Improving Video-On-Demand Performance with Prefetching

Farnoosh Falahatraftar

Submitted to the
Institute of Graduate Studies and Research
in partial fulfillment of the requirements for the Degree of

Master of Science
in
Computer Engineering

Eastern Mediterranean University
June 2013
Gazimağusa, North Cyprus

Approval of the Institute of Graduate Studies and Research

Prof. Dr. Elvan Yılmaz
Director

I certify that this thesis satisfies the requirements as a thesis for the degree of Master of Science in Computer Engineering.

Assoc. Prof. Dr. Muhammed Salamah
Chair, Department of Computer Engineering

We certify that we have read this thesis and that in our opinion it is fully adequate in scope and quality as a thesis for the degree of Master of Science in Computer Engineering.

Assoc. Prof. Dr. Işık Aybay
Supervisor

Examining Committee

1. Assoc. Prof. Dr. Işık Aybay

2. Assoc. Prof. Dr. Muhammed Salamah

3. Asst. Prof. Dr. Gürcü Öz

ABSTRACT

Over the past few years multimedia communications has become essential parts of people's daily life. In this context, video streaming is attracting extensive attention and is becoming one of the most popular activities over the Internet. However, video streaming supports a large number of simultaneous users and consumes more network bandwidth as compared to other internet applications. So, implementations that can improve video streaming efficiency are of particular importance. On the other hand, the spectacular development in Peer-to-Peer (P2P) technologies presents scalability and support for large number of users worldwide.

In this work, we consider a prefetching mechanism in a P2P Video-on-Demand (VoD) system and study performance of using this prefetching method on our model. We compute the prefetching time for one video segment and then divide our idle time into several slices of prefetch activities. The prefetched segments are the segments which are not available on other peers in the network, therefore those must be prefetched from the server directly.

With using this prefetching mechanism, the idle time of the system is reduced and consequently, the efficiency of the system will be improved.

Keywords: Prefetching, VoD Systems, Peer-to-Peer Networks, Efficiency

ÖZ

Son yıllarda multimedia iletişim konusu insan yaşamında vazgeçilmez bir yer tutmaktadır. Bu bağlamda, akan video uygulamaları, Internet üzerindeki en yaygın uygulamalar arasında yer almaktadır. Ancak akan video, aynı anda bır çok kullanıcıya ulaşması gereken ve dığer internet uygulamalarına göre daha büyük bant genişliğine ihtiyaç duyan bır uygulamadır. Bu nedenle, akan video uygulamalarında etkinlik ve hızı artırabilen yaklaşımlar önem kazanmaktadır. Aynı ağ üzerindeki bilgisayarların birbirine destek olmasını sağlayan P2P (Peer-to-Peer) tekniği de bu konuda yararlı olmaktadır.

Bu çalışmada, P2P akan video uygulamaları için önceden-getirme (Prefetching) yöntemini kullanarak sistem etkinliginin arttırılması amaçlanmıştır. Önceden-getirme işlemi, sistemin boş (Idle) zamanlarında yapılmaktadır. Aynı ağdaki diğer bilgisayarlarda bölümler (segment) için önceden-getirme işlemi uygulamaktadır.

Önceden-getirme yöntemi ile sistemin boş geçirdigi zaman azaltılmakta ve böylece sistem etkinligi artırılmaktadır.

Anahtar kelimeler: Önceden-getirme, Akan Video Sistemleri, Peer-to-Peer Ağlar, Sistem Etkinliği

To my Father and Mother

ACKNOWLEDGMENTS

I would like to thank Assoc. Prof. Dr. Işık Aybay for his continuous support and guidance in the preparation of this study. Without his invaluable supervision, all my efforts could have been short-sighted.

I wish to express my thanks to all the members of the Department of Computer Engineering at Eastern Mediterranean University.

I owe quit a lot to my family who allowed me to travel all the way from Iran to Cyprus and supported me all throughout my studies. I would like to dedicate this study to them as an indication of their significance in this study as well as in my life. Besides, a number of friends had always been around to support me morally. I would like to thank them as well.

TABLE OF CONTENTS

ABSTRACT III

ÖZ IV

DEDICATION V

ACKNOWLEDGMENTS VI

LIST OF TABLES IX

LIST OF FIGURES X

1 INTRODUCTION 1

2 LITERATURE REVIEW 3

2.1 Peer-to-Peer Video-on-Demand Systems 3

2.2 LCBBS Module 4

2.3 Different Prefetching Models 5

3 P2P TECHNOLOGY IN LAN NETWORKS 7

3.1 Introduction 7

3.2 Ethernet Frame Format 8

3.3 Context Switching 9

3.4 CPI 10

3.5 Clock Rate 11

4 PREFETCHING MODEL 13

4.1 Introduction 13

4.2 Prefetching Model Proposed by this Study 14

4.2.1 Prefetching a Segment 15

4.2.2 Transmission Time 17

4.2.3 Unpacking Time 17

4.3 Timing Dicussions 19
4.3.1 Assume no Packet Loss 19
4.3.2 Assume Lost Packets 26
4.4 Efficiency Discussion 29
4.5 Assumptions Review 32
5 PREFETCHING SIMULATION RESULTS 34
5.1 The Simulation Model 34
5.2 Results and Discussion 35
5.3 Comparison with Similar Studies 40
CONCLUSION 43
REFERENCES 45
APPENDIX 48

LIST OF TABLES

Table 1. Packet Availability Table 16

Table 2. Assumptions Considered In This Study 33

Table 3. Our Prefetching Model versus Other Prefetching Strategies.....40

Table 4. Samples of Different System Schedules.....48

Table 5. Samples of Different System Schedules.....49

Table 6. Samples of Different System Schedules.....50

Table 7. Samples of Different System Schedules.....51

Table 8. Samples of Different System Schedules.....52

Table 9. Samples of Different System Schedules.....54

Table 10. Samples of Different System Schedules.....55

Table 11. Samples of Different System Schedules.....56

Table 12. Samples of Different System Schedules.....57

Table 13. Samples of Different System Schedules.....58

LIST OF FIGURES

Figure 1. System Architecture 8

Figure 2. A Basic 10/100 Ethernet Frame Format 8

Figure 3. A Case Which Shows Segments Available/Unavailable Locally 14

Figure 4. Dividing Idle Time into Segments of Prefetching Time 14

Figure 5. Prefetching Steps 19

Figure 6. Creating Table Program with 1399 Zero Values 20

Figure 7. Prefetching Analysis for 1399 Packets 21

Figure 8. Getting Packet Program with 770 Instructions One Group.....21

Figure 9. Getting Packet Program with 359 Instructions.....22

Figure 10. Check Program for Finding Any Zero Value. 22

Figure 11. Final Check Program Assuming No Lost Packets.....23

Figure 12. Prefteching Time Analyze In The Case of No Packet Loss 25

Figure 13. Dividing Each 256 Packets into One Group.....26

Figure 14. Time Needed for First Step 27

Figure 15. Time Needed for Second Step 28

Figure 16. An Example of a System Schedule 30

Figure 17. Impact of Using Idle Time for Segment Prefetching On Efficiency 33

Figure 19. Efficiency with Using Prefetching Model.....38

Figure 20. Busy and Idle Time in Millisecond for the First 10 Samples.....40

Chapter 1

INTRODUCTION

Today video on demand systems are used by millions of users around the world. These users are served by enormous servers which are employed for streaming video data. It is expensive to stream massive numbers of data videos on the Internet with high quality and less time latency. In order to reduce the costs, we can use the Peer-to-Peer (P2P) technology which makes use of available resources of peers; moreover, we can use the advantages of prefetchig strategy to ensure play back continuity.

The requested video is divided to small blocks which are called segments. The user can watch his/her requested video by downloading (or prefetching) these segments from peers or from the main sever. With prefetching mechanism a peer can get a video segment earlier than display time.

In peer to peer VoD systems, each peer can use available resources of other peers; in fact peers can act both as a provider and as a consumer. When a peer needs a video, it sends a request for that to other peers. Then, other peers check their buffer for requested video segments and if any segment is found, they send back a list of available segments of requested video to the requesting peer. The mentioned peer collects all responses from its peers and finds out which segments are not available. If the prefetching mechanism takes a lot of time, it will not be suitable because it will

be better to download and display normally instead of prefeching segments. Therefore, measuring the time needed for prefetching mechanism is very important.

Several recent works proposed prefetching mechanisms based on priority of chunks. In one approach, chunks closer to the current playing position have more importance for prefetching [3]. These systems also use a scheduler to define the order of packets to be transmitted from the queues. In another approach, just one segment after the currently played segment can be prefetched by full speed but the one after next cannot be prefetched, after prefetching peer release the occupied bandwidth for other peers [2].

Prefetching strategies based on priority and mathematical computations will take more time, but time is a main factor in peer to peer VoD systems. So, the aim is to reduce the latency more and more. In this study, we focus on the location of requested video segments. The main idea is that the chunks that are only available in main server must be prefetched first. So, all the segments that are not available in other peer's buffers have a priority for prefetching. In our work, we calculate time needed for prefetching segments upon basic assumptions and different circumstances. We develop a prefetching model and perform a detailed analytical study and simulation which takes into consideration factors like the context switching time, average clock cycles per instruction and clock cycle time.

Chapter 2

LITERATURE REVIEW

2.1 Peer-to-Peer Video-on-Demand Systems

Video-on-Demand (VoD) is a compelling application, but it is also costly. VoD is costly due to the load it places on video source servers. Some researchers have proposed using peer-to-peer (P2P) techniques to shift some load from servers to peers. This technique has been used successfully for file downloading and live streaming.

VoD differs from other Internet media applications in several important ways. First, in VoD a user can begin a VoD session at any time and seek to any position during playback. In live streaming, a stream begins at the same time for everyone and users cannot seek forward and backward arbitrarily. Second, VoD has strict real-time constraints while file downloading does not.

For VoD, the next segment is more important than a later one while any new file part is good for file downloading. VoD is more challenging than live streaming or file downloading because of user-control operations and real-time bonds.

The peer-to-peer networks are attractive for VoD since, they can provide a data distribution model which reduce the costs and increase the scalability of video distribution. For reducing server's workload, the P2P attempts to use the peer's upload bandwidth. Satisfying the application requirements of as many end users as

possible with sustainable server bandwidth costs is the Internet media streaming's final goal. For maintaining streaming to end users in traditional client/server architecture in large scale, vast data centers are used. The bandwidth cost on servers increases rapidly as the user population increases, and may not be manageable in corporation with limited resources.

Peer to Peer technology comes for declining server utilization. For instance, in a network consisting of some peers, at first other peers may not have copies of requested data and it must be downloaded from servers. However, as time goes, buffers of other peers will contain the popular data in network and consequently number of downloads from the server will be decrease.

Many methods are proposed for improving efficiency of VoD systems with P2P technology. Some examples are use of client back-end buffering system [1] or multi-channeling. Data Prefetching has also been proposed as a technique for reducing the access latency [7]. In this work we consider prefetching mechanism as an approach to amending the utility of VoD systems.

2.2 LCBBS Module

An LCBBS (client back-end buffering system) can be installed and used on each peer in a LAN. The segment available table and a two level buffer are the LCBBS important parts. First level buffer is used for storing the names of videos and maximum number of segments. The second level buffer is used for storing actual video data and FIFO strategy is used for storing data in these two level buffers.

If LCBBS seeks on LAN for video segments, SAT is created. When one segment exists in first level local buffer, the segment status is "local". Otherwise, LCBBS module searches on LAN peers and if a peer has a copy of the segment, the related status for that segment is "LAN". Finally the "remote server" status is for the segments which do not exist on local buffer and LAN. Then the remote server transferred these segments.

In LCBBS, a message is sent by the communication system to all nodes in LAN. The LCBBS puts segments's number which exist in first level buffer, into response message and multicast it in LAN. Then, the SAT of peer which needs to this video will be updated after receiving this message. These segments can be downloaded from the remote server directly, in the case that the message is lost.

The remote server calculates necessary byte of video segment's addresses for executing remote server transfer function. Then this function requests these data from the remote server. Eventually, after downloading data from remote server completely, the system assigns "local" status for segment in SAT and its information will be recorded in first level buffer.

With increasing buffer size, LCBBS improves the responding time and also it reduces start up latency and total stopping time for each video [1].

2.3 Different Prefetching Models

In the past years, several methods have been proposed for multimedia prefetching. [8] For example, a basic random strategy based on segments which are close to the play back position. This segment selection is randomly and probability is inversely

related to the number of replicas of segment. Generating lot of overhead for sharing the segment's information in large network is drawback of this method. An optimal off-line prefetching algorithm and a heuristic prefetching algorithm were proposed in [9]. It is shown that with using appropriate prefetching policies the performance of layered video can be improved.

Another approach is [7] proposed a cooperative prefetching strategy that decreases the overhead significantly. Moreover, for selecting the best peer for supplying of contents, they suggest a scheduling mechanism.

