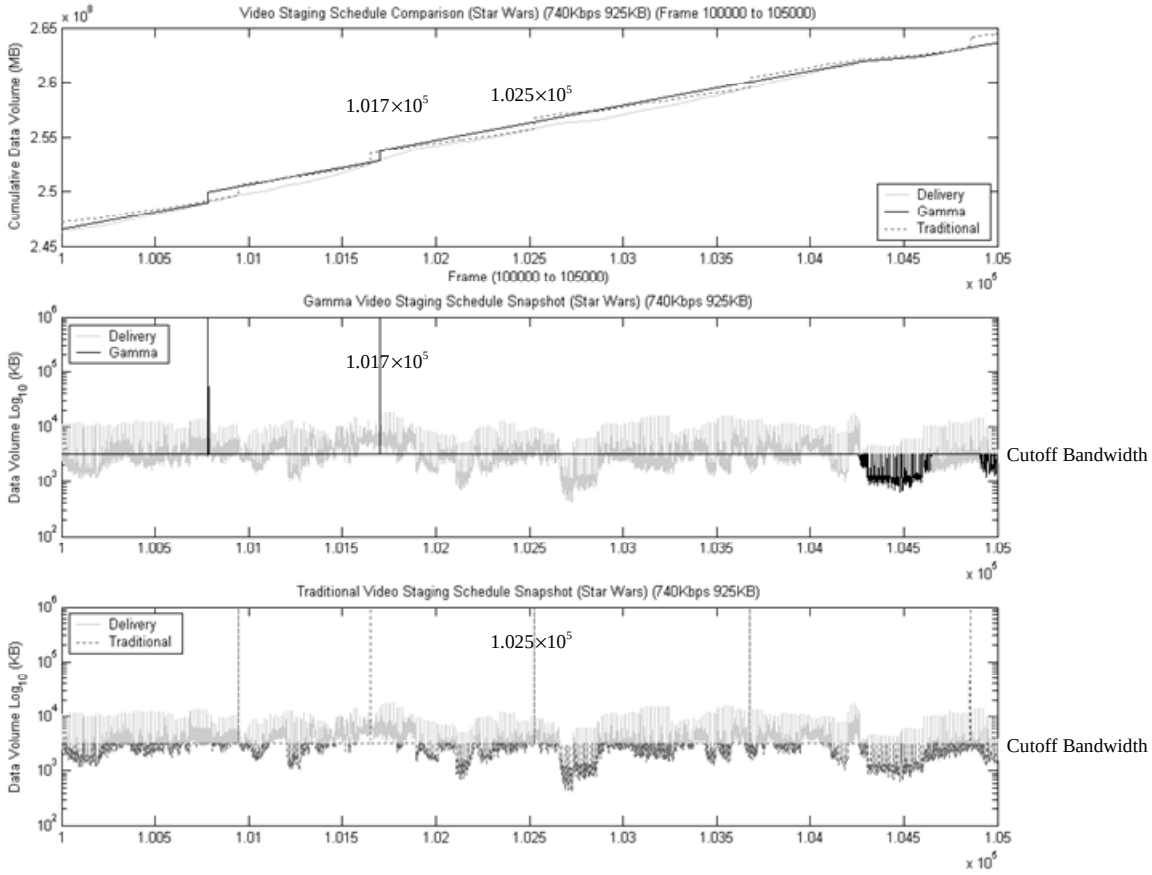


Figure 41: Video staging schedules of the proposed and traditional algorithms (740Kbps 925KB)

The topmost graph in Figure 41 shows the cumulative volume of data of the original schedule and video staging schedules obtained from the network and the disk using the proposed and traditional algorithms. The lower two graphs show the exact amount of data obtained for each frame plotted in logarithmic scale. Darker lines in the middle and the bottom graphs represent the results for the gamma (algorithm Γ) and the traditional video staging algorithms respectively. The lighter lines in these two graphs represent the original required amount of data in the delivery schedule. The impulses in these two graphs indicate the times when there are disk accesses, such as times around 1.017×10^5 in the middle graph and around 1.025×10^5 in the lower graph, while the remainder indicates network retrievals. The middle graph shows that the network part of the proposed arrival schedule for Γ maintains its network access at 740Kbps most of the time while

the traditional network schedule drops below 740Kbps very often during the period. In other words, the schedule of Γ has a better utilization of the network compared with the traditional one. Consequently, the traditional schedule has to compensate the loss from the network by the disk schedule. While the volume of data retrieved by a disk access is limited each time by the memory buffer size, the more data the disk schedule has to retrieve, the more disk accesses it will introduce. As shown in the graph, schedule Γ has to make only two disk accesses compared with five in the traditional schedule. When we cut the network bandwidth by half to 370Kbps as shown in Figure 42, both the traditional and Γ schedules utilize the network bandwidth efficiently and therefore their disk access requirements are both 10 during the period. However, if we increase the network bandwidth to 1110Kbps as shown in Figure 43, schedule Γ again gives a better network bandwidth utilization. As seen in the figure, even though schedule Γ does not require any disk access within the period, the traditional schedule still requires three disk accesses. Figure 44 and Figure 45 shows the result by changing the memory buffer size to 462KB and 1388KB respectively while maintaining the network bandwidth as 740Kbps. They both indicate that schedule Γ requires fewer disk accesses as this reduction is obtained from the utilization of wasted network bandwidth in the traditional algorithm which will not be affected by changing the memory buffer size.

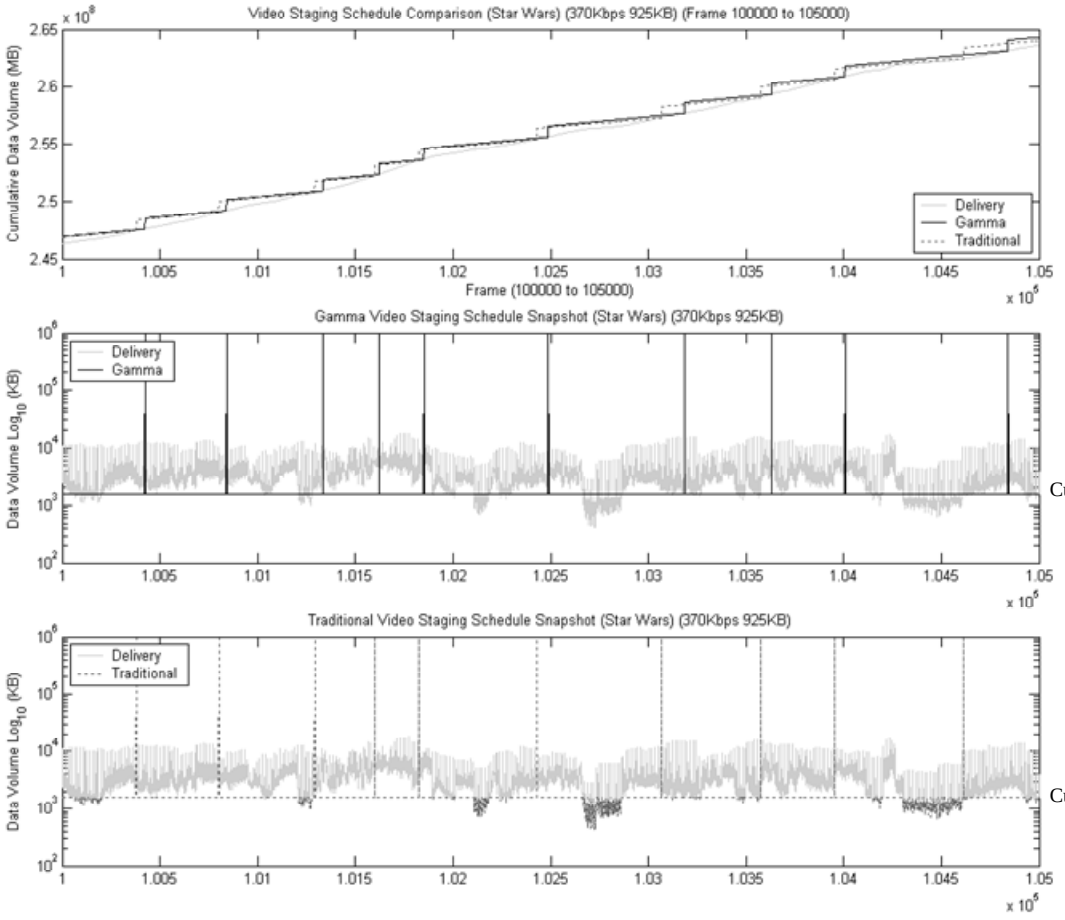


Figure 42: Video staging schedule of the proposed and traditional algorithms (370Kbps 925KB)

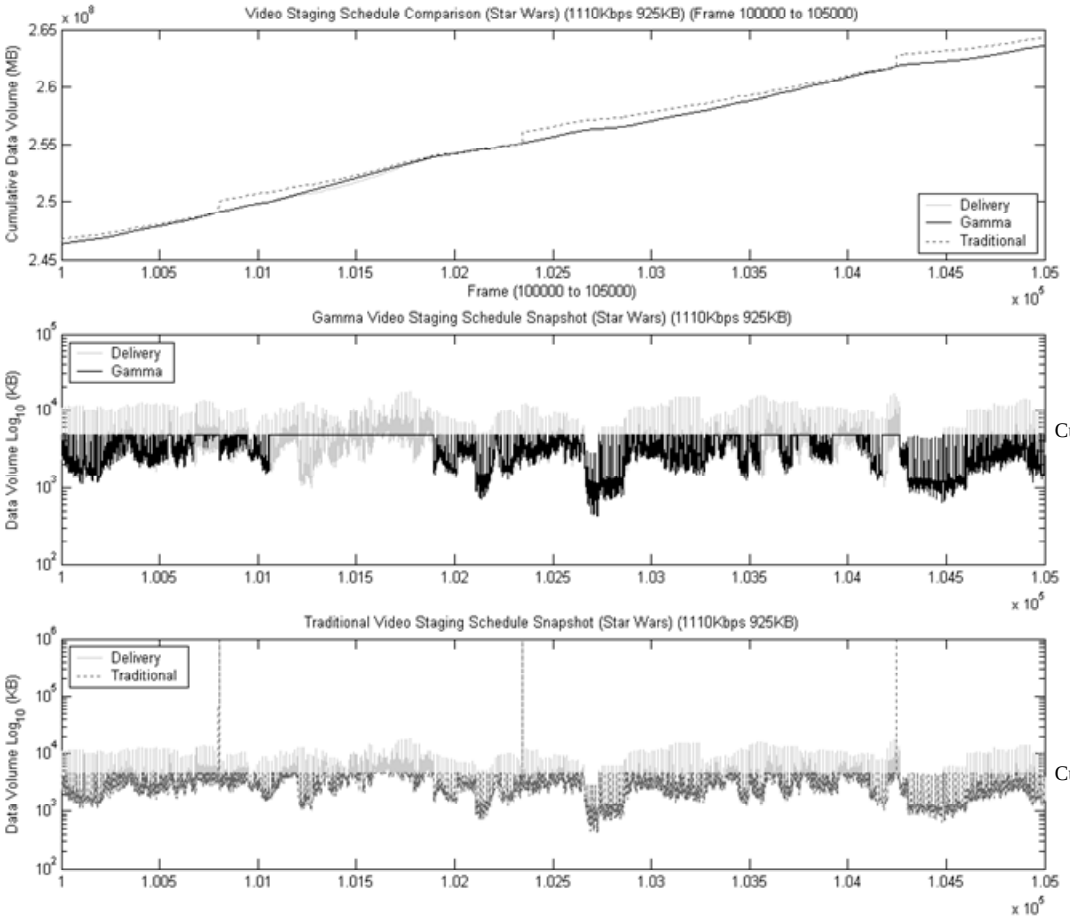


Figure 43: Video staging schedule of the proposed and traditional algorithms (1110Kbps 925KB)

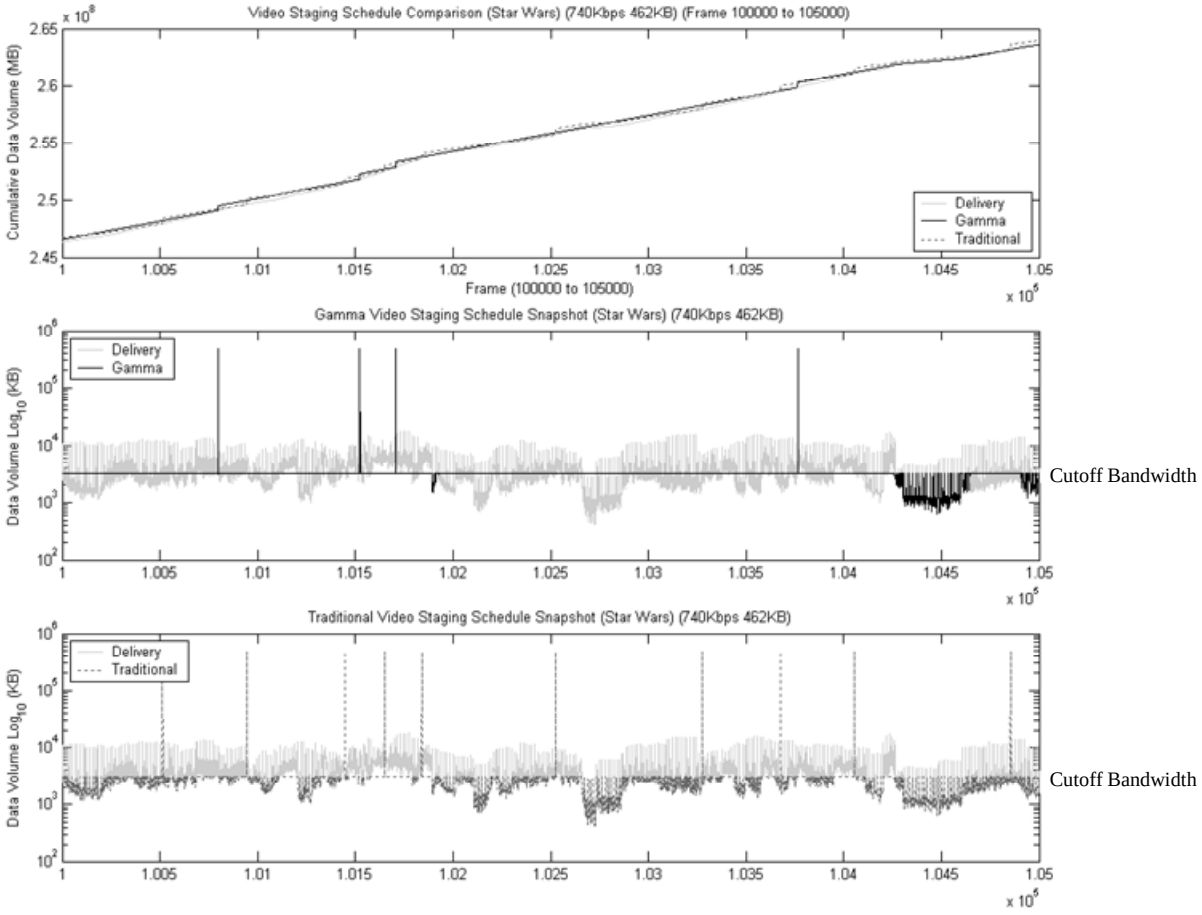


Figure 44: Video staging schedule of the proposed and traditional algorithms (740Kbps 462KB)

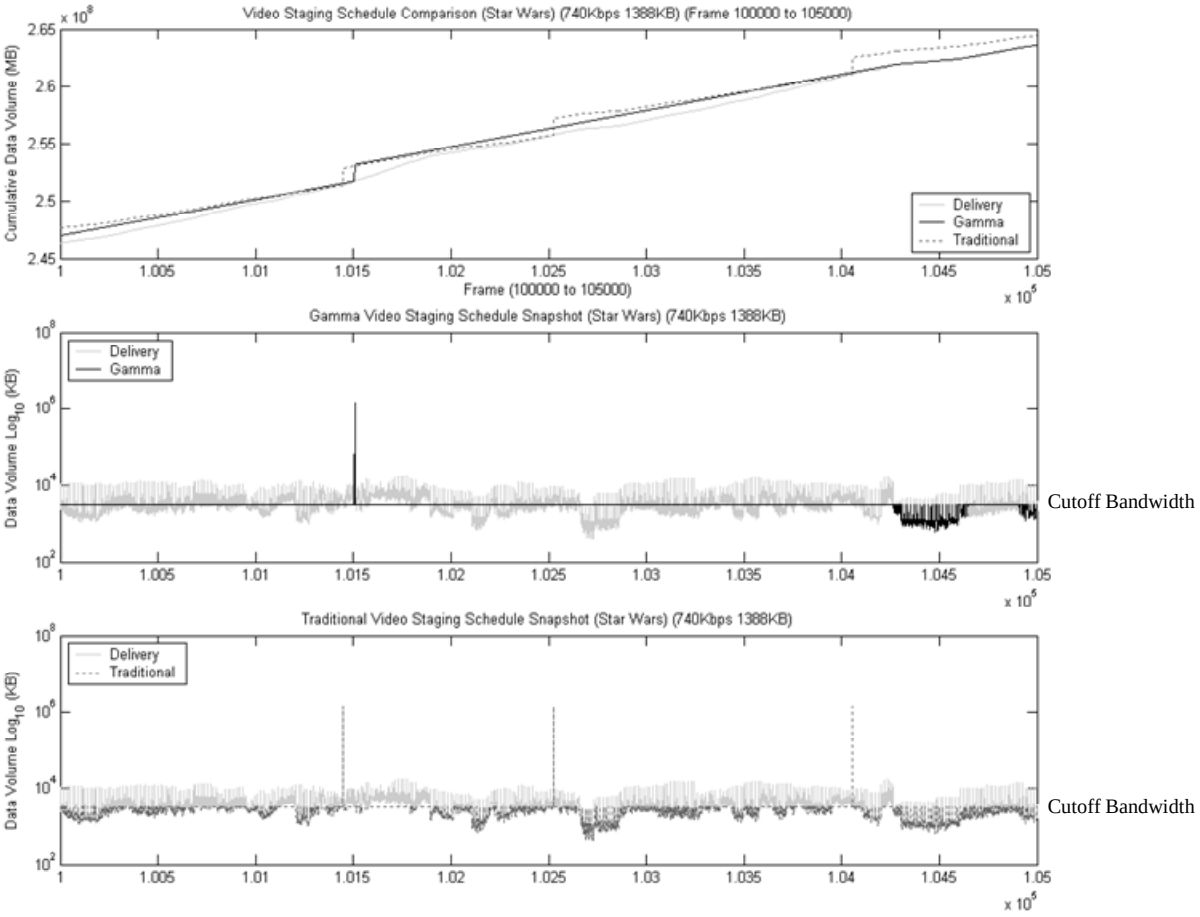


Figure 45: Video staging schedule of the proposed and traditional algorithms (740Kbps 1388KB)

Figure 46 shows a comparison on the volume of data retrieved from the network for the complete video title using Γ and the traditional algorithm under different network bandwidths and memory buffer constraints. We observe that given the same amount of incoming network bandwidth, algorithm Γ retrieves data closer to network bandwidth bound than the traditional algorithm. This implies that algorithm Γ has utilized its network bandwidth better. In addition, as the network bandwidth is increasing, the amount of data retrieved by algorithm Γ increases much faster than that received by the traditional one, as revealed in the analysis above.

