COST AWARE TCP SCHEDULER FOR BANDWIDTH AGGREGATION

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF INFORMATICS OF
THE MIDDLE EAST TECHNICAL UNIVERSITY

BY

SERHAT AĞIRBAŞ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
IN
INFORMATION SYSTEMS

JUNE 2018

COST AWARE TCP SCHEDULER FOR BANDWIDTH AGGREGATION

Submitted by **Serhat Ağırbaş** in partial fulfillment of the requirements for the degree of **Master of Science in Information Systems, Middle East Technical University** by,

Prof. Dr. Deniz Zeyrek Bozşahin
Dean, **Graduate School of Informatics** _____

Prof. Dr. Yasemin Yardımcı Çetin
Head of Department, **Information Systems** _____

Assoc. Prof. Dr. Altan Koçyiğit
Supervisor, **Information Systems, METU** _____

Dr. Cüneyt Sevgi
Co-Supervisor, **Computer Tech. & Inf. Systems, Bilkent University** _____

Examining Committee Members:

Assoc. Prof. Dr. Aysu Betin Can
Information Systems, METU _____

Assoc. Prof. Dr. Altan Koçyiğit
Information Systems, METU _____

Assist. Prof. Dr. Bilgin Avenoğlu
Computer Engineering, TED University _____

Assoc. Prof. Dr. Pekin Erhan Eren
Information Systems, METU _____

Assoc. Prof. Dr. Oumout Chouseinoglou
Industrial Engineering, Hacettepe University _____

Date: **11.06.2018**

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name : Serhat Ağırbaş

Signature : _____

ABSTRACT

COST AWARE TCP SCHEDULER FOR BANDWIDTH AGGREGATION

Ağırbaş, Serhat

MSc, Department of Information Systems

Supervisor: Assoc. Prof. Dr. Altan Koçyiğit

Co-Supervisor: Dr. Cüneyt Sevgi

June 2018, 83 pages

Constant bit rate, time sensitive data delivery is needed for many network applications and these applications usually require high throughput and less variable delay. When such applications run on mobile devices, the bandwidth available and the other characteristics of the primary network connection may not be sufficient to provide necessary quality of service. On the other hand, most of the mobile devices are multihomed that is they are equipped with more than one network interface, hence they can be connected to more than one network simultaneously. Therefore, the bandwidth aggregation is a viable option for better quality of service in such cases. This thesis tackles constant bit rate data delivery problem by utilizing multiple network connections to satisfy bandwidth requirements in a cost effective manner specifically for time sensitive file streaming applications. The proposed method, called Cost Aware TCP Scheduler (CATS), aggregates the resources of two network connections which are available on the client device in a cost aware manner to deliver enough bandwidth for streaming applications. One of the connections is called the free connection and it is considered to have fluctuating throughput and variable delay but no monetary cost of use. The second connection is the paid connection that provides higher throughput and less variable delay but there is monetary cost associated with data transfer. CATS schedules data transfer over these connections to ensure timely delivery of data while minimizing the data transfer cost. Experimental results show that CATS maximizes the utilization of the free connection and minimizes the utilization of the paid connection thereby reducing the total monetary cost without causing significant quality degradation.

Keywords: Bandwidth Aggregation, Cost Aware Scheduler, Time Sensitive Data Transfer

ÖZ

BANT GENİŞLİĞİ BİRLEŞTİRME İÇİN MALİYET FARKINDALIKLI TCP ZAMANLAYICI

Ağırbaş, Serhat

Yüksek Lisans, Bilişim Sistemleri Bölümü

Tez Yöneticisi: Doç. Dr. Altan Koçyiğit

Ortak Tez Yöneticisi: Dr. Cüneyt Sevgi

Haziran 2018, 83 sayfa

Birçok ağ uygulaması zamana duyarlı sabit bit hızı aktarımına ihtiyaç duymaktadır ve bu uygulamalar genellikle yüksek veri ve daha az değişken gecikme gerektirmektedir. Bu tür uygulamalar mobil cihazlarda çalıştırıldığında, birincil ağ bağlantısının mevcut bant genişliği ve diğer özellikleri gerekli hizmet kalitesini sağlamak için yeterli olmayabilir. Diğer yandan, mobil cihazların çoğu birden fazla ağ arayüzü ile donatılmışlardır ve aynı anda birden fazla ağa bağlanabilirler. Bu nedenle, birden fazla ağın bant genişliğinin birleştirilmesi, bu gibi durumlarda daha kaliteli hizmet alınması için uygulanabilir bir seçenektir. Bu tez, özellikle zamana duyarlı dosya transferi uygulamaları için bant genişliği gereksinimlerini uygun maliyetli bir şekilde karşılamak üzere çoklu ağ bağlantılarını kullanarak sabit bit hızı veri aktarımı problemini ele almaktadır. Cost Aware TCP Scheduler (CATS) olarak adlandırılan yöntem, kullanıcı cihazında bulunan iki ağ bağlantısının kaynaklarını, duraksız aktarım uygulamaları için yeterli bant genişliği sağlamak amacıyla maliyeti gözeterek birleştirir. Bağlantılardan biri ücretsiz bağlantı olarak adlandırılır ve bu bağlantının dalgalanan veri hacmine ve değişken gecikmeye sahip olduğu kabul edilir, ancak parasal kullanım maliyeti yoktur. İkinci bağlantı, daha yüksek veri hacmi ve daha az değişken gecikme sağlayan ücretli bağlantıdır, ancak bu bağlantının veri aktarımıyla ilişkili parasal maliyet vardır. CATS, veri aktarım maliyetini en aza indirirken, verilerin zamanında gönderilmesini sağlamak için bu bağlantılar üzerinden veri aktarımını planlamaktadır. Deneysel sonuçlar, CATS'in ücretsiz bağlantı kullanımını en üst düzeye çıkardığını ve ücretli bağlantı kullanımını en aza indirdiğini ve böylece belirgin bir kalite bozukluğuna yol açmadan toplam parasal maliyeti düşürdüğünü göstermektedir.

Anahtar Kelimeler: Bant Genişliği Birleşimi, Maliyet Farkındalıklı Zamanlayıcı, Zamana Duyarlı Veri Transferi

To My Family…

ACKNOWLEDGMENTS

First of all, I would like to express my gratitude to my supervisors Assoc. Prof. Dr. Altan Koçyiğit and Dr. Cüneyt Sevgi for their valuable guidance, support and patience throughout this study.

I am also grateful to Yasemin Öktem for her sincere support and assistance while preparing this thesis study.

Finally, I would like to thank my family for their love, support and encouragement.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

CATS	Cost Aware TCP Scheduler
FL	Free Link
FC	Free Connection
PL	Paid Link
PC	Paid Connection
MPTCP	Multipath TCP
SCTP	Stream Control Transmission Protocol
MCTT	Measured Chunk Transfer Time
CTT	Chunk Transfer Time
RTT	Round Trip Time
SCTT	Smoothed Chunk Transfer Time
TEMCTT	Time Elapsed Measured Chunk Transfer Time
USCTT	Updated Smoothed Chunk Transfer Time
FCTT	Free Connection TCP Throughput
LT	Wall Clock Time of First Chunk Downloaded Over Free Connection
FT	Wall Clock Time of Last Chunk Downloaded Over Free Connection
CC	Chunk Count Downloaded Over Free Connection
MBS	Mbit Size for Each Chunk
PCSCTT	Paid Connection Smoothed Chunk Transfer Time
FCSCTT	Free Connection Smoothed Chunk Transfer Time
CUR	Chunk Usage Rate
MTT	Measured TCP Throughput
BLEST	Blocking Estimation
HOL	Head of Line
ESPA	Energy Usage Performance Aware

CHAPTER 1

INTRODUCTION

1.1. Motivation

Demand for higher Internet bandwidth is on the rise globally. Nielsen's Law of Internet Bandwidth [1] states that a user's connection speed grows 50% each year. Particularly with the introduction of mobile devices, people are connected to the Internet at any time of their daily lives. Moreover, emerging technologies like Internet of Things (IOT) come up with the result that more and more devices connect to the Internet and require more bandwidth every passing day. In addition to advances in networking, increasing processing power enables clients to employ more sophisticated applications. High quality video streaming, social networking, online gaming and many other have become common and widely used services.

Improvement of Internet connectivity is not always sufficient to provide adequate throughput for some of the client applications. Applications like electronic e-mail and web browsing are elastic which means they can tolerate delay, loss and bandwidth fluctuations. For example, consider web, even if throughput of underlying connection is low, user just experiences high latency but still the main functionality will be provided. On the other hand, some applications such as voice over IP, video conference over IP and video streaming are sensitive to bandwidth and delay variations and they require guaranteed bandwidth and low delay variation for proper operation. For example, consider stored video streaming which is a type of media streaming in which the data from a video file is continuously delivered via the Internet to a remote user. In this scenario, if the throughput of underlying connection is lower than the video playout rate, client continuously experiences video pauses and this deteriorates user experience. Thus, in such scenarios high throughput data delivery is important.

Many methods have been developed for enhancing available throughput for client applications. One of these methods is bandwidth aggregation which corresponds to

utilization of more than one access network for one client application. Today, most of the mobile devices are equipped with more than one network interface. Hence, bandwidth aggregation is possible by turning on these multiple interfaces and transferring data via connections established on multiple access networks. TCP pooling is one of the most widely employed techniques for bandwidth aggregation. In TCP pooling, multiple heterogeneous TCP connections are combined to provide more bandwidth for client applications. Transmission Control Protocol (TCP) [2] has been commonly used as a reliable communication protocol between hosts for more than 40 years for point to point data transfer over a single connection. A recent proposal called Multipath TCP (MPTCP) [3] extends TCP to utilize multiple networks for a single connection. MPTCP is backward compatible with TCP and it maintains separate data paths via multiple network interfaces and combines these data paths to one logical connection for one application. MPTCP is commonly associated with the multihomed devices such as mobile phones that have both Wi-Fi and cellular network interfaces. The scheduler of MPTCP decides which file segments will be delivered over which network interface. Therefore, employed scheduling algorithm of MPTCP determines the utilization ratio of each network interface.

Various TCP pooling methods and MPTCP schedulers have been developed to obtain maximum throughput to improve client experience. However, most of these methods do not take into account monetary cost associated with the utilization of access networks. Usually, cellular interfaces provide lower delay, higher throughput and lower packet loss rate when compared to Wi-Fi network interfaces of mobile devices. However, cellular networks may have large monetary cost for each byte transferred while Wi-Fi network interfaces are free. Therefore, the scheduler may also take into account the monetary cost associated with data transfer in addition to providing higher quality of service. This thesis investigates the use of multiple interfaces to minimize monetary cost while not causing performance degradation. The main goal is to provide sufficient throughput to client application with minimum monetary cost possibly by the maximum utilization of free connections.