One prefetching model is based on user seeking behaviors [16]. The authors suggest guided seek which is based on segment access information. This information gained from seeking statistics in the current or previous segments. The guided seek is different from random seek. The amount of access to a segment effects on its popularity, so the popular segment is requested and visited more repeatedly.

Chapter 3

P2P TECHNOLOGY IN LAN NETWORKS

3.1 Introduction

In Peer-to-Peer (P2P) technology, peers download text or video data from other peers in network. For example, a peer downloads data from other peers if the other peers have got the requested data, otherwise the peer must download what it needs from the server directly. In this approach, we assume that we have many peers in the local network which can download/upload data from/to other peers and each one has appropriate buffer space for storing data, the server is contacted in the condition that peers cannot do anything for other one. Each peer provides the content to other peers.

Figure 1 shows our system architecture. The server is close to the clients, so we ignore the propagation time. However, this model can also be used with modification (adding propagation) for remote server systems. The communication network can be LAN, Wireless LAN or Mobile cellular system, in our study we suppose that it is LAN and also the type of our network is 100 Mbps Ethernet with 1500 byte payload size for each packet.

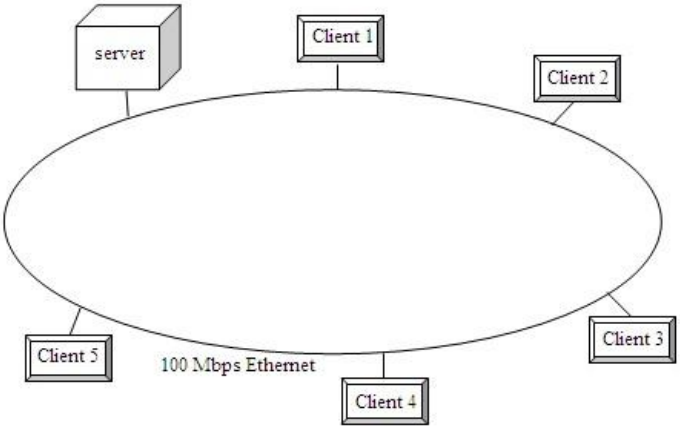

Figure 1. System Architecture

3.2 Ethernet Frame Format

We assume the communication system is a 100 Mbps Ethernet. Figure 2 shows a 10/100 Mbps Ethernet frame which includes [12]:

Preamble	7 octets
Start Frame Delimiter	1 octets
Destination Address	6 octets
Source Address	6 octets
Length /Type	2 octets
Payload	46 octets to 1500 octets
Pad	
Frame Check Sequence	4 octets

Figure 2. A Basic 10/100 Ethernet Frame Format [12]

3.3 Context Switching

In a computer system, the scheduler which is inside the operating system maintains a queue of executable threads for each process priority level. These are known as ready threads. When a processor becomes available for further processing, the system performs a context switch.

The most common reasons for a context switch are:

- The time slice allocated for a process has passed.
- A thread with a higher priority is ready to run.
- A running thread needs to wait for some peripheral/memory actions.

Context switching performs in some CPUs which have hardware support for it, and also it can be executed by the operating system software.

The state of the process includes all the registers that the process may use particularly the program counter and also other operating system specific data. This data is stored in a data structure called a process control block (PCB).

The contents of a CPU's registers and program counter at any point in time are the "context". Context switching executes the next activities:

1. Suspending one process and storing the CPU's context for that process.

2. Choosing the next process to be run, retrieving the context of that process from memory and restoring it in the CPU's registers

3. Returning to the location indicated by the program counter (returning to the line of code at which the next process was interrupted) in order to resume the process.

Some processors context switching times:

- Intel 5150: ~1900ns/process context switch, ~1700ns/thread context switch
- Intel E5440: ~1300ns/process context switch, ~1100ns/thread context switch
- Intel E5520: ~1400ns/process context switch, ~1300ns/thread context switch
- Intel X5550: ~1300ns/process context switch, ~1100ns/thread context switch
- Intel L5630: ~1600ns/process context switch, ~1400ns/thread context switch

Therefore, we consider 2 microseconds for context switching time. (2μsecs equals to 2000ns) [10, 11].

3.4 CPI

In computer architecture, cycles per instruction (CPI) is a term used to describe one aspect of a processor's performance: the number of clock cycles spent that happen when an instruction is being executed. CPI is the average number of clock cycles per instruction.

$$CPI = \text{CPU clock cycles} / \text{Instruction count} \quad (1)$$

$$\text{CPU Clock Cycles} = \text{Sum}(CPI_i * Count_i) \quad (2)$$

The average CPI from processors like MIPS, Intel and etc is 10, so we suppose that CPI is 10.

3.5 Clock Rate

Computers are constructed using a clock that runs at a constant rate and determines when events take place in hardware. Clock rate is the number of cycles per second; it is typically measured in megahertz or gigahertz. One megahertz is equal to one million cycles per second, while one gigahertz equals one billion cycles per second.

A 1.8 GHz CPU is not necessarily twice as fast as a 900 MHz CPU. Because different processors usually use different architectures. For instance, one processor may need more clock cycles to complete an instruction than another processor. If the 1.8 GHz CPU can execute an instruction in 4 cycles, but it takes 7 cycles in 900 MHz CPU, the 1.8 GHz processor will be more than twice as fast as the 900 MHz processor [17].

Clock rate is the reverse of clock period: 1/cycle time. So, the execution time can be:

$$\text{Execution time} = \text{clock cycles} / \text{clock rate} \quad (3)$$

For example, in group of Intel Core i7Extereme Processors, i7-990x has maximum clock speed of 3.46 GHz and i7-965 has minimum clock speed with 3.20 GHz. So the clock time for each one is 0.28 ns and 0.31 ns respectively.

Here we assume that our processor is i7-3960X with 3.30 GHz clock rate. Therefore:

$3.3 * 10^9 = 1/$ clock time

Clock time$=1/3.3*10^9=0.30*10^{-9}$

Clock time= 0.3 ns (3.3 Billion Cycles per second)

In group of Intel Core i5 Processors, clock speed has minimum number of 2.30 GHz and maximum number of 3.40 GHz. Consequently, clock time is 0.29 ns minimally and 0.43 ns maximally.

Clock time$=1/2.30*10^9=0.43*10^{-9}$ ⟶ Clock time= 0.43 ns

Clock time$=1/3.40*10^9=0.29*10^{-9}$ ⟶ Clock time= 0.29 ns

Chapter 4

PREFETCHING MODEL

4.1 Introduction

Prefetching mechanisms are based on making a decision for choosing chunks. Although each prefetching mechanism has a different rule for segment selecting, a segment must have priority to be prefetched. For instance, popularity is a factor for prioritizing the segments and a segment which is most popular has the highest priority. The prefetching strategies are currently based on priority and mathematical solutions [16, 2, and 18] which may take a lot of time. Time is the main factor in peer to peer VoD systems and all of the work in this aspect is for reducing the time latency. Mentioning this point, we try to focus on location of requested video segments. A segment has high priority if it is not available in the local buffer and other peers do not have it. In this approach unavailable segments must be prefetched directly from main server. When a peer needs a video, it sends a request for that video to other peers in network. Then other peers check their buffer for requested video segments and if they have segments of it, they send back a list of available segments of requested video to the peer which has requested for this video. The mentioned peer collects all responses from the peers and finds out which segments are not available and should be prefetched from the server directly. Figure 3 shows one case of the available/unavailable segments of a requested video. The priority for prefetching is based on location of segments which are not in peer and are in the

main server .All the segments that are not available in other peers' buffers have a priority for prefetching.

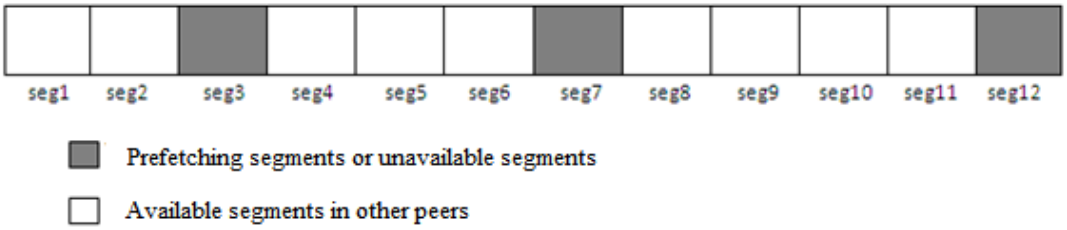

Figure 3. A Case Which Shows Segments Available/Unavailable Locally

4.2 Prefetching Model Proposed by this Study

In our prefetching model, we calculate prefetching time for segments with a fixed 2 Mbyte size and take the advantage of idle time. As Figure 4 indicates, if one segment needs 'x' time for prefetching, we divide our idle time into one or several slices of 'x' times. 2 Mbyte fixed size is chosen with reference to the discussion given in [1].

When idle time begins, the peer executes context switching and starts prefetching and checks P2P ports periodically. the must do context switching again when finishing the idle time and starting download from other peers, but if this context switching is requested while prefetching a segment, then the peer has to delay context switching to the end of segment retrieval.

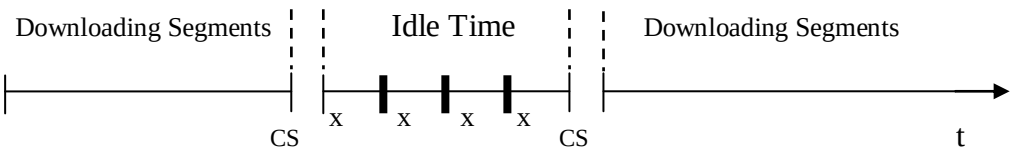

Figure 4. Dividing Idle Time into Segments of Prefetching Time (CS is Context Switching)

Without prefetching in idle time, we have two statuses: idle and busy, therefore efficiency of the system can be calculated as:

$$\frac{\text{Busy}}{\text{Busy} + \text{Idle}} \qquad (6)$$

In our prefetching model, idle time is used for prefetching segments. We have three statuses for our system: idle time, segment retrieval and busy. Busy time starts when peer downloads segments from other peers. So the efficiency of the system will can be calculated as:

$$\frac{\text{Busy} + \text{Segment Retrieval}}{\text{Busy} + \text{Segment Retrieval} + \text{Idle}} \qquad (7)$$

For the next step, we must calculate time needed for prefetching a 2Megabyte segment and use it as a scale to find out that how much of idle time we can use for prefetching segments.

4.2.1 Prefetching a Segment

In order to calculate time needed for prefetching a segment, we should compute time in each step of the prefetching mechanism. Prefetching a segment contains several parts: packet transmission, unpacking [20], receiving data and sorting them, checking for lost packets and buffering.

Firstly we calculate time for prefetching a segment in the best case with an assumption that we do not have any lost packet. Then we discuss the worst case where we have some lost packets.

When we say "no packet loss" it means that all packets of a segment are received correctly. We consider (Table 1) which includes the status of packets. Initially, all check bits are zero. Whenever a packet is received the corresponding zero check bit value will be changed to 1. After receiving each group of 256 packets, the system checks whether all check bits are 1. If there is a zero check bit, this indicates a lost packet which may be received later. When sending packets from server is finished, the peer must search this table to find out which packets were not received.

After each search on table for lost packets, if the system does not find any lost packet, it must sort packets in the buffer according to their packet numbers.

Table 1. Packet Availability Table

Packet number	Check bit
1	0
2	0
3	0
4	0
.	.
.	.
.	.
256	0
.	.
.	.
.	.
512	0
.	.
.	.
.	.
768	0
.	.
.	.
.	.
1024	0
.	.
.	.
.	.
1280	0
.	.
.	.
.	.
1399	0

4.2.2 Transmission Time

We consider a 2 Megabyte segment [1] which must be divided into 1399 packets since we assume that we use 100 Mbps Ethernet with a maximum payload size of 1500 bytes (100Mbps=13107200 bytes per second, 2 Megabytes = 2097152 bytes). Accordingly we can compute T_s (time for transmitting one segment) as:

$$Ts = \frac{\text{Size of a segment (byte)}}{\text{Band width (byte/sec)}} \qquad (8)$$

T_s = 2097152/13107200=0.16 second (160000 microseconds)

4.2.3 Unpacking Time

We need to consider the time for unpacking each packet (frame) [19]. We assume this unpacking process takes 121.44 microseconds [4] for each packet in our system.

$$\text{Time for unpacking k packet} = k * T_{unpack} \qquad (9)$$

Where T_{unpack} is time for unpacking one packet, T_{unpack}= 121.44, k=1399

121.44*1399=169894.56 microseconds≈ 170 milliseconds

For considering the sorting of packets in the buffer, there are certain sorting algorithms like heap sort, bubble sort and quick sort. Merge sort is the one that is appropriate for large amount of data. Actually, we have a counter that will be increased after each received packet and when counter equals to (256*a) where 'a' can be 1, 2…5 Merge sort algorithm is used to sort all packets. Figure 5 shows prefetching steps:

```
Start
Counter = 0
While Server is sending packets
{
Get packet and change corresponding zero value to 1
in packet availability table
Increase counter by one
For ( a=1; a<6; a++)
{
 If counter = (a*256) then
   {Check (a*256) entity of table for zero value
    If all (a*256) entity of table have value of 1 then  sort them in buffer
    }
}
}
L1: search on table for zero values
    If find zero value then
   {Send request for corresponding packet to the server
    %Server starts to send requested packets again
    While server is sending packet
       {Get packet and change corresponding zero value to 1 in packet
availability table }
Go to L1
}
If no zero value then
Sort all 1399 packets
End
```

Figure 5. Prefetching Steps

4.3 Timing Discussions

4.3.1 Assume no Packet Loss

As we mentioned before, we need 160000 microseconds for transmitting 1399 packets in addition to 170 milliseconds for unpacking them.