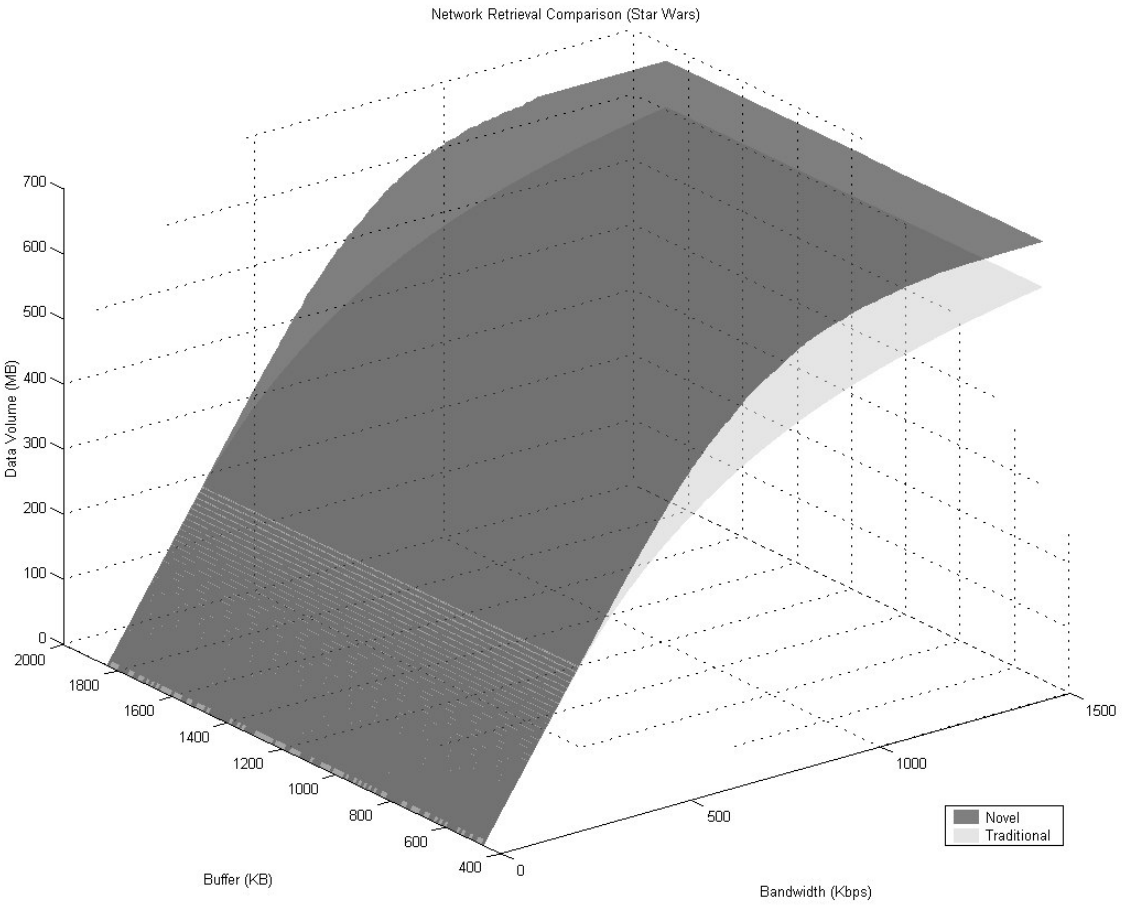


Figure 46: Network Retrieval Comparison

Figure 47 shows the amount of data stored in the disk for both algorithms in logarithmic scale. As the traditional algorithm does not fully utilize the reserved network bandwidth, the disk has to supplement the difference accordingly. As a result, the storage requirement for the traditional algorithm is higher than Γ with the same incoming bandwidth and memory buffer size. Figure 48 confirms the analysis of Figure 37. The number of disk accesses is directly proportional to the disk storage requirement. Figure 48 also shows that the disk access requirement of Γ is lower than that of the traditional algorithm, particularly when the bandwidth is large. This is because the network utilization of the traditional video staging algorithm decreases as the bandwidth increases; in order to compensate for this under-utilization, the disk storage requirement has to increase. This leads to a comparatively higher disk access requirement than in algorithm Γ .

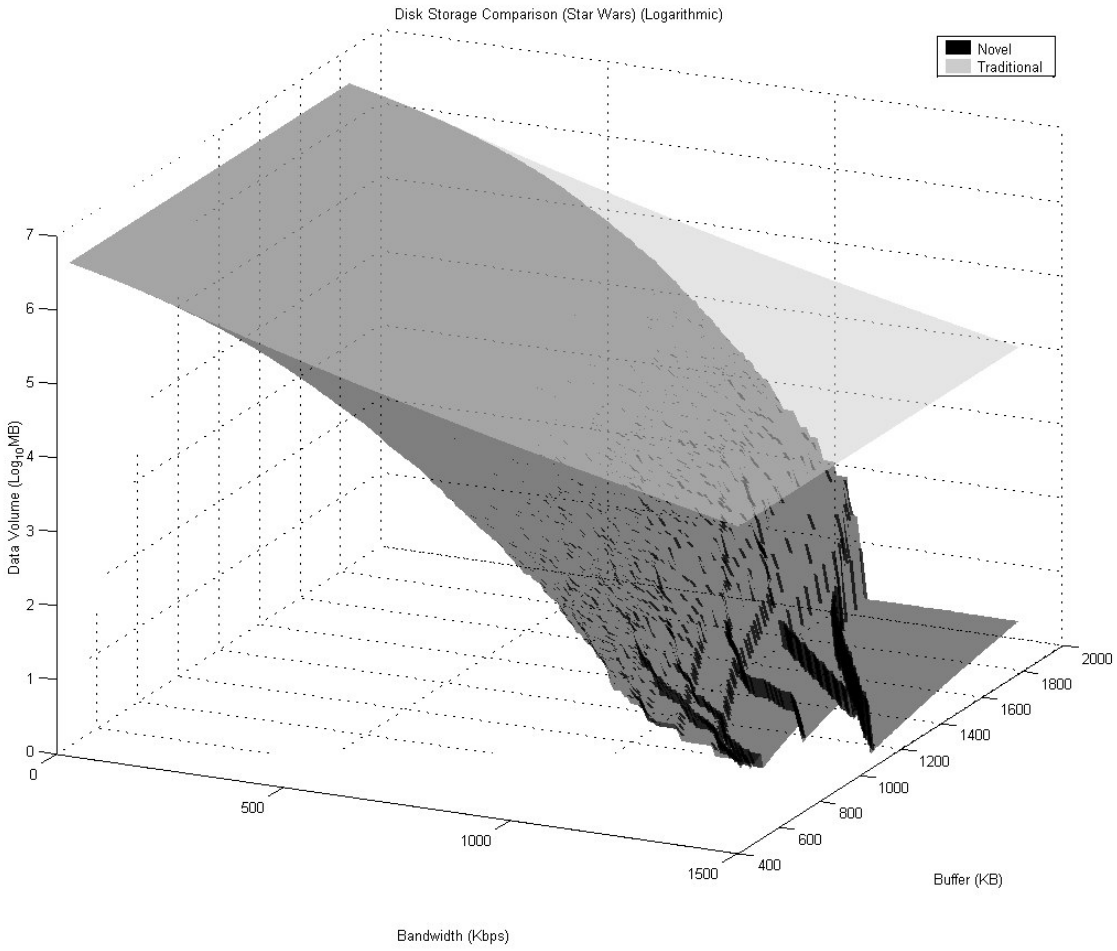


Figure 47: Disk Storage Comparison (Logarithmic)

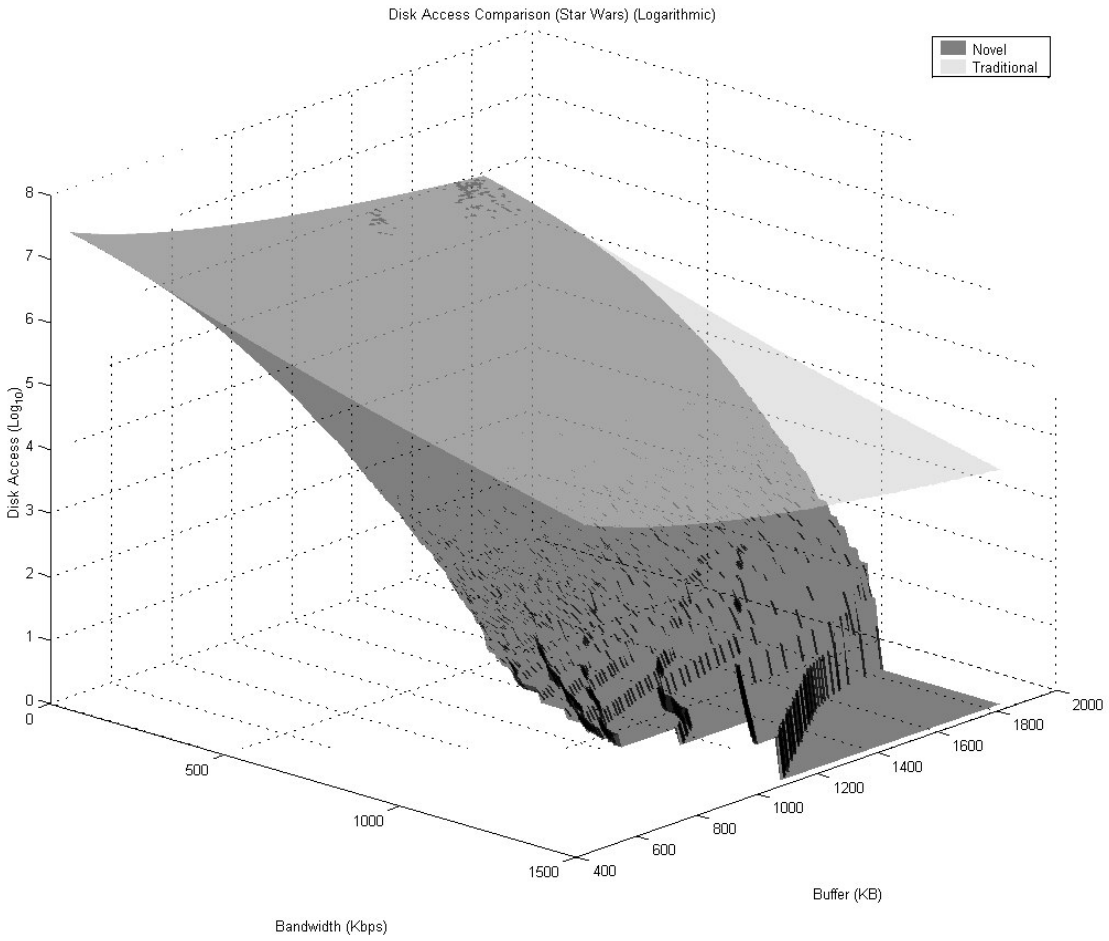


Figure 48: Disk Access Comparison (Logarithmic)

4.5 Summary

Video staging provides an effective method for relaxing the bandwidth requirement between the video server and the proxy server. It separates a video stream into two parts: one is obtained through the video server and the other is obtained from the local disk cache. Owing to the temporal locality of video streams, network bandwidth can be saved. However, the traditional video staging algorithm has not yet optimized the throughput of a proxy server, since network bandwidth is not the only resource required to support a connection; disk bandwidth and memory space are also essential. The maximum number of connections that a system can support is the minimum of the maximum number supportable by either of the disk, network or memory alone. Therefore, if a system is running out of any one of these resources, it cannot support any further connections, even

if it has the others in abundance. Hence, we should pursue the optimal usage of all these three kinds of resource in order to optimize the system throughput. In addition, we need to attain a specific pattern of usage schedule suitable for each of the resources so that when the resource is shared among different connections, its throughput can still be optimized.

In our analysis of algorithm Φ, we have shown that Φ constitutes an optimal schedule towards memory usage. As for the video staging algorithm Γ, it provides us with a way to interchange network bandwidth and memory usage within the proxy server. We have performed a series of simulations to verify the performance of algorithm Γ. We have shown that it provides us with the flexibility to increase memory usage in order to create a usage pattern more suitable for multiple connections. We have also shown that the proposed algorithm allows us to increase the number of disk accesses in order to maintain a fixed buffer size when the incoming bandwidth decreases. The proposed algorithm also gives a better performance in utilizing reserved resources. They are the benefits obtained from heterogeneous interchange of resource usage.

In short, in this section, we have,

1. Developed a video staging mechanism
2. Based on the properties of this mechanism, developed a resource utilization map depicting the relationship on the use of disk and network bandwidth as well as memory buffer

5 Grand Scale Scheduling for Admission Control

In Chapter 4, we have presented a video staging scheduling scheme for video streaming proxy services. This scheduling scheme allows usage interchange among different resources including network bandwidth, disk bandwidth, and memory. As a result, we are able to prevent bottlenecks forming from overuse of any one of these resources. Thus, we can utilize these resources efficiently.

A video streaming proxy server is required to handle multiple video streams concurrently; therefore, it must be able to guarantee all the resources required by these streams by reserving them in advance. A scheduler is responsible for reserving these resources during admission control upon the acceptance of a new request. A simple scheduling scheme can issue these reservations based on the maximum requirement but it will not be able to achieve a satisfactory utilization rate, since the actual requirement varies from time to time and underutilization is likely. Hence, some have proposed scheduling schemes such as Constant-Data-Length (CDL) and Constant-Time-Length (CTL) to improve utilization though an adoption of two-level scheduling. However, these approaches usually focus on disk bandwidth alone and ignore the memory constraint. In addition, these two layer approaches do not completely solve the problem of waste. To improve utilization, we analyze the relationship between the sum of all schedules and their actual resource requirements. Based on this relationship, we have developed the grand scale scheduling scheme by taking into account both disk bandwidth and memory usage. In addition, resource utilization has been raised through the interchange of reservations between different video streams.

5.1 Resource Usage in System Perspective

The operation of a video streaming proxy server is extremely resource demanding, even with today's elevated technology standard, as clients are not only voluminous but also exigent. In order to maintain service quality, a video proxy has to be equipped with enough memory, disk and network bandwidth. It has to adopt an admission control scheme to ensure resource sufficiency so that any admitted client can receive an acceptable quality of service.

Let us further elaborate on the resource requirements of a proxy server in its three modes of operation: forwarding, cache out, and cache in. We assume that a video streaming proxy is installed in a gateway separating the LAN from the WAN (e.g., the Internet) as in Figure 49. In this

forwarding mode, the proxy server only forwards the data it receives to the clients that request them. The data received from the ingress network through the WAN have a network bandwidth constraint of B ; these data will be held in the memory buffer with a size constraint M and sent out subsequently through the egress network with an unlimited bandwidth, comparing with the ingress network bandwidth.

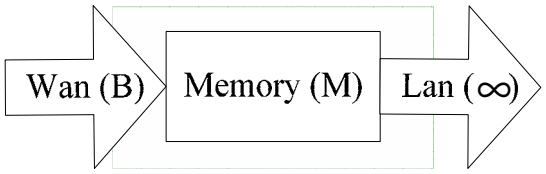


Figure 49: Proxy operating in forward mode

For the second case, we consider the cache-out operation mode of a proxy server with a double buffer system. In this case, all the video streams have already been stored in the local storage of the proxy server and are retrieved from the disk and sent to clients directly upon request, as shown in Figure 50. In this case, the data retrieval will be constrained by the disk bandwidth Δ . For the final case, we consider the proxy operating in the cache-in mode. In addition to forwarding the video streams from the ingress network to the egress network, the proxy also copies the video data to the local storage as shown in Figure 51.

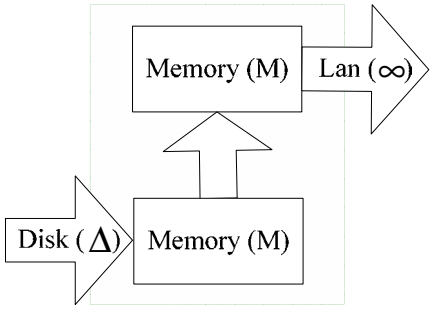


Figure 50: Proxy operating in cache-out mode

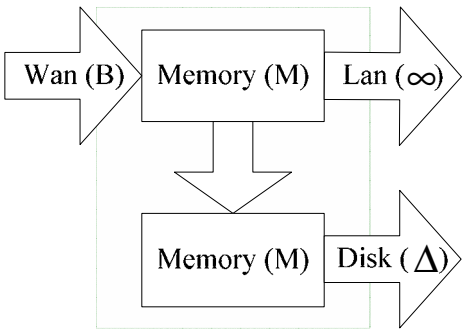


Figure 51: Proxy operating in cache-in mode

To ensure that the resource requirements lie within the system limit, a selection scheme is required to determine whether a new client should be admitted or rejected. This operation is called admission control. The simplest way to determine whether a new client can be admitted to the system is first to assume that the all-time requirement of the streams is equal to their maximum requirement. Hence, by comparing the maximum requirement with the system limit, we can determine whether a new client can be admitted. Obviously, such an approach oversimplifies the problem and will lead to a waste of resources.

In practice, a client will be admitted if the system is able to provide the required resources. However, as the resource requirement of a video stream requested by the client is closely related to the way it is scheduled; different scheduling schemes thus generate different criteria of admission control. The majority of the existing works in admission control focus on disk scheduling and deal with a physical disk access mechanism, which involves physical constraints such as seek time, rotation time, placement and so forth. They strive to achieve the most efficient plan for disk usage [36][37]. This is natural, as the disk bandwidth has always been the bottleneck of a video streaming platform. However, researches have already begun to take network bandwidth into account as well [38]. The emergence of disk arrays and gigabits networks has slackened these bandwidth constraints and they are no longer the only decisive factor in video streaming platform design. As a result, recent works have shown a trend towards combining all resources into the consideration including network bandwidth, disk bandwidth and memory [39][41]. In addition, some scheduling approaches [43][25][32][5][21][35] have provided the means to interchange the

usage of different supporting resources while providing the necessary guarantee for the playback of each stream.

Traditional approaches to providing admission control can be classified into three categories: deterministic, statistical, and predictive. The deterministic approach considers the aggregated worst-case requirement for admittance. Its decision is fail-safe but it usually over-reserves resources, leading to low throughput. The statistical approach makes use of certain parameters to estimate the overall usage of resources while the predictive approach takes a further step of speculating on the resource usages when deciding the admittance. These approaches reduce the waste from over-reservation but they are no longer fail-safe.

To reduce the wastage in the deterministic approach, a data scheduling approach based on disk scheduling known as the generalized constant-data-length (CDL) scheme was built [40]. The original constant-data-length scheme makes reservation of memory buffer and disk bandwidth into a stream according to its maximum requirements. Resources are wasted during the time when requirements are not at their peak. In view of this, the generalized constant-data-length scheme was proposed to shorten the duration of each reservation. This scheme uses a window to divide a stream into many different parts and evaluates the maximum resource requirements of each part independently. As a result, it can be determined that the peak reservation may be required only at a particular window, so a much lower reservation can be made for the rest, thus reducing wastage. A similar generalized constant-time-length (CTL) scheme has been proposed in the same work. In [42], this similar concept has also been applied to network scheduling.

In [43], a pair of two-layer scheduling schemes, Round-based CTL and CDL-EDF, was proposed. These two-layer schemes separate data access from data scheduling in the traditional disk scheduling concept. In the first layer, either the CTL or CDL scheme is adopted as the data access policy. The objective is to determine the deadline by which part of the data must be made available in the memory. The second layer of scheduling in the round-based CTL scheme assigns disk access in cycle with a fixed period similar to CTL. The second layer of the CDL-EDF scheme reschedules the data access deadlines in the Earliest-Deadline-First manner. In these approaches, the worst-case requirement obtained from the first layer scheduling is no longer reserved for all time, as this can be rescheduled into the second layer. However, the drawback in these two-layer

schemes is their memory consumption. Take a stream in CDL-EDF for example. A memory buffer of size equal to the "constant-data-length" of the CDL scheme is required at the time when the first layer scheme makes its decision. As the second layer EDF scheme may reschedule the disk access to an earlier time, a memory buffer of size large enough to store both the unconsumed data and the data to access is required. The memory buffer size will be doubled if the EDF scheme is allowed to reschedule the disk access up to the time of the last disk access.

All variants of CDL and CTL schemes operate on the client individually. They provide a disk schedule and a memory schedule based solely on the stream required by a client. The lack of communications between these individual considerations reduces the opportunity of reaching the best utilization of resources. For instance, the memory buffer assigned to a stream cannot be re-assigned to other streams even if it is not in use. The key to such a possibility in resource sharing is in the knowledge of the whole. However, to consider the requirements of all clients at all times can be a very tedious process that can slow down the admission control process dramatically.

5.2 Grand Scale Scheduling

Let us consider a scenario in which our proxy streaming server is operating in the cache-out mode and serving n clients $c_0, c_1, \ldots, c_{n-1}$ while they are making requests to different video streams with the data consumption characterized by these functions $s_0(t), s_1(t), \ldots, s_{n-1}(t)$ respectively. The function $s_v(t)$ actually indicates the playback schedule of the video stream requested by client v. As pointed out earlier, the major competing resource in the cache-out mode is the disk bandwidth. When admitting a new client, we need to ensure that the disk bandwidth is big enough to provide the data required by the clients in time.

We begin by determining the lower bound volume of data for which an individual stream will be transferred from the disk with a specific number of disk accesses within any confined period of a schedule.