1.2.Objective and Scope

The purpose of this thesis study is to provide a bandwidth aggregation method for transfer of constant bit rate time sensitive data over two heterogeneous access networks called the free link and the paid link. Free connection established over the free link is considered to have fluctuating throughput and variable delay, while paid connection established over the paid link considered having higher throughput and less variable delay. To provide enough throughput rather than maximizing throughput is the primary approach employed to minimize paid link costs. This necessitates the maximization of free utilization while minimizing the paid link usage.

CATS is compatible with the applications involving time sensitive data transfer like stored video file streaming. However, CATS can be extended to support applications such as videoconferencing and teleconferencing over IP.

In order to exemplify the problem addressed by CATS, consider a scenario in which a stored video file is to be played out by a client. In this scenario a server streams the stored video file and a client plays out the video chunks as they are delivered to it and the playout rate is 5Mbit/s. The client also has two network interfaces; one of them is Wi-Fi interface which has 3Mbit/s throughput while the other one is 4G/LTE interface which has 10Mbit/s throughput. In this scenario, free link corresponds to Wi-Fi, while paid link corresponds to 4G/LTE. Possible options to transfer video file to the client and associated problems can be listed as;

1) Client tries to stream complete video file over Wi-Fi network interface but the Wi-Fi throughput isn't enough for video playout rate, client experiences continuous pauses.
2) Client streams complete video file over 4G/LTE interface and experiences a good video streaming, but because of complete streaming done over 4G/LTE interface, monetary cost will be high.
3) It is possible to aggregate the bandwidth of multiple interfaces for this existing method. So the client may have a total of 13Mbit/s throughput even if only 5Mbit/s is enough. Therefore, unnecessarily large cellular cost might be caused by over utilization of 4G/LTE network and underutilization of Wi-Fi network.

In this scenario, CATS system aims to stream the complete video file over both interfaces but tries to minimize the use of 4G/LTE network. That is, the video file chunks are downloaded over 4G/LTE network only when Wi-Fi network is not enough. In this example, video file has 5Mbit/s playout rate and Wi-Fi throughput is 3Mbit/s. So, only the difference, 2Mbit/s is aimed to be downloaded over 4G/LTE network. More importantly, while doing this, the pauses in the playout is minimized. That is, quality of service is preserved.

1.3.Outline

This thesis includes 5 chapters. In Chapter 2, existing methods for bandwidth aggregation are inspected in detail.

In Chapter 3, proposed algorithm is explained in detail. Each logic component for proposed algorithm is explained separately and the implementation of the proposed algorithm is placed.

In Chapter 4, experimental results done with the implementation is placed. Experimental results are compared with expected results and success of proposed algorithm is evaluated.

In the last chapter, conclusions are drawn by referring to Chapter 4. Future works to be done to improve proposed solution are suggested.

CHAPTER 2

BACKGROUND AND RELATED WORKS

When TCP/IP has been specified 40 years ago, network applications, devices and the Internet were much more different than they are today. In our day, applications require higher bandwidth and lower latency than ever before because most of the data is stored in cloud instead of local repositories and thus an excessive amount of data streamed from cloud to local computers over the Internet. To be able to meet the needs of today's applications, mobile devices are now equipped with more than one interfaces such as Wi-Fi and 4G/LTE. However, TCP communication protocol is not capable of utilizing more than one interface for a single connection. Hence a TCP session can be established through one network interface attached to each end-node. At this point, many connection pooling and bandwidth aggregation methods and protocols have been proposed to meet the need for more bandwidth. This chapter provides a brief review of such methods.

2.1. MPTCP

Multipath TCP (MPTCP) [3], which is an extension of TCP, comes forward as a solution for the problem of utilizing multiple access networks while achieving reliable delivery provided by TCP. MPTCP is currently standardized by IETF [4].

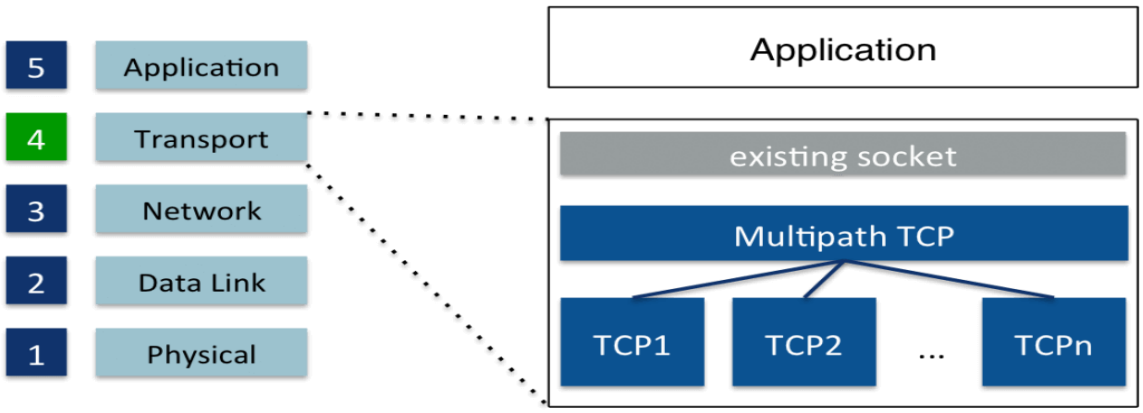

Figure 1 MPTCP Architecture [5]

MPTCP is a backward compatible extension to TCP and it can use multiple connections established through multiple network interfaces for a connection (i.e., a bi-directional data stream). Hence bandwidths of these network interfaces are combined to have one higher bandwidth logical connection as shown in Figure 1. The data streams pushed by each end-point are divided into several data chunks which are transferred through multiple access networks. Hence, the application can maximize the throughput by means of bandwidth aggregation on multiple network interfaces.

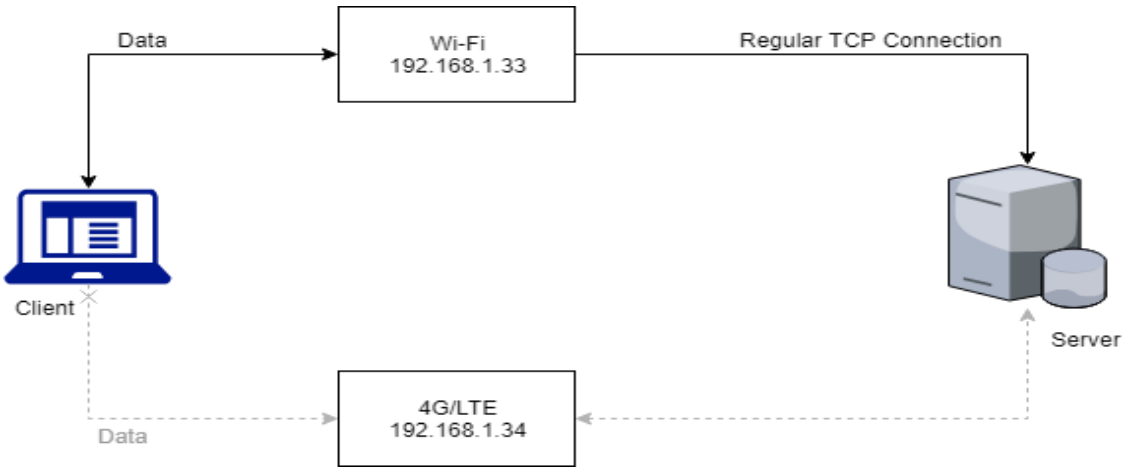

Figure 2 A Client and A Server Using a Traditional TCP Connection

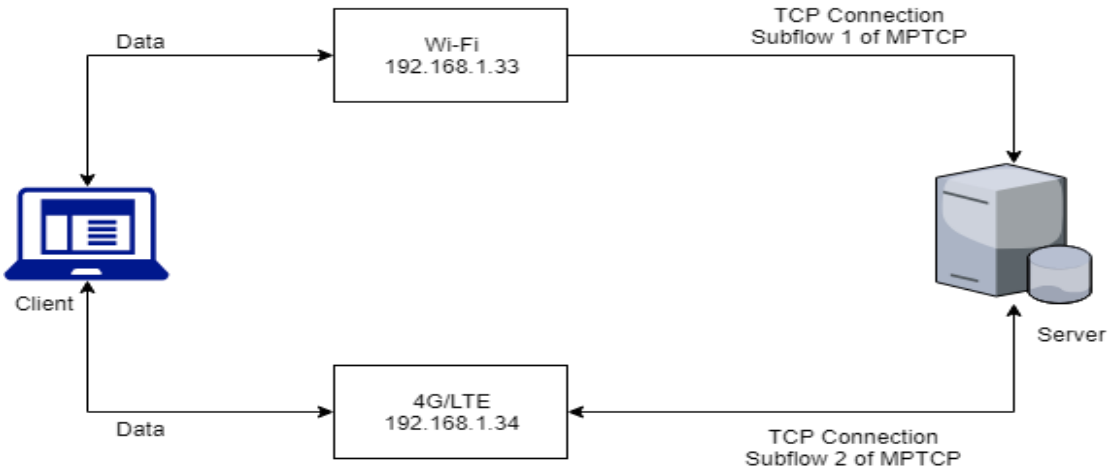

Figure 3 A Client and a Server using an MPTCP Connection

In Figure 2, a client that has two different network interfaces attached to two different access networks is illustrated. In this case, the client can use these access networks to connect to a server through the Internet. However, if the client and the server are connected to each other using standard TCP, only one of the access networks (e.g., the Wi-Fi interface) can be utilized to exchange data. Alternatively, the client and the server can be connected to each other using MPTCP (see Figure 3). Hence, both access networks can be utilized by the client to transfer data to/from the server. In this case, both end points divide the input data streams into chunks and transfer those chunks over two different connections established through the Wi-Fi and the 4G interfaces. Thus, the client can achieve higher throughput. Moreover, the client can also keep one of these interfaces as a hot spare that will be used whenever the primary access network is unavailable. Since both interfaces are active at any time, failover to the spare connection can be performed very quickly. This is another advantage brought by MPTCP and especially useful for the mobile devices.