Creating the packet availability table with 1399 zero values take 12.597 microseconds as we explained in Figure 6.We assume that clock time is 0.0003 microsecond. This means that each clock cycle is equal to 0.0003 microseconds and the number the of clock cycles needed for each instruction (CPI) is 10. Thus:

$$T_i = CPI * Clock\ time \qquad (10)$$

Hence, T_i is time needed for each instruction, so an instruction needs 0.003 microseconds for execution. As discussed in Figure 6, we have 4199 instructions.

$$Running\ time = T_i * (number\ of\ instructions) \qquad (11)$$

4199*0.003=12.597 microseconds

After receiving each 256 packet group, the sort algorithm should be run. Figure 7 illustrates this point.

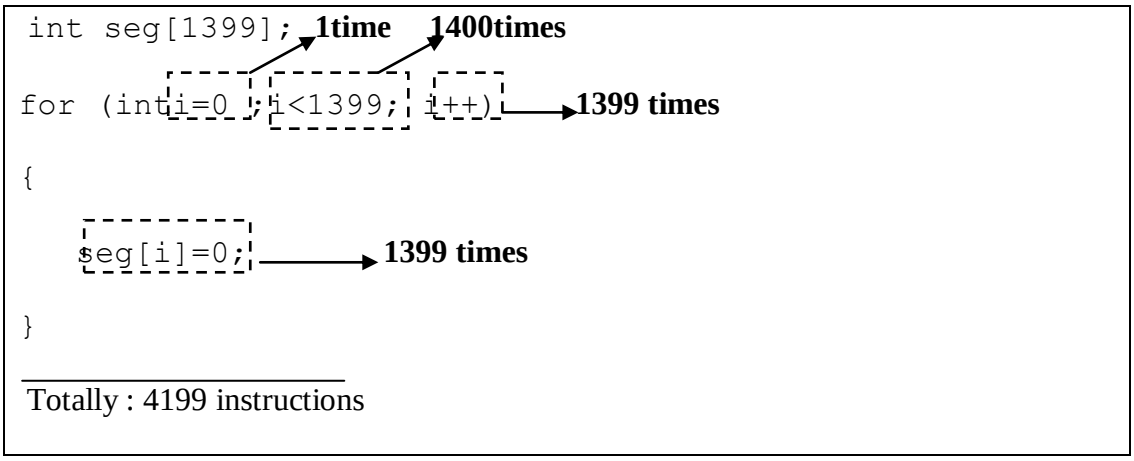

Figure 6. Creating Table Program with 1399 Zero Values

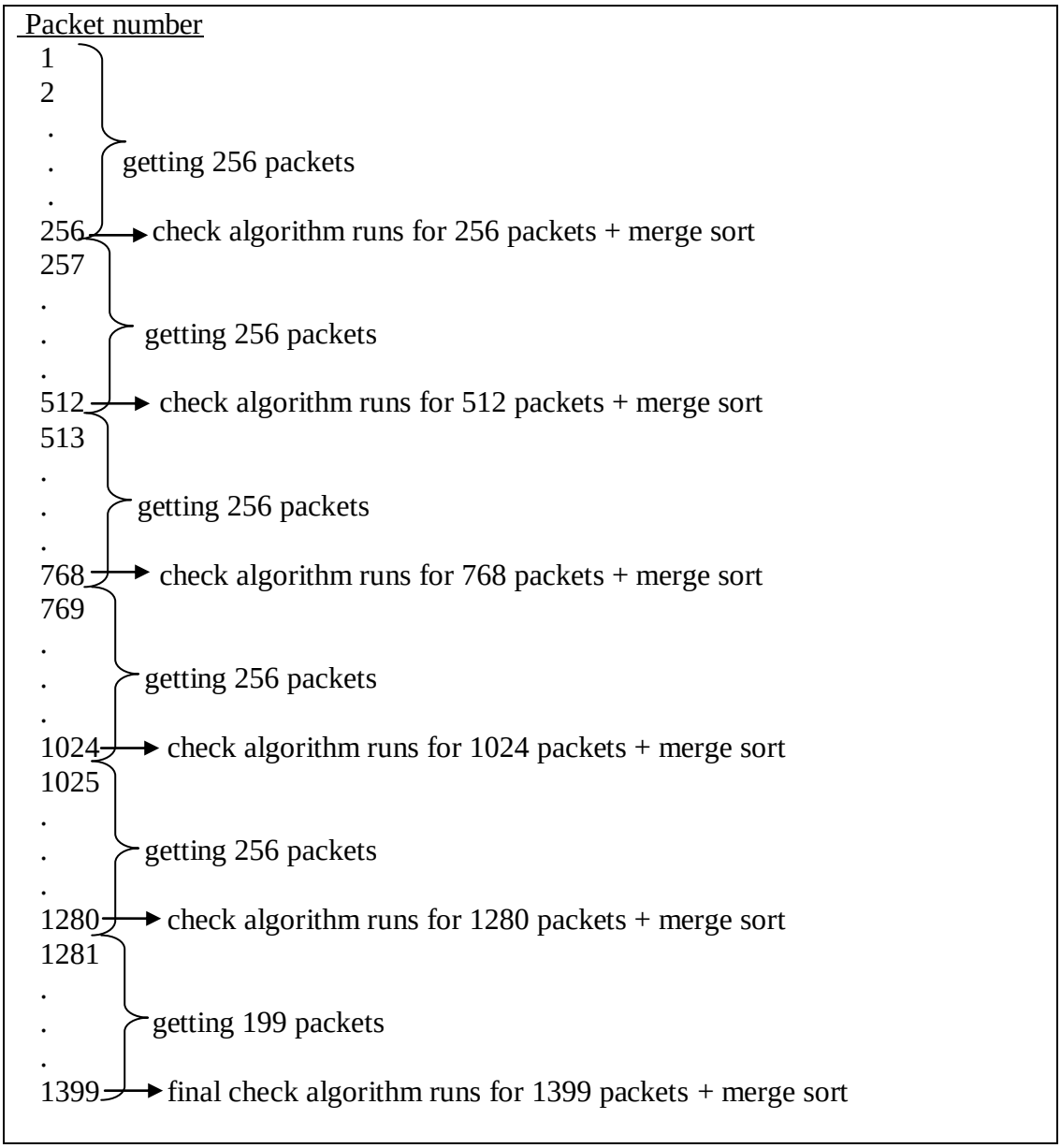

Figure 7. Prefetching Analysis for 1399 Packets

For calculating all steps indicated in Figure 7, we must compute the number of instructions in each step. Figure 8 and Figure 9 represent getting packets algorithm for two different groups, Figure 10 and Figure 11 indicate check algorithm and final check algorithm respectively.