Lemma 5-1:

If a CDL schedule has i disk accesses with each disk access retrieving a data size of b in an arbitrary period (t', t''), then the minimum amount of data transferring from the disk for this schedule is $\max(0, i-1)b$ within this period.

Proof:

If a CDL schedule does not require any disk accesses in the period (t', t''), the data required to be transferred from the disk must be 0. If a CDL schedule requires 1 disk access of amount b in the period (t', t''), the data required to be transferred from the disk will be lower bounded to 0. This is because, as each data transfer takes time, the actual data transfer will be done outside the period (t', t'') if the disk access request is made exactly at the end of the period, i.e. t''. Hence, the minimum amount of data transferred from the disk for this schedule is 0 within this period. In general, if a CDL schedule has i disk accesses, with each disk access retrieving a data size of b in an arbitrary period (t', t''), then the minimum amount of data transferring from the disk for this schedule is $\max(0, i-1)b$ within this period.

Now, we suppose there is no client leaving or joining the system. If we confine our view to a specific period (t', t''), we shall find that some streams do not require any disk access and some streams require several. If we denote the number of streams having i disk accesses of size b_k as $\nu(i,k)$ within this period, then according to lemma 5-1, the data transferred from the disk as requested by all these streams within this period will be lower bounded by $\sum_{i=0}^{\infty}\sum_{k=1}^{\infty}\nu(i,k)\max(0,i-1)b_k$. In addition, if we let $\mu(k)$ be the number of schedules that will retrieve b_k data from the disk in each access, we shall have $\sum_{i=0}^{\infty}\nu(i,k)=\mu(k)$. This is obvious since, at any period of time, the total number of streams making disk accesses of size b_k is a constant, irrespective of the number of disk accesses they are making.

Next, we turn to construct the so-called grand scale function $S(t)$ by summing up all the individual schedules such that $S(t)=\sum_{v=0}^{n-1}s_v(t) \quad \forall 0 \le t < \infty$, as shown in Figure 52. We can view $S(t)$ as the schedule of a single virtual stream supported by a server. We arbitrarily set the

size of each virtual disk access of this virtual stream to be $\mathrm{M}_{\text{out}} = \sum_{k=0}^{\infty} \mu(k) b_k$, which indicates the total amount of buffer memory that will be allocated to all streams. We shall show later that by setting $\mathrm{M}_{\text{out}} = \sum_{k=0}^{\infty} \mu(k) b_k$, we can easily relate M_{out} with the actual amount of data transferred from the disk within any period of time.

By setting $\mathrm{M}_{\text{out}} = \sum_{k=0}^{\infty} \mu(k) b_k$, we shall have its corresponding CDL memory schedule $M(t)$. At each time a virtual disk access in this grand scale function is scheduled, we may assume that $S(t) = M(t)$. Moreover, if t_τ denotes the time of the τ^{th} disk access, then we are assured that $M(t_\tau) - M(t_{\tau-1}) = \mathrm{M}_{\text{out}} = \sum_{k=0}^{\infty} \mu(k) b_k$. Therefore, we also have $S(t_\tau) - S(t_{\tau-1}) = \mathrm{M}_{\text{out}} = \sum_{k=0}^{\infty} \mu(k) b_k$, which means that the volume of data of all schedules required from the disk within this period is equal to $\sum_{k=0}^{\infty} \mu(k) b_k$. We should bear in mind that these disk accesses are virtual, and do not represent actual disk accesses. However, as mentioned previously, these virtual disk accesses have a close relationship with the actual disk accesses. They help us to simplify the problem, since now we only need to deal with one virtual stream rather than n streams.

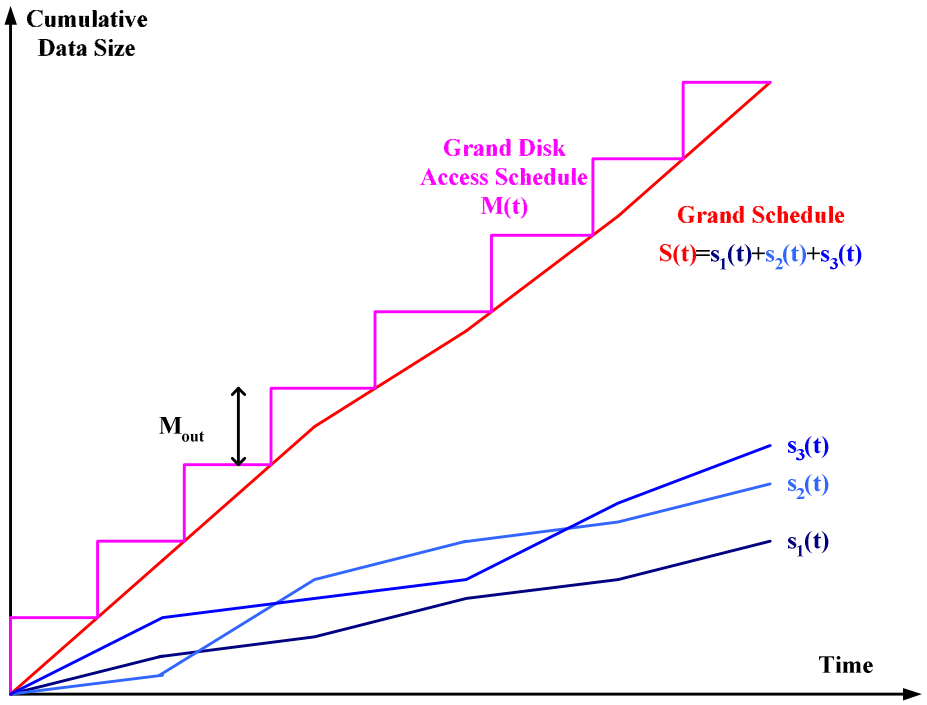


Figure 52: Grand Scale Schedule

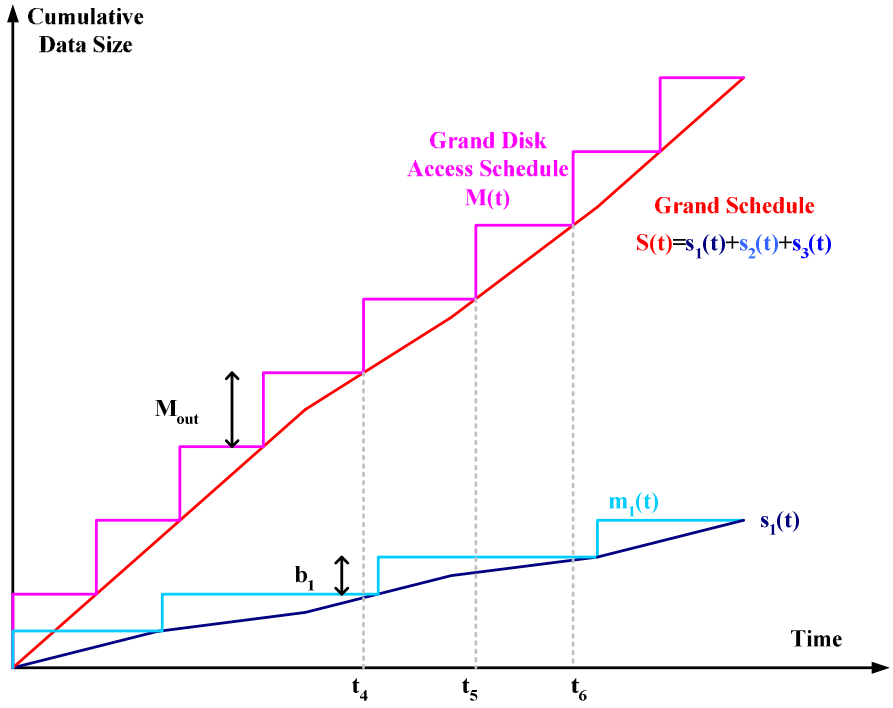


Figure 53: Grand Scale Schedule and Individual Schedule

Now, let us confine our view to a particular period of time $(t_{\tau-1}, t_\tau)$, which is embraced by two virtual disk accesses of the grand scale schedule. During this confined period, the grand scale schedule will require virtually an exact size of data $\sum_{k=0}^{\infty} \mu(k) b_k$ from the disk. As for the actual schedules, we know from the discussion above that a minimum of $\sum_{i=0}^{\infty} \sum_{k=1}^{\infty} \nu(i,k) \max(0, i-1) b_k$ data will be transferred from the disk. However, the data transferred due to all actual schedules inside the confined period should not be larger than $\sum_{k=0}^{\infty} \mu(k) b_k$, which is the total amount of buffer allocated to contain the data to be delivered. Hence,

$$\sum_{i=0}^{\infty} \sum_{k=1}^{\infty} \nu(i,k) \max(0, i-1) b_k \le \sum_{k=1}^{\infty} \mu(k) b_k \qquad <1>$$

The above equation shows that, although some streams may retrieve data a few times larger than b_k from the disk, there should be some streams that retrieve data less than b_k, since their summation is bounded by a constant $\sum_{k=0}^{\infty} \mu(k) b_k$. To be specific, we find that, within a confined period embraced by two virtual disk accesses of the grand scale schedule, there is a rule governing the number of streams with more than two actual disk accesses to the number of streams with less than two actual disk accesses.

Lemma 5-2:

The number of streams with actual disk accesses larger than 2 is confined by the number of streams with actual disk accesses smaller than 2 within a specific period embraced by two virtual disk accesses of their corresponding grand scale schedule.

$$\sum_{i=3}^{\infty} \sum_{k=1}^{\infty} \nu(i,k)(i-2)k \le \sum_{i=0}^{1} \sum_{k=1}^{\infty} \nu(i,k) k$$

Proof:

From inequality<1>, we can deduce the following by expanding and rearranging the terms:

$$\sum_{i=1}^{\infty} \sum_{k=1}^{\infty} \nu(i,k)(i-1) b_k \le \sum_{i=0}^{\infty} \sum_{k=1}^{\infty} \nu(i,k) b_k$$

$$\sum_{k=1}^{\infty} v(2,k)b_k + \sum_{k=1}^{\infty} 2v(3,k)b_k + \ldots + \sum_{k=1}^{\infty} (\iota - 1)v(\iota,k)b_k + \ldots$$

$$\leq \sum_{k=1}^{\infty} v(0,k)b_k + \sum_{k=1}^{\infty} v(1,k) + \ldots + \sum_{k=1}^{\infty} v(\iota,k)b_k + \ldots$$

$$\sum_{k=1}^{\infty} v(3,k)b_k + 2\sum_{k=1}^{\infty} v(4,k)b_k + \ldots + \sum_{k=1}^{\infty} (\iota - 2)v(\iota,k)b_k + \ldots$$

$$\leq \sum_{k=1}^{\infty} v(0,k)b_k + \sum_{k=1}^{\infty} v(1,k)b_k$$

$$\sum_{i=3}^{\infty}\sum_{k=1}^{\infty} v(i,k)(i-2)b_k \leq \sum_{i=0}^{1}\sum_{k=1}^{\infty} v(i,k)b_k$$

Without loss of generality, we may assume that $b_k = kb$ *, then we will obtain*

$$\sum_{i=3}^{\infty}\sum_{k=1}^{\infty} v(i,k)(i-2)k \leq \sum_{i=0}^{1}\sum_{k=1}^{\infty} v(i,k)k$$

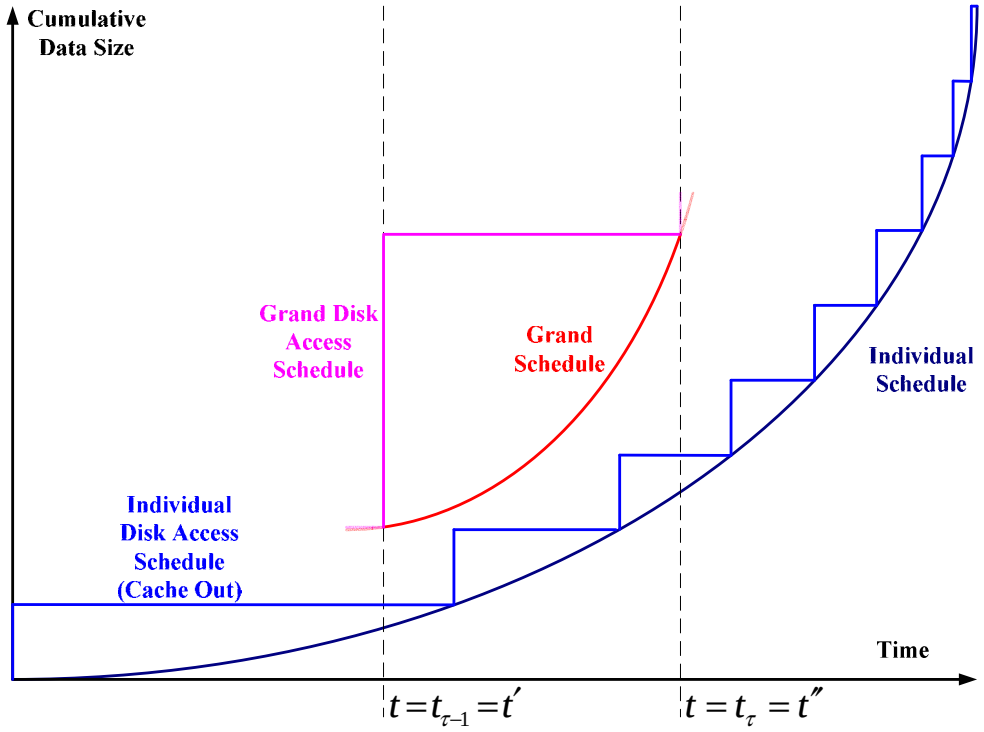


Figure 54: Cache-Out Grand Schedule

Figure 54 depicts an example showing the relationship between an actual schedule and its corresponding grand scale schedule operating in the cache-out mode. It shows that the actual

schedule is required to make two disk accesses during the interval between two virtual disk accesses in the grand scale schedule. Lemma 5-2 states that there may be other schedules making more disk accesses during this interval but they will always be compensated by schedules making fewer or zero disk access. From the result of Lemma 5-2, we discovered the relationship associating the actual amount of data transferred from the disk to the amount of data required by a virtual disk access within a particular period of time.

Theorem 5-3:

The total amount of data transferred from the disk due to all actual schedules will not exceed two times the data required by the grand scale schedule within the period confined by two consecutive virtual disk accesses.

$$\frac{\sum_{i=0}^{\infty}\sum_{k=1}^{\infty}\nu(i,k)ik}{\sum_{i=0}^{\infty}\sum_{k=1}^{\infty}\nu(i,k)k} \le 2$$

Proof:

From Lemma 5-2, we have the following:

$$\sum_{i=3}^{\infty}\sum_{k=1}^{\infty}\nu(i,k)(i-2)k \le \sum_{i=0}^{1}\sum_{k=1}^{\infty}\nu(i,k)k$$

$$\sum_{i=3}^{\infty}\sum_{k=1}^{\infty}\nu(i,k)ik \le \sum_{i=0}^{1}\sum_{k=1}^{\infty}\nu(i,k)k + 2\sum_{i=3}^{\infty}\sum_{k=1}^{\infty}\nu(i,k)k$$

$$\sum_{k=1}^{\infty}\nu(0,k)(0)k + \sum_{k=1}^{\infty}\nu(1,k)(1)k + \sum_{k=1}^{\infty}\nu(2,k)(2)k + \sum_{i=3}^{\infty}\sum_{k=1}^{\infty}\nu(i,k)ik$$

$$\le \sum_{k=1}^{\infty}\nu(0,k)(0)k + \sum_{k=1}^{\infty}\nu(1,k)(1)k + \sum_{k=1}^{\infty}\nu(2,k)(2)k + \sum_{i=0}^{1}\sum_{k=1}^{\infty}\nu(i,k)k + 2\sum_{i=3}^{\infty}\sum_{k=1}^{\infty}\nu(i,k)k$$

$$\sum_{i=0}^{\infty}\sum_{k=1}^{\infty}\nu(i,k)ik \le 2\sum_{k=1}^{\infty}\nu(0,k)(0)k + 2\sum_{k=1}^{\infty}\nu(1,k)(1)k + 2\sum_{k=1}^{\infty}\nu(2,k)(2)k + 2\sum_{i=3}^{\infty}\sum_{k=1}^{\infty}\nu(i,k)k$$

$$\sum_{i=0}^{\infty}\sum_{k=1}^{\infty}\nu(i,k)ik \le 2\sum_{i=0}^{\infty}\sum_{k=1}^{\infty}\nu(i,k)k$$

$$\frac{\sum_{i=0}^{\infty}\sum_{k=1}^{\infty}\nu(i,k)ik}{\sum_{i=0}^{\infty}\sum_{k=1}^{\infty}\nu(i,k)k} \le 2$$

In other words, if we construct a grand scale schedule with buffer size $\mathrm{M}_{\mathrm{out}} = \sum_{i=0}^{\infty}\sum_{k=1}^{\infty} v(i,k)b_k$, then within a confined period defined by two consecutive virtual disk accesses, the actual data transferred from the disk in this period can be as much as $2\mathrm{M}_{\mathrm{out}}$. Theorem 5-3 is not only applicable to the cache-out operation; it can be shown that, with a minor modification, theorem 5-3 is also applicable to the cache-in operation.

5.3 Admission Control Mechanism

We can employ the result obtained from Section 5.2 to develop an admission control mechanism for a video streaming proxy server performing the cache-out operation. Suppose we have a system configured with $\mathrm{M}_{\mathrm{out}}$ buffer memory and we assume that each virtual disk access of the grand scale schedule will require $\mathrm{M}_{\mathrm{out}}$ from the disk. With this assumption, a virtual disk schedule will be generated that defines the times when the virtual disk accesses will occur. While $\mathrm{M}_{\mathrm{out}}$ is a constant, the times of the virtual disk accesses will change according to the number of stream requests that have been accepted. In general, the times between two consecutive virtual disk accesses will be decreasing as the number of streams increases (this means more virtual disk accesses are required), although there can be exceptions. Now, we denote d_i as the length of the period between virtual disk accesses $i-1$ and i . Theorem 5-3 indicates that there can be as much as $2\mathrm{M}_{\mathrm{out}}$ data transfer from the disk during the period (t_{i-1}, t_i) . Assuming the disk system has a total bandwidth Δ , and then the minimum period between two consecutive virtual disk accesses is given by $d_{\min} = \frac{2\mathrm{M}_{\mathrm{out}}}{\Delta}$. When admitting a stream, we need to check whether $d_i \geq d_{\min} \quad \forall i$. Depending on the actual disk access policy, it may be necessary to add an offset ε to the calculation to account for the rotational and seek latency. To summarize, the policy for admitting a new video stream to a system with a virtual disk access period d_i and a $\mathrm{M}_{\mathrm{out}}$ memory constrained grand scale schedule can be described as follows:

$$\begin{cases} d_i < \dfrac{2\mathrm{M}_{\mathrm{out}}}{\Delta} + \varepsilon & \text{reject} \\ d_i \geq \dfrac{2\mathrm{M}_{\mathrm{out}}}{\Delta} + \varepsilon & \text{accept} \end{cases} \qquad \forall i : i > 0; \varepsilon > 0 \qquad <2>$$

We formulate the criteria above based on the expectation of maximum memory buffer usage over a period confined by two virtual disk accesses on the grand scale schedule. We have made several assumptions in the formulation. First, we assume that there are no dominating video streams that have a very high memory buffer requirement compared with the others. Since the dominating stream may give rise to a transient fluctuation in the memory buffer requirement that largely exceeds the expectation, the criteria may fail on these occasions. Second, we assume that the number of video streams involved is not very small in order to guarantee the first assumption. Finally, we also assume that the playback requests are not simultaneous. The effect of simultaneous playbacks is similar to that of having a dominating stream at the moment of commencement when all the streams are trying to fill up their buffers at the same time.

The criteria can be further relaxed if we make a further assumption regarding the playback requests. Normally, not all schedules meet their worst-case requirement at the same time. On average, their requirement is equal to half of the worst-case requirement. Some schedules ask for more and some for less, but their combined requirements can be cancelled out if the number of schedules is large enough and their average requirement is roughly the same. Under this assumption, the actual data transfer from the disk within two virtual disk accesses can only be half of the original one, i.e. $\mathrm{M}_{\mathrm{out}}$. Therefore, the admission control mechanism for the relaxed system remains the same, but the criterion can be adjusted as follows:

$$\begin{cases} d_i < \dfrac{\mathrm{M}_{\mathrm{out}}}{\Delta} + \varepsilon & \text{reject} \\ d_i \geq \dfrac{\mathrm{M}_{\mathrm{out}}}{\Delta} + \varepsilon & \text{accept} \end{cases} \qquad \forall i : i > 0; \varepsilon > 0 \qquad <3>$$

We can satisfy the disk bandwidth and memory space constraints for admission control of the cache-in and forwarding operations in a similar way to that of the cache-out operation.