The scheduler, which decides on which data chunk (i.e., TCP segment) will be sent via which access network (i.e., TCP subflow), is the most crucial component of MPTCP. The objective of a scheduler is to maximize the throughput by determining data chunk transfer time, order, and transmission medium. A successful scheduler may improve delay and other quality characteristics as well as throughput. Therefore, in MPTCP, scheduler design has a significant effect on the performance.

MPTCP has a Linux Kernel Implementation [6] with three out of the box schedulers [7]. The Lowest Round Trip Time First (LowRTT) scheduler is the default scheduler for the MPTCP implementation. LowRTT scheduler sends data over the subflow that has the lowest estimated RTT value, until its congestion window is filled. In other words, LowRTT scheduler firstly estimates RTT values for each subflow, and then it continuously sends data over the one that has lowest RTT value, until it reaches the limit of congestion window. Scheduling according to the lowest RTT value increases quality of service. On the other hand, this may not be the best option if the user pays for amount

of data transferred over the access network providing lowest RTT. Hence, one of the main goals of this thesis is to minimize monetary cost as well as improving quality of service.

Another scheduler available in MPTCP Linux implementation is the Round-Robin (RR) scheduler [7]. The RR scheduler selects the sublows and sends data over them in a round-robin fashion. Hence, it distributes load evenly on subflows. The RR scheduler sends a fixed amount of data (one segment by default) before switching to the next subflow. The RR scheduler ensures the maximum utilization of all subflows as well as distribution of data evenly on to the subflows. However, fully utilizing all the subflows may not be needed in all cases. For example, if one of the subflows is via a free access network, the other is via a paid access network, and the free connections throughput is almost sufficient for the application, RR scheduler will unnecessarily utilize the paid access network resulting in high monetary cost. Hence, one of the main goals of this thesis is to maximize the utilization of free access networks and to utilize the paid access network only when it is really needed.

The third available scheduler in MPTCP Linux implementation is the Redundant Scheduler, which sends each segment over all subflows, redundantly. This scheduler would be beneficial especially when reliability and timely delivery are main issues. However, the Redundant Scheduler is, in many cases, inefficient.

There are many studies that propose novel MPTCP schedulers to improve data transfer performance. In [8], authors propose a method to improve performance of LowRTT scheduler by freezing the slow path. As it is explained above, LowRTT scheduler sends data segments over the fast path until the congestion window is full. Then, the following segments are sent over other slower paths. In the proposed method, if the RTT value difference between the slow and the fast path is significant, the slow path is frozen and the next data segment waits until the congestion window of fast path is available again. The slow path is utilized and segments are sent over it if only if the fast path's congestion window is full and the remaining segment count is above some predefined threshold value.

Yang also proposes a method to improve default LowRTT scheduler [9] and it is claimed that sending chunks over different subflows based on only RTT values causes congestion and packet losses. Therefore, throughput and other quality of service characteristics deteriorate. Authors state that a good packet scheduler needs to select the subflow based on not only its RTT but also its congestion level. The proposed scheduler estimates the available capacity for each subflow. After available capacity estimation for each subflow, authors' scheduler calculates an "OCCUPIED" value by using outstanding packets and available capacity estimation for each subflow. For the next chunk, scheduler selects a subflow by checking "OCCUPIED" and RTT values. It selects the subflow, which has the minimum RTT value and whose "OCCUPIED" value is under the predefined threshold. Although the scheduler proposed by the authors

increases the throughput of default LowRTT scheduler under congested network conditions, it may over-utilize a paid access network to maximize the throughput.

Head of Line (HoL) blocking occurs when the segments received over the fast path aren't processed because of excessively delayed earlier segments sent over slow path. Blocking Estimation (BLEST) [10] scheduler is proposed to decrease HoL blocking probability. At the beginning, BLEST scheduler operates similarly with the default LowRTT scheduler and it sends segments over the fast path until its congestion window is full. Then, the scheduler calculates the possibility of HOL blocking if the subsequent chunks are sent over slow path. If the probability is larger than the predefined threshold, the following segments are not sent until an empty spot in the congestion window of the fast path is available. BLEST scheduler tries to eliminate HOL problem but it still uses LowRTT scheduler as a baseline which causes utilization of the fast path as much as possible which may be very expensive for the user.

In [11], MPTCP and Software Defined Network Controller (SDN) [12] are combined to improve the default Round-Robin scheduler of MPTCP implementation. The RR scheduler tries to distribute load over subflows by pushing a predefined number of segments to each subflow at each round. The improvement proposed by the authors is to set a different number of segments to transmit over each subflow in each round. Hence, the main idea is to transform the Round-Robin scheduler to Weighted-Round-Robin Scheduler. The number of segments to send in each round over each subflow is determined by the SDN Controller by considering the current situation of the underlying network. In this method, the users can also limit the utilization, hence the monetary cost, of 4G/LTE connection. However, the mechanisms to achieve optimal utilization have not been detailed in this work. In the experimental evaluation, different tests carried out to see whether the WRR scheduler is compatible with the SDN controller. Moreover, even if the 4G/LTE path is limited with the SDN controller, some of the chunks will be still downloaded over 4G/LTE path even if it is not needed. For example, consider a video streaming scenario in which the video play out rate is 1Mbit/s, Wi-Fi link speed is 3Mbit/s and 4G/LTE link speed is 5Mbit/s. SDN controller sets the number of segments send in each round for Wi-Fi connection to 10 while it is set to 1 for 4G/LTE connection. Even if the weight for 4G/LTE is limited in this scenario, 4G/LTE will be still be utilized even when the Wi-Fi link's bandwidth is sufficient to download all chunks before scheduled playout times. SDN Controller just considers the current situation of underlying network, it does not take into account the time sensitive nature of the data that is to be sent.

In [13], [14], and [15] novel mechanisms are proposed to solve the issues in existing MPTCP schedulers such as HOL blocking and out of order delivery. However, none of these mechanisms focus on cost aware data delivery. In [16] and [17], energy saving mechanisms have been proposed to decrease the energy consumption for mobile devices when MPTCP is employed, but neither of these mechanisms consider cost aware scheduling problem.

2.2. SCTP

Stream Control Protocol (SCTP) [18] is another protocol that enables multihoming. SCTP is standardized by IETF. SCTP is a message oriented transport protocol and the main purpose of it is the successful handover of data transfer to alternative interface when the primary interface fails. An extension containing various schedulers for SCTP has been standardized by IETF [19]. Schedulers such as Round Robin Scheduler, First Come First Served Scheduler and Priority Based Scheduler are available in this extension. However, none of these schedulers aims to stream time sensitive file in a cost efficient manner, to aggregate bandwidths and to achieve minimum cost. Instead they mainly focus on the problems like HoL blocking and successful handover.

In [20], authors propose a cost aware SCTP scheduler which transmits packets in a cost-efficient manner for multi-homed vehicles. They propose a cost model which optimizes how much money the client can pay, how much data the client wants to transfer (traffic demand), delay profile of data and probability of Wi-Fi availability (Reachable hot spots always change as the vehicle is moving). Cost efficiency is one of the objectives of this study, and they considered delay tolerant data transfer. So, they don't specifically consider time sensitive data transfer. Moreover, this work considers the cellular network as the primary access network and Wi-Fi is utilized only when it is available.

2.3. Cost Aware Schedulers

There are also scheduler proposals that consider monetary cost and energy consumption in addition to throughput. Such schedulers are inspected in this section.

In [21], an algorithm for resource efficient video streaming for mobile devices is proposed. The proposed algorithm minimizes the weighted sum objective of 4G/LTE connection money cost and mobile device energy consumption. Firstly, they model the scheduling problem as a Markov Decision Process and accordingly a scheduling algorithm based on dynamic programming is developed. Then they develop a heuristic algorithm that approximates the optimal scheduler. The algorithm proposed utilizes the Wi-Fi connection only when the Wi-Fi connection's throughput is enough to continue video streaming. If not, the subsequent chunks are transferred over the cellular network. This study focus on optimization of energy consumption and monetary cost of cellular network usage but if the Wi-Fi access network is not sufficient for video streaming, it is just switched off. However, in this thesis, we consider maximum utilization of free access network even if the throughput achievable is just only a fraction of the required bandwidth.

ESPA [22] is a 4G/LTE Wi-Fi interfaces bandwidth aggregation algorithm which uses multi-cost function for mobile phone file transfers. ESPA's cost function includes battery life, data usage quota for cellular interface and file transfer completion time. It compromises of three main parts; ESPA Network Interface Manager, ESPA Equalizer

and File Segment Manager. ESPA Equalizer takes cost formulation and weighting factors for battery usage, 4G/LTE quota usage, and file transfer completion time. ESPA Network Interface manager selects the best available interface in reference to estimated throughputs. ESPA server receives feedback from ESPA Network Interface Manager and dynamically sends individual segments of file to one of the available interfaces. ESPA algorithm offers a good solution that involves cost, energy and the transfer completion time for the files which are not necessarily time sensitive. That is, ESPA doesn't specifically propose a solution for time sensitive data transfer.

In [23], bandwidth aggregation for cellular and Wi-Fi access networks is aimed together with cost-effectiveness for video streaming applications. Authors employ split-layer SVC encoding and split video file in spatial basis to three parts as base layer (BL), and two enhancement layers EL1 and EL2. Proposed system initially starts with downloading BL and EL1 layers over the cellular network while downloading EL2 layer over the Wi-Fi network. If the Wi-Fi network is in an excellent condition and sufficient to download more than EL2 layer, EL1 layer and if possible, BL layer are also downloaded over the Wi-Fi network. Otherwise, the Wi-Fi network is used to download only the enhancement layer EL2, because BL and EL1 layers are considered as more crucial layers for video playout. If the user chooses quota saving, only BL and EL layers of video file can be downloaded over the cellular network in case the Wi-Fi network is insufficient for these layers. Otherwise, if the user chooses the maximum quality, EL2 layer also can be downloaded over the cellular network in case the Wi-Fi network is insufficient. This study pays attention to the video quality over cost efficiency. At the beginning the cellular network utilized to download BL and EL1 layers however, if the Wi-Fi does not provide necessary bandwidth, the cellular network is utilized to download all layers including the base one. However, in this thesis, our aim is to utilize free connection as much as possible throughout the data transfer. Moreover, the authors employ a client-side request scheduling method which requests the next chunk after receiving the notification of the previous one. In other words, they employ a stop and wait method. However, if the network causes large delay, the proposed method can utilize the free Wi-Fi connection less than its capacity since, Wi-Fi connection will be idle during the propagation and queuing delay periods. In the experimental evaluations, only the observed Wi-Fi throughput is given, if these values were compared with the predefined or limited maximum Wi-Fi throughput, utilization problem could be observed.