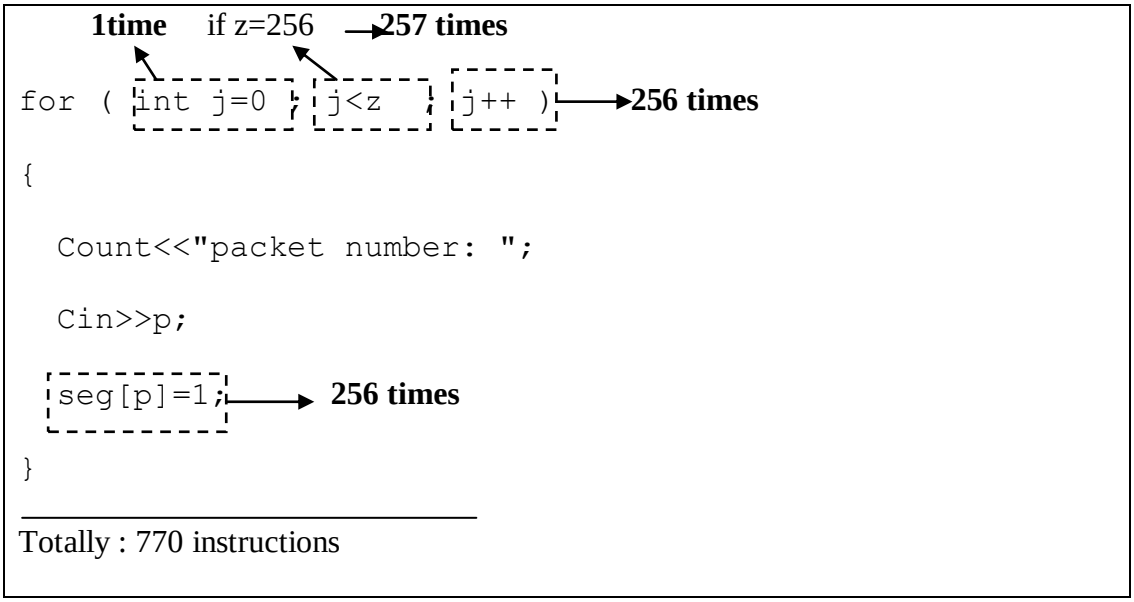

Figure 8. Getting Packet Program with 770 Instructions One Group

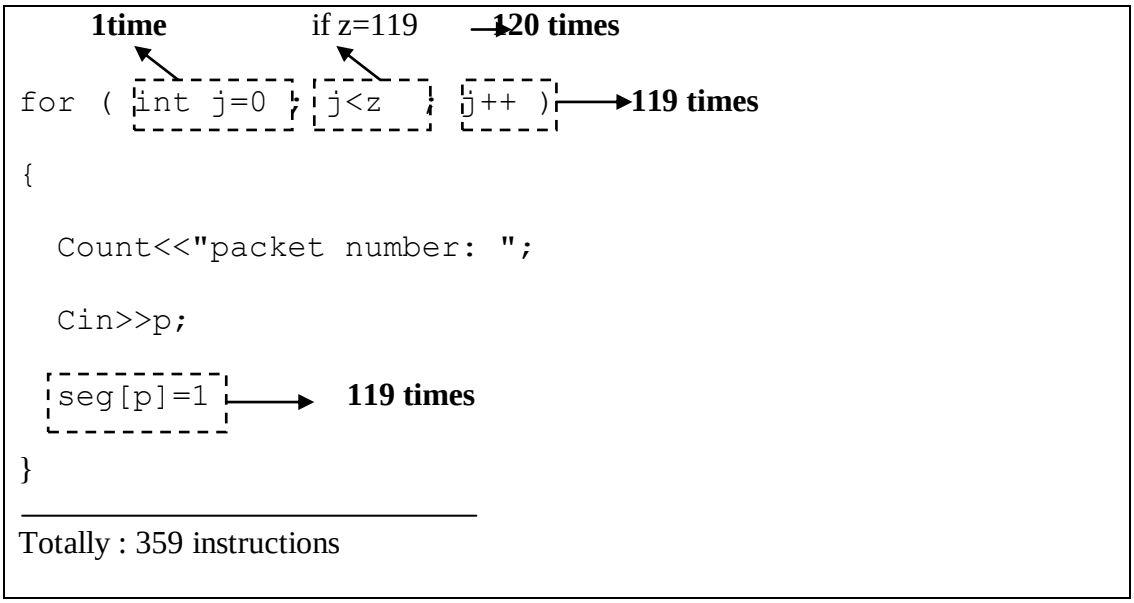

Figure 9. Getting Packet Program with 359 Instructions for One 119 Packet Group

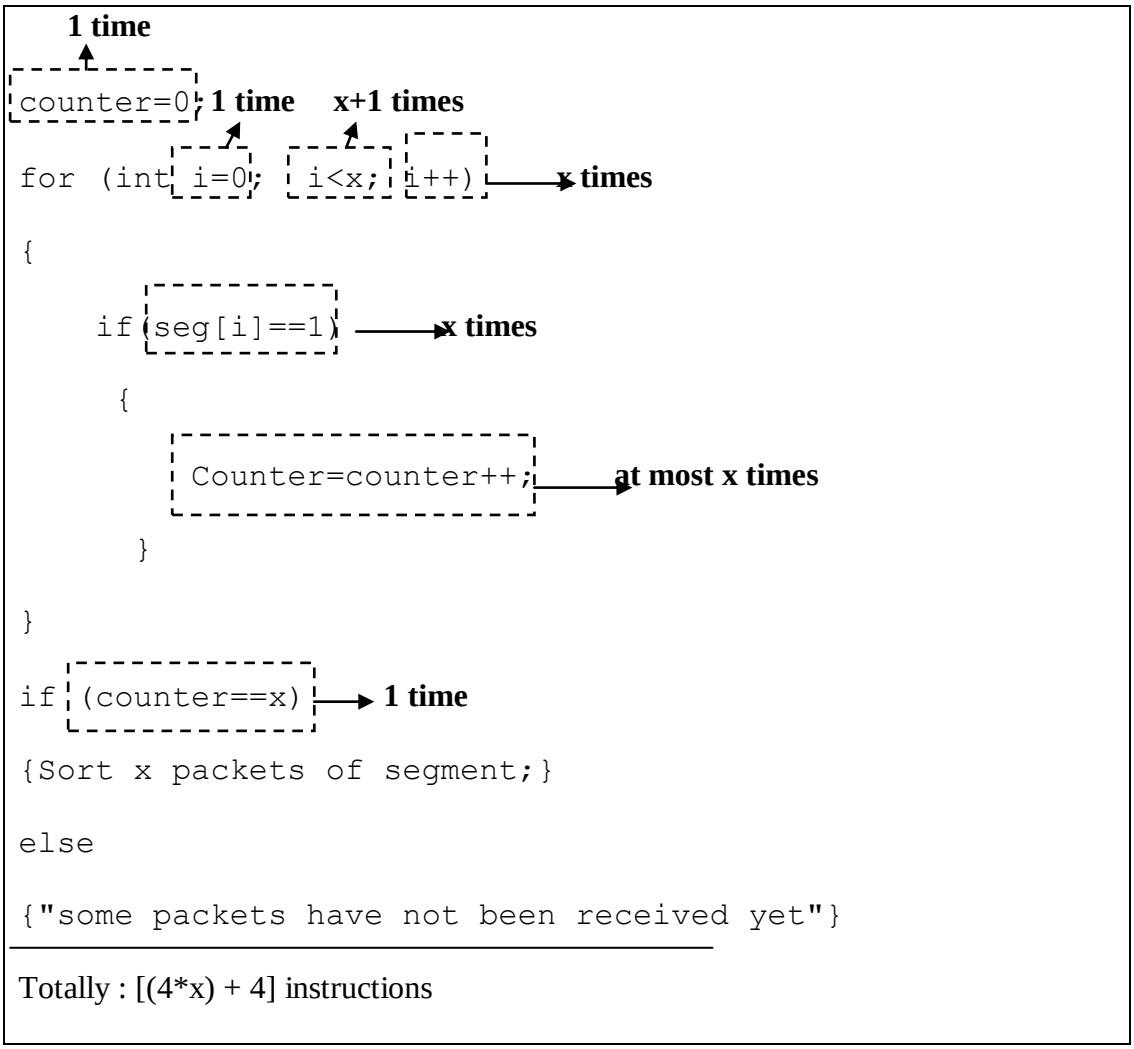

Figure 10. Check Program for Finding Any Zero Value. It Runs After Each 256 Group of Received Packets

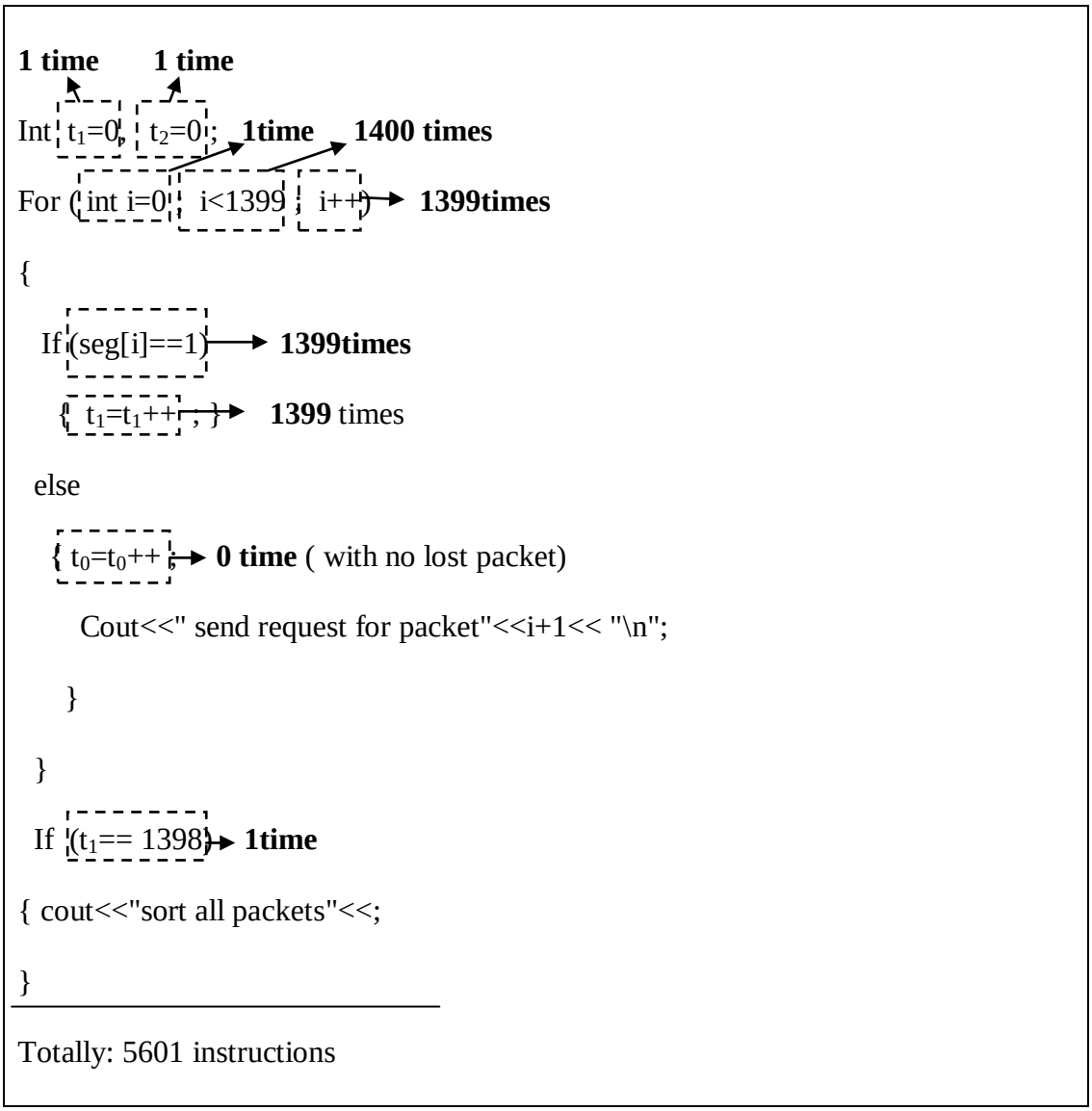

Figure 11. Final Check Program Assuming No Lost Packets ($t_1$and t_0 Indicate Number of Received Packet and None Received Packet Respectively)

In Figure 7 with the assumption that we do not have any packet loss, getting packet algorithm (Figure 8) runs 5 times with 256 packets and 1 more time with 119 packets (Figure 9).

From (11)⟶ 770*0.003=2.31 microseconds

359*0.003=1.07 microseconds

Check program must execute for 5 times that in each run with different values of 'x' (256, 512, 768, 1024 and 1280).

From (11) ⟶ (4*256) + 4=1028 , 1028*0.003=3.084 microseconds

(4*512) + 4=2052 , 2052*0.003=6.156 microseconds

(4*768) + 4=3076 , 3076*0.003=9.228 microseconds

(4*1024) + 4=4100 , 4100*0.003=12.3 microseconds

(4*1280) + 4=5124 , 5124*0.003=15.372 microseconds

Merge sort has to run after check algorithm execution. Totally it takes 26 milliseconds (26000 microseconds) for sorting packets [13, 14, 15].

The final check algorithm runs for finding any lost packet and it needs 5601 instructions if we do not have any packet loss, consequently:

From (11) ⟶ 5601*0.003=16.8 microseconds

Next figure shows time needed for each part of prefetching except transmission time and unpacking time.

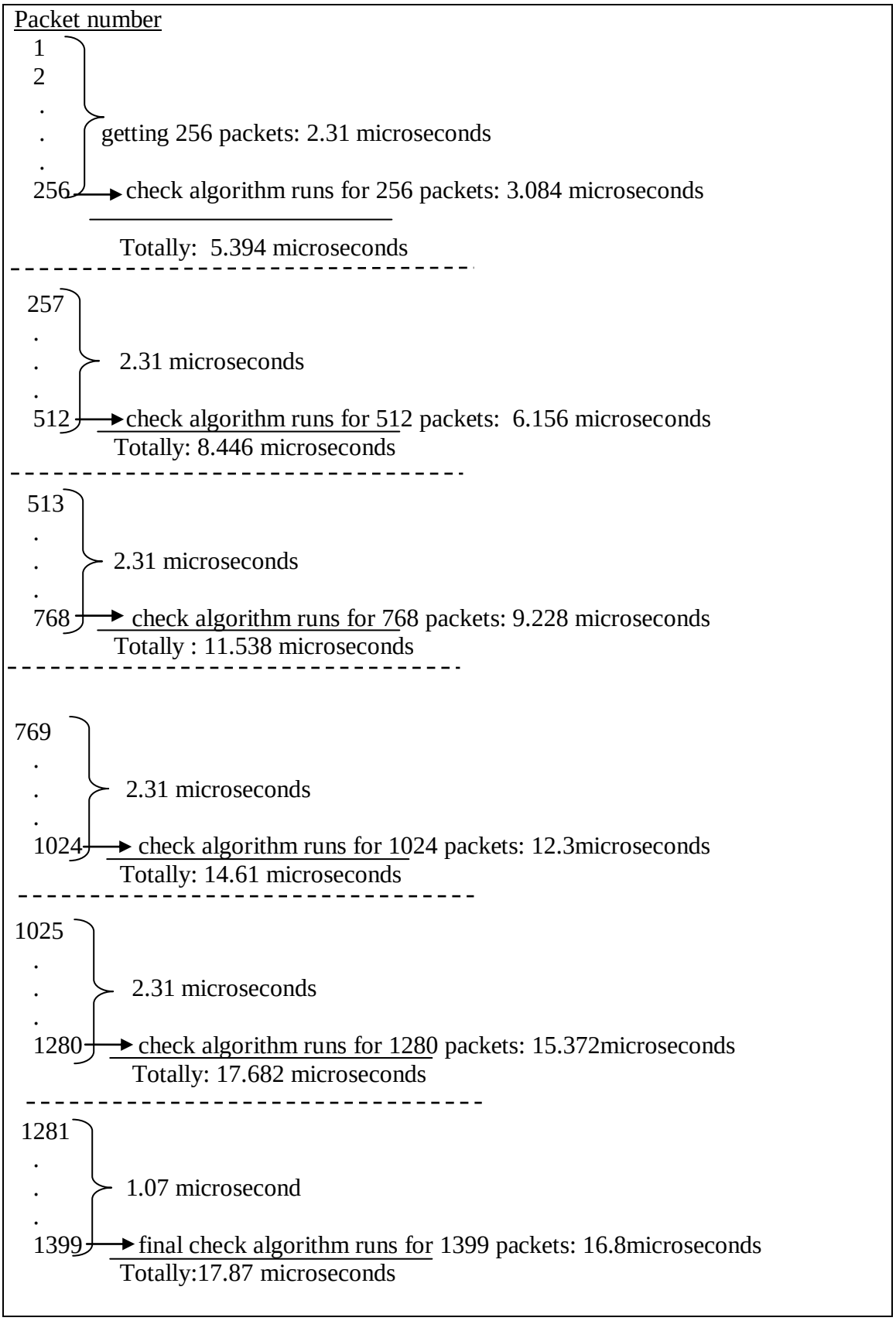

Figure 12. Prefteching Time Analyze In The Case of No Packet Loss

In order to compute time needed for prefetching a segment from server to client buffer, we must add transmission time and unpacking time with calculated values in Figure 12.

160000microseconds ⟶transmission time for 1399 packets

169894.56 microseconds ⟶unpacking time for 1399 packets

12.597 microseconds ⟶creating packet availability table

5.394 microseconds

8.446 microseconds

11.538 microseconds

14.61 microseconds

17.682 microseconds

17.87 microseconds

26000 microseconds ⟶sorting time (totally)

Totally: 355982.697microseconds ≈ 356 milliseconds

4.3.2 Assume Lost Packets

In this case we assume that approximately half of packets are not received, this means 700 packets ([1399/2] ≈700). If we divide 1399 packets into five groups of 256 packets and one with 119 packets (look at Figure 13).

A	B	C	D	E	F
256	256	256	256	256	119

Figure 13. Dividing Each 256 Packets into One Group

As discussed before, the check algorithm runs after each 256 received packets. Actually it checks if these 256 received packets are in one group or not. If yes, it sorts packets. Moreover, the final check algorithm runs whenever server stops

sending packet to node whether all 1399 packets received or not. We assume that we have some lost packets in each group (worst case).