5.4 Performance

Several criteria may serve as the figures of merit for the comparison between different scheduling and admission control schemes. The first is speed. Naturally, clients expect to receive responses to their requests promptly from the video streaming server. The second concern is admission capacity. Systems with similar resource configuration but employing different admission control schemes may have different capacities due to their resource utilization effectiveness. The higher the utilization rate, the better the admission scheme is.

We may verify the performance of the proposed scheme over these criteria through simulations. Firstly, we create an ordered set of video streams, each 1000s long, through random selection from a two hours title, “Star Wars”. Then, we schedule these streams by different scheduling schemes until the system reaches its capacity and then we record the average decision time. Instead of conducting an exhaustive test over all combinations of different parameters, we assume that there is no interrelation between the two parameters and we select one set of parameters as a norm before conducting the test over one parameter each time. The normal setting we selected is 160MBps disk bandwidth, 1GB memory, clip length 1000s, test length 10000s, and buffer size equal to 15s times average bandwidth of the video stream under schedule. The same set of experiments has also been done on the other video titles. However, as the result thus obtained is very similar. We have excluded them.

We have selected several scheduling schemes for comparison. They are two one-level schemes – CDL (Constant-Data-Length) and CTL (Constant-Time-Length) – and several two-level schemes – such as CDL-CDL, CDL-CTL and CTL-CDL – and the grand scheduling scheme G-CDL (Grand-Constant-Data-Length). We have assumed that all of these two-level schemes adopted double buffer and we have ignored the crossover tests. Therefore, their actual schedules may consume more memory than our result indicates. This makes our result a theoretical maximum for any practical implementation that has included the checks such as those in the schemes as suggested by Shau, et al. in [43].

CDL	8.3344
CTL	8.0886
CDLCDL	14.6456
CDLCTL	15.0096
CTLCDL	15.1182
GCDL	15.7637

Table 1: Average Processing Time (ms)

Table 1 lists the average processing time taken in different schemes to admit a client. To summarize, two-level schemes require about double the time for completion compared with the one-level schemes. Moreover, the time taken by a corresponding grand scheduling scheme is roughly the same. Figure 55 depicts the relationship between capacities of different scheduling schemes and disk bandwidths. It shows that the grand scale schedule provides higher acceptance with different disk bandwidth compared with the other schemes.

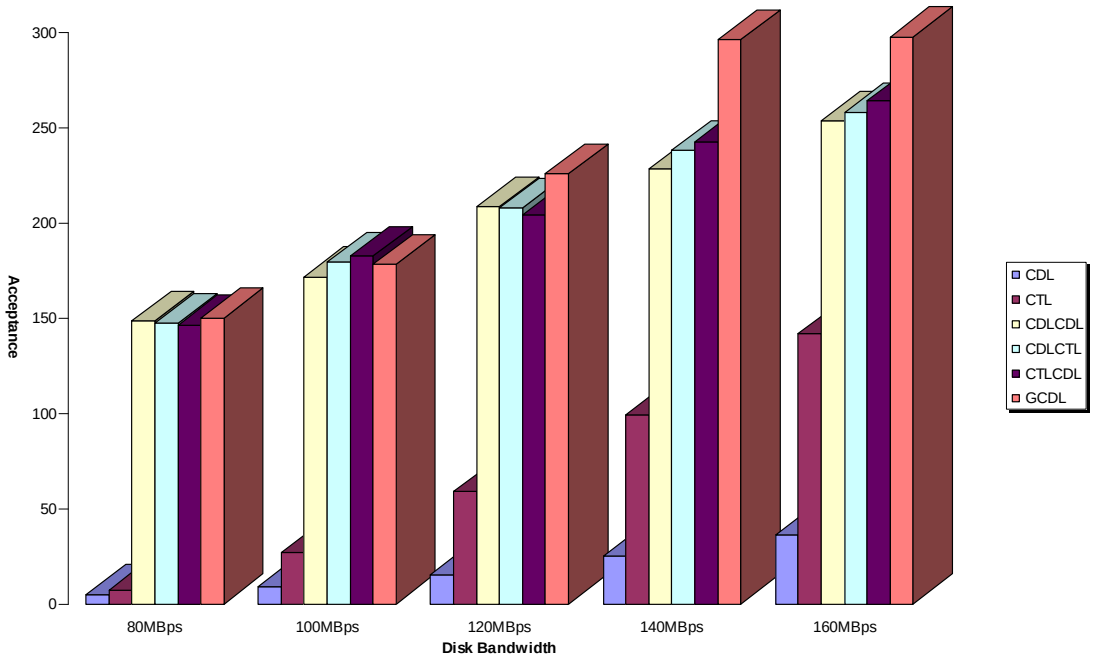


Figure 55: Acceptance variation with Disk Bandwidth

Figure 56 depicts the relationship between the capacity and the variation in memory size and it shows that the grand scale schedule performs better than the other schemes. However, we also observe that the variation in capacity against memory size is not linear. The variation stems from the modulus relation between disk bandwidth and memory size as shown in Figure 57.

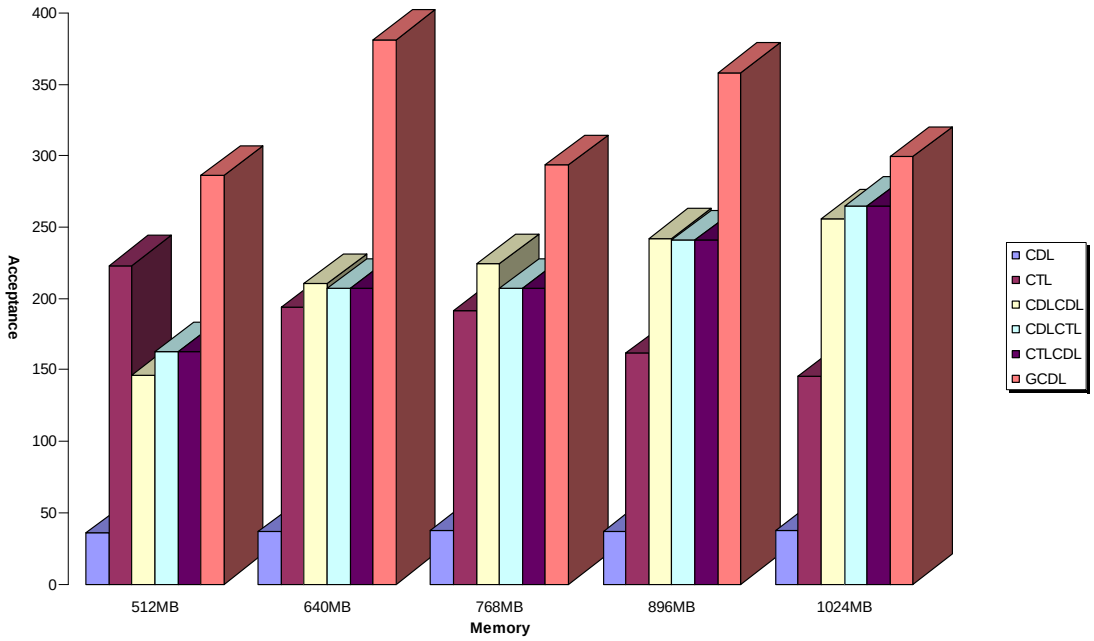


Figure 56: Acceptance variation with Memory Size

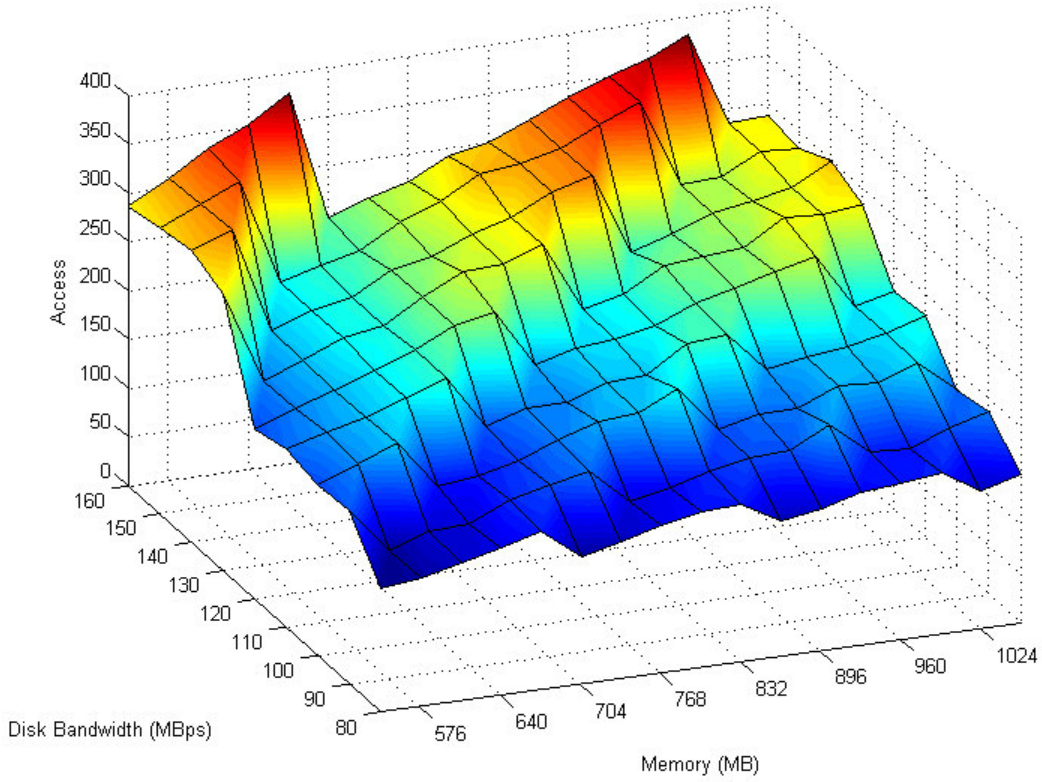


Figure 57: Acceptance variation with Memory Size and Disk Bandwidth

The time span of the test indirectly controls the density of requests. Each of the playback requests represents a video stream of 1000s in length; when the time span is, for instance, 2000s, all of these requests will overlap together. Figure 58 shows that the density of the requests has no significant effect on the capacity.

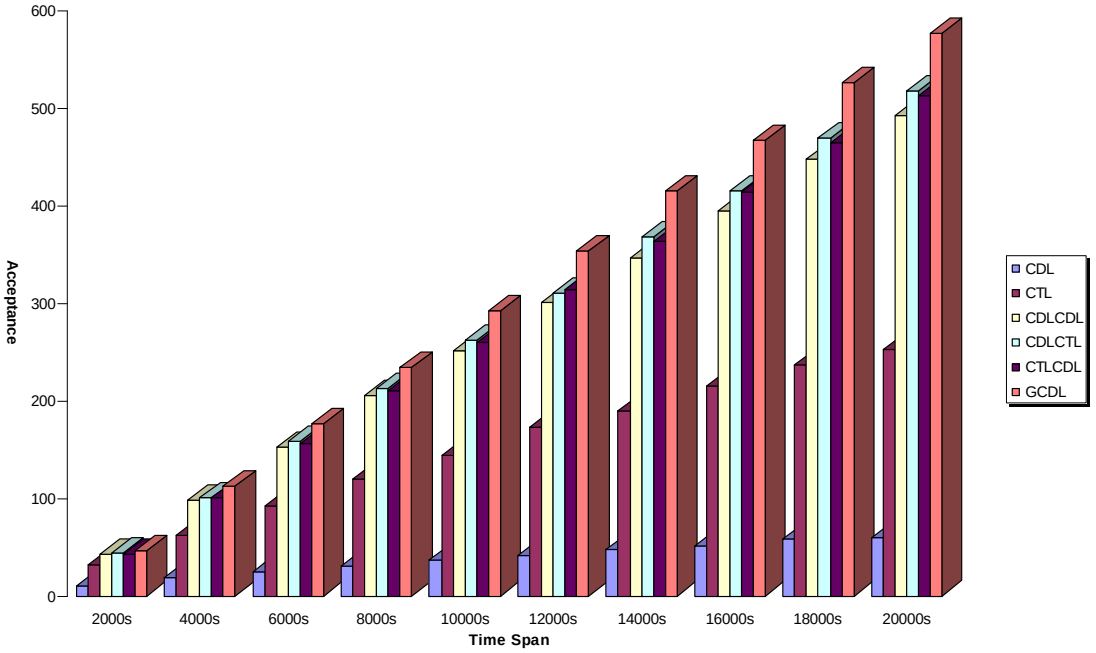


Figure 58: Acceptance variation with Time Span

Figure 59 shows the relationship between capacity and buffer time. The length of the buffer time controls the buffer size allocated to the stream, which is equal to the product of the buffer time and the average bit-rate of the stream. These figures indicate that the grand scale schedule delivers a stable performance in response to buffer time increments.

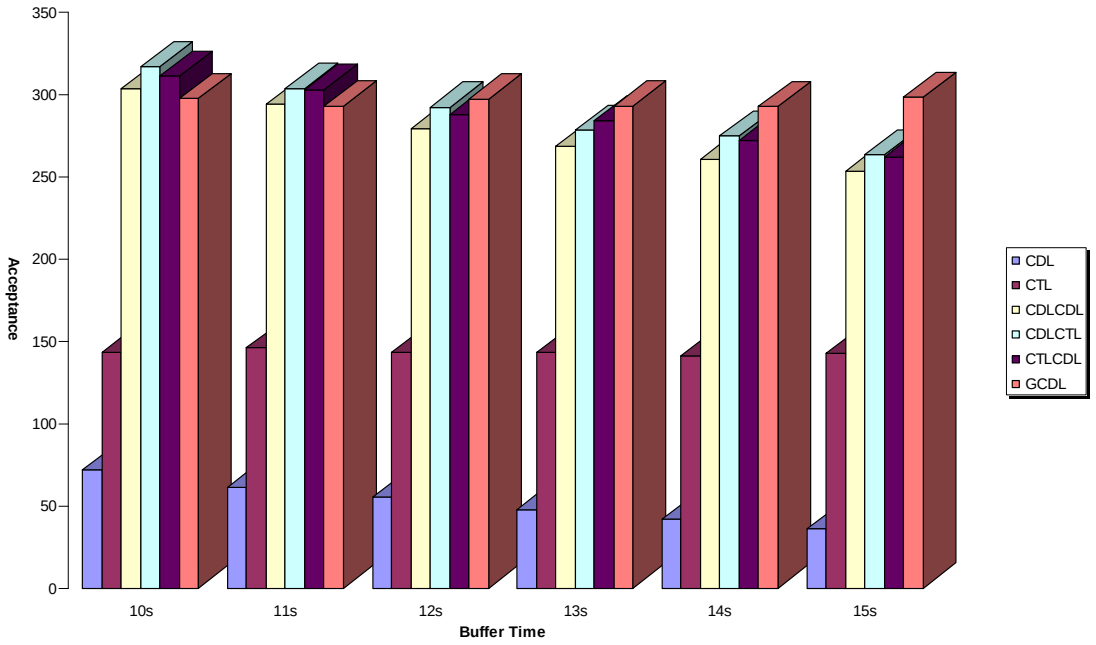


Figure 59: Acceptance variation with Buffer Time

As previously mentioned, both CTL and Generalized-CTL schemes require us to designate a fixed length of time as the period of the disk access cycle. The proxy server is filled up whenever one of these periods is full, even if its neighboring periods are still empty. The grand scale schedule does not maintain a fixed length period. Instead, it changes the length of the period according to the actual requirement to ensure all of these periods are fully utilized. As a result, the situation above will not likely occur, as its neighboring period will be able to relieve some of the loading in the peak period to allow more acceptances. On the other hand, it offers a gain to CDL schemes in another way. All CDL schemes assume a fixed buffer requirement within a period of time, even if the buffer is not used. The grand scale schedule provides a means to utilize them. Under any CDL scheme, the buffer requirement between any two disk accesses is fixed. The buffer usage will decrease monotonically as the data are delivering to the client. Even so, this reserved buffer cannot be reclaimed for other uses, as it is impossible to determine the exact volume that has been freed. In grand scale scheduling, a constant buffer reservation is still required, but it is only required between the two virtual disk accesses on the grand scale schedule. As a result, it is not necessary that the required buffer requirement be constant for the individual stream in the period

between its own disk accesses; it is allowed to drop as it transverses the virtual disk accesses of the grand schedule. In this way, these relinquished memory buffers can be used to support additional streams.

5.5 Summary

The video streaming proxy server is a resource intensive application even with today's technology standards. In order to increase the cost-effectiveness of the infrastructure, a proper scheduling scheme is essential. The two-level scheduling schemes are able to fulfill this requirement, but there is room for improvement as there is still wastage in the underlying CDL or CTL first level schemes. We propose grand scale scheduling as an alternative solution. Rather than considering resource requirements individually, the grand scale scheduling scheme considers them as a whole. It makes use of the relationship between the aggregated schedule and the actual requirements of each individual schedule. With the aid of this information, it is no longer necessary to reserve resources at their maximum utilization rate throughout the whole servicing period. These reservation periods can be shortened in a similar way, as provided by the two-level schemes. But our proposed scheduling scheme can further increase the utilization rate by interchanging resource usage within these reservation periods, resulting in lower reservation requirements. Our simulations show that the average processing time of grand scale schedules is roughly the same compared with that of the other two-level scheduling schemes. Moreover, it also yields a higher acceptance, especially when the buffer requirement of individual stream increases.

In short, in this section, we have,

1. Formulated a mathematical relationship between the Grand Scale Schedule and its corresponding memory buffer requirement

2. Based on this relationship, an admission control scheme is proposed, which have taken disk bandwidth and memory buffer availability into account

3. Excessive reservation on memory buffer due to CDL scheduling is interchanged in an inherent manner through the admission control

6 Admission Control With Early "Black Sheep" Detection

Resource availability-based admission control schemes usually offer a binary accept or reject decision on the request of an individual video stream to a video streaming proxy server. An acceptance is granted whenever the server can guarantee the availability of resource required to support the stream. Otherwise, it will reject the request. This is the greedy approach. However, our study has discovered that an acceptance of the request of some streams may jeopardize the capacity of a video streaming proxy server. More specifically, a server will have its capacity greatly reduced after the acceptance of the request of one particular stream – a black sheep. If we are able to screen out such stream in advance, we can maintain the normal server capacity by handling it carefully. When it comes to a multiple server platform, we may divert that particular stream to another server. Yet, there are numerous settings available. We have developed several schemes to facilitate such diversion and have determined their strength and weakness in different scenarios, which we will cover in this Chapter.

6.1 Admission Control

As we have mentioned in previous chapters, admission control remains an important issue in video streaming proxy server design. The server must reserve a certain amount of resources for each incoming streaming request. It cannot admit any further requests when such reservation fails; hence, it is said to have reached its capacity. In most situations, however, we have to anticipate these requirements and tend to reserve much more than enough. As a result, some resources are wasted and the server cannot attain its maximum capacity.

Traditionally, admission control schemes have been devised based on the availability of individual resource, such as network bandwidth [42], disk bandwidth [44][48][49][52][53][37], memory [51] and processing power [39]. Admission control of a video stream requires exercising admission control policy over all of these resources, which is laborious. It is better to have a unified resource requirement estimator in the design of a video streaming proxy server [45][50] to simplify the process.

Requirement anticipation is a tedious task in itself, due to the varying bit-rate of a video stream. Variable-Bit-Rate video streaming is in the mainstream of video streaming nowadays, as it is able to preserve the level of visual quality even for motion intensive scenes. However, it also

creates a problem in resource reservation as those resource requirements become time variant. To avoid waste, the server has to reserve different amount of resource over different times for different video streams, which increases the complexity of resource management. To face with this problem, the current admission control schemes often make use of different simplified models for resource usage estimation. These schemes can be further classified into three categories: deterministic, statistical and predictive. Deterministic admission schemes consider the worst-case requirement and therefore they always provide a feasible solution. However, since they consider the worst case, they are also comparably inefficient. Statistical schemes decide solely based on the usage statistics of the server and the admitted stream, while predictive schemes predict these usages from previous knowledge. Both of these schemes may provide an infeasible result, but their operations are simpler.