CHAPTER 3

COST AWARE CONNECTION POOLING

Cost Aware TCP Scheduler (CATS) mainly focuses on constant bit-rate time sensitive data delivery between a client and a server by utilizing connection pooling. Stored video file streaming can be considered as one of the use-cases supported by the current implementation of CATS. However, the main approach proposed in CATS can be extended to cover various use-cases such as teleconferencing and videoconferencing over multiple channels.

In subsequent discussions in this thesis, when we refer to constant bit-rate time sensitive data, we are actually referring to data stored at the server and is segmented into chunks that are consumed at the client in time orderly manner. A video file when streamed between a server and a client can be considered as a typical example of high bit-rate time sensitive data. Most of the video servers like YouTube [21] store and transfer the video files in the form of chunks. And these chunks are delivered to the client which plays out the received chunks in a continuous manner.

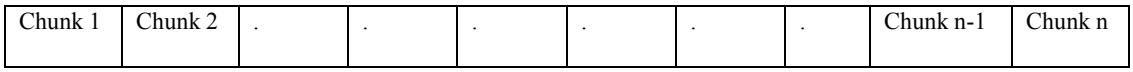

Figure 4 A Time Sensitive File Segmented in Chunks

In Figure 4, an example of file segmented into n chunks, is visualized. Each chunk comprises fixed size data bytes that will be used at a certain time interval after the file download/usage has been started. In CATS, the chunks, which are stored in the server, are numbered from Chunk 1 to Chunk n. Client requests and uses received chunks according to the scheduled usage time.

In CATS, Transmission Control Protocol (TCP) [2] is used to transfer chunks from a server to a client. TCP is indeed segmenting the data pushed by application into data packets but the applications utilizing TCP are oblivious to TCP segmentation. So CATS employs its own file segmentation.

CATS client application starts consuming data when a chunk is received entirely. Hence, from the client's point of view, the fact that processing a chunk can only be possible

when it is received entirely. Chunks also serve as jitter avoidance buffers. Moreover, the client usually uses the current chunk while the subsequent chunks are being downloaded. As such, the length of chunks determines the chunk usage time and this is a configurable parameter in CATS.

CATS primarily employs connection pooling for the timely delivery of file chunks. We assume that the stored time sensitive data is delivered from a server to a client over two parallel TCP connections each of which has different characteristics. First connection is called the free connection (FC). FC is established on a free link (FL). A FL (and FCs utilizing that FL) is considered to have fluctuating throughput and variable delay. However, there is no monetary cost associated with FL usage. The second connection is called the paid connection (PC) and it is established on a paid link (PL). A PL (and PCs established on that PL) is considered to have higher throughput and lower variable delay. As its name implies, there is monetary cost associated with PC usage.

In this thesis, we assume a scenario in which there is a client application processing high bit-rate time sensitive data transferred from a server. We also assume that this client device has at least two network interfaces such as Wi-Fi and cellular data. In this scenario, we consider that the Wi-Fi link is FL and the cellular link such as 4G/LTE is the PL.

We propose CATS to improve user experience by aggregating bandwidths of multiple links while minimizing the monetary cost incurred by the PL usage. CATS is based on bandwidth aggregation which is the utilization of multiple TCP connections for a single application. The strength of CATS lies in its consideration of monetary cost as well as QoS parameters in TCP connection pooling as opposed to the previous studies. Most of the previous works done on TCP connection pooling aim to maximize throughput by using multiple connections over separate links without considering monetary cost or limitations on any of the connections. In CATS, the goal is providing enough throughput to ensure timely delivery of chunks to the client rather than maximizing throughput and for this purpose the FC is utilized as much as possible. The three goals of this thesis are summarized as follows:

- Timely delivery of chunks to the client,
- Maximum Utilization of FC,
- Minimum Utilization of PC.

3.1.Optimal Bandwidth Pooling

This section investigates the role of different chunk scheduling mechanisms that can be used to distribute load on the multiple heterogeneous channels on achieving abovementioned goals for streaming time sensitive data. In order to simplify the presentation, we consider a video streaming over multiple channels scenario in which a server is streaming a video file to a mobile phone which has two network interfaces; Wi-

Fi and 4G/LTE. Suppose the client has established two connections to the streaming server over these interfaces. The connection established via Wi-Fi interface represents the FC and the connection established via 4G/LTE interface represents the PC. The video file streamed from the server is segmented in chunks and the chunks are transferred to the client and played out. The notation given in above Figure 5 will be employed in the following discussion and illustrations.

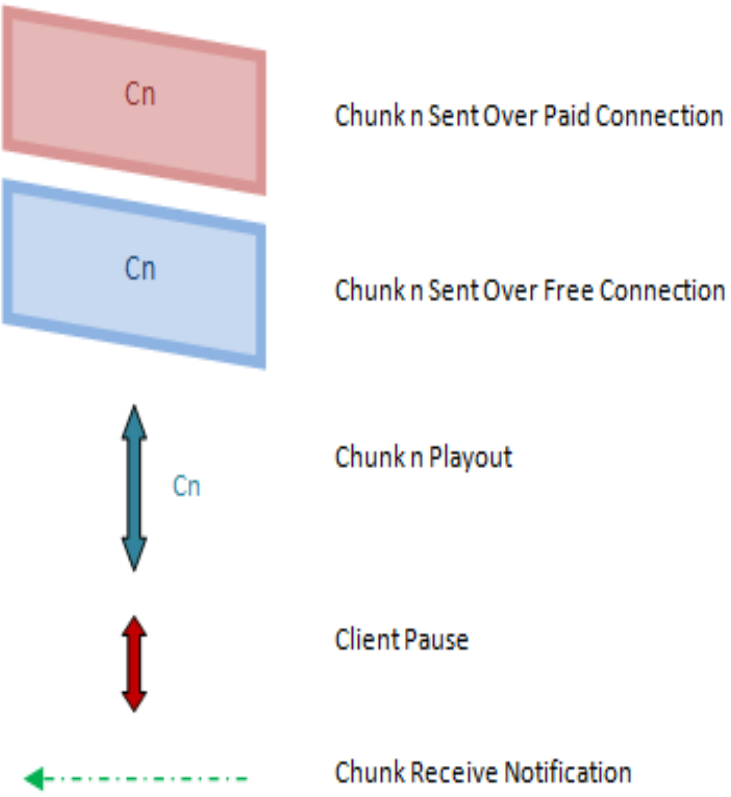

Figure 5 Chunk Download Figure Notation

3.1.1. Scenario 1: FC Throughput Higher Than the Video Playout Rate

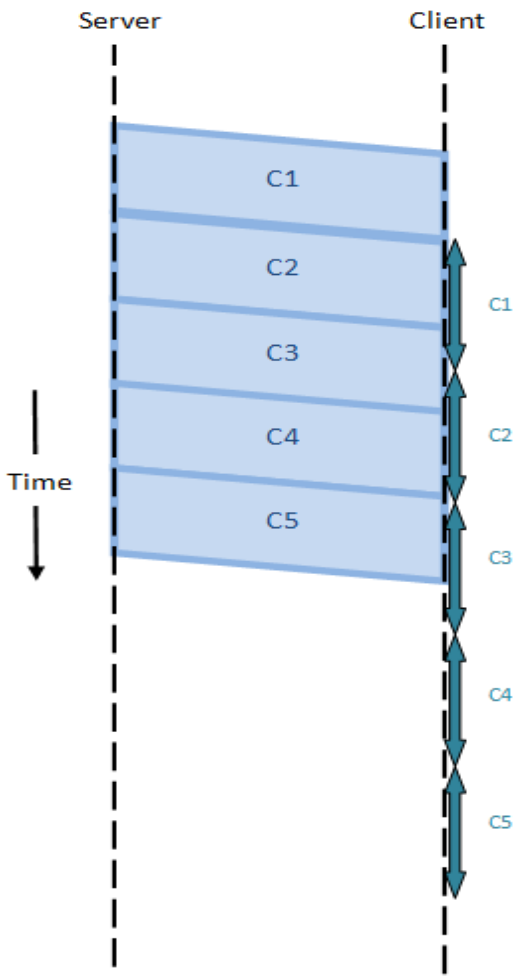

Figure 6 Scenario 1: FC Throughput Higher Than the Video Playout Rate

In scenario 1, client's free connection throughput is assumed to be higher than the video playout rate. As it is shown Figure 6, after the arrival of the first chunk, C1, client can start the playout of C1 and in the meantime the subsequent chunks are downloaded over free connection. For example, client downloads C2 and a bit of C3 while C1 is being played. This will be true for the rest of the chunks. As it can be seen from Figure 6, the client can stream the complete video file over only FC without any video pause. Hence, we can conclude that, if FC's throughput is higher than the video playout rate, no additional connection and a special mechanism is needed for timely delivery of data.

3.1.2. Scenario 2: FC Throughput is the Same with the Video Playout Rate

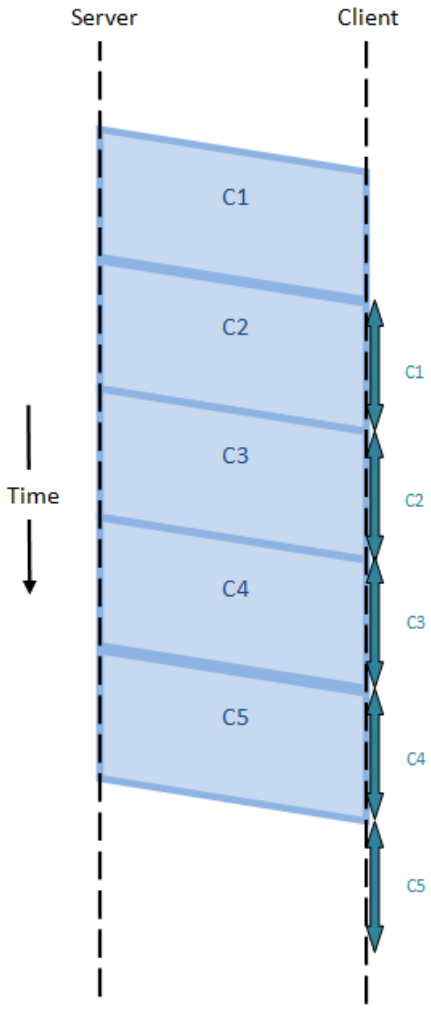

Figure 7 Scenario 2: FC Throughput is the Same with the Video Playout Rate

In scenario 2, the client's free connection throughput is exactly the same with the video playout rate. As it is shown in Figure 7, the client starts video playout after the arrival of C1 and C2 can be downloaded while C1 is being played out. This will be repeated for the subsequent chunks. In this scenario, as long as the client fully utilizes FC, all of the chunks can be downloaded over FC without any video pause as it can be seen from Figure 7.