So, when 256 packets are received, the check algorithm runs for finding that whether these 256 packets belong to group A? Answer is no in this case because we suppose that we have lost packet in each group. This happens similarly after receiving another 256 packets. Consequently, we get 512 packets until here, besides, after receiving 187 more packets, the system recognizes that sending packets from server is finished. So final check program starts to find lost packets (Figure 13). After that, the peer sends a request for lost packets to the server and server again sends these 700 lost packets. Thus, the check algorithm runs two times after receiving each 256 packets of 700 packets and the final check algorithm again executes (after receiving 188 remains packets) for finding any packet loss when no packets received from server (Figure 14).

Therefore, we need time for getting 699 packets, running the check algorithm two times and also the final check algorithm for first step (Figure 14):

Getting 256 packets: 2.31 microseconds

Check algorithm runs for 256 packets: 3.084 microseconds

Getting another 256 packets: 2.31 microseconds

Check algorithm runs for 512 packets: 6.156 microseconds

Getting 187 packets: 1.689 microseconds

Final check algorithm runs for 1399 packets: 16.8microseconds

Totally (for first step): 32.349 microseconds

Figure 14. Time Needed for First Step

After sending request for 700 lost packets (second step), again we need time for getting 700 packets, running the check algorithm for two times and time for executing the final check algorithm. Moreover, sorting time for sorting 1399 packets must also be considered.

Getting 256 packets: 2.31 microseconds

Check algorithm runs for 256 packets: 3.084 microseconds

- -

Getting another 256 packets: 2.31 microseconds

Check algorithm runs for 512 packets: 6.156 microseconds

- -

Getting 188 packets: 1.698 microseconds

Final check algorithm runs for 1399 packets: 16.8microseconds

Sorting time: 26000

Totally (for second step): 26032.358 microseconds

Figure 15. Time Needed for Second Step

Although just half of all packets of a segment received in first step, the server sent all 1399 packets. So we have two transmissions times, first one for transmitting 1399 packets in first step and second one for retransmission of 700 lost packets in second step. We assume no packet loss in the second transmission. As far as half of all packets are lost so we can assume that retransmission time for this half should be 0.8 second.

Beside, these 700 packets must be unpacked, so from (9)

121.44*700=85008 microseconds

These retransmission and unpacking process execute in time after first step.

Therefore, time measurement in the case of half of packets of a segment is lost:

160000microseconds ⟶ transmission time for 1399 packets

84886.56microseconds ⟶ unpacking time for 699 packets

12.597microseconds ⟶ creating packet availability table

32.349 microseconds ⟶ from Figure 14

80000microseconds ⟶ transmission time for 700 packets

85008microseconds ⟶ unpacking time for 699 packets

26032.358 microseconds ⟶ from Figure 15

Totally: 435971.864 microseconds≈ 436 milliseconds

From what we discussed above, prefetching time for a segment is 356 milliseconds in the best case where all packets are received completely and 436 milliseconds in the worst case where half of the packets are lost.

4.4 Efficiency Discussion

Our prefetching model is proposed for reducing the idle time by using it for prefetching segments which are not available in other peers in the network. As formula (6) indicates, efficiency and idle time have a sort of inverse relation with each other and we try to reduce idle time for improving system efficiency. Therefore, formula number (6) is improved to what we see in formula (7).

In this section, we use the results which are presented in the previous chapter about how long prefetching a segment takes, and we calculate the total time needed for prefetching all segments. Then we subtract this prefetching time from our idle time. Finally, we use formula (7) for computing efficiency for simulating our system and we provide some results.

Firstly, we propose some examples to clarify our simulation model.

First example: If we have a system schedule like the one in Figure 16, and segment prefetching time is 0.436 seconds (according to what we explained before in this chapter) and we prefetch 8 segments during idle time, how can we calculate efficiency in this system? (Assuming context switching time is 2 microseconds)

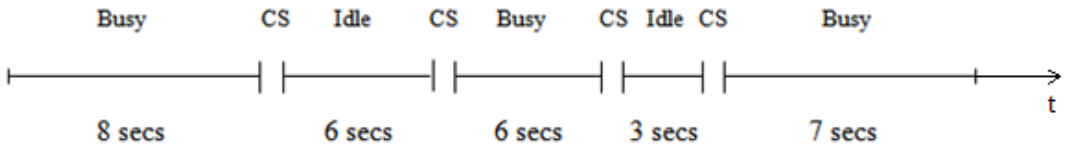

Figure 16. An Example of a System Schedule (CS: Context Switching)

In this case, the total time for prefetching 8 segments will be:

8*0.436=3.488 seconds

Total idle time: 6+3=9 seconds

Total busy time: 8+6+7=21seconds

Real idle time= Total idle time - time for prefetching 8 segments

Real idle time: 9-3.488=5.512 seconds

From (6): Efficiency without prefetching:

E=Busy/Busy + Idle

E=21 / (21+9) = 0.7

From (7): Efficiency with prefetching:

E_p= (Busy + Segment retrieval)/ (Busy + new Idle + Segment retrieval)

E_p = (21+3.488) / (21+5.512+3.488) = 0.81

Second example: Now assume that we have the same schedule as in figure 16 and we need 15 segments to prefetch from server, in this case:

Total time for prefetching 15 segments is: 15*0.436=6.54 seconds

Therefore, for this example, new idle time: 9-6.54 =2.46 seconds

From (7), efficiency with prefetching can be computed as:

E_p= (21+6.54) / (21+2.46 +6.54) = 0.91

Considering these examples, for the first case, we use from less than 50% of our total idle time by prefetching 8 segments (is about 38% of total idle time) and we get an efficiency of more than 80 percent improved from 70 percent. It means around 10 percent of improviement on efficiency. In the second example, when we use more than 50% of idle time by prefetching 15 segments (which takes about 72% of total idle time), the resulting efficiency is about 90 percent. This is 20 percent higher than the efficiency of the case without any prefetching.

In the previous examples, if we assume that segment prefetching time is 0.356 second (as discussed before, one segment preftching time with no packet loss), then efficiency can be calculated as:

Total time for prefetching 8 segments: 8*0.356=2.848 seconds

Therefore, real idle time: 9-2.848=6.152 seconds

From (7): E_p= (21 + 2.848) / (21 + 6.152 + 2.848) = 0.79

So, there is about 9 percents rise in efficiency, compared with efficiency with no prefetching segments in idle time.

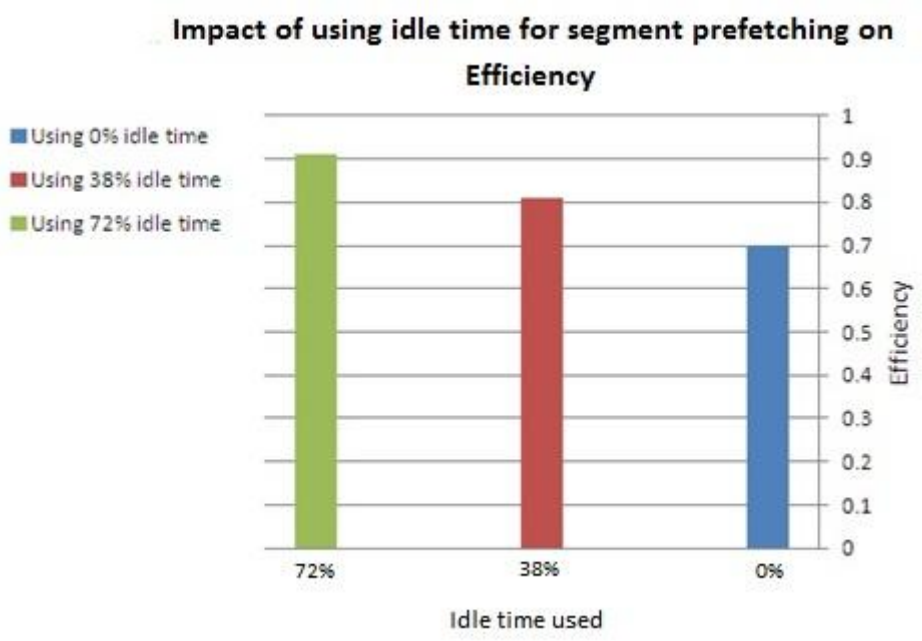

Figure 17. Impact of Using Idle Time for Segment Prefetching On Efficiency for Previous Examples

These examples indicate that we can improve efficiency if we use idle time for prefetching segments. As figure 17 indicates, when we use more than 70 percent of idle time for prefetching, we obtain efficiency about 90%. This amount is reduced to 81% when we use nearly 40% of idle time, which is still better than efficiency with no prefetching.

4.5 Assumptions Review

The next table contains all our assumptions which we need for obtaining our results in next chapter.

Table 2. Assumptions Considered In This Study

Assumption	Type or Value
Communication Network	LAN
Type of Network	100 Mbps Ethernet
Context Switching Time	2 microseconds
CPI(clock per instruction)	10
Clock Time	0.0003 microseconds
Transmission Time (for 2 MB segment)	0.16 second (160000 microseconds)
Unpacking Time (for 2 MB segment)	170 milliseconds(169894.56 microseconds)
Sorting Time (for 1399 packets)	26000 microseconds
One Segment Prefetching Time (without any lost packet)	~ 356 milliseconds
One Segment Prefetching Time (with lost packets, worst case)	~ 436 milliseconds
Propagation time	Ignored

Chapter 5

PREFETCHING SIMULATION RESULTS

5.1 The Simulation Model

In this chapter, we are going to discuss the results of simulation studies for a prefetching system. We use 100 system schedules like our example in chapter 4. Each schedule has different values for busy time and idle time sections during 20 seconds of simulation time. After each switching for idle time or busy time, we consider 2 microseconds for context switching. So we simulate our system by dividing 20 seconds of total time into sections which indicate busy and idle status of our system. We run this simulation for 100 times with different types of partitioning for busy and idle statuses. We use Matlab for simulating our system.

Values allocated for busy and idle time sections in each of the 100 samples are generated randomly. The assumption is that, the sum of them plus context switching time should be equal to 20000 milliseconds (20 seconds), which is the total simulation time.

In each run of our simulation model, we calculate total busy time and total idle time in addition to total time including context switching times that we have during this 20 second interval. Then we use formula (6) and (7) for computing efficiencies with and without prefetching.

Finally, we have two figures for illustrating the results of our simulation model for these 100 samples. The main goal of this study was to improve efficiency by using prefetching, which is shown to be achieved by the simulation study.

5.2 Results and Discussion

Tables 2, 3, 4, 5, 6, 7, 8, 9, 10 and 11 summarize our data and results for simulation. We have a variation of data for each part. The busy time is 4790 milliseconds minimum, and 15670 milliseconds maximum, because of data choice. On the other hand, 4290 milliseconds and 15170 milliseconds are minimum and maximum amounts for idle time respectively. We test our data with at most 9 segments for prefetching, beacause if we consider that each segment takes 436 milliseconds for prefetching (0.436 second, from results of chapter 4) then with 4290 millieconds, which is minimum idle time that we have, we can prefetch 9 segments, as:

$$\left\lfloor \frac{4290}{436} \right\rfloor = 9$$

From the given tables, efficiency is 0.2395 minimum and 0.7835 maximum without prefetching. These amounts rise to 0.4357 and 0.9797 in minimum and maximum with our prefetching model. Efficiency has a mean of 0.531 without any prefetching and a mean of 0.728 with using our prefetching model.

As results of one hundred samples indicate, in each sample we improve efficiency by almost 0.2 on the average, which is 20 percent (see figure 18 and figure 19).

From Figure 19, for our one hundred samples and with our assumption, efficiency has the highest amount where idle time is minimum. It shows that whenever we use from idle time more we can reach higher efficiency values. As explained above, we prefetch 9 segments, so we use:

9*436= 3924 milliseconds for prefetching.