6.1.1 Grand Scale Scheduling Scheme

To solve the abovementioned problems, we have proposed a deterministic scheduling scheme – Grand Scale Scheduling [55] – as described in the previous chapter. It was developed based on CDL (Constant-Data-Length) and CTL (Constant-Time-Length) [54], as well as its generalized form, generalized CDL [40] and CTL [41]. The grand scale scheduling provides a simplified deterministic approach for the analysis of both the memory and disk bandwidth required to support a set of video streams in a streaming proxy server. We shall show later that it gives rise to a heuristic function that accurately indicates the possible increase in loading when accepting a new video stream request. Based on such information, an efficient admission control scheme can be devised.

While the resource requirement of a video stream is different at different times, it is known that by providing appropriate startup delays for video streams can relief the fierce resource competition between them. Hence upon the admission of a new stream, the proxy server needs to decide whether to start delivering the video stream right away or a startup delay is required. Based on the heuristic function as mentioned above, it is possible for us to determine the optimal delay by means of an exhaustive search so that the increase in loading due to the acceptance of a request can be kept to the minimum. While an exhaustive search requires much computational time and may increase the latency in starting the service, experimental results show that selecting the best delay for loading reduction every time may not necessarily allow more acceptance in the long run than a

randomly chosen delay. This suggests that optimal delay evaluation for minimal loading, although possible, is not worthy.

Besides, our study also shows that we should not accept every stream request even if we have enough resource to support. In some cases, the admission of a new stream may increase the loading dramatically and decrease the capacity of the server such that the number of admissions can no longer be guaranteed. Nevertheless, we can avoid it simply by rejecting the request of such problematic stream. Similar arguments and results have also been reported in [46][47][36], in which the advantage of non-greedy admission control over the greedy approach was presented in terms of client rejection ratio. To early detect problematic streams, we propose to make use of the heuristic function as mentioned above. Different amount of startup delay can be applied to see if we can relief the resource competition such that a problematic stream can be converted back to a normal stream. However, it is possible that we may never find a suitable startup delay. We call a problematic stream to be a "Black Sheep" if a suitable startup delay cannot be found to convert it back to a normal stream. For single server platform, there can be two strategies in this case. We may simply reject the request of the black sheep, however, it would lead to service discrimination. If rejecting service request is not allowed, we have no choice but to admit the black sheep with the delay that gives the best heuristic. We have incorporated these schemes into the SSRC (Single Server Rejection and Correction) mechanism as discussed later in this chapter.

In a multiple server platform, we gain an additional option by reintroducing the black sheep to another server within the platform. Due to the different loading condition of the servers, a black sheep of one server can be a normal stream in another. We have developed three MSRC (Multiple Server Rejection and Correction) mechanisms for this purpose. In fact, these mechanisms facilitate a kind of interchange of services between servers, allowing them to selectively entertain the service requests that are best suitable to their own service condition.

6.1.2 Heuristic Function

Traditional disk scheduling schemes [54] for video streaming are either CDL- or CTL-based. One of their advantages is that they have simplified the scheduling problem by reducing its degree of freedom. They have fixed a variable in the original VBR (variable-bit-rate) video stream. The CDL scheme fixes the data size of each disk access, so that we can focus the scheduling problem on the duration between disk accesses. The CTL scheme on the other hand fixes the duration

between disk accesses, leaving the volume as the only free variable. Subsequently, some two-layer scheduling schemes such as GCDL [40], GCTL [41] and CDL-EDF [43] were proposed. These schemes facilitate the design of admission control mechanism for supporting multiple streams over traditional CDL and CTL schemes.

Effectively, GSS (Grand Scale Scheduling) provides a solution to multiple streams scheduling similar to that of GCDL. In GCDL, each stream will be scheduled by CDL for the first level of scheduling, so that each of them will obtain a disk access schedule. Then, for the second level of scheduling, these disk access schedules will be combined together to form one disk access schedule. Finally, this aggregated schedule will be scheduled again by CTL. As a result, we are able to tell whether a stream can be accepted by checking the disk bandwidth requirement in each of these CTL periods of the final schedule. Since a period in CTL is fixed, variations in a stream make some of these periods bear higher load and the others lower. Unlike GCDL, GSS does not maintain a fixed period; instead, its period changes according to the loading. In an area with heavier loading, the period will be shorter and vice versa. GSS schedules the aggregated disk schedule with CDL, again with buffer pre-fetching employed [87]. Therefore, the length of these periods reflects the actual loading of the server.

To facilitate the discussion below, let us briefly recall the operation of GSS. Firstly, an aggregated schedule S (or the so-called grand scale schedule) should be generated by summing the schedule of all admitted streams in the proxy server. Then, the CDL scheduling is applied to S with buffer size $\mathrm{M}/2$, where M is the total memory buffer size. A virtual disk access schedule can then be obtained. Such disk accesses are virtual since they are only the vehicle for us to evaluate the system loading. There are no such disk accesses in reality. As it is stated in the last chapter, the minimum time between any two consecutive virtual disk accesses of the grand scale schedule $\min\{d_{k+1} - d_k\}$ decides how long a proxy server can take to retrieve M data from the disk. Practically, this minimum decreases monotonically as the server admits additional stream and it will continue to decrease when admitting more streams until the quotient $M / \min\{d_{k+1} - d_k\}$ reaches the system disk bandwidth D. At that time, the system is said to have reached its maximum capacity. Hence, the function $\min\{d_{k+1} - d_k\}$ is infinite when the system is empty,

becoming $M/2D$ when the system is full. Therefore, we can adopt this function $h: \min\{d_{k+1} - d_k\} \ \forall k$ as a heuristic to estimate the system loading.

It is interesting to note that, in theory, h may not necessarily decrease on every new stream acceptance. It is because h only reflects the region with the heaviest loading; it is a localized loading indicator. If the newly accepted stream does not increase the loading at the original heaviest loading region nor does it creates another region to replace the original one, h would remain unchanged.

6.1.3 Service Startup Delay

Heuristic function h gives us some idea on the impact to the system loading for a given set of video streams. We can decide whether a system has enough resource to admit a stream by comparing the heuristic output with the available disk bandwidth. This decision, as mentioned above, needs not be binary. Rather than accepting or rejecting a stream request, we can also choose to delay the startup time to relief the problem that it will impose to the system. The optimal delay time can also be estimated by using the proposed heuristic function.

For simplicity's sake, let us consider the cache-out operation of a video streaming proxy server. When the server receives a cache-out request, it can acquire the corresponding delivery schedule of the video stream from its record. Based on this, the server can determine the heuristic function, which is generated from the grand scale schedule as mentioned above. When delivering the video streams, disk accesses required from different schedules may be concentrated in a region, creating a heavy loading requirement at a certain point. Such condition can be indicated by the heuristic function, which shows the peak loading across all times. Hence when admitting a new stream, one should check if the peak region is further loaded to an unacceptable level. Figure 60 shows the effect of admitting a problematic stream on the heuristic. In the figure, the optimal line (dash line) is plotting the heuristic in logarithmic scale against an ordered set of streams in which no problematic stream is found. The pink line is also a plot of the heuristic against this set of streams, with an exception that the "20" stream is replaced with a problematic one. The other lines are similar in that the "40", "60" and "80" streams are the problematic streams, respectively. In this example, the problematic stream is a stream that demands an average bandwidth 10 times bigger

than the average bandwidth of the other streams in the set. From the figure, we observe that whenever the proxy server admits a problematic stream, the heuristic drops substantially.

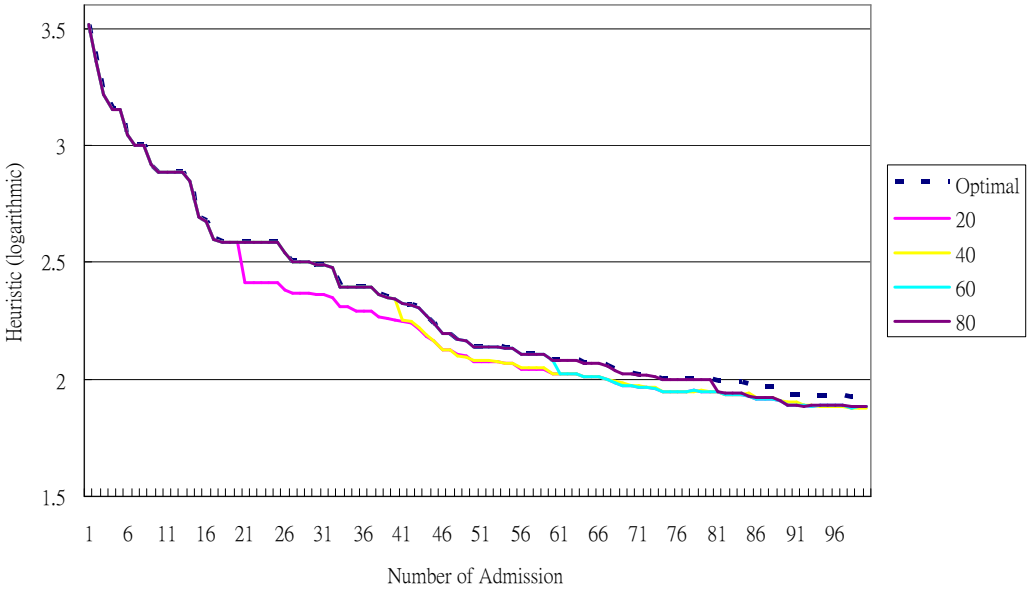


Figure 60: Effect of problematic streams to the heuristic

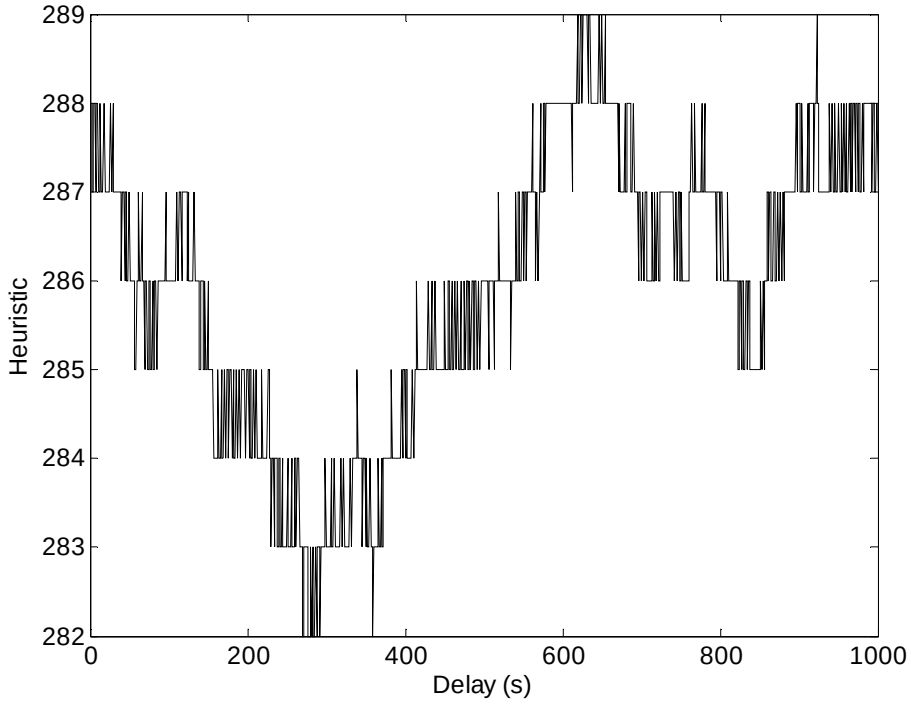


Figure 61: Heuristic vs. Delay

One of the solutions to deal with the problem is to shift the startup time of a stream in order to avoid and offload the peak region. Figure 61 below illustrates this effect by examining the loading of a system with heuristic h (peak loading) over different startup delay τ . It shows that there do exist some τ that can reduce the loading requirement.

Intuitively, we may assume that the maximum server capacity must be able to achieve by having an exhaustive search of the optimal delay τ on every admission. However, this conjecture does not necessarily hold in practice. It is because, although we claim to choose the best delay on every admission, it is indeed only the best based on the knowledge of the given set of video streams that have been admitted in the system. It can never take care of the video streams that will be admitted in the future as they have not been admitted yet. It is possible that the best delay that we derived based on the current set of admitted video streams be later found to be not so good when we consider also the video streams that will be admitted in the future. Besides, it is known that the best delay will produce the slightest additional load to the peak loading region. For other regions, the effect of this delay is random. It means that the loading in other regions increases

similar to starting the service with a random delay. With the effort of optimal delay search, the growth in loading in the peak region is slower than other regions and eventually, the original peak region may be replaced by another one, of which the increase in loading was random. They explain why the result obtained from an optimal search strategy is similar to the result obtained from starting the service with a random delay in the long run. Having said so, in practice, starting the service with a totally random delay may accidentally align the peak region of the admitting video stream with that of the admitted streams. To avoid the problem, a simple comparison between a few random startup delays will still be useful. It will be described in more details in Section 6.2.1.

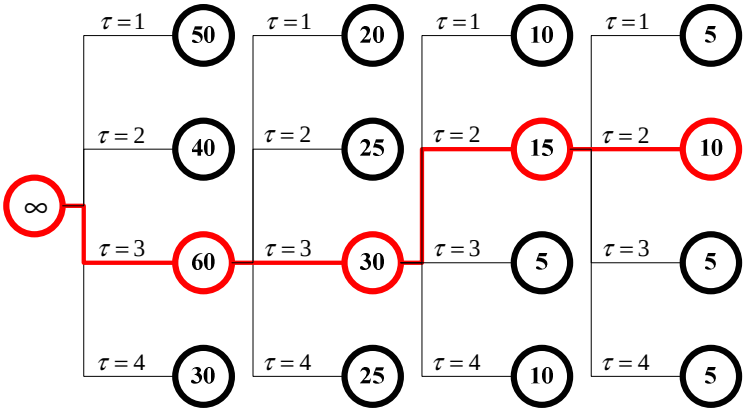


Figure 62: Admission Tree with Maximum Strategy

The observation above is justified with the following experiment. To better understand the different admission strategies, we picture them with a tree-like diagram as shown in Figure 62. Each level in the tree represents the service request of a new stream while each node represents the peak heuristic after the admission of a stream with the indicated startup delay. The red line shows the admission history with an admission strategy which always chooses the path with the highest heuristic, i.e. the lowest anticipated loading in the peak region. Figure 63 shows the anticipated loadings given by the heuristic function from the first to the twentieth admissions with different admission strategies. These strategies include: maximum (MAX), minimum (MIN), median (MED) and random (RAND1, RAND2 and RAND3). For the maximum strategy, we always follow the link that leads to the highest heuristic valued node. The minimum and median strategies are constructed similarly but the link that leads to the minimum and median heuristic value node is chosen, respectively. They are implemented just to provide a comparison. For the random strategy, we determine the startup delay by comparing a few randomly selected delays and selecting the best

of them. We repeat the experiment with 3 sets of randomly selected delays. It is seen in Figure 63 that all strategies yield a similar result. In addition, Figure 64 shows that the ratio between the heuristic obtained from different strategies does not vary over 20%, even after 100 admissions. This indicates that none of them can significantly raise the server capacity in the long run. However, the searching mechanism associated with the maximum, minimum and median strategies requires substantially more processing time to complete. More results can be found in Section 6.3.

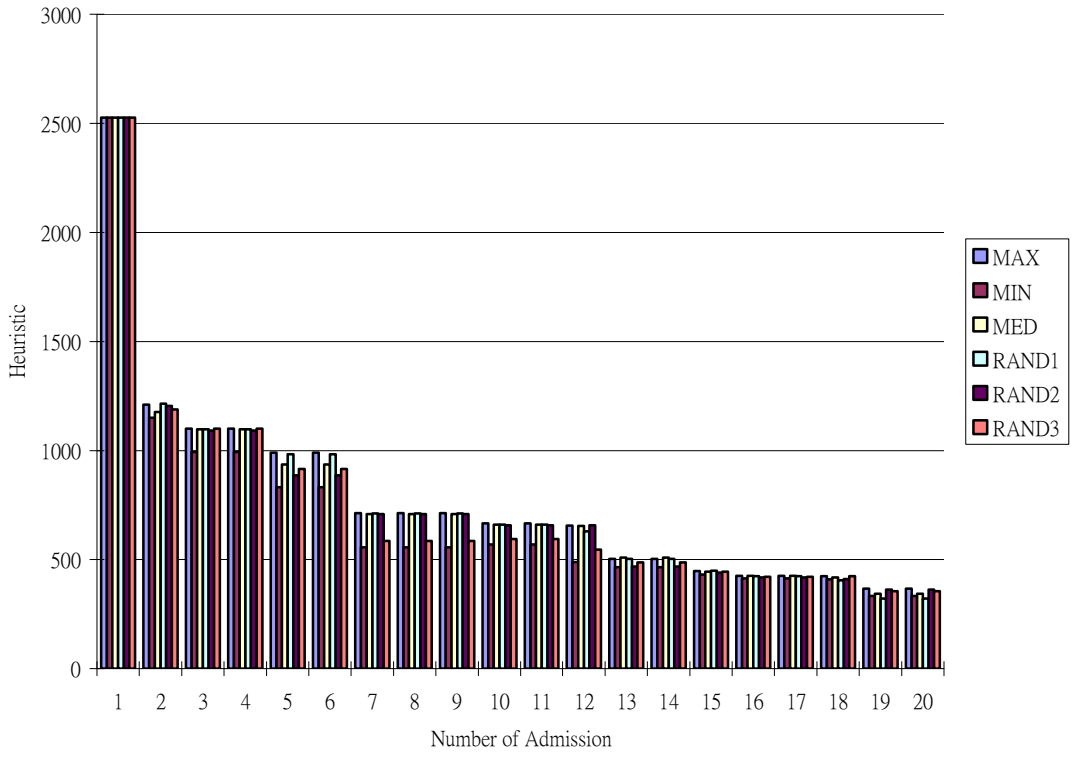


Figure 63: Heuristic from different Admission Strategy

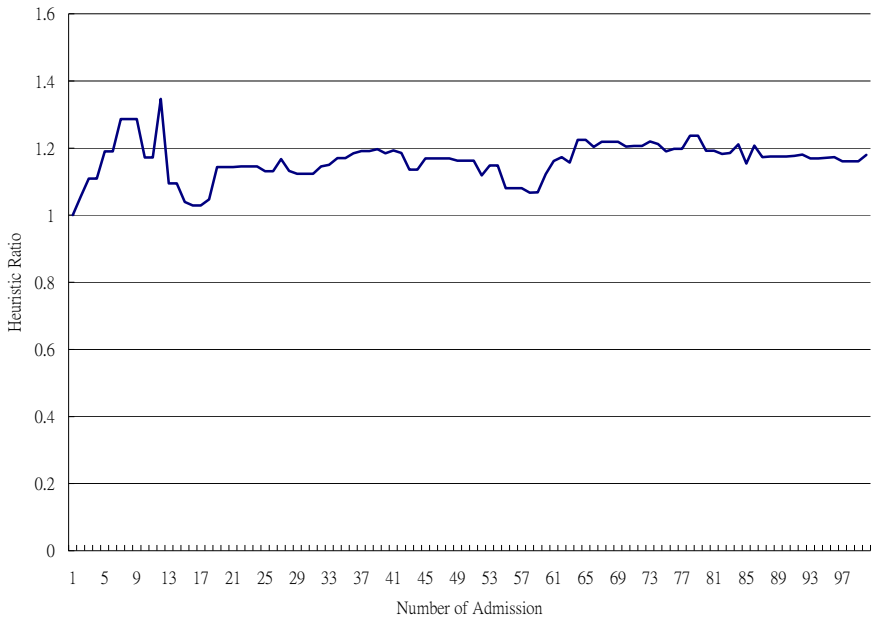


Figure 64: Heuristic Ratio between different Admission Strategies

6.2 The Black Sheep

In contrast with an optimal startup delay, bad startup delays are defined as delays that will substantially increase the system loading. An admission of a stream with bad startup delay can reduce the system's capacity substantially. As mentioned in Section 6.1.1, we call a problematic stream to be a black sheep if we have no choice but use a bad startup delay for that stream. In the following sections, we shall describe some methods that will be useful to perform the searching of delay with a hope to avoid a problematic stream from becoming a black sheep. If black sheep are inevitable, a set of mechanisms is devised to deal with them, particularly in the multi-server platform.