3.1.3. Scenario 3: FC Throughput is smaller than the Video Playout Rate

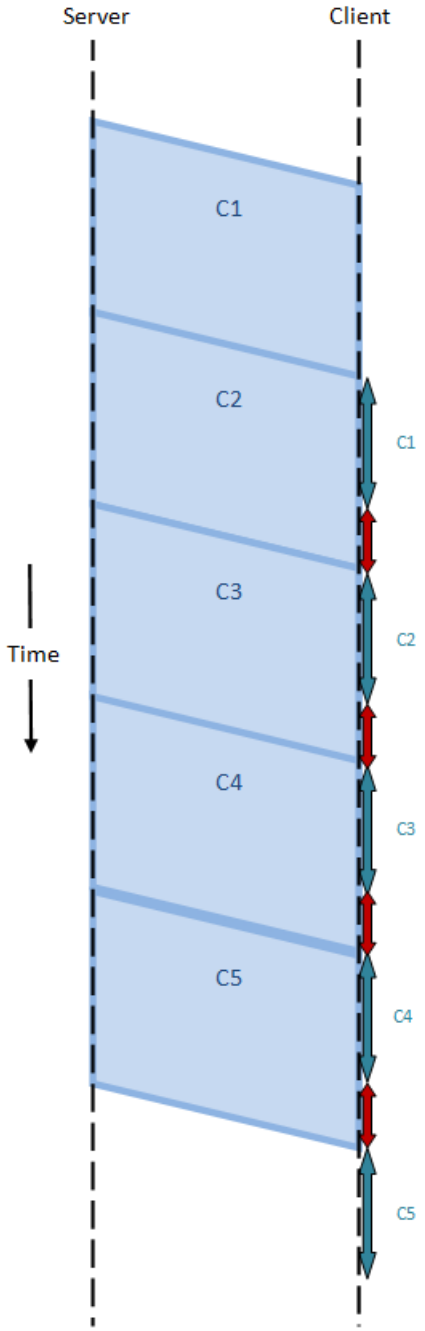

Figure 8 Scenario 3: FC Throughput is smaller than the Video Playout Rate

In the scenario 3, the FC's throughput is smaller than the video playout rate. As it is shown in Figure 8, the client experiences video pauses through complete video file streaming when only the FC is used. For example, the client starts video playout when C1 is delivered and while playing out C1, C2 could be downloaded. However, as shown in Figure 8, when C1's playout is complete, C2 will not be ready as the FC's throughput is lower than the video playout rate and the client has to pause until C2's transfer completes. Video pause is the most undesirable situation while video streaming and it deteriorates the quality of video playout. To be able to cope with this situation, various possible solutions will be explained.

3.1.3.1. Solution 1: Use Only PL

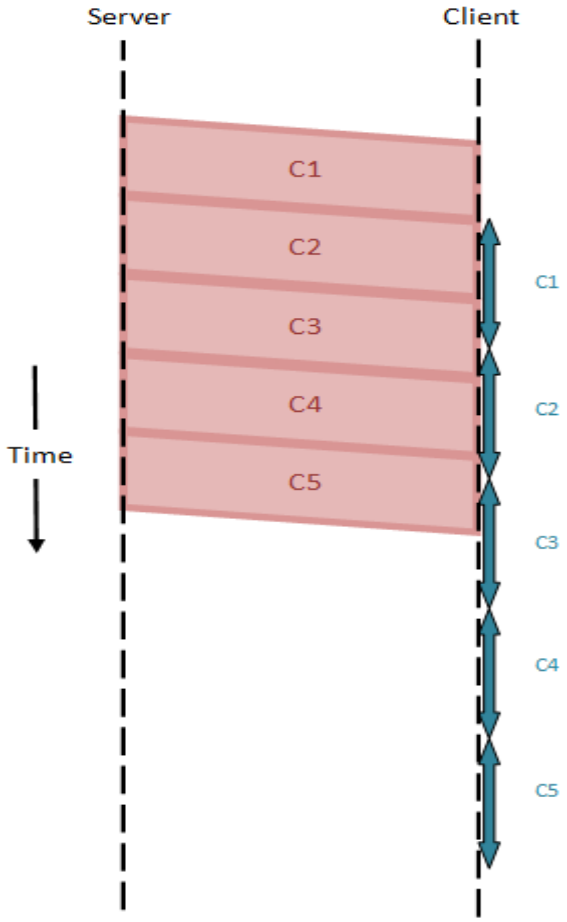

Figure 9 A Cost Inefficient Solution to Pause Problem in Scenario 3

Simplest solution for the client video pause problem could be to stream all video chunks over the PC as shown in Figure 9. In this scenario we assumed that the PC's throughput is considerably higher than the video playout rate. If the server streams the entire video

file over the PC, all the chunks will be delivered to the client before their scheduled playout times and there won't be any video pause. On the other hand, client faces with the cost of using PC. That is, data transferred over paid connection has a monetary cost and downloading a complete video file over only paid connection can cost too much. Furthermore, the client also wastes the FC bandwidth as it is not used. Even though, FC's throughput is lower than video playout rate, it can still be used to transfer some of the chunks while avoiding video pause and this will reduce the overall cost of the video transfer.

3.1.3.2. Solution 2: Bandwidth Aggregation

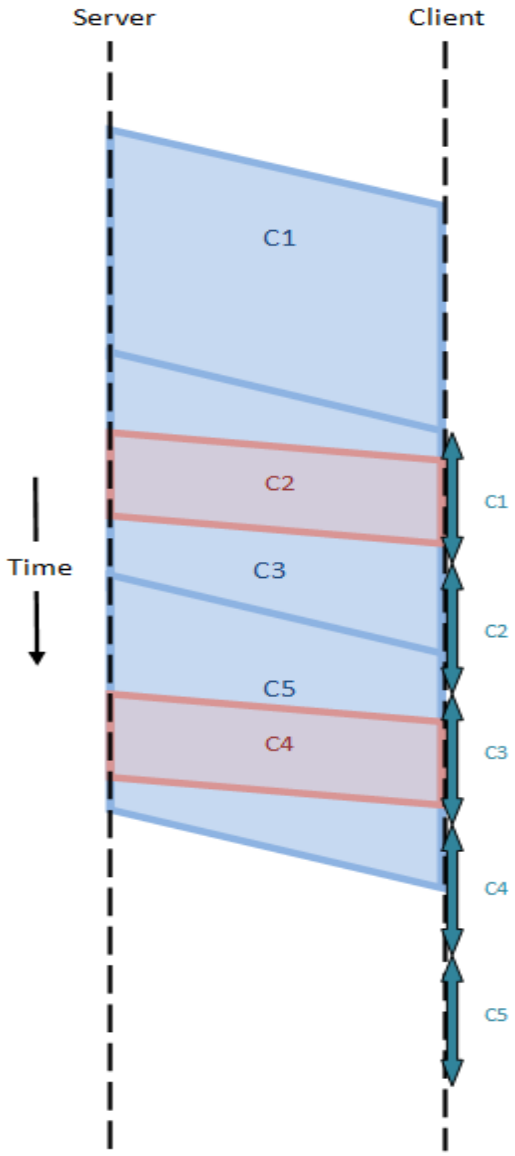

Figure 10 A Cost efficient Solution to Pause Problem in Scenario 3

In Figure 10, utilizing both FC and PC to ensure timely delivery of data chunks while minimizing the monetary cost is illustrated. In this scenario, FC and PC are pooled to stream video file in a manner that achieves previously stated CATS goals. As it can be seen from the Figure 10, some of the chunks such as C2 and C4 are transferred over the PC and the rest of the chunks are transferred over the FC. After the delivery of C1, the client starts video playout. If C2 was transferred over the FC, it would not be delivered to the client before the scheduled playout time and the client would experience a video pause. Alternatively, C2 is transferred over the PC and delivered to the client before its

scheduled playout time. In the meantime, FC is also being utilized for transferring C3 and delivered to the client after C2 is played out completely. Hence, the monetary cost is minimized and the playout quality is maximized.

3.1.4. Scenario 4: Variable FC Throughput

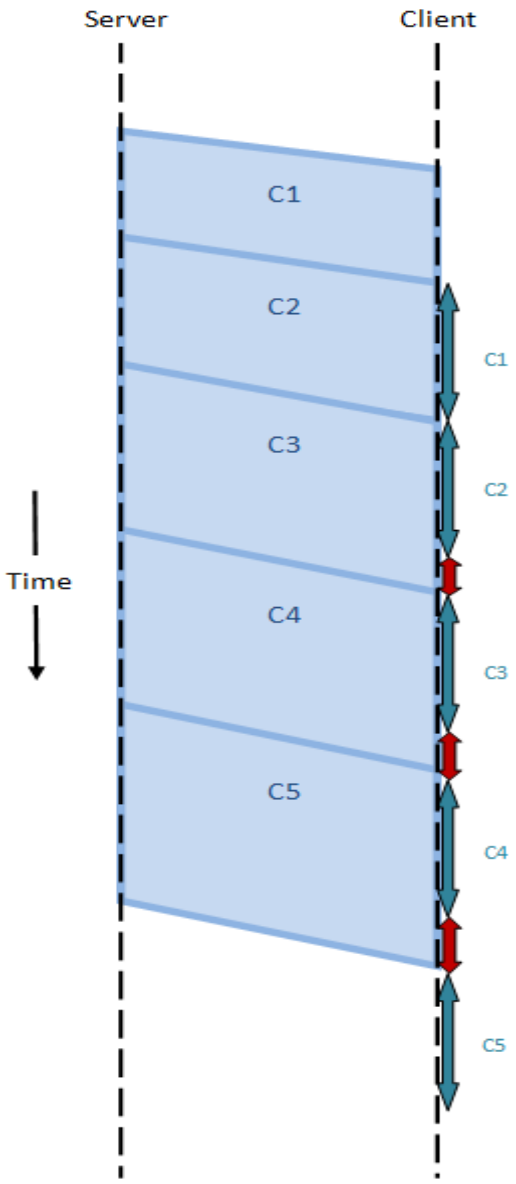

Figure 11 Scenario 4: Variable FC Throughput

In the previous scenarios, throughput of the FC is assumed to be constant. However, this assumption is not always valid in real networks. In Figure 11, the case in which the FC's

throughput is decreasing in time is illustrated. In this scenario, the client receives C1 and then the video playout starts. The FC's throughput is still sufficient to transfer C2 before its scheduled playout time. However, as it can be seen from Figure 11, as throughput of FC decreases, download time of the subsequent chunks increase. For example, when playout of C2 is complete, client pauses as C3 is not completely received. The same happens to the following chunks as long as the FC's throughput remains smaller than the playout rate.