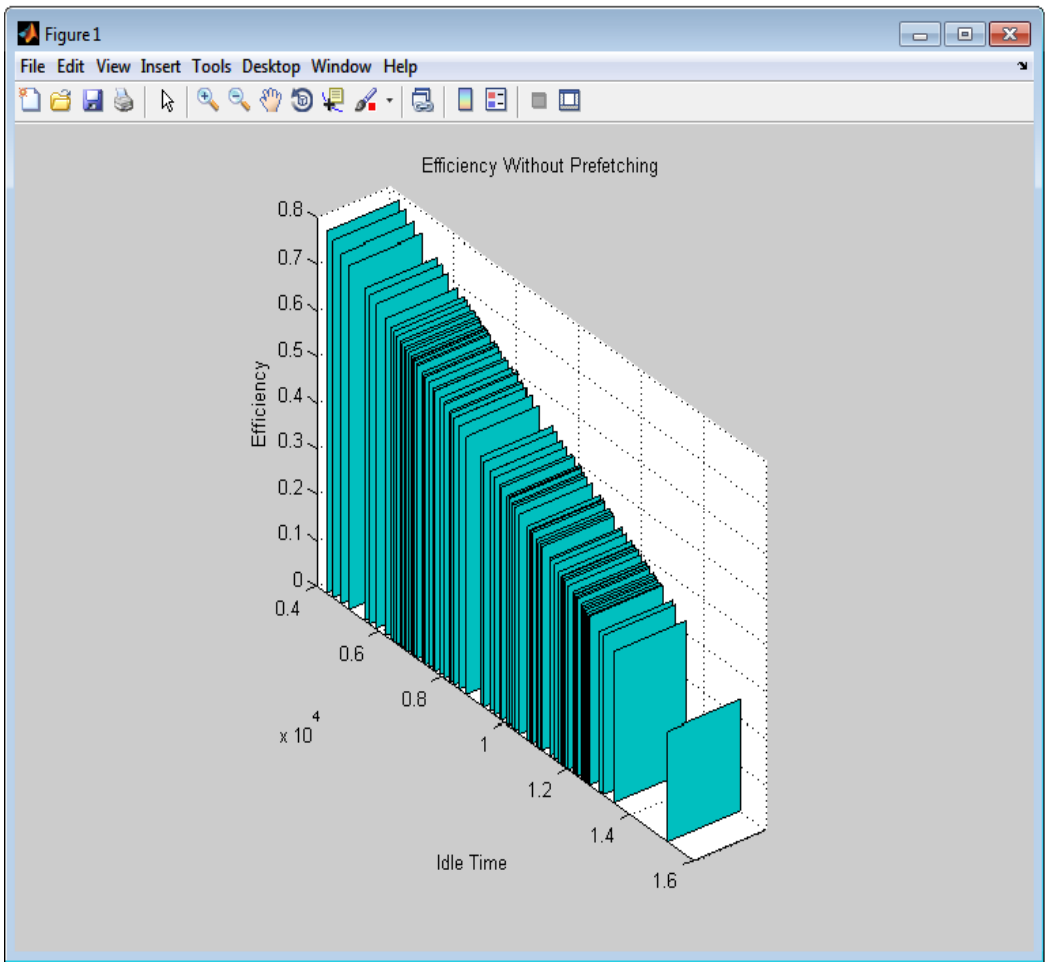

Figure 18. Efficiency without Prefetching (Idle Time Is in Millisecond)

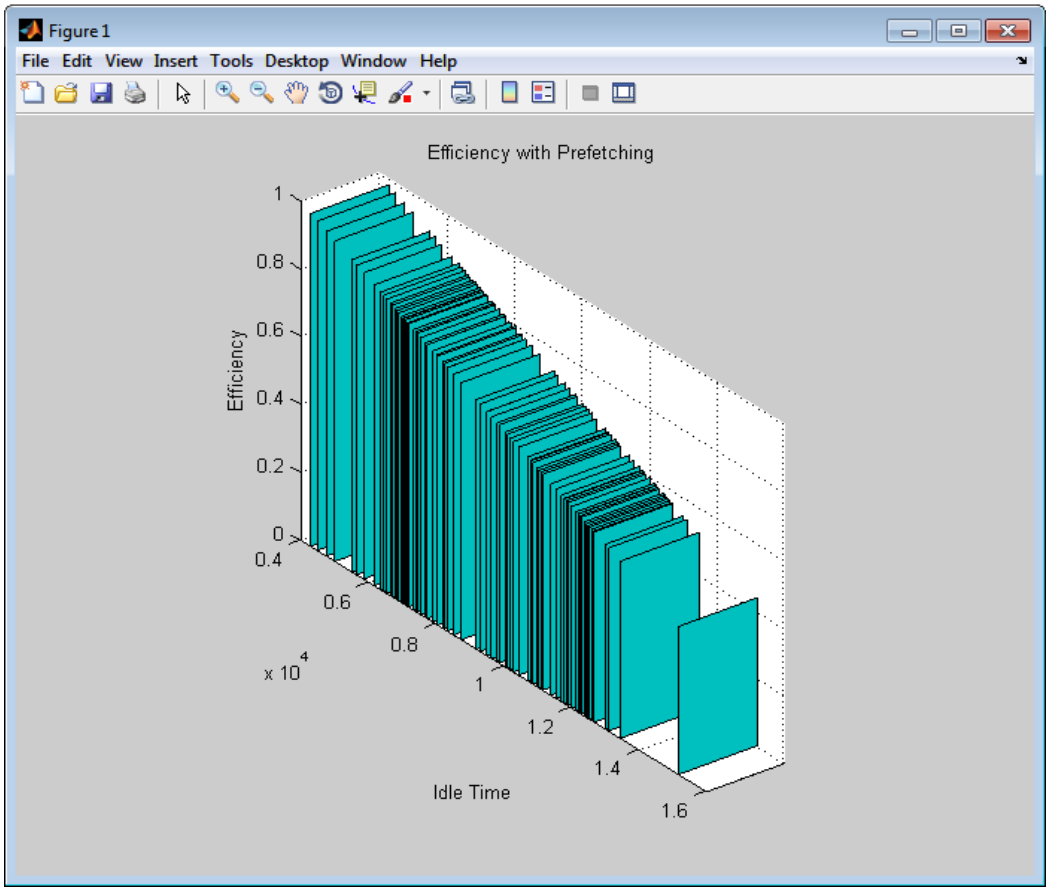

Figure 19. Efficiency with Using Prefetching Model (Idle Time Is in Millisecond)

Therefore, when idle time is 4290 milliseconds and we use 3924 milliseconds of that for prefetching, it means we take more part of idle time for prefetching and the real idle time will be short. So efficiency will be increased as Figure 19 indicates.

In case of maximum idle time, if we prefetch 9 segments, we use 25% of our idle time. So, real idle time will be 11246 milliseconds and consequently efficiency will be less compared to other 99 cases.

All of our values for maximum and minimum busy and idle times and the number of prefetching segments depend on the schedules of our samples. Of course if we generate different schedules these results may change. However, the main part is that

with any different values, which we use as samples we have improvement in efficiency of the system with this prefetching model.

When running the program in Matlab simulator, we generate various schedules, and consequently, we generate different values as total busy and idle times. For example, in some samples the total busy time is less than the total idle time. Note that these values represent just 20 seconds of a system schedule. So during this time, total busy time of the system may be less than total idle in some cases. In this sort of samples, we see that efficiency is low.

If we increase the number of prefetched segments in each sample, we reach to higher efficiency values with prefetching (E_p) and also if we decrease the number of prefetched segments, the efficiency is lower in each sample.

Each sample starts with busy time and also finishes with busy time. We check whether the sum of busy and idle time plus context switching time is equal to 20 seconds (20000 milliseconds). If when data is finished, computation steps for calculating E and E_p will start. Otherwise getting data from user mode will continue.

The number of busy and idle part in each sample are different. As Tables (4-12) show, we have samples with at least 21 of busy/idle times and at most 33 values of a system schedule. Figure 20 plots total busy and total idle times only for the first table (Table 4), For example, first pair of bar charts is for first column (sample) of table 4 and the second pair is for second column of table 4 and so on.

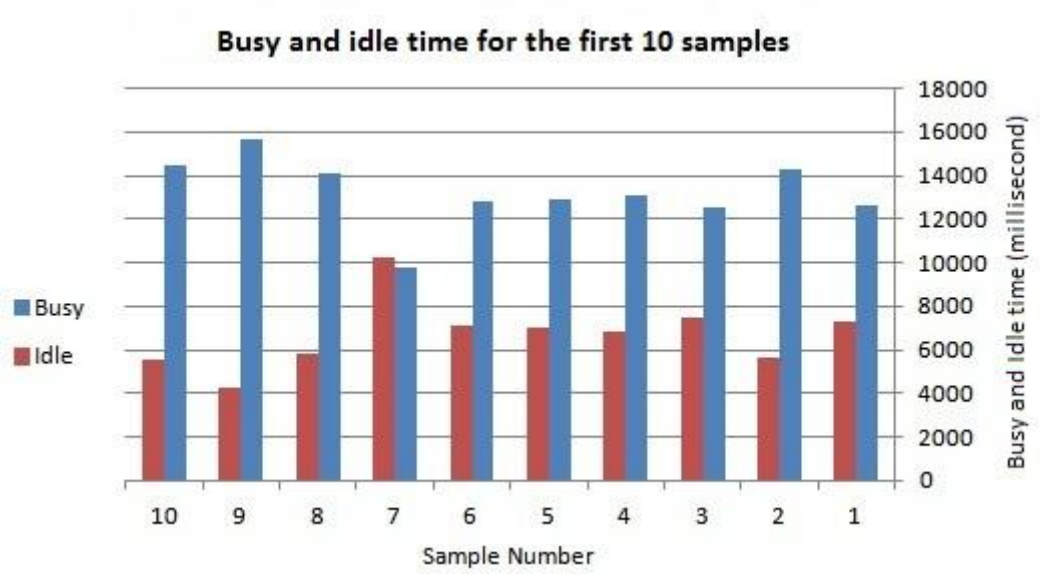

Figure 20. Busy and Idle Time in Millisecond for the First 10 Samples of Table 4

Our model shows that if we assign more idle time for prefetching segments, we can reach higher efficiency values in the system.

Up to this point, we have assumed that the server responds to our prefetch requests immediately, what happens if the server is busy? As discussed in chapter 4, when idle time starts, peer sends a request for prefetching segments to the server. If the server is busy exactly at that time, the peer may find out that its request is rejected by the server. So, the peer should send this request periodically while the idle time continues and the server has not yet accepted segment request from the peer. As soon as the server starts sending packets, the system uses the rest of idle time for prefetching, even though the rest of idle time is not enough for prefetching all requested segments. So in this case, efficiency becomes less because of the idle time which spends for getting respond from the server than efficiency when the server immediately starts sending segments.

All our computations were upon the assumption that the time for prefetching a segment is 0.436 second, but as discussed in chapter 4 this time is calculated with the condition that we have packet loss. Actually in the best case, this time reduces to 0.356 second. In real life, the prefetching time needed for one segment can be some value between these two values.

5.3 Comparison with Similar Studies

Table 3 compares between three other studies and our study in different aspects.

In another study on prefetching [18], continuous-time and discrete-time are the factors. In continuous-time model, video frames can be transmitted continuously across frame periods. We use discrete-time prefetching. [18] Indicates that continuous-time prefetching may decrease the starvation probability and this may lead to better performance compared to discrete-time prefetching.

Table 3. Our Prefetching Model versus Other Prefetching Strategies

Study	Video Slicing	Model	Results
Prefetching Optimization in P2P VoD application with guided seek [16]	Each video divided into several segments	The module of the optimal prefetching takes the segment access probability as input and determines the optimal segments for prefetching and optimal cache replacement policy.	The proposed prefetching scheme optimally determines which segments will be prefetched and cached, based on the segment access probability. The optimized prefetching scheme could minimize the expected seeking delay at each viewing position.
Prefetching with Differentiated Chunk Scheduling [3]	Each video divided into several chunks	Use different queues: "urgent target" and "prefetching target". The segments near to playback position are placed in urgent target. Prefetching queue contains segments with latest play back time. Urgent segments have higher priority than prefetching segments.	Differentiated chunk scheduling mechanism can achieve high peer bandwidth utilization. Using queue-based signaling between peers and the content source server, the amount of workload assigned to a peer is proportional to its available upload capacity, which leads to high bandwidth utilization.
Optimal off-line prefetching algorithm and Heuristic prefetching algorithm [9]	Each video divided into several substreams	The optimal prefetching policy determines how to allocate the bandwidth at time t to the M substreams in order to minimize the average distortion. To implement this prefetching policy, the server peer needs to keep track of the prefetch buffer content and an estimate of the average available bandwidth for a request (heuristic prefetching).	When a substream is lost, the client may have a sufficient "reservoir" for that sub-stream, so that playback continues without any quality degradation.
Our prefetching model	Each video divided into several segments	We calculate prefetching time for one segment, if it takes 'x' time, then we divide our idle time into slices of 'x' times. In our prefetching model, idle time is used for prefetching segments which are unavailable in local buffer and other peers.	We use 100 system schedules and our results show that we have about 20% improvement in performance of the system by using our prefetching model.

Chapter 6

CONCLUSION

Within this dissertation, we discussed how to improve the performance of peer to peer video-on-demand systems by using a prefetching method. This prefetching mechanism will be executed during idle times of the processor. Therefore, idle time of the system will be reduced, and consequently efficiency of the system will be improved. Our simulation studies show this improvement can be about 20%, which is considerably high.

We compute one segment's prefetching time based on some assumptions and conditions like type of the communication network, type of the network, processor clock speed and context switching time. Changes in these attributes will, of course, change the results. For example, if we use a 10 Mbps Ethernet network, then time needed for prefetching a segment may increase.

Moreover, we assume that the peer sends a request for prefetching segments to the server and the server starts sending segments instantly. However, the server may be busy at that moment and may not send segments immediately. It is a controversial issue that can probably happen in our prefetching model, and if we miss our idle time because of the server delays, the efficiency of our system may decrease.

Furthermore, this situation that the server is busy also may happen in the case that we have some lost packets and peer has to send new requests to the server for retransmission them. If the server is busy, prefetching that video segment would take a longer time. The peer should wait until all packets are received correctly and completely from the server. So the server busy status is a subject which we could not discuss comprehensively in this work.