6.2.1 Single Server Rejection and Correction

As mentioned above, we may have a better chance to avoid bad startup delay by using an optimal search, however, this is very time-consuming. We show that a random search is much simple and can give a performance similar to the optimal search in the long run. However in practice, a random selection scheme will have a higher chance of introducing a bad startup delay,

as the chosen delay might happen to be a bad one. Therefore, we have to devise a remedy for the random selection scheme.

We observed that there is a subtle difference between the two types of black sheep occurrences. Type I is caused by the existence of a heavy loading requirement in the stream to admit. This requirement is intrinsic to the stream therefore it will persist on different startup delay. In other words, suppose we have an original grand scale schedule which has already provided a flat platform as shown in Figure 65, due to the heavy loading requirement of the stream, we will not be able to find any startup delay τ that can provide a better schedule.

Type II occurrence is caused by the roughness of the grand scale schedule. As shown in Figure 66, a new stream requesting admission may or may not fit properly into the grand scale schedule depending on the startup delay. This would result in a varying loading. In order to avoid a bad selection, we may make a set of selections and pick the best one. Intuitively, the size of the set is difficult to determine, as it must be kept as small as possible while still allowing the mechanism to remain functional. Conversely, we may adopt a best-effort approach by accepting the first delay which is not categorized as bad.

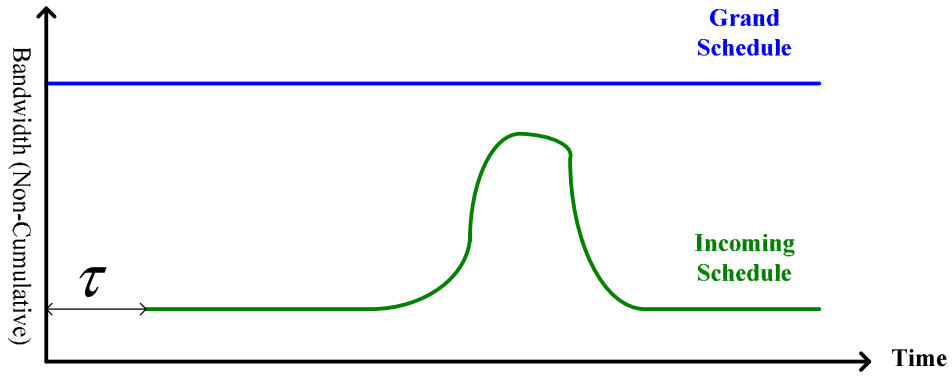


Figure 65: Type I Black Sheep

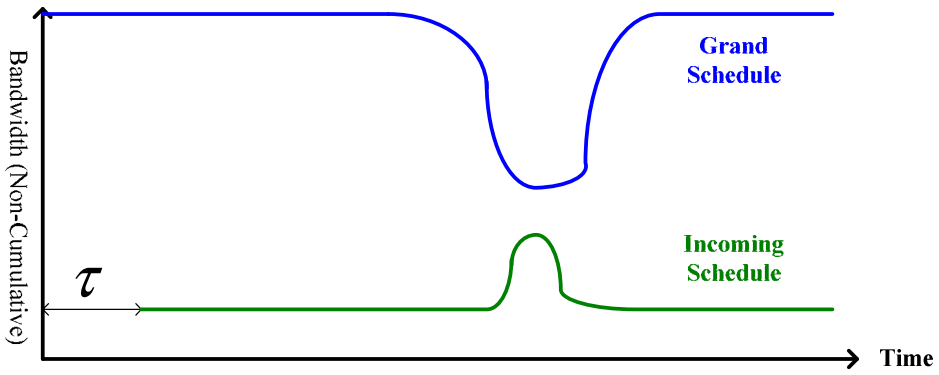


Figure 66: Type II Black Sheep

The determination of the searching range may need to be further elaborated. It is always possible to search from the beginning to the end of all admitted video streams in the system in order to look for the best delay for the admitting stream. However, since a delay for too long will not be acceptable to the client, a search for such a long period of time of course is unnecessary. The next possible searching range is the minimum distance between the heavily loaded regions in the grand scale schedule, with a hope that by delaying the admitting stream, we can put the heavily loaded regions of that stream to the regions that are not loaded heavily in the system. However, it is noticed that a heavily loaded region as shown in the grand scale schedule can be easily last for more than 120s. A delay of the admitting stream with such duration will still be unacceptable to the client. In fact, normally, we may assume those virtual disk accesses in a grand scale schedule within the region of heaviest loading are periodic and the distance between them is equal to the heuristic. This is because the length of a heavily loaded region in any video stream usually spans across several virtual disk accesses. The vicinity of the peak loading region in the grand scale schedule will also approach to the peak. Thus, we may assume that the periods between all virtual disk accesses around this heavily loaded region are nearly the same and equal to the heuristic. Consequently, in order to shorten the search range, we can limit the search over a period the same as the heuristic, that is, the minimum distance between virtual disk accesses. We know that the result obtained from searching the second period will be nearly equal to the first one. To play safe, we may in practice extend the search range to a certain fixed multiple of the heuristic to cater for the exceptional cases.

Until now, our definition of black sheep remains conceptual, in that its admission will substantially affect the heuristic of the server. However, in practice, the server requires a concrete

definition of a black sheep so that it can categorize a stream and carry out the remedial. We can recognize a black sheep by comparing its resultant heuristic with the norm. On every admission, when we examine the heuristic h for the overall system loading, we will obtain a generally decreasing curve against the number of admissions, as shown in Figure 60. In this curve, a black sheep will be represented by a dramatic drop in the heuristic corresponding to a substantial increment in overall loading. Conversely, a set of normal streams will not generate a curve with such fall. Therefore, we can construct a lower limit l for the heuristic curve so that whenever it falls below this we can tell that the stream under admission is a black sheep.

Suppose we have an ordered set of streams with the same constant bit rate r and they are being requested one by one with an available buffer size b . Now, right after the first admission, the heuristic function $h(1)$ will give $\frac{b}{r}$, as this is always the time taken to empty the buffer. We can also evaluate $h(2)$ after the second admission as $\frac{b}{2r}$, as shown in Figure 67. Therefore, we can determine that $h(n)=\frac{b}{nr}$ in this specified case, where n is the number of admissions. Now, we consider a different set of streams having variable constant bit rate $r_1, r_2, \ldots, r_n$. Intuitively, the heuristic function thus obtained becomes $h(n)=\frac{b}{\sum_{i=1}^{n} r_i}$. We can assign the lower bound as $l=\overline{l}=h(n)=\frac{b}{\sum_{i=1}^{n} r_i}$ for the categorization in a constant bit rate stream set.

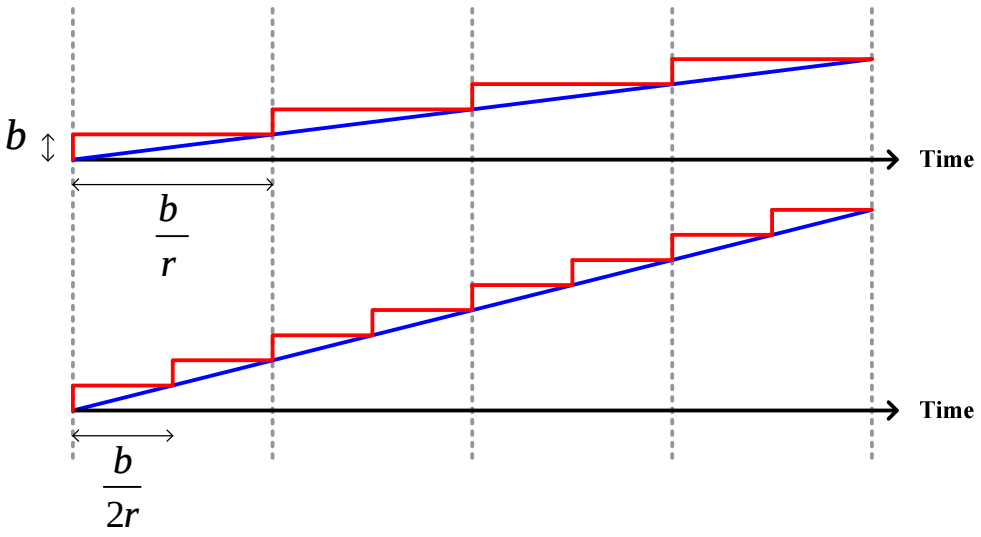


Figure 67: Construction of Optimal Heuristic Curve

In practice, video streams are not of a constant bit rate and therefore their schedules will not be presented in straight lines but curves. Consequently, the heuristic function will give a lower value (Majorization effect). Therefore, we cannot simply categorize variable bit rate streams by a constant bit rate heuristic function. However, the philosophy behind the remedy is to select a startup delay that allows the incoming stream to fit in the existing schedules within the server only, though it may not be the fittest. The resulted schedule will be evened out to some extent, resembling a schedule generated by a set of constant bit rate streams. In this way, we may construct the variable bit rate lower bound $\tilde{l}$ by adopting the constant bit rate heuristic function with a reasonable tolerance ν as $\tilde{l}(n) = \bar{l}(n+\nu)$. Given $r_{\text{average}} = \frac{\sum_{i=1}^{n} r_i}{n}$, $r_j = r_{\text{average}}$ as a preset bit rate $\forall n < j \le n+\nu$, we have $\tilde{l} = \frac{b}{\sum_{i=1}^{n+\nu} r_i}$. This produces a lower bound $\tilde{l}$ which tolerates several more streams than $\bar{l}$ so that it can accommodate the variation of the variable bit rate streams. When the tolerance ν is set to 1, this means that the server is ready to accept a new stream if its bit rate does not derivate too much from the average bit rate of those streams that have been admitted. Now, we can formulate our remedy into a complete SSRC algorithm as follows:

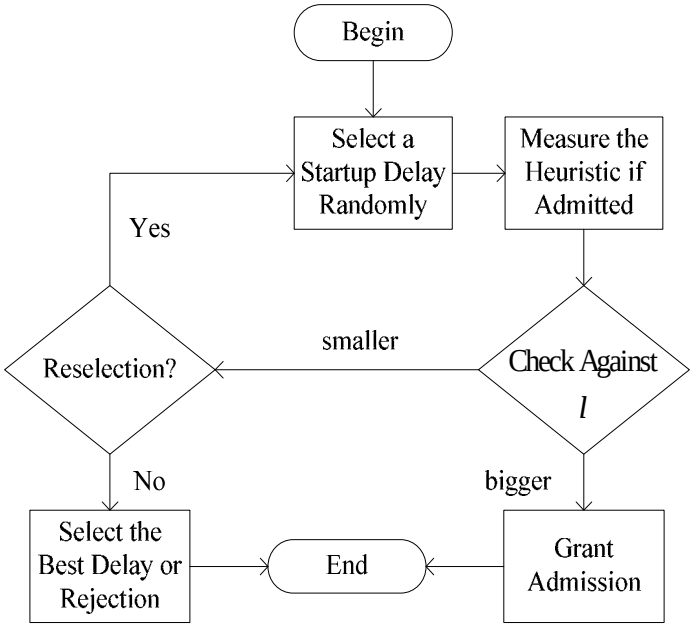


Figure 68: SSRC Algorithm

6.2.2 Multi-Server Rejection and Correction

We can extend our single server black sheep algorithm SSRC to a multi-server platform to formulate a multi-server rejection and correction algorithm MSRC. In SSRC, we have to accept an unsatisfactory startup delay or even consider rejection. However, the multi-server platform provides another option: such streams can be passed to another server for admission. As mentioned in the previous section, the occurrence of a black sheep can be categorized into two types. The type II occurrence is affected by the history of admittance of an individual server. Therefore, a stream which has been classified as black sheep in one server due to type II reason can be a normal stream in another server due to different acceptance histories. Based on this, we are able to formulate several rejection and correction algorithms in a multi-server platform. We will compare their performance in a later section.

The first MSRC algorithm to introduce is MSRC-S1, as shown in Figure 69. This algorithm is a simple extension to the SSRC algorithm. Beginning from the initial server, video streaming requests are sent to each server one after another until any of them accept it. In this way, the initial server will always be filled up first, followed by the second and so on, creating an unbalanced loading. This algorithm has a major drawback, since even when some servers are getting full, a stream must still seek acceptance starting from the first server. It is because a server may reject one

stream but accept another; we may overlook a possible acceptance if we simply bypass the acceptance check with the server that has indicated as full. However, a stream will then waste its time asking for acceptance from those servers already filled up, while most of the servers at the back are still empty and would have accepted them immediately.

To ease the problem, we introduce another MSRC algorithm, MSRC-S2, in Figure 70. Unlike MSRC-S1, we do not designate a fixed initial server for this algorithm. Indeed, we rotate the initial server on each acceptance. In this way, the loading will be balanced between all servers. When a server is getting full, so also will the other servers. The time taken between request and acceptance for each server will be roughly the same, irrespective to the initial server choice. However, as indicated in the previous section, the search range of SSRC decreases when loading increases; hence, the time taken to finish the search will also become shorter. This shortening effect is immediately beneficial to MSRC-S1 but not to MSRC-S2 until the whole round is served. Therefore, we have a tradeoff between the short-term and long-term benefits.

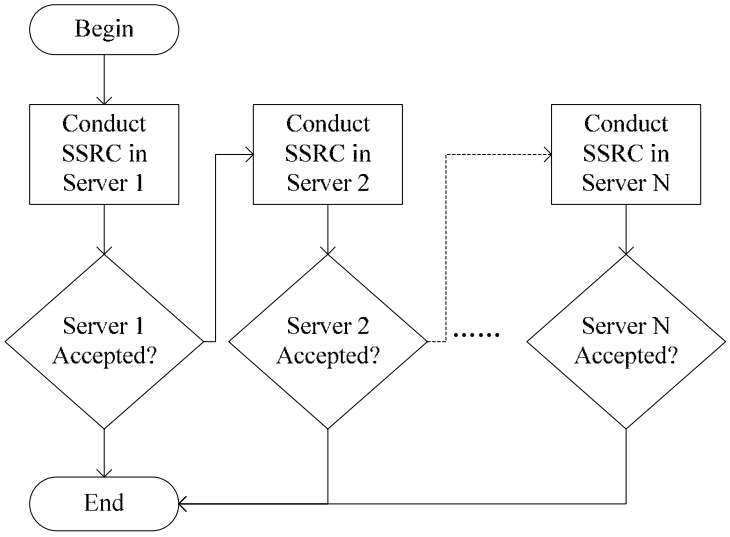


Figure 69: MSRC-S1 Algorithm

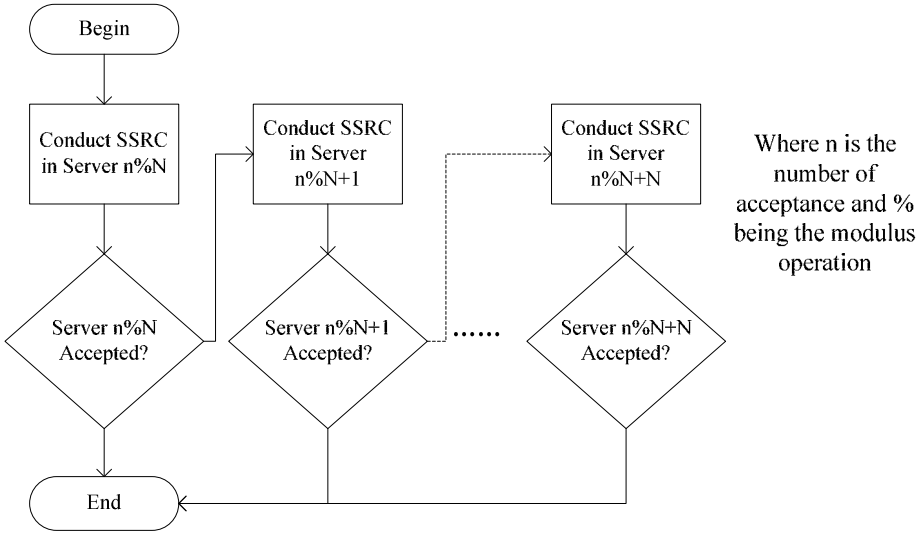


Figure 70: MSRC-S2 Algorithm

The last algorithm to introduce is the MSRC-P, as shown in Figure 71. In this algorithm, we will exploit the parallel processing power provided by a multi-server platform by distributing requests to all servers and allowing the server that anticipates the lowest loading increment to provide the service. Hence, we can always find the best fit server. In this way, the chosen server becomes unpredictable, depending on the fitness of the incoming schedule to those servers. Yet, we are still able to anticipate its processing time. As MSRC-P has to gather the heuristic anticipated by all servers, it will always be bounded by the slowest server response. Its plot against incoming requests will begin similarly to MSRC-S2, but fall more gently as some servers may be able to keep a lower loading for a longer time as compared with MSRC-S2.

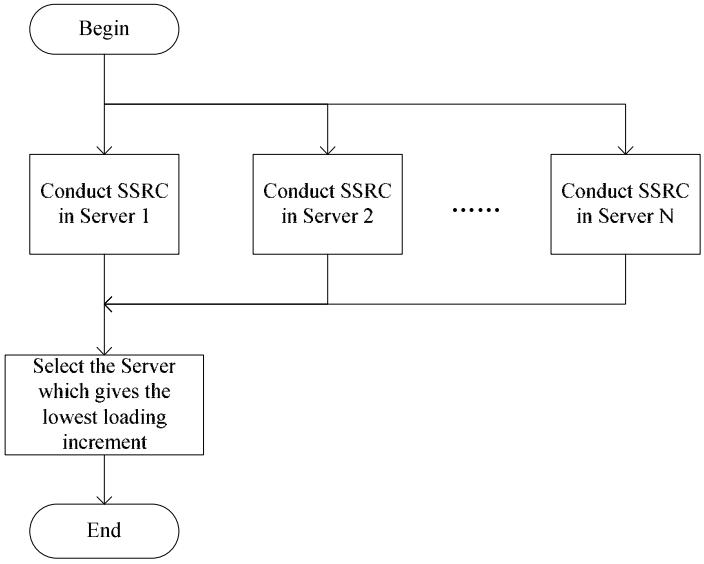


Figure 71: MSRC-P Algorithm

The three algorithms have their strengths and weaknesses in different aspects. Yet, it also allows the construction of a hybrid platform consisting of servers with different algorithms applied in order to obtain a more feasible solution. Besides, it is not a restriction that one machine should host only one server. In fact, the number of servers that a machine can host is only limited by the number of disks it possesses. This can increase the flexibility of the design and implementation of a multi-server platform.

6.3 Simulation Results

In this section, we will present a series of simulation results on the capacities of different video streaming proxy server platforms with different admission control mechanisms. These platforms include single server (SSRC), multiple servers in serial (MSRC-S1), round-robin (MSRC-S2) and parallel (MSRC-P). Admission strategies include greedy, reject, optimal, random and lazy. Under the greedy admission strategy, we allow the server to accept incoming stream requests until it can no longer accept any more. For the reject strategy, we will reject any request on black sheep without any correction. In the optimal strategy, we will conduct a search for the best startup delay. In both the random and lazy strategies, we will only limit the search over a randomly selected set of delay to about 50% of the full range as specified in Section 6.2.1. We will select the best delay under the random strategy; and we will only select the first acceptable delay and terminate the search under the lazy strategy.

We have conducted the simulations on a Pentium III 800MHz machine with 100 video stream requests for the single server platform and 500 video stream requests for the multiple server platform (5 servers). These video streams are synthetic and are created from an agglomeration of randomly selected parts of a video trace “Star Wars”. This is to ensure a balance in randomness as well as reality. We have simulated four stream sets in these platforms. The first set, “N”, is a normal set with streams, having an average bandwidth of 1Mbps in 10000s length. The purpose of the “N” set is to illustrate the admission capacity of normal streams; we expect all schemes to accept a roughly similar amount of streams. The second set, “BS1”, is a black sheep set with streams normally having an average bandwidth of 1Mbps, but with an insertion of black sheep streams having an average bandwidth of 10Mbps in every tenth stream. This set of video streams is employed to demonstrate how those admission strategies identify and handle the black sheep and the capacity reduction when they were not identified. The third set, “BS2”, is a set of streams with an average bandwidth of 1Mbps. This time 80% of these streams are identical. Each stream within this set of stream is well-behaved. They are not black sheep in type Ⅰ. However, when the server accepts them without introducing any start-up delay, their heavily loaded regions will be superimposed together and form a peak in the grand scale schedule resembling the occurrence of a black sheep in type Ⅱ. The fourth set “BS3” is constructed in a similar way as “BS2”. Except that we have raised a portion of the delivery schedule (200s in the middle) from those identical streams by an amount equal to their average bit-rate. As a result, the effect stemmed from black sheep will occur earlier. Finally, each server is equipped with a 250MB memory buffer and disk bandwidth of 40Mbps.