3.1.4.1. Solution: Client Notifications and On Demand Use of the PC

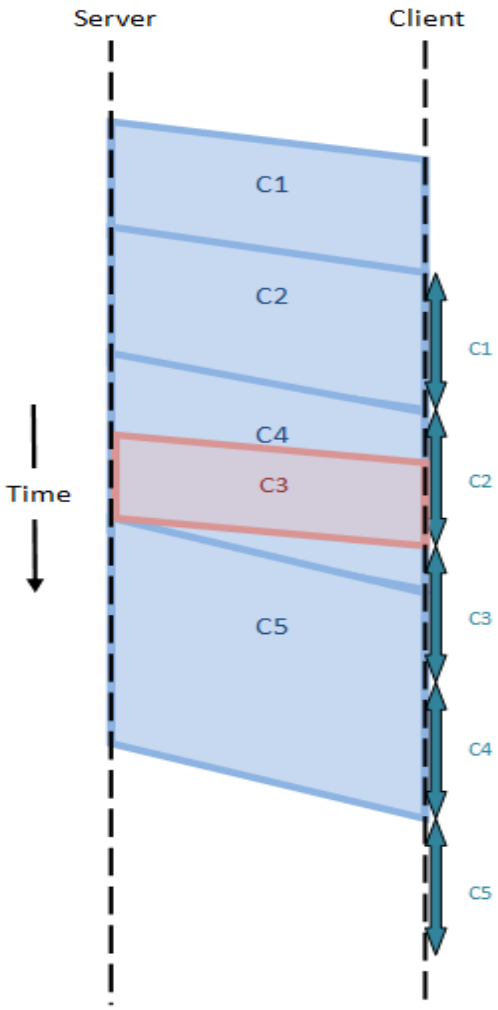

Figure 12 CATS Solution to Scenario 4

In Figure 12, on demand use of the PC is illustrated. In this scenario, the FC's throughput decreases in time and the chunks which can be delivered to the client before the scheduled playout times are pushed to the FC. If a chunk cannot be delivered to the client timely over the FC, it is transferred over the PC. For example, C3's estimated

delivery time over the FC is predicted to be large so it is going to miss the scheduled playout time. Therefore, it is transferred over the PC and it is delivered to the client timely. While C3 is being transmitted over PC, the FC is used to transfer C4 in order not to waste FC's bandwidth.

In order to determine whether a chunk can be delivered timely over the FC (and also over the PC) the server should be able to predict the delivery time of the chunk. Feedback from the client can be used for this purpose. In Figure 13, how the client notifications can be used as a feedback to predict the delivery time is illustrated. For each complete chunk received, the client sends a separate notification. The server can record the time at which a chunk is pushed to the FC (or PC) and uses the client's notification for that chunk to measure delivery time. By considering the scheduled playout time of the next chunk to be sent and recent delivery time measurement, the connection that should be utilized to the transfer the chunk can be determined.

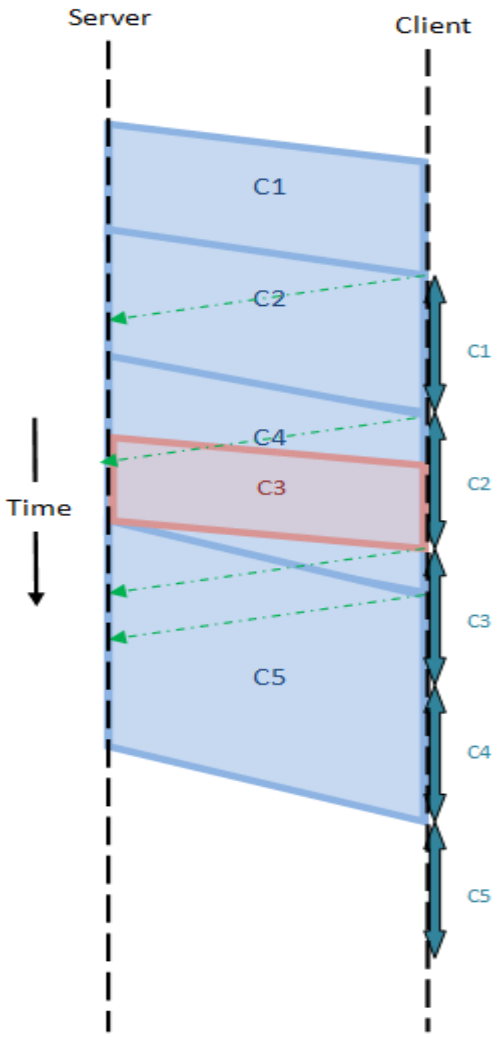

Figure 13 CATS Notification Mechanism

3.2. Overview of Cost Aware TCP Scheduler

In this section, fundamental concepts that characterize the CATS and the mechanisms employed by CATS are explained.

CATS consists of two modules: the CATS client and the CATS server. The CATS client requests a time sensitive file stored on a CATS server. Upon receiving the request, the CATS server sends the chunks of the file through two TCP connections to the CATS client. These connections are FC and PC established over FL and PL attached to the device running the CATS client as shown in Figure 14. Both FC and PC are initiated by the CATS client. The CATS server is mainly responsible for selecting the connection to be used while sending each chunk to the CATS client. The CATS sever also keeps track

of the chunk delivery times and determine when each chunk's transmission is initiated based on the feedback provided by the CATS client.

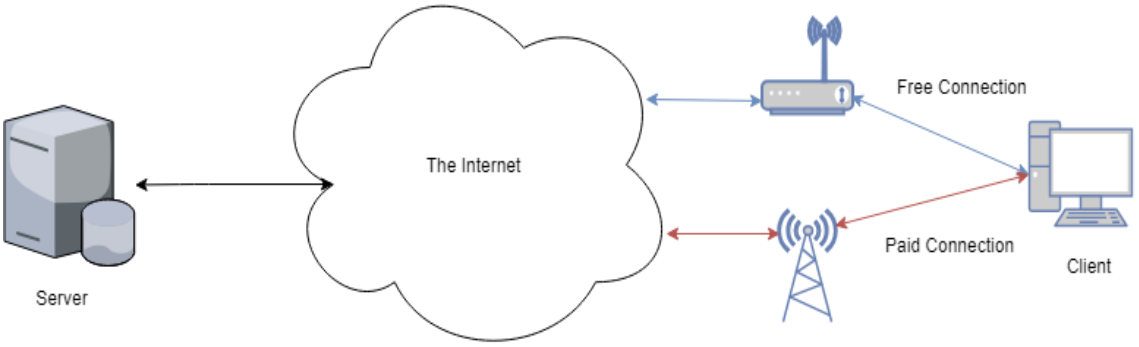

Figure 14 The Client and The Server Connection in CATS

As it will be explained in more detail in the subsequent sections of this chapter, there are notification and time synchronization mechanisms in CATS. Hereinafter, the terms, the client and the CATS client, and similarly the server and the CATS server will be used interchangeably in the rest of the thesis. Client sends notification packets to server side for each received chunk and these feedback packets are sent over paid connection. Client also sends timestamps to the server in order to synchronize their time and these synchronization packets are also sent over paid connection. In CATS, the server is the side that is responsible for making the important decisions about data transfer in the proposed algorithm. In the server, the requested file is segmented into a number of chunks and these chunks are sent to the client after completing the initial handshake with the client. The server is responsible for identification of the order of chunks to be sent, the connection to be utilized for each chunk, and the time at which each chunk's transmission starts.

The client is simply responsible for the straightforward system operations or tasks such as initializing and introducing the connections as free and paid connections at the beginning and requesting the file. For each chunk received during the file download process, the client sends notification packets to the server. The server uses these notification packets to compute MCTT (Measured Chunk Transfer Time). The client also sends time synchronization packets to the server after consumption of the chunks to facilitate time synchronization.

In the next section, we discuss the proposed mechanisms executed at the client and the server in more detail. All logic components and mechanisms for client and server sides will be explained under separate titles.

3.2.1. Communication Between The Client and The Server

CATS employs 6 different types of packets exchanged between the client and the server to execute the proposed algorithm. These packets are “DATA_CHUNK”, “INIT_TRANSFER”, “TIME_SYNC”, “NOTIFICATION”, “END_TRANSFER”, and “INIT_CONNECTION”. All of these packets use the CATS’s general packet structure shown in Figure 15. The packets are differentiated with a packet type field values. Moreover, each packet type has its own set of communication commands. The 4-byte packet type header field identifies the type of the packet. The 4-byte length field identifies the length of the data. Data field is the portion of the packet where actual user data is carried.

3.2.1.1. Packet Structure

Packet Type	Length	Data
(4 Byte)	(4 Byte)	(Packet Data Byte Size)

Figure 15 General Packet Structure of CATS

3.2.1.2 Packet Types

The types of packets used by CATS are listed in the Table 1.

Table 1 Packet Types

Name of Packet	**Description**
DATA_CHUNK	These packets are sent from the server to the client. Each chunk of a file is carried in DATA_CHUNK packets.
INIT_TRANSFER	These packets are sent from the client to the server. Client sends INIT_TRANSFER packets after the first initialization, to start the file transfer.
TIME_SYNC	These packets are sent from the client to the server. Client sends TIME_SYNC packets to synchronize time with the server.
NOTIFICATION	These packets are sent from the client to the server. The Client sends NOTIFICATION packets to notify the server about each received chunk.
END_TRANSFER	These packets are sent from the server to the client to notify that all chunks have been sent and the file transfer is completed.
INIT_CONNECTION	These packets are sent from the client to the server to notify each connection whether it is free or paid connection.

3.3. The CATS Client

The client streams time sensitive data in CATS. During the data streaming, client downloads and processes the chunks by time order. The server is responsible for sending the chunks to the client and running the scheduling algorithm. The tasks handled by the client are depicted in Figure 16 by using a state diagram.