As mentioned before, the type of network was considered LAN in our work. We also assumed the server is on the same LAN. Actually, in many cases, the server will be remote. We can implement our prefetching model upon other types of networks and investigate efficiency of system in each one. For instance, in a wireless LAN network or in mobile cellular systems which use base stations, communication between base stations and peers in the peer to peer network are significant. So, for the future work we can evaluate efficiency of the system with our prefetching model in different type of network with more challenging conditions and limitations than a LAN network.

REFERENCES

[1] H. Sarper, I. Aybay. , "Improving VoD Performance with LAN Client Back-End Buffering", IEE Multimedia, VOL.14, issue: 1, pp: (48-60), 2007.

[2] G. Deng, T. Wei, C. Chen, W. Zhue, B. Wang, D.R. Wu. , "Moderate Prefetching Strategy Based on Video Slicing Mechanism for P2P VoD streaming System", 4th IET International Conference on Wireless, Mobile and Multimedia, 2011.

[3] U. Abbasi, G. Simo, T. Ahmed., "Differentiated Chunk Scheduling for P2P Video-on-Demand System", The 8th Annual IEEE Consumer Communication and Networking Conference, pp: (622-626), 2011.

[4] H. Sampathkumar. "Using Time Division Multiplexing To Support real-time Networking on Ethernet", Master thesis of Computer Science and Engineering, University of Madras, Chennai, 2002.

[5] D.A. Patterson, J. L. Hennessy, Computer Organization and Design, Elsevier, 2005.

[6] J. Summers, T. Brecht, D. Eager, B. Wong., "To Chunk or Not to Chunk: Implications for HTTP Streaming Video Server Performance", Proceedings of ACM NOSSDAV, 2012.

[7] U. Abbasi, T. Ahmed., "COOCHI_G: Cooperative Prefetching Strategy for P2PVideo-on-Demand System", IEEE Consumer Communication and Networking Conference, 2011.

[8] P. Garbacki, D.H.J. Epema, J. Pouwelse, M. Steen, “Offloading Servers with Collaborative Video on Demand”. In Proc of International Workshop on Peer-to-Peer Systems (IPTPS '08) 2008.

[9] Y. Shen, Z. Liu, S. Panwar, K. Ross, Y. Wang, “On the design of prefetching strategies in a peer-driven Video on demand systems”. In Proceedings of IEEE International Conference on Multimedia and Expo, pp: (817-820), 2006.

[10] http://techtips.salon.com/differences-intel-processors-2586.html
(All web address: last visited in May 2013)

[11] http://blog.tsunanet.net/2010/11/how-long-does-it-take-to-make context.html

[12] M. Simmons, Ethernet Theory of Operation, Microchip, Microchip Technology INC, 2008.

[13] http://www.academia.edu/1908921/An_Experiment_to_Determine_and_Compare_Practical_Efficiency_of_Insertion_Sort_Merge_Sort_and_Quick_Sort_Algorithms

[14] http://warp.povusers.org/SortComparison/integers.html

[15] http://www.cs.uml.edu/~pkien/sorting/

[16] Y. He, L. Guan, "Prefetching Optimization in P2P VoD Applications", First International Conference on Advances in Multimedia, pp: (110-115), 2009.

[17] http://www.techterms.com/definition/clockspeed

[18] S. Oh, B. Kulapala, A.W. Richa, and Martin Reisslein, "Continuous-Time Collaborative Prefetching of Continuous Media", IEEE Transactions on Broadcasting, VOL. 54, pp: (36-52), 2008.

[19] R. Diwan, S. Thakur, "Role of Data Link Layer in OSI", International Research Journal, VOL. 1, issue: 7, pp: (24-26), 2010.

[20] J. Jasperneite, J. Imtiaz, M. Schumacher, K. Weber. "A Proposal for a Generic Real-Time Ethernet System", IEEE Tranactions on Industrial Informatics, VOL.5, issue: 2, pp: (75-85), 2009.

APPENDIX

Table 4. Samples of Different System Schedules

	1	2	3	4	5	6	7	8	9	10
1	400	500	1000	650	1000	500	800	700	600	600
2	262	450	300	700	500	600	300	240	400	500
3	800	1000	700	1250	700	1000	600	800	500	500
4	500	130	500	500	300	300	500	600	200	236
5	1000	800	900	500	600	700	700	400	1000	700
6	600	500	600	700	600	400	700	500	300	400
7	900	3000	2000	2000	500	900	1000	800	700	800
8	800	900	500	300	200	500	200	500	500	600
9	500	750	700	650	500	2000	500	1000	900	900
10	300	600	1000	950	500	600	500	700	600	500
11	2500	2500	2500	300	2000	600	700	900	2000	1000
12	400	500	200	400	400	240	400	500	500	300
13	1350	920	500	200	700	800	800	800	700	800
14	658	580	400	500	900	400	100	500	1000	400
15	750	3500	1000	600	800	1500	900	1300	2500	500
16	1000	100	800	800	300	500	700	600	200	600
17	2250	700	900	1000	500	700	2000	700	500	700
18	650	30	152	350	450	300	500	500	400	800
19	3000	1500	2000	550	700	800	400	700	1000	800
20	340	500	600	800	500	600	300	400	800	500
21	1000	500	400	500	450	900	900	900	900	600
22			600	1500	500	100	600	300	152	400
23			500	100	700	500	700	600	2000	1500
24			200	600	100	400	500	500	600	700
25			1000	600	800	700	1000	1000	1000	800
26				900	636	600	400	800		500
27				200	1100	400	900	500		600
28				800	400	1000	500	500		400
29				400	900	400	700	600		900
30				400	500	500	640	300		200
31				240	700	500	500	800		500
32					300					300
33					200					400
Total Busy time	14450	15670	14100	9740	12850	12900	13100	12500	14300	12600
Total Idle time	5510	4290	5852	10200	7086	7040	6840	7440	5652	7336
E	0.7225	0.7835	0.705	0.487	0.6425	0.645	0.655	0.625	0.715	0.63
E_p	0.9187	0.9797	0.9012	0.6834	0.8387	0.8412	0.8512	0.8212	0.9112	0.8262

Table 5. Samples of Different System Schedules

	1	2	3	4	5	6	7	8	9	10
1	500	900	400	30	400	240	600	2250	600	500
2	800	200	600	1500	900	650	700	650	100	1000
3	300	800	500	500	500	700	800	3000	500	800
4	600	400	500	500	700	1250	700	340	500	500
5	500	240	236	450	640	500	800	1000	600	600
6	700	650	700	1000	500	500	800	400	436	300
7	700	700	400	130	800	700	800	262	700	800
8	1000	1250	800	800	300	2000	500	800	500	700
9	200	500	600	500	600	300	1000	500	200	240
10	500	500	900	3000	500	650	240	1000	900	800
11	500	700	500	900	700	950	900	600	400	600
12	700	2000	1000	750	700	300	500	900	600	400
13	400	300	300	600	1000	400	800	800	500	500
14	800	650	800	2500	200	200	500	500	1000	800
15	100	950	400	500	500	500	500	300	500	500
16	900	300	500	920	500	600	400	2500	700	1000
17	700	400	600	580	700	800	700	400	300	700
18	2000	200	700	3500	400	1000	300	1350	800	900
19	500	500	800	100	800	350	700	658	500	500
20	400	600	800	700	100	550	600	750	800	800
21	300	800	500	500	900	800	900	1000	600	500
22	900	1000	600		700	500	300		1800	1300
23	600	350	400		2000	1500	600		600	600
24	700	550	1500		500	100	500		700	700
25	500	800	700		400	600	1000		400	500
26	1000	500	800		300	600	500		900	700
27	400	1500	500		900	900	500		500	400
28	900	100	600		600	200	500		800	900
29	500	600	400		700	800	600		400	300
30	700	400	900		500	400	1300		500	600
31	640	600	200		1000	400	400		400	500
32			500						750	
33			300						450	
Total Busy time	7340	10640	7736	4790	12540	10440	11600	10770	8150	8540
Total Idle time	12600	9300	12200	15170	7400	9500	8340	9190	11786	11400
E	0.367	0.532	0.3868	0.2395	0.627	0.522	0.58	0.5385	0.4075	0.427
E_p	0.5632	0.7282	0.583	0.4357	0.8232	0.7182	0.7762	0.7347	0.6037	0.6232

Table 6. Samples of Different System Schedules

	1	2	3	4	5	6	7	8	9	10
1	700	700	700	800	1350	200	400	600	1100	900
2	30	300	600	400	658	1000	500	500	400	400
3	1500	200	400	400	750	1000	500	200	900	600
4	500	1000	1000	240	1000	300	500	1000	500	500
5	500	500	400	650	2250	700	600	300	700	1000
6	450	700	500	700	650	500	1000	700	300	500
7	1000	300	600	1250	3000	900	300	500	200	700
8	130	600	1000	500	340	600	700	900	1000	300
9	800	600	300	500	1000	2000	400	600	500	800
10	500	500	700	700	400	500	900	2000	700	500
11	3000	200	400	2000	262	700	500	500	300	800
12	900	500	900	300	800	1000	2000	700	600	600
13	750	500	500	650	500	2500	600	1000	600	1800
14	600	2000	2000	950	1000	200	600	2500	500	600
15	2500	400	600	300	600	500	240	200	200	700
16	500	700	600	400	900	400	800	500	500	400
17	920	900	240	200	800	1000	400	400	500	900
18	580	800	800	500	500	800	1500	1000	2000	500
19	3500	300	400	600	300	900	500	800	400	800
20	100	500	1500	800	2500	152	700	900	700	400
21	500	450	500	1000	400	2000	300	152	900	500
22		700	700	350		600	800	2000	800	400
23		500	300	550		400	600	600	300	750
24		450	800	800		600	900	400	500	450
25		500	600	500		500	100	1000	450	600
26		700	900	1500			500		700	100
27		100	100	100			400		500	500
28		800	500	600			700		450	500
29		636	400	600			600		500	600
30		1100	500	900			400		700	436
31		400	500	200			1000		100	700
32		900							800	500
33		500							636	200
Total Busy time	15670	7686	6940	10300	11212	13300	7440	6852	8786	12850
Total Idle time	4290	12250	13000	9640	8748	6652	12500	13100	11150	7086
E	0.7835	0.3843	0.347	0.515	0.5606	0.665	0.372	0.3426	0.4393	0.6425
E_p	0.9797	0.5805	0.5432	0.7112	0.7568	0.8612	0.5682	0.5388	0.6355	0.8387

Table 7. Samples of Different System Schedules

	1	2	3	4	5	6	7	8	9	10
1	500	600	800	800	800	1000	500	500	152	700
2	600	436	500	700	636	400	200	700	2000	500
3	400	700	600	240	1100	262	900	640	600	200
4	900	500	400	800	400	800	400	500	400	900
5	200	200	900	600	900	500	600	800	600	400
6	500	900	200	400	500	1000	500	300	500	600
7	300	400	500	500	700	600	1000	600	200	500
8	400	600	300	800	300	900	500	500	1000	1000
9	600	500	400	500	200	800	700	700	1000	500
10	500	1000	600	1000	1000	500	300	700	300	700
11	500	500	500	700	500	300	800	1000	700	300
12	236	700	500	900	700	2500	500	200	500	800
13	700	300	236	500	300	400	800	500	900	500
14	400	800	700	800	600	1350	600	500	600	800
15	800	500	400	500	600	658	1800	700	2000	600
16	600	800	800	1300	500	750	600	400	500	1800
17	900	600	600	600	200	1000	700	800	700	600
18	500	1800	900	700	500	2250	400	100	1000	700
19	1000	600	500	500	500	650	900	900	2500	400
20	300	700	1000	700	2000	3000	500	700	200	900
21	800	400	300	400	400	340	800	2000	500	500
22	400	900	800	900	700		400	500	400	800
23	500	500	400	300	900		500	400	1000	400
24	600	800	500	600	800		400	300	800	500
25	700	400	600	500	300		750	900	900	400
26	800	500	700	1000	500		450	600		750
27	800	400	800	800	450		600	700		450
28	500	750	800	500	700		100	500		600
29	600	450	500	500	500		500	1000		100
30	400	600	600	600	450		500	400		500
31	1500	100	400	300	500		600	900		500
32	700	500	1500		700		436			600
33	800	500	700		100		700			436
Total Busy time	11600	7650	9136	8240	8950	6510	13150	13040	11752	7486
Total Idle time	8336	12286	10800	11700	10986	13450	6786	6900	8200	12450
E	0.58	0.3825	0.4568	0.412	0.4475	0.3255	0.6575	0.652	0.5876	0.3743
E_p	0.7762	0.5787	0.653	0.6082	0.6437	0.5217	0.8537	0.8482	0.7838	0.5705