The following tables show the result obtained from SSRC. In SSRC, if we have completed the search for a satisfactory startup delay which does not yield then we will have to select the best from those unsatisfactory ones. Another option is to simply reject the admission. The following result tabularized the SSRC with rejection. Hence, the result obtained below is the best result obtainable as we do not have to accept any unsatisfactory startup delays which will hamper the system throughput.

	Average Time(s)	Number of Admission
Greedy	0.0114	40
Reject	0.0113	40
Optimal	0.7586	39
Random	0.3923	40
Lazy	0.2884	40

Table 2: SSRC (Normal)

	Average Time(s)	Number of Admission
Greedy	0.0212	22
Reject	0.0112	40
Optimal	0.7692	39
Random	0.4357	38
Lazy	0.3457	41

Table 3: SSRC (BS1)

	Average Time(s)	Number of Admission
Greedy	0.0308	39
Reject	0.0284	42
Optimal	0.7768	41
Random	0.4079	42
Lazy	0.3601	45

Table 4: SSRC (BS2)

	Average Time(s)	Number of Admission
Greedy	0.0340	36
Reject	0.0312	39
Optimal	0.7687	39
Random	0.4251	39
Lazy	0.3519	40

Table 5: SSRC (BS3)

As expected, Table 2 does not indicate any difference between these schemes in terms of capacity: all of them admit about 40 streams. This is quite reasonable, as the disk bandwidth is exactly 40 times the average bandwidth of a single stream. However, the time taken to conduct an admission is very different. The optimal search takes about 67 times longer to complete a search compared with the greedy selection scheme. This implies that an optimal search investigates about 67 startup delays on average per admission, while the random scheme investigates about 34 startup delays, and lastly, the lazy scheme conducts about 25 investigations.

Table 3 shows their performance on “BS1”. It shows that all schemes except the greedy one have successfully found out and rejected those black sheep. The throughput of these schemes is

better than the greedy one as they do not admit a black sheep which occupies an excessive amount of resources in the peak loading region. Table 4 shows their performance on “BS2”. In this case, the performance of the greedy scheme is only slightly poorer than the other schemes. As those streams in “BS2” are not black sheep by themselves, therefore the greedy scheme will accept them, until a point which they begin to possess the characteristics of black sheep. We have observed from the result, capacity reduction caused by this effect is not very large. It only drops from the expected 40 admissions to 39. However, all the other schemes provide more than 40 admissions. It is because the last three schemes have spent some effort on the search of a better startup delay, which allows a smaller peak loading. To the surprise, the capacity obtained from the reject scheme also reaches 40 for both “BS1” and “BS2”. Even the reject scheme does not search for a better delay for a stream and simply reject it, the variation between the different startup time of the incoming streams serves a similar effect as a random delay. Therefore, it produces a result approaching the other searching schemes. However, since it does not search, there will always be a few streams with their peak regions accidentally crash in the peak regions of the system and are rejected. Hence it can never produce the best result. In “BS3”, we have increased the bit-rate of a particular region to emphasize the region of heaviest loading of those identical streams. With this emphasis, we observe that the effect of black sheep is amplified and the capacity for the greedy scheme has dropped to 36, while the others can still maintain the nominal capacity of about 40.

			Number of Admission					
		Average Time(s)	#1	#2	#3	#4	#5	Total
Greedy	Serial	0.0530	46	36	36	35	35	188
Reject	Serial	0.0482	46	39	39	40	40	204
Optimal	Serial	2.4453	45	35	39	40	40	199
Random	Serial	1.2258	45	40	41	39	40	205
Lazy	Serial	1.9858	46	40	42	42	41	211

Table 6: MSRC-S1 (N)

			Number of Admission					
		Average Time(s)	#1	#2	#3	#4	#5	Total
Greedy	Serial	0.1051	33	12	21	21	20	107
Reject	Serial	0.0495	44	41	41	40	40	206
Optimal	Serial	2.5229	44	40	39	39	39	201
Random	Serial	1.3053	44	40	40	41	39	204
Lazy	Serial	2.1926	46	40	40	41	36	203

Table 7: MSRC-S1 (BS1)

			Number of Admission					
		Average Time(s)	#1	#2	#3	#4	#5	Total
Greedy	Serial	0.0524	40	38	40	39	38	195
Reject	Serial	0.0498	40	40	42	41	41	204
Optimal	Serial	2.3361	41	40	41	41	41	204
Random	Serial	1.2397	41	40	41	40	41	203
Lazy	Serial	1.9313	41	40	41	40	42	204

Table 8: MSRC-S1 (BS2)

			Number of Admission					
		Average Time(s)	#1	#2	#3	#4	#5	Total
Greedy	Serial	0.0534	37	38	39	37	37	188
Reject	Serial	0.0491	40	40	40	41	40	201
Optimal	Serial	2.4834	39	40	41	40	39	199
Random	Serial	1.2942	40	40	40	40	40	200
Lazy	Serial	2.1377	41	43	41	40	40	205

Table 9: MSRC-S1 (BS3)

For the serial multiple server platform serving the “N” stream set, as shown in Table 6, we see that the average admission time for the greedy scheme is about 5 times longer than in the single server case, as it must consider servers even if they have already been filled up. Apart from this, as in SSRC case, all schemes achieve similar performance in terms of capacity.

Table 7 shows the result when it is serving “BS1”. Similarly, admission begins from server #1 and progresses to #5. However, different schemes now behave in a different way. All schemes with rejection mechanisms have not accepted those black sheep streams while the greedy scheme has accepted them and it leads to a dramatic decrease in capacity.

Table 8 shows the result for “BS2”. A similar conclusion as in the “SSRC” case can be drawn. The effect of black sheep is small. However, in “BS3” as shown in Table 9, the effect of black sheep is a little bit larger. Yet, its effect is still incomparable to the case in “SSRC”. This is because any rejected stream will be re-introduced to another server in “MSRC”, and they will behave as a normal stream in the new server.

			Number of Admission					
		Average Time(s)	#1	#2	#3	#4	#5	Total
Greedy	Roundrobin	0.0437	37	39	35	41	39	191
Reject	Roundrobin	0.0444	40	39	40	40	40	199
Optimal	Roundrobin	2.1548	40	40	39	39	40	198
Random	Roundrobin	1.2041	40	40	39	41	35	195
Lazy	Roundrobin	1.8747	41	41	36	41	40	199

Table 10: MSRC-S2 (N)

			Number of Admission					
		Average Time(s)	#1	#2	#3	#4	#5	Total
Greedy	Roundrobin	0.0941	20	20	28	29	11	108
Reject	Roundrobin	0.0462	40	41	41	35	40	197
Optimal	Roundrobin	2.5366	39	39	35	39	40	192
Random	Roundrobin	1.2445	39	38	39	40	40	196
Lazy	Roundrobin	2.2971	40	41	40	40	41	202

Table 11: MSRC-S2 (BS1)

			Number of Admission					
		Average Time(s)	#1	#2	#3	#4	#5	Total
Greedy	Roundrobin	0.0427	41	37	39	38	41	196
Reject	Roundrobin	0.0440	41	39	40	39	39	198
Optimal	Roundrobin	2.0517	40	35	39	40	40	194
Random	Roundrobin	1.0231	43	39	39	39	40	200
Lazy	Roundrobin	1.7211	41	40	39	40	40	200

Table 12: MSRC-S2 (BS2)

			Number of Admission					
		Average Time(s)	#1	#2	#3	#4	#5	Total
Greedy	Roundrobin	0.0422	39	43	37	37	40	196
Reject	Roundrobin	0.0431	40	41	41	40	39	201
Optimal	Roundrobin	2.1582	39	42	35	41	39	196
Random	Roundrobin	1.0684	39	40	40	40	40	199
Lazy	Roundrobin	1.8019	41	41	41	42	41	206

Table 13: MSRC-S2 (BS3)

Table 10 to Table 13 show results obtained for the Round Robin platform. Their performances in terms of capacity are roughly the same as for MSRC-S1. In overall, MSRC-S2 offers a 10% to 15% shorter operation time than MSRC-S1. However, the overall admission number for the Round Robin platform is also a bit lower than the Serial platform. In Table 7, we can easily observe that server #1 admits many more streams than the other servers. This is because the admission history of server #1 in the serial platform has a history of accepting “non-Black

Sheep” streams. It tends to reject any black sheep stream and re-introduce them to other servers. And the other servers will try to accept those rejected black sheep streams. This is not the case in the Round Robin platform since the requests of clients will be entertained starting with different servers.

			Number of Admission					
		Average Time(s)	#1	#2	#3	#4	#5	Total
Greedy	Parallel	0.1607	38	38	39	40	38	193
Reject	Parallel	0.1546	41	41	39	39	41	201
Optimal	Parallel	2.4094	40	39	40	41	39	199
Random	Parallel	1.3455	40	39	41	39	39	198
Lazy	Parallel	2.0993	41	42	42	42	42	209

Table 14: MSRC-P (N)

			Number of Admission					
		Average Time(s)	#1	#2	#3	#4	#5	Total
Greedy	Parallel	0.2888	19	28	22	19	21	109
Reject	Parallel	0.1505	41	40	40	42	40	203
Optimal	Parallel	2.7128	39	40	39	40	39	197
Random	Parallel	1.4638	40	39	40	39	40	198
Lazy	Parallel	2.3104	36	39	41	40	40	196

Table 15: MSRC-P (BS1)

			Number of Admission					
		Average Time(s)	#1	#2	#3	#4	#5	Total
Greedy	Parallel	0.1561	37	35	39	41	41	193
Reject	Parallel	0.1442	40	41	42	41	42	206
Optimal	Parallel	2.5090	41	40	41	40	42	204
Random	Parallel	1.3758	40	41	41	42	41	205
Lazy	Parallel	2.0959	41	41	42	41	44	209

Table 16: MSRC-P (BS2)

			Number of Admission					
		Average Time(s)	#1	#2	#3	#4	#5	Total
Greedy	Parallel	0.1543	37	42	40	37	40	196
Reject	Parallel	0.1486	42	41	40	40	41	204
Optimal	Parallel	2.4656	41	39	41	39	41	201
Random	Parallel	1.3508	41	40	41	39	39	200
Lazy	Parallel	2.0683	41	42	42	44	41	210

Table 17: MSRC-P (BS3)

Table 14 to Table 17 show results obtained for the parallel platform. Similar performance is noted as the Round Robin platform in terms of capacity. It seems that the parallel platform requires

the longest operation time. However, as all servers in this platform can operate in parallel, the actual time taken by each admission is divided by the number of servers. In this way, the MSRC-P would outperform the other schemes in terms of operation time.

From the results above, the following general conclusions can be drawn:

1. No matter in the single server platform or the multi-server platform, early detecting black sheep and rejecting them can significantly increase the server capacity, particularly when we consider in practical situation, both type I and type II black sheep may be requested at the same time by the clients.
2. Introducing a random startup delay to the admitting streams can give some improvement in server capacity. Optimal search, while can be laborious, may not necessarily improve over random search.

6.4 Summary

Admission control is an essential task in resource scheduling within a video streaming proxy server. A good admission control mechanism is able to maximize capacity with an instant response. These two requirements are self-contradictory, as the maximization in capacity can only be obtained by an extensive search of the system loading, which becomes time consuming. To speed up the search, we can make use of a heuristic offered by grand scale scheduling as an indicator of the loading of the server. The use of these heuristic results is indeed a trade-off between accuracy and time. We are able to determine this heuristic with a single execution of grand scale scheduling.

With this heuristic, we can measure the increase in loading for each stream in each server. We noticed that this loading increment was not unique to the admission of the same stream on every server. Also, it varies with different startup delays. Moreover, the acceptance of some streams may increase the loading dramatically. As a result, we have developed the concept of black sheep. We classify a stream as a black sheep if its acceptance creates a substantial loading increment and reduces server capacity by comparing its load to a standardized loading curve. We may correct it by applying a different startup delay, transferring the loading to another server or simply rejecting it. Between these approaches, random schemes can deliver the highest capacity within a reasonably short time; they basically employ an optimal with a randomly chosen subset of the complete searching space. Therefore, it also trades accuracy with time.

A multiple server platform offers more flexibility than a single server platform. The “Rejection and Correction” mechanism provides a criterion to decide which stream a server should accept. In this way, each server can select its best video stream to fulfill the objective of capacity increment. This is literally fulfilled by the sharing of resources across different servers within the platform.

In short, in this section, we have,

1. Make use of the Grand Scale Schedule as an heuristic to indicate a video streaming proxy server loading
2. Based on this heuristic, developed an early detection technique for “Black Sheep”
3. Developed several multiple platforms admission scheme with rejection and correction capability

7 Conclusion

The video streaming proxy server is one of the most important components in today's video streaming infrastructure. However, due to its nature of operation it requires special design for resource management. It must guarantee all the required resources for every service it has agreed to support so that the quality-of-service can be guaranteed. Yet, at the other end of the spectrum, the capacity of the proxy server could be decimated because of these guarantees. As it is likely that the amount of these guaranteed resources that has been requested is more than the actual need, this leads to waste, which reduces the overall capacity. Much effort has been made to reduce the wastes. Ideally, the best solution is to match up the need and the guarantee as closely as possible. In this piece of work, we have evaluated previous solutions to this match-up problem and determined that resource usage interchange is the key. In our investigation, resource usage interchange exists in two different forms: heterogeneous and homogeneous. Heterogeneous interchange allows an interchange of usage between different resources for a single video stream. It results in higher flexibility for the streaming proxy servers in resource allocation. Homogeneous resource usage interchange deals with the sharing of a kind of resource for different video streams. It can be further extended to the system level that allows the capacity of servers be shared among themselves. In this work, we have proposed a number of mechanisms to actualize the concepts of heterogeneous and homogeneous resource interchange. Significant improvements in server performance, particularly in their capacity, are achieved that fully justify our claims.

7.1 Processing Power Management

Real-time system is nevertheless the most appropriate platform for the construction of a video streaming proxy server. It is capable of providing real-time support and giving a processing power guarantee. The contractual platform that we have adopted in this work is capable of these functions and our experiment has shown that it has outperformed its time-sharing counterpart. Furthermore, we have also implemented the direct processing power transfer mechanism that facilitates the transfer of processing power within a group of operations. This mechanism provides a homogeneous interchange of resource usage so that any excessively reserved processing power can be relinquished and transfer to other operations. Although the transfer is conducted in an implicit manner within members of the group, their aggregate can still be expressed explicitly. Under this organization, the uncertainty in processing power requirements for any specific

operation in servicing a variable-bit-rate video stream within a proxy server can be entertained and the capacity can be guaranteed at the same time.

7.2 Stream Level Resource Management

Apart from processing power, memory, network and disk bandwidth are the other essential resources required to support a video stream inside the proxy server. Based on the principle of video staging, we have developed a novel video staging algorithm that not only provides the basic function of video staging, but also incorporating the idea of video smoothing. Under this algorithm, usage of several resources is considered. They include a bounded ingress network bandwidth, and disk bandwidth in the forms of a disk schedule and memory buffer. Our algorithm does not only improve the utilization of these resources but also facilitate the interchange of their usage; this forms the basis of the heterogeneous resource usage interchange.

7.3 System Level Resource Management

At the system level we have firstly proposed a grand scale scheduling algorithm to provide homogeneous resource usage interchange on memory buffer usage. This algorithm is proposed to reduce the wastage generated by the resource guarantee, which is essential to the quality of service. It is known that the worst case of wastage is in the guarantee of peak resource requirements. Since no stream has peak resource requirement at all times, the residue is wasted throughout the delivery of the whole stream. Improvements were made by shortening the period of wastage; as a result, the proxy server is only required to evaluate and guarantee the peak resource requirement of each stream confined in one window alone. Our algorithm provides a further improvement by analyzing the sum of the actual requirements from all streams, then allowing them to share the guaranteed resources. This leads to an increment in capacity.

Secondly, we have applied the grand scale schedule to formulate a heuristic that can inform us about the loading of the proxy server. Also, we have drawn a nominal loading curve from this heuristic which indicates the normal change on heuristics upon the acceptance of a new stream. With this heuristic and the nominal loading curve, we are able to evaluate the effect of the admittance of a stream to further admittance, allowing us to diversify the service provided by each server within a multiple server platform according to their own service history. As a result, each

server may serve the clients that best suit to its own service condition and in turn improve its capacity.

7.4 Comprehensive Strategy

We can construct a comprehensive admission control strategy to make use of these different levels of management schemes. At first, the video streaming proxy server may decide the proper network bandwidth, disk bandwidth and memory buffer it plans to serve a video stream. In this stage of admission, we may optimize the network bandwidth usage for the system. For a multiple server platform, we may proceed to conduct an MSRC scheme to decide which server will be used. In both the single and multiple server platforms, GSS is used to optimize the memory buffer and disk bandwidth usage. As a result, the optimal usage of network bandwidth, disk bandwidth and memory buffer indirectly maximizes the throughput of the video streaming proxy server.

7.5 Future Work

Video streaming is a developing technology and there are many areas yet to be explored. Recently, ubiquity has become a hot topic in the light of the announcement of the 3rd generation mobile communication standard. Nowadays, video playback is no longer restricted to cinemas or television sets, but encompasses desktop computers, notebook computers, handheld computers and even mobile phones. The video signal may be transmitted through the air, copper wire or optical fibers. To work with these various combinations, scalability is the only solution. However, this will create many versions of the same video stream in different resolutions and they will be treated as different streams with current technology. Further investigation is required in order to develop efficient mechanisms to manage these highly correlated video streams. We believe that some of the ideas of the current work can be extended to deal with the problem. For instance, the MPS as mentioned in Chapter 3 has been built in the transcode function that can facilitate the resolution conversion of video streams. The video staging smoothing algorithm as mentioned in Chapter 4 can be extended to allow only a fixed number of bit planes to be retrieved through network. The remaining bit planes can be retrieved directly from the cache of the proxy server. Hence the network bandwidth will not be greatly affected due to the requests of the higher resolution version of the same stream. Finally the admission control schemes as proposed in Chapter 5 and Chapter 6 will also be useful although the suggested grand scale scheduling method may need to be slightly modified to take care of the fact that many streams may have the same peak

regions although the magnitude of the peaks may be different according to the resolution of the streams. We believe that further investigation in that direction will be fruitful.

References

[1] Gebhard, H., Lindner, L., “Virtual Internet broadcasting”, IEEE Communications Magazine, Volume 39 Issue 6, pp. 182-188, June 2001.

[2] Qian Zhang, Wenwu Zhu, Ya-Qin Zhang, “Resource allocation for multimedia streaming over the Internet”, IEEE Transactions on Multimedia, Volume 3 Issue 3, pp. 339-355, Sept. 2001.

[3] Chia-Wen Lin, Jian Zhou, Jeongnam Youn, Ming-Ting Sun, “MPEG video streaming with VCR-functionality”, IEEE Transactions on Circuits and Systems for Video Technology, Volume 11 Issue 3, pp. 415-425, March 2001.

[4] Fabmi, H., Latif, M., Sedigh-Ali, S., Ghafoor, A., Liu, P., Hsu, L.H., “Proxy servers for scalable interactive video support”, Computer, Volume 34 Issue 9, pp. 54-60, Sept. 2001

[5] Zhi-Li Zhang, Yuewei Wang, Du, D.H.C., Dongli Su, “Video staging: a proxy-server-based approach to end-to-end video delivery over wide-area networks”, IEEE/ACM Transactions on Networking, Volume 8 Issue 4, pp. 429-442, Aug. 2000

[6] “Netscape Proxy Server Administrator's Guide for Windows NT”, Chapter 7, http://developer.netscape.com/docs/manuals/proxy/adminnt/revpxy.htm

[7] Schulzrinne, H., Rao, A., and Lanphier, R., “Real-Time Streaming Protocol (RTSP)”, RFC 2326, April 1998.

[8] Schulzrinne, H., Casner, S., Frederick, R., and Jacobson, V. "RTP: A Transport Protocol for Real-Time Applications", RFC1889, January 1996.

[9] Schulzrinne, H. and Fokus, G.M.D., “RTP Profile for Audio and Video Conferences with Minimal Control”, RFC 1890, January 1996.