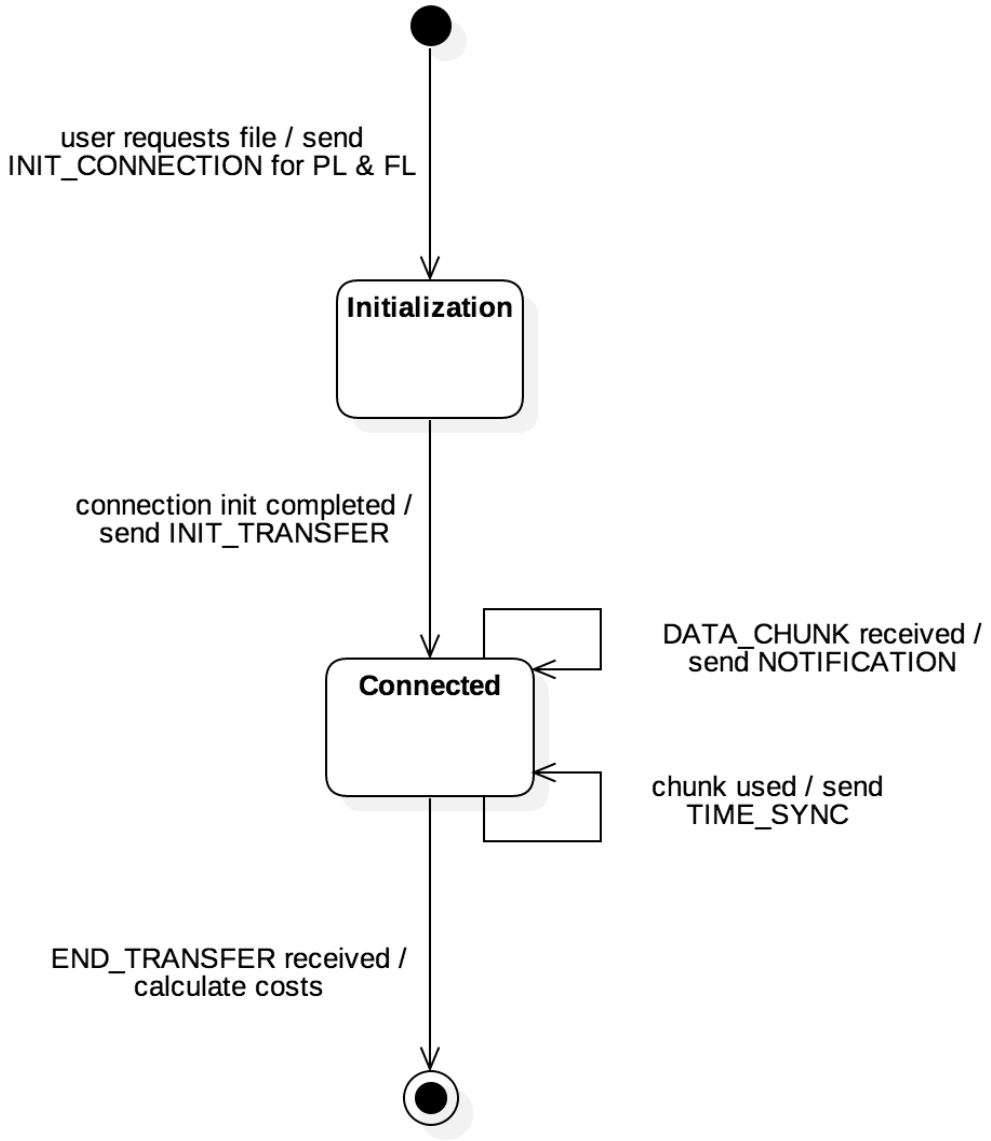

Figure 16 Tasks Handled by The Client

3.3.1. Connection Initilization

In CATS, the client connects to the server over two different TCP connections simultaneously; free and paid TCP connections. From the stand point of the server, two connections are identical, they only have specific port numbers and IP addresses. However, in order our proposed algorithm to be executed, the server needs to differentiate, which flow represents the free connection and which one represents the

paid connection. At system startup, the client is responsible of creating the connections and introducing them to the server.

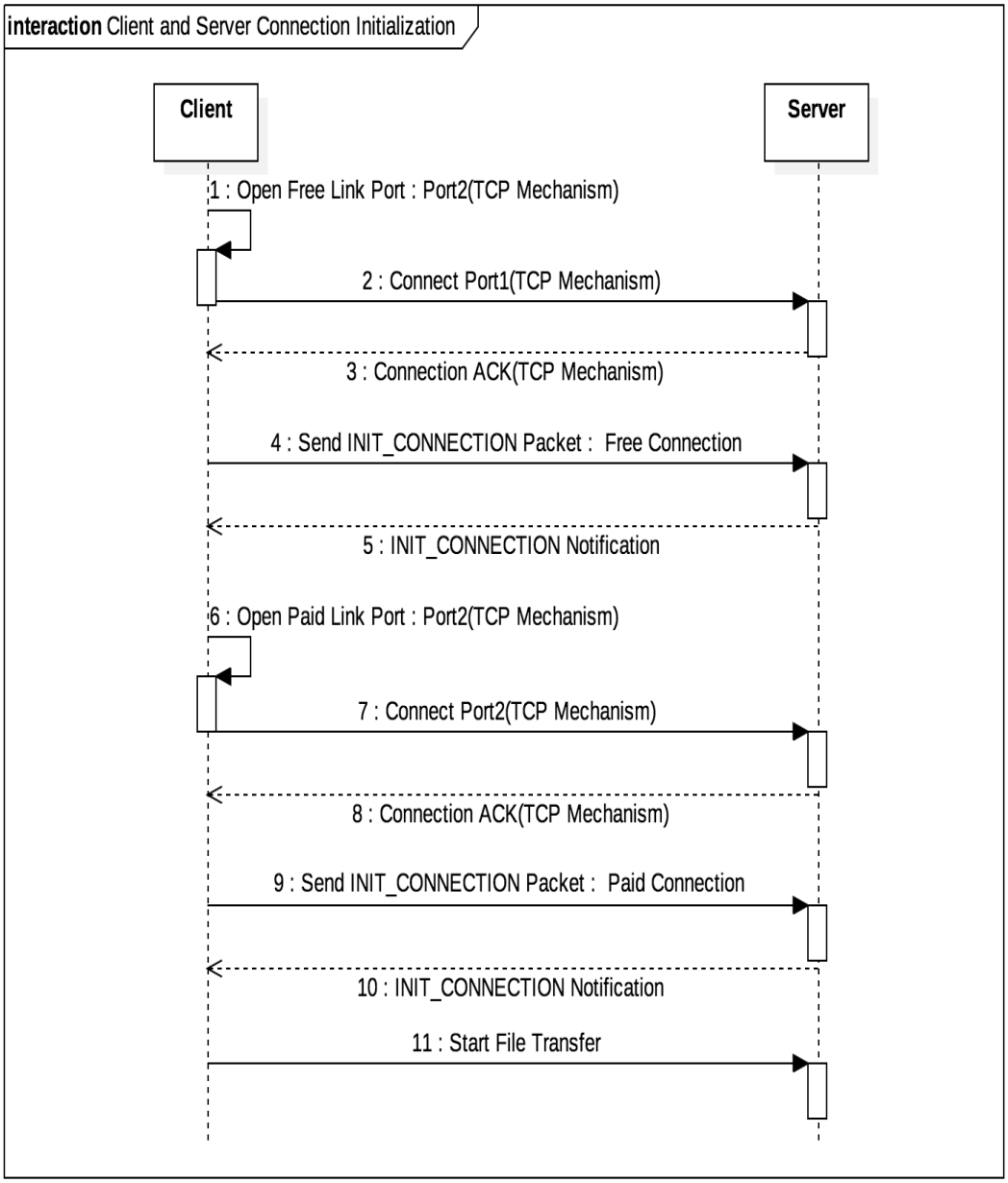

Figure 17 The Client and The Server Connection Initialization Process

In Figure 17, initialization process between the server and the client is visualized. Firstly, the client creates a TCP socket from the free link and connects it to the server. Then, client sends an "INIT_CONNECTION" packet for the connection and ensures

that server stamps this first connection as free connection. Secondly, the client creates a TCP socket from the paid link and connects it to the server. As it is done for free connection, client sends an "INIT_CONNECTION" packet for the paid connection and ensures that the server stamps this connection as the paid connection. After this point, to be able to start data streaming, the client sends "INIT_TRANSFER" packet to the server. When the server receives "INIT _TRANSFER" packet, the chunks will be sent over the free and the paid connections as it will be explained in Section 3.4.

3.3.2. Chunk Receive Notification Mechanism

In CATS, TCP is the only transport layer protocol in all means of communications. TCP has its own ACK mechanism, and CATS is operating on the top of TCP layer. However, CATS cannot reach and thus cannot use the ACK mechanism of TCP. This is the reason why we introduce an additional ACK mechanism. This CATS specific ACK mechanism is called CATS notification mechanism.

The Client sends notification packets to the server for the received chunks and these notification packets play a very critical role for the consistent operation of the proposed algorithm. The received notifications are the only way for the server to obtain MCTT (Measured Chunk Transfer Time) for the sent chunks. As the server receives the notifications for the sent chunks, it can estimate instant throughput of underlying network by obtaining MCTT for each chunk. Another reason for exploiting the notification packets is giving the information to the server whether any chunk which is sent over free link will reach to client before its scheduled usage time or not and whether duplicate send will be needed or not over paid link to prevent the client pause as it will be described in Section 3.4.3.1.