Table 8. Samples of Different System Schedules

	1	2	3	4	5	6	7	8	9	10
1	600	600	3500	400	500	600	1000	436	600	1000
2	300	400	100	700	500	900	400	700	900	400
3	800	900	700	600	500	200	500	500	200	900
4	700	200	30	400	600	800	500	200	800	500
5	240	500	1500	1000	1000	400	500	900	400	700
6	800	300	500	400	300	240	600	400	240	640
7	600	400	500	500	700	650	1000	600	650	500
8	400	600	450	500	400	700	300	500	700	800
9	500	500	130	600	900	1250	700	1000	1250	300
10	800	500	1000	1000	500	500	400	500	500	600
11	500	236	800	300	2000	500	900	700	500	500
12	1000	700	500	700	600	700	500	300	700	700
13	700	400	3000	400	600	2000	2000	800	2000	700
14	900	800	750	900	240	300	600	500	300	1000
15	500	600	900	500	800	650	600	800	650	200
16	800	900	600	2000	400	950	240	600	950	500
17	500	500	2500	600	1500	300	800	1800	300	500
18	1300	1000	500	600	500	400	400	600	400	700
19	600	300	920	240	700	200	1500	700	200	400
20	700	800	580	800	300	500	500	400	500	800
21	500	400	500	400	800	600	700	900	600	100
22	700	500		1500	600	800	300	500	800	900
23	400	600		500	900	1000	800	800	1000	700
24	900	700		700	100	350	600	400	350	2000
25	300	800		300	500	550	900	500	550	500
26	600	800		800	400	800	100	400	800	400
27	500	500		600	700	500	500	750	500	300
28	1000	600		900	600	1500	400	450	1500	900
29	800	400		100	400	100	700	600	100	600
30	500	1500		500	1000	600	600	100	600	700
31	500	700		500	400	400	400	500	400	500
32		800						500		
33		500						600		
Total Busy time	8540	8836	14950	7540	12900	9900	13500	12886	9900	8400
Total Idle time	11400	11100	5010	12400	7040	10040	6440	7050	10040	11540
E	0.427	0.4418	0.7475	0.377	0.645	0.495	0.675	0.6443	0.495	0.42
E_p	0.6232	0.638	0.9437	0.5732	0.8412	0.6912	0.8712	0.8405	0.6912	0.6162

Table 9. Samples of Different System Schedules

	1	2	3	4	5	6	7	8	9	10
1	400	1500	500	650	500	500	200	700	500	400
2	900	500	600	3000	500	500	500	640	500	700
3	200	500	436	340	450	600	300	500	500	600
4	500	450	700	1000	1000	300	400	800	450	400
5	300	1000	500	400	800	800	600	300	1000	1000
6	400	130	200	262	130	700	500	600	130	400
7	600	800	900	800	500	240	500	500	800	500
8	500	500	400	500	3000	800	236	700	500	500
9	500	3000	600	1000	900	600	700	700	3000	500
10	236	900	500	600	750	400	400	1000	900	600
11	700	750	1000	900	600	500	800	200	750	1000
12	400	600	500	800	2500	800	600	500	600	300
13	800	2500	700	500	500	500	900	500	2500	700
14	600	500	300	300	100	1000	500	700	500	400
15	900	920	800	2500	580	700	1000	400	920	900
16	500	580	500	400	3500	900	300	800	580	500
17	1000	3500	800	1350	920	500	800	100	3500	2000
18	300	100	600	658	700	800	400	900	100	600
19	800	700	1800	750	30	500	500	700	700	600
20	400	500	600	1000	500	1300	600	2000	30	240
21	500	30	700	2250	1500	600	700	500	1500	800
22	600		400			700	800	400		400
23	700		900			500	800	300		1500
24	800		500			700	500	900		500
25	800		800			400	600	600		700
26	500		400			900	400	700		300
27	600		500			1000	1500	500		800
28	400		400			600	700	1000		600
29	1500		750			500	800	400		900
30	700		450			300	500	900		100
31	800		600			800	600	500		500
32	500		100				400			
33	600		500				900			
Total Busy time	11700	15200	12786	11440	7280	9240	12200	7400	15670	13400
Total Idle time	8236	4760	7150	8520	12680	10700	7736	12540	4290	6540
E	0.585	0.9562	0.6393	0.572	0.364	0.462	0.61	0.37	0.7835	0.67
E_p	0.7812	0.9562	0.8355	0.7682	0.5602	0.6582	0.8062	0.5662	0.9797	0.8662

Table 10. Samples of Different System Schedules

	1	2	3	4	5	6	7	8	9	10
1	400	100	500	500	100	500	340	500	500	600
2	240	700	400	600	500	1000	1000	500	500	500
3	650	30	700	300	500	400	400	600	600	1000
4	700	1500	600	800	600	900	262	436	300	800
5	1250	500	400	700	436	500	800	700	800	500
6	500	500	1000	240	700	700	500	500	700	600
7	500	500	400	800	500	640	1000	200	240	300
8	700	450	500	600	200	500	600	900	800	800
9	2000	1000	500	400	900	800	900	400	600	700
10	300	130	600	500	400	300	800	600	400	240
11	400	800	1000	800	600	600	500	500	500	800
12	950	500	300	500	500	500	300	1000	800	600
13	300	3000	700	1000	1000	700	2500	500	500	400
14	650	900	400	700	500	700	400	700	1000	500
15	200	750	900	900	700	1000	1350	300	700	800
16	500	600	500	500	300	200	658	800	900	500
17	600	2500	2000	800	800	500	750	500	500	1000
18	800	500	600	500	500	500	1000	800	800	700
19	1000	920	600	1300	800	700	2250	600	500	900
20	350	580	240	600	600	400	650	1800	1300	500
21	550	3500	800	700	1800	800	3000	600	600	800
22	800		400	500	600	100		700	700	500
23	500		1500	700	700	900		400	500	1300
24	1500		500	400	400	700		900	700	600
25	100		700	900	900	2000		500	400	700
26	600		300	300	500	500		800	900	500
27	600		800	600	800	400		400	300	700
28	900		600	500	400	300		500	600	400
29	200		900	1000	500	900		400	500	900
30	800		100	800	400	600		750	1000	300
31	400		500	500	750	700		450	800	500
32					450			600		
33					600			100		
Total Busy time	9650	13600	12900	11900	12386	12040	13790	7650	8540	11900
Total Idle time	10290	6360	7040	8040	7550	7900	6170	12286	11400	8040
E	0.4825	0.68	0.645	0.595	0.6193	0.602	0.6895	0.3825	0.427	0.595
E_p	0.6787	0.8762	0.8412	0.7912	0.8155	0.7982	0.8857	0.5787	0.6232	0.7912

Table 11. Samples of Different System Schedules

	1	2	3	4	5	6	7	8	9	10
1	500	600	500	600	3000	640	2000	580	900	500
2	700	500	300	400	340	500	600	3500	200	800
3	300	1000	400	1000	1000	800	400	100	500	300
4	200	800	600	400	400	300	500	700	300	600
5	1000	500	500	500	262	600	200	3000	400	500
6	500	500	500	500	800	500	1000	1500	600	700
7	700	600	236	600	500	700	300	500	500	700
8	300	300	700	1000	1000	700	700	500	500	1000
9	600	800	400	300	600	1000	500	450	236	200
10	600	700	800	700	900	200	900	1000	700	500
11	500	240	600	400	800	500	600	130	400	500
12	200	800	900	900	500	500	2000	800	800	700
13	500	600	500	500	300	700	500	500	600	400
14	500	400	1000	2000	2500	400	700	30	900	800
15	2000	500	300	600	400	800	1000	900	500	2000
16	400	800	800	600	1350	100	2500	750	300	900
17	700	500	400	240	658	900	200	600	1000	700
18	900	1000	500	800	750	700	500	2500	800	100
19	800	700	600	400	1000	2000	400	500	400	500
20	300	900	700	1500	2250	500	1000	920	500	400
21	500	500	800	500	650	400	800	500	600	300
22	450	800	800	700		300	900		700	900
23	700	500	500	300		900	152		800	600
24	500	300	600	800		600	600		800	700
25	450	600	400	600		700	1000		500	500
26	500	700	1500	900		500			600	1000
27	700	500	700	100		1000			1500	900
28	100	700	800	500		400			400	400
29	800	400	500	400		900			700	500
30	636	900	600	700		500			800	700
31	1100	1300	400	500		700			500	640
32	400		900						600	
33	900		200						400	
Total Busy time	12750	9840	7936	7540	9170	13240	8052	7760	10436	9740
Total Idle time	7186	10100	12000	12400	10790	6700	11900	12200	9500	10200
E	0.6375	0.492	0.3968	0.377	0.4585	0.662	0.4026	0.388	0.5218	0.487
E_p	0.8337	0.6882	0.593	0.5732	0.6547	0.8582	0.5988	0.5842	0.718	0.6832

Table 12. Samples of Different System Schedules

	1	2	3	4	5	6	7	8	9	10
1	300	900	658	800	300	900	100	200	700	400
2	800	500	750	500	400	500	600	1000	500	900
3	700	700	1000	500	600	700	900	500	1000	500
4	240	640	2250	600	500	300	200	700	400	700
5	800	500	650	300	500	200	800	300	900	300
6	600	800	3000	800	236	1000	400	600	500	200
7	400	300	340	700	700	500	240	600	700	1000
8	500	600	1000	240	400	700	650	500	640	500
9	800	500	400	800	800	300	700	200	500	700
10	500	700	262	600	600	600	1250	500	800	300
11	1000	700	800	400	900	600	500	500	300	600
12	700	1000	500	500	500	500	500	400	600	600
13	900	200	1000	800	1000	200	700	2000	500	500
14	500	500	600	500	300	500	2000	700	700	200
15	800	500	900	1000	800	500	300	900	700	500
16	500	700	800	700	400	2000	650	800	100	500
17	1300	400	500	900	500	400	950	300	200	2000
18	600	800	300	500	600	700	300	500	500	400
19	700	100	2500	800	700	900	400	450	500	700
20	500	900	400	500	800	800	200	700	700	900
21	700	700	1350	1300	800	300	500	500	400	800
22	400	2000		600	500	500	600	450	800	300
23	900	500		700	600	450	800	500	1000	500
24	300	400		500	400	700	1000	700	900	450
25	600	300		700	1500	500	350	100	700	700
26	500	900		400	700	450	550	800	2000	500
27	1000	600		900	800	500	800	636	500	450
28	800	700		300	500	700	500	1100	400	500
29	500	500		600	600	100	1500	400	300	700
30	500	1000		500	400	800	400	900	900	100
31	600	400		1000	900	636	600	500	600	800
32					200	1100		700		636
33					500	400		300		1100
Total Busy time	12000	7800	10098	12200	12500	8086	10140	8886	9500	12250
Total Idle time	7940	12140	9862	7740	7436	11850	9800	11050	10440	7686
E	0.6	0.39	0.5049	0.61	0.625	0.4043	0.507	0.4443	0.475	0.6125
E_p	0.7962	0.5862	0.7011	0.8062	0.8212	0.6005	0.7032	0.6405	0.6712	0.8087

Table 13. Samples of Different System Schedules

	1	2	3	4	5	6	7	8	9	10
1	500	900	1500	1000	400	200	1350	200	400	300
2	500	152	500	1000	400	800	658	900	600	200
3	600	2000	500	300	1000	400	750	400	500	1000
4	1000	600	450	700	500	400	1000	600	200	500
5	300	400	1000	500	500	240	2250	500	1000	700
6	700	600	130	900	500	650	650	1000	300	300
7	400	500	800	600	600	700	3000	500	700	600
8	900	200	500	2000	300	1250	340	700	500	600
9	500	1000	3000	500	1000	500	1000	300	900	500
10	2000	1000	900	700	700	500	400	800	600	200
11	600	300	750	1000	400	700	262	500	2000	500
12	600	700	600	2500	900	2000	800	800	500	500
13	240	500	2500	200	500	300	500	600	700	2000
14	800	900	500	500	2000	650	1000	1800	1000	400
15	400	600	920	400	600	950	600	600	2500	700
16	1500	2000	580	1000	600	300	900	700	200	900
17	500	500	3500	800	240	400	800	400	500	800
18	700	700	100	900	800	200	500	900	400	300
19	300	1000	500	152	400	500	300	500	1000	500
20	800	2500	230	2000	1500	600	2500	800	800	450
21	600	200	500	600	500	800	400	400	900	700
22	900	500		400	700	1000		500	152	500
23	100	400		600	300	350		400	2000	450
24	500	1000		500	800	550		750	600	500
25	400	800		200	600	800		450	1000	700
26	700				900	500		600		100
27	600				100	1500		100		800
28	400				500	100		500		636
29	1000				400	600		500		1100
30	400				700	600		600		400
31	500				600	900		436		900
32								700		500
33								500		700
Total Busy time	7540	9100	15470	6852	8140	9840	11212	7286	14100	12950
Total Idle time	12400	10852	4490	13100	11800	10100	8748	12650	5852	6986
E	0.377	0.455	0.7735	0.3426	0.407	0.492	0.5606	0.3643	0.705	0.6475
E_p	0.5732	0.6512	0.9697	0.5388	0.6032	0.6882	0.7568	0.5605	0.9012	0.8437