[10] Handley, M., and Jacobson, V., "SDP: Session Description Protocol", RFC 2327, April 1998.

[11] Siu, Y.M., “O.S. Support for Cluster Computing”, Technical Report, Department of Electronic Engineering, The Hong Kong Polytechnic University, 1996.

[12] Cheuk, W.K., Li, C.K. and Chan, W.Y., “Contractual Operating System – Design and Problems”, Proceedings of the 1st IEEE Computer Society International Workshop on Cluster Computing, pp. 255-260, 1999, Adelaide, Australia.

[13] Lam, W.K., “An Analysis of the Contractor Model of Networked Parallel Computing”, TROCC No.II — Relationship with Existing Paradigms, Computer Research Institute, Tsinghua University, 1995.

[14] Lam, W.K., “A Preliminary Investigation on Contractor Technology”, TROCC No.III — Implementations Issues, Computer Research Institute, Tsinghua University, 1996.

[15] Lam, W.K., “A Brief Review on the Development of Contractual Computing”, Proceedings, International Workshop on Computational Science and Engineering, IWCSE’97, pp. 171-181, 1997.

[16] Lam, W.K., “The Design of Contractual Mechanisms in Networked Environments”, Proceeding, ISCA 12th International Conference - Parallel and Distributed Computing System, pp. 339-344, 1998.

[17] Lam, W.K., Li, S.L., Cheuk, W.K.and Li, C.K., “The Contractors - An Alternative Paradigm for Worldwide Virtual Computing”, Proceedings, Euromicro workshop on parallel and distributed processing, pp. 230-236, 1999.

[18] Cheuk, W.K., “Design and Implementation of Cluster Computing System”, MPhil dissertation, The Hong Kong Polytechnic University, pp. 73-102, 2001.

[19] Gallmeister, B.O., “POSIX.4, Programming for the real world”, O’Reilly & Associates, Inc, pp. 203-204, 1995.

[20] “About Darwin Streaming Server”, Apple Computer, Inc, © 2001 Apple Computer, http://www.publicsource.apple.com/projects/streaming/AboutDarwinStreamingServer.pdf

[21] Wei-hsiu Ma and Du, David H.C., “Reducing Bandwidth Requirement for Delivering Video Over Wide Area Networks With Proxy Server”, IEEE Transactions on Multimedia, Vol. 4 No. 4, pp. 539-559, Dec 2002.

[22] Thiran, P., Le Boudec, J.-Y. and Worm, F., “Network calculus applied to optimal multimedia smoothing”, INFOCOM 2001, Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. IEEE Proceedings, Volume 3, pp. 474-483, 2001.

[23] Qi Wang, Zixiang Xiong, Feng Wu, and Shipeng Li, “Optimal Rate Allocation for Progressive Fine Granularity Scalable Video Coding”, IEEE Signal Processing Letters, Vol 9 No2, pp. 33-39, Feb 2002.

[25] Rexford, J. and Towsley, D., “Smoothing variable-bit-rate video in an internetwork”, IEEE/ACM Transactions on Networking, volume 7 issue 2, pp. 202-215, Apr. 1999.

[26] Mahanti, A., Williamson, C. and Eager, D., “Traffic analysis of a Web proxy caching hierarchy”, IEEE Network, volume 14 issue 3, pp. 16-23, May-June 2000.

[27] Dogan, S., Cellatoglu, A., Uyguroglu, M., Sadka, A.H., and Kondoz, A.M., “Error-resilient video transcoding for robust internetwork communications using GPRS”, IEEE Transactions on Circuits and Systems for Video Technology, volume 12 issue 6, pp. 453-464, Jun 2002.

[28] Goose, S., Schneider, G., Tanikella, R., Mollenhauer, H., Menard, P., Le Floc'h Y., and Pillan, P., “Toward improving the mobile experience with proxy transcoding and virtual composite devices for a scalable bluetooth LAN access solution”, Proceedings of the Third International Conference on Mobile Data Management 2002, pp. 169-170, 2002.

[29] Gorinsky, S., Baruah, S., and Stoyen, A., “Boosting the network performance via traffic reshaping”, Proceedings of the Sixth International Conference on Computer Communications and Networks, 1997, pp. 285-290, 1997.

[30] Lombaedo, A., Schembra, G., and Morabito, G., “Traffic specifications for the transmission of stored MPEG video on the Internet”, IEEE Transactions on Multimedia, volume 3 issue 1, pp. 5-17, March 2001.

[31] Wei-Hsiu Ma and Du, D.H.C., “Reducing bandwidth requirement for delivering video over wide area networks with proxy server”, 2000 IEEE International Conference on Multimedia and Expo, ICME 2000, volume 2, pp. 991-994, 2000.

[32] Rexford, J., Sen, S., and Basso, A., “A smoothing proxy service for variable-bit-rate streaming video”, 1999 Global Telecommunications Conference, GLOBECOM '99, volume 3, pp. 1823-1829, 1999.

[33] Sen, S., Rexford, J., and Towsley, D., “Proxy prefix caching for multimedia streams”, Proceedings of the Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies, INFOCOM '99, volume 3, pp. 1310-1319, 1999.

[34] Cheuk, W.K., Hsung, T.C., and Lun, Daniel P.K., “Design and Implementation of Contractual Based Real-time Scheduler for Multimedia Streaming Proxy Server”, Multimedia Tools and Applications, Netherlands. (Accepted)

[35] Cheuk, W.K., and Lun, Daniel P.K., “Video Staging in Video Streaming Proxy Server”, The IEEE International Conference on Multimedia & Expo, 2004, Taipei. (Accepted)

[36] Lee, W., Srivastava, J., and Won-Ho Lee, “Adaptive disk scheduling algorithms for video servers”, Proceedings of International Conference on Parallel Processing, pp. 363-370, 21-24 Sept. 1999.

[37] Tzli-Cker Chiueh and Vernick, M., “An empirical study of admission control strategies in video servers”, Proceedings of International Conference on Parallel Processing, pp. 313-320, 10-14 Aug. 1998.

[38] Jagannathan, S., Tohmaz, A., Chronopoulos, A., and Cheung, H.G., “Adaptive admission control of multimedia traffic in high-speed networks”, Proceedings of the 2002 IEEE International Symposium on Intelligent Control, pp. 728-733, 27-30 Oct. 2002.

[39] Qazzaz, B., Moreno, J., Xiao, J., Hernandez, P., Suppi, R., and Luque, E., “Admission control policies for video on demand brokers”, Proceedings of International Conference on Multimedia and Expo 2003, Vol. 2, pp. 529-532, 6-9 July 2003.

[40] Biersack, E., Thiesse, F., and Bernhardt, C., “Constant data length retrieval for video servers with variable bit rate streams”, Proceedings of the Third IEEE International Conference on Multimedia Computing and Systems, 1996, pp. 151-155, 17-23 June 1996.

[41] Biersack, E., and Thiesse, F., “Statistical admission control in video servers with variable bit rate streams and constant time length retrieval”, Proceedings of the 22nd EUROMICRO Conference EUROMICRO 96 'Beyond 2000: Hardware and Software Design Strategies', pp. 633-639, 2-5 Sept. 1996.

[42] Hadar, O., and Greenberg, S., “Statistical multiplexing and admission control policy for smoothed video streams using e-PCRTT algorithm”, Proceedings of International Conference on Information Technology: Coding and Computing, 2000, pp. 272-277, 27-29 March 2000.

[43] Sambit Shau, Zhi-Li Zhang, Kurose, J.and Towsley, D., “On the efficient retrieval of VBR video in a multimedia server”, Proceedings of IEEE International Conference on Multimedia Computing and Systems '97, pp. 46-53, 3-6 June 1997.

[44] KyungOh Lee and Yeom, H.Y., “A Dynamic Scheduling Mechanism for an Effective Admission Control for Variable-Bit-Rate Video Streams”, Proceedings of The Twenty-Second Annual International Computer Software and Applications Conference 1998, COMPSAC '98, pp. 614-619, Aug. 1998.

[45] Meliksetian, D. Feng-Kuo Yu, Frank, and Chen, C.Y.R., “Methodologies for Designing Video Servers”, IEEE Transactions on Multimedia, Vol. 2 Issue 1, pp. 62-69, March 2000.

[46] Wonjun Lee and Srivastava, J., “Reserve-based Disk Admission Control and Bandwidth Scheduling Strategy for Continuous Media Servers”, Proceedings of the Ninth International Conference on Computer Communications and Networks, 2000, pp. 232-237, 16-18 Oct. 2000.

[47] Lee, W., and Srivastava, J., “Negotiated disk admission control in video streaming”, Electronics Letters, Vol. 35 Issue 21, pp. 1810-1812, 14 Oct 1999.

[48] In-Hwan Kim, Jeong-Won Kim, Seung-Won Lee and Ki-Dong Chung, “Measurement-Based Adaptive Statistical Admission Control Scheme for Video-on-Demand Servers”, Proceedings of the 15th International Conference on Information Networking 2001, pp. 471-478, 31 Jan-2 Feb 2001.

[49] Sun-Euy Kim and Das, C.R., “A Reliable Statistical Admission Control Strategy for Interactive Video-On-Demand Servers with Interval Caching”, Proceedings of the International Conference on Parallel Processing 2000, pp. 135-142, 21-24 Aug. 2000.

[50] Lui, J.C.S. and Wang, X.Q., “An Admission Control Algorithm for Providing Quality-of-Service Guarantee for Individual Connection in a Video-on-Demand System”, Proceedings of the Fifth IEEE Symposium on Computers and Communications 2000, ISCC2000, pp. 456-461, 3-6 Jul 2000.

[51] Keunhyung Kim and Seog Park, “Schemes for utilizing efficiently Disk bandwidth and Buffer in Video Server”, Proceedings of the Eighth International Conference on Parallel and Distributed Systems 2001, ICPADS2001, pp. 549-554, 26-29 Jun 2001.

[52] Xia, Z.,. Yen, I.L, and Li, P., “An Aggressive Distributed Admission Control Policy for Streaming Media”, Proceedings of the 12th International Conference on Computer Communications and Networks 2003, ICCCN2003, pp. 131-136, 20-22 Oct 2003.

[53] Kwon, J.B., and Yeom, H.Y., “An Admission Control Scheme for Continuous Media Servers Using Caching”, Proceeding of the IEEE International Conference on Performance, Computing, and Communications Conference 2000, IPCCC00, pp. 456-462, 20-22 Feb 2000.

[54] Chang, E., and Zakhor, A., “Admissions Control and Data Placement for VBR Video Servers”, Proceedings of the IEEE International Conference Image Processing 1994, ICIP94, Vol.1, pp. 278-282, 13-16 Nov 1994.

[55] Cheuk, W.K. and Lun, D.P.K., “Grand Scale Scheduling for Admission Control”, International Symposium on Intelligent Multimedia, Video and Speech Processing, ISIMP2004, 20-22 Oct 2004, Hong Kong. (Accepted)

[56] Makaroff, D., Neufeld, G., and Hutchinson, N., “Design and Implementation of a VBR Continuous Media File Server”, IEEE Transactions on Software Engineering, Vol. 27 Issue 1, pp. 13-28, Jan 2001.

[57] Shiao-Li Tsao, Yueh-Min Huang, Shiang-Chun Liou, Jar-Ching Lin, Jen-Wen Ding, Jyh-Ming Wang and Ho-Li Wang, “An Efficient Data Retrieval Scheme to Improve Service Capacity of a VBR Video Server”, Proceedings of International Conference on Information, Communications and Signal Processing 1997, ICICS97, Vol. 2, pp. 1088-1092, 9-12 Sept 1997.

[58] Jones, Michael B., “Adaptive Real-Time Resource Management of Supporting Composition of Independently Authored Time-Critical Services”, Proceedings of the Fourth Workshop on Workstation Operating Systems, IEEE Computing Society, Raleigh-Durham, November 1993.

[59] Jones, Michael B., Leach, Paul J., Draves Richard P., and Barrera III, Joseph S., “Modular Real-Time Resource Management in the Rialto Operating System”, Technical Report, MSR-TR-95-16, Microsoft Research, 1995.

[60] Jones, Michael B., and Barrera III, Joseph S., “An overview of the Rialto Real-Time Architecture”, Technical Report, MSR-TR-96-13, Microsoft Research, 1996.

[61] Jones, Michael B., Roşu, Daniela, and Roşu, Marcel-Cãtãlin, “CPU Reservations and Time Constraints: Efficient, Predictable Scheduling of Independent Activities”, Proceedings of the 16th ACM Symposium on Operating Systems Principles, pp. 198-211, 1997.

[62] Jones, Michael B., “The Microsoft Interactive TV Systems: An Experience Report”, Technical Report, MSR-TR-97-18, Microsoft Research, 1997.

[63] Jones, Michael B., and Regehr, John, “Issues in Using Commodity Operating Systems for Time-Dependent Tasks: Experiences from a Study of Windows NT”, Technical Report, MSR-TR-98-29, Microsoft Research, 1998.

[64] Jones, Michael B., Regehr, John and Saroiu, S., “Two case studies in predictable application scheduling using Rialto/NT”, Proceeding of the IEEE 7th Symposium on Real-Time Technology and Applications 2001, pp. 157-164, 30 May-1 June 2001.

[65] Nieh, Jason, and Lam, Monica S., “The Design, Implementation and Evaluation of SMART: A Scheduler for Multimedia Applications”, Proceedings of the 16th ACM Symposium on Operating Systems Principles, pp. 184-197, 1997.

[66] Rao, K.R. and Yip, P., “Discrete Consine Transform: Algorithms, Advantages, Applications”, New York: Academic, Aug. 1990.

[67] JPEG 2000. [Online] Available: http://www.jpeg.org/JPEG2000.htm

[68] Turaga, D., and van der Schaar, M., “Wavelet coding for video streaming using new unconstrained motion compensated temporal filtering,” Proceeding of International Workshop in Digital Communications: Advanced Methods for Multimedia Signal Processing, Capri, Italy, pp. 41–48, Sept. 2002.

[71] LeGall, D.J., “MPEG: A video compression standard for multimedia applications”, ACM Communications, vol.34, pp. 46-58, 1991.

[70] Skodras, A., et al, “The JPEG-2000 still image compression standard”, IEEE Signal Processing Magazie, pp. 36-58, Sept., 2001.

[71] ISO/IEC International Standard 11172, “Coding of moving pictures and associated audio for digital storage media up to about 1.5 Mbits/s”, November 1993.

[72] ISO/IEC International Standard 13818, “Information technology - Coding of audio-visual objects”.

[73] ISO/IEC International Standard 14496, “Information technology - Coding of audio-visual objects”, January 2000.

[74] International Telecommunication Union, “Video Coding for Low Bitrate Communication”, ITU-T Recommendation H.263, 1996.

[75] Cheuk, W.K., Li, C.K., Lun, Daniel P.K., and To, T.P., “Design of a Process Scheduler with Guaranteed Resource Allocation”, Proceedings of IASTED Networks, Parallel and Distributed Processing, and Applications Conference 2002, NPDPA2002, pp. 235-240, Tsukuba, Japan, Oct 2002.

[76] Gu, Z. and Shin, K.G., “Algorithms for effective variable bit rate traffic smoothing”, Proceedings of the IEEE International Conference on Performance, Computing, and Communications, 2003, pp. 387-394, 9-11 April 2003.

Optimal smoothing for guaranteed service

[77] Le Boudec, J.-Y. and Verscheure, O., IEEE/ACM Transactions on Networking, Vol. 8 , Issue 6, pp. 689-696, Dec. 2000.

[78] Ng, J.K.-Y. and Song, Shibin, “A video smoothing algorithm for transmitting MPEG video over limited bandwidth”, Proceedings of the Fourth International Workshop on Real-Time Computing Systems and Applications , pp. 229-236, 27-29 Oct. 1997.

[79] Ye, Dejian., Cam, Barker., Xiong, Zixiang. and Zhu, Wenwu., “Wavelet-Based VBR Video Traffic Smoothing”, IEEE Transactions on Multimedia, Vol.6, No.4, pp. 611-623, Aug. 2004.

[80] Huang, Shih-Yu., “Improved techniques for dual-bitstream MPEG video streaming with VCR functionalities”, IEEE Transactions on Consumer Electronics, Vol.49, Issue 4, pp. 1153-1160, Nov. 2003.

[81] Bjork, N. and Christopoulos, C., “Transcoder architectures for video coding”, IEEE Transactions on Consumer Electronics, Vol. 44, Issue. 1, pp. 88-98, Feb. 1998.

[82] Lin,Chia-Wen., Chen, Yung-Chang and Sun, Ming-Ting., “Dynamic region of interest transcoding for multipoint video conferencing”, IEEE Transactions on Circuits and Systems for Video Technology, Vol.13 , Issue 10, pp. 982-992, Oct. 2003.

[83] Hung, Bin-Feng and Huang, Chung-Lin, “Content-based FGS coding mode determination for video streaming over wireless networks”, IEEE Journal on Selected Areas in Communications, Vol. 21 , Issue 10, pp. 1595-1603, Dec. 2003.

[84] Van der Schaar, M. and Radha, H., “A hybrid temporal-SNR fine-granular scalability for Internet video”, IEEE Transactions on Circuits and Systems for Video Technology, Vol. 11 , Issue 3, pp. 318-331, March 2001.

[85] Van der Schaar, M. and Radha, H., “Adaptive motion-compensation fine-granular-scalability (AMC-FGS) for wireless video”, IEEE Transactions on Circuits and Systems for Video Technology, Vol. 12 , Issue 6, pp. 360-371, June 2002.

[86] Wong, Chi-Wah A. and Kwok, Yu-Kwong, “On a region-of-interest based approach to robust wireless video transmission”, Proceedings of the 7th International Symposium on Parallel Architectures, Algorithms and Networks, 2004, pp. 385-390, 10-12 May 2004.

[87] Fitzek, F.H.P. and Reisslein, M., “A prefetching protocol for continuous media streaming in wireless environments”, IEEE Journal on Selected Areas in Communications, Vol. 19, Issue 10, pp. 2015-2028, Oct. 2001.

[88] Guan, Dongliang and Yu, Songyu, “A two-level patching scheme for video-on-demand delivery”, IEEE Transactions on Broadcasting, Vol. 50, Issue 1, pp. 11-15, March 2004.

[89] Guan, Dongliang and Yu, Songyu, “A new patching channel schedule scheme for video multicast”, IEEE Transactions on Consumer Electronics, Vol. 49, Issue 2, pp. 342-347, May 2003.

[90] Eager, D., Vernon, M. and Zahorjan, J, “Minimizing bandwidth requirements for on-demand data delivery”, IEEE Transactions on Knowledge and Data Engineering, Vol. 13, Issue 5, pp. 742-757, Sept.-Oct. 2001.

[91] Shi, R.T., Shao, L., Pei, Y.Z. and Xie, D., “A novel stream merging algorithm for VOD servers”, Proceedings of the 2003 Joint Conference of the Fourth International Conference on Information, Communications and Signal Processing, 2003 and the Fourth Pacific Rim Conference on Multimedia, Vol. 3, pp. 1982-1986, 15-18 Dec. 2003.

[92] Goodenough, John B., and Sha, Lui, “The priority ceiling protocol: A method for minimizing the blocking of high-priority Ada tasks”, Special Report, CMU/SEI-88-SR-4, Mar. 1988.

[93] Lipari, G., Buttazzo, G., and Abeni, L., “A bandwidth reservation algorithm for multi-application systems”, Proceedings of the IEEE Conference on real-time computer systems and applications, pp. 77-82, Oct. 1998, Hiroshima, Japan.

[94] Caccamo, M., Lipari, G., and Buttazzo, G., "Sharing resources among periodic and aperiodic tasks with dynamic deadlines", Proceedings of the IEEE Real-Time Systems Symposium, pp. 284-293, Dec. 1999, Phoenix, Arizona.

[95] Abeni, L., and Buttazzo, G., "Resource reservation in dynamic real-time systems", Real-Time Systems, Vol. 27, No. 2, pp. 123-167, Jul. 2004.

[96] Bini, E., and Buttazzo, G., "Schedulability analysis of periodic fixed priority systems", IEEE Transactions on Computers, Vol. 53, Issue 11, pp. 1462-1473, Nov. 2004.

[97] Ng, Joseph. K., “A reserved bandwidth video smoothing algorithm for MPEG transmission”, Journal of Systems and Software, Vol. 48, Issue 3, pp. 233-245, Nov. 1999.