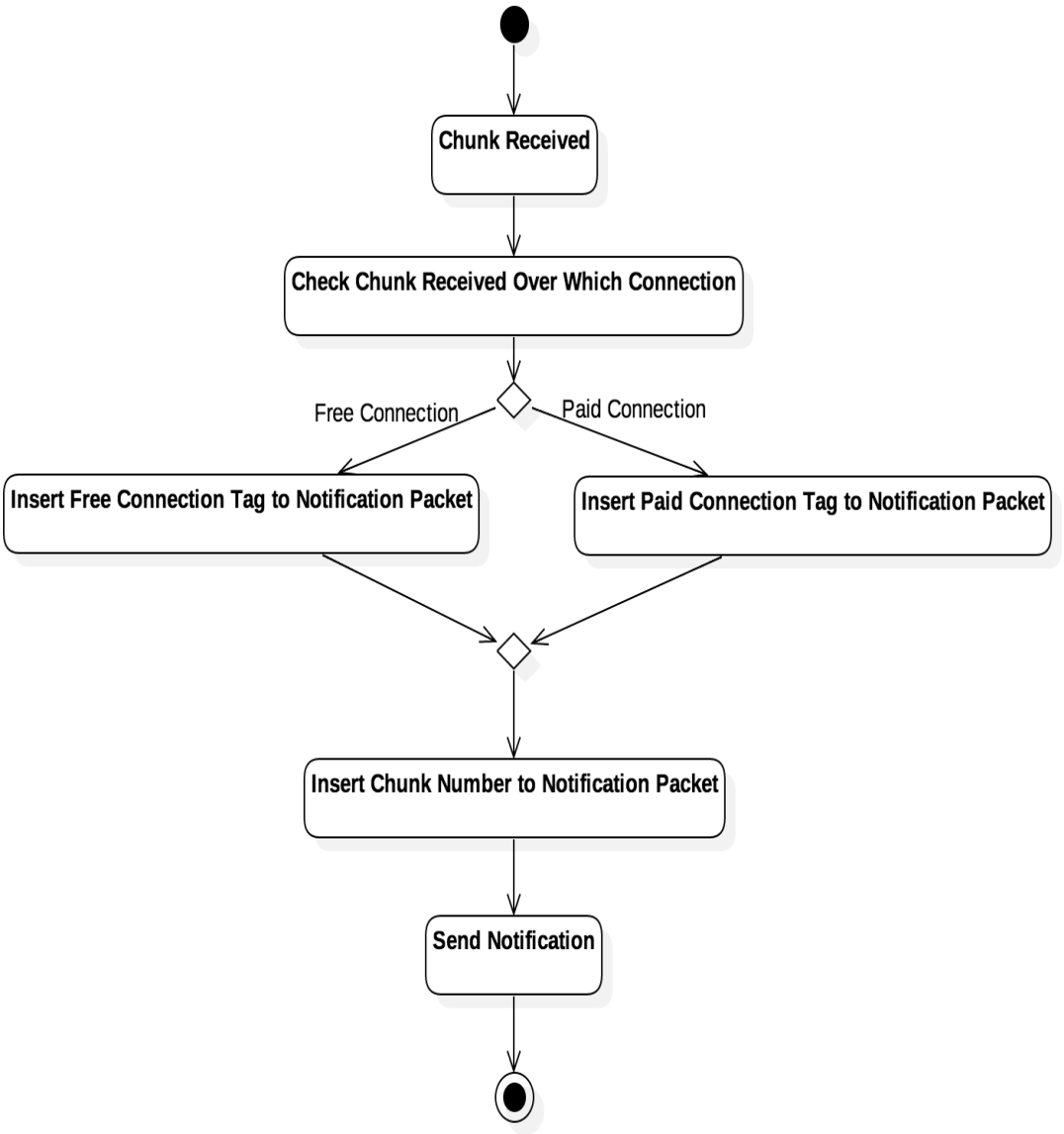

Figure 18 The Notification Mechanism Performed by the Client

The server sends each chunk to the client with their sequence numbers, thus client can identify the chunks by checking their sequence numbers. Client sends notification packets for each chunk also with their sequence numbers, thus server can identify each received notification packet belonging to which chunk.

Another important piece of information placed in the notification packets is the connection type which the client received chunk over. The server may send each chunk to the client over one of two available connections, free and paid connections, moreover in some cases a chunk may be sent twice over both connections. As it will be explained

in the Section 3.4.2.1, server obtains Measured Chunk Transfer Time (MCTT) values for each chunk sent over each connection separately. Therefore, it is necessary to put the information about which chunk received over which connection into the notification packets as it is shown in Figure 18.

The client sends notification packets over the paid connection. CATS is designed to stream stored time sensitive file without pausing the client even in case of undesired network conditions such as high jitter, lower throughput etc. CATS has only MCTT values as feedback from the network. All the estimations and calculations are based on the obtained MCTT values. Therefore, the paid connection is chosen to deliver notification packets faster which would result more accurate MCTT results and better estimations. The main drawback of sending notification packets over the paid connection is its monetary cost. On the other hand, as the overhead of these notification packets are relatively small when compared with the entire file to be streamed, the cost of using the paid link can be considered as negligible.

3.3.3. Chunk Usage and Time Synchronization Packets

CATS can be used in various use-cases. In the domain of this thesis study, time sensitive file streaming is handled. Therefore, the client side is considered as a time sensitive file user which uses chunks of file by time order and pauses when the next chunk is not received yet.

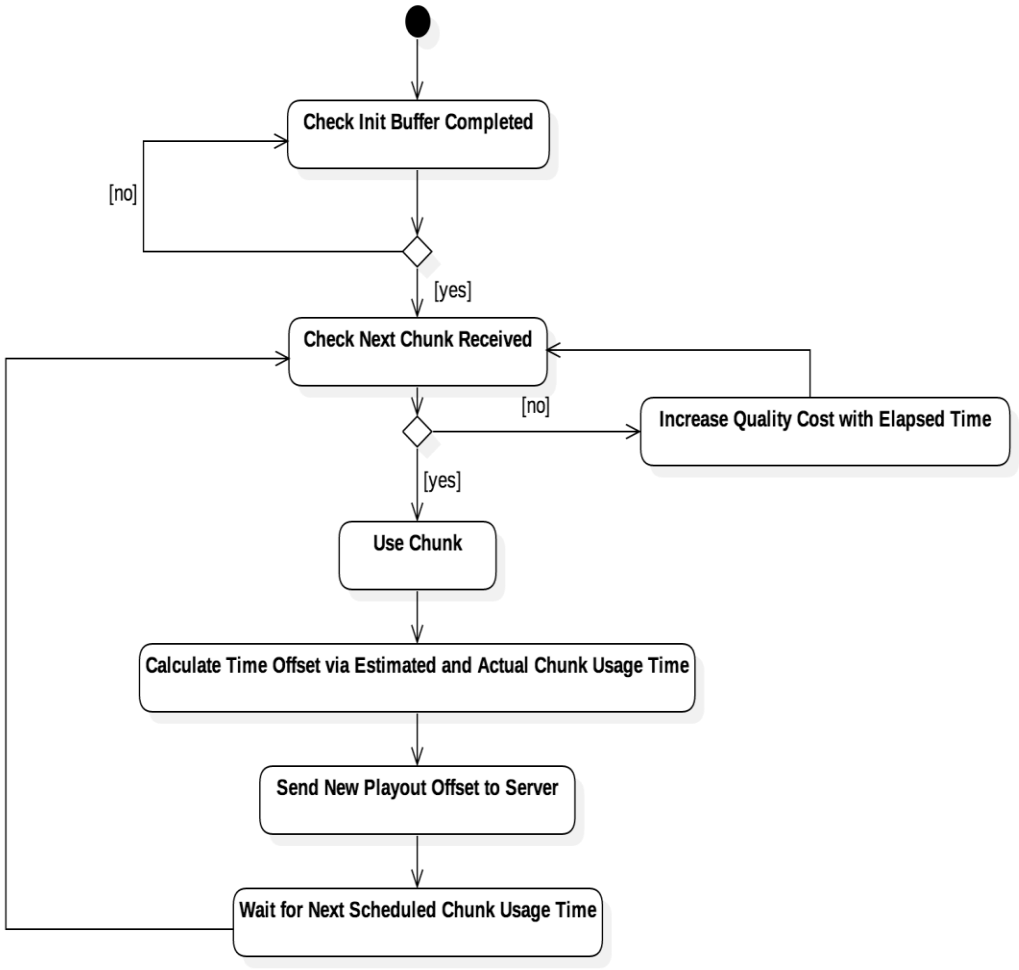

Figure 19 The Chunk Usage Mechanism Performed by the Client

The client has an initial buffer of 10 seconds before it starts to consume chunks. That is, the client downloads the first chunk, but chunk usage starts after 10 seconds. As shown in Figure 19, the client consumes the chunks one by one after the previous chunk's scheduled usage period is completed. If the next chunk is missing, the system updates the quality cost, which will be explained in the next part.

Another key feature of the client is sending time synchronization packets. Proposed CATS system is meant to be working and implemented in two different computers; as client and server. It is observed that, due to many reasons like; client pause, processing delays etc. scheduled usage times of chunks may change throughout the playout. Contrary to the server, client can easily detect this change, since it knows the exact wall clock time of each chunk's usage time. Sending playout time to the server for each chunk would be a solution but this would also be unnecessary transmission of many

bytes. Instead of sending timestamps for each chunk, the use of general offset value for chunk usage would be easier and requires fewer bytes to be transmitted.

The Client gets wall clock time when the first chunk is used, and stores this time value as the Start Time of the file streaming. Each chunk's usage time is explicitly calculated with the usage of first chunk with the formula below;

$$Estimated\ Usage\ Time = Start\ Time + ChunkNumber * Chunk\ Time\ Period \quad (1)$$

For the rest of the chunks, the client calculates difference between the estimated and actual usage time offset as:

$$Offset = Estimated\ Usage\ Time - Actual\ Usage\ Time \quad (2)$$

Client sends this offset value to server and, thus server can accurately update usage times for each chunk. Client sends offset value to server once again only if difference between current offset value and last sent offset value reaches a predefined threshold value. In CATS, this threshold value is set to 10% of a chunk's time period which is determined by carrying out several exploratory experiments.

3.3.4. The Cost Calculation

One of the tasks of client side in CATS is cost calculation. At the end of the time sensitive file streaming, system's success is evaluated with these calculated costs and algorithm improvement is done in the light of these costs.

3.3.4.1. The Money Cost

Client counts all the chunks which are downloaded over the paid connection. At the end of the file completion and streaming, money cost is calculated as the number of chunks downloaded over paid connection.

3.3.4.2. The Quality Cost

As explained in the previous section, the client has a chunk usage mechanism. If a chunk is available at the scheduled playout time, the client uses this chunk. But if the chunk is not available, the client pauses and the client quality cost measures the pause duration for each chunk missing its scheduled playout time. At the end of the streaming, the quality cost calculator sums all pause durations.

3.4. The CATS Server

The server is the brain of the proposed algorithm. After the initialization, server side makes the scheduling for the chunks transfer. The objectives of the scheduling algorithm are maximum free connection utilization, minimum paid connection usage and minimum client pause.

The server operation is summarized in Figure 20. In order to explain the mechanisms employed by the CATS Server, a set of scenarios given below are used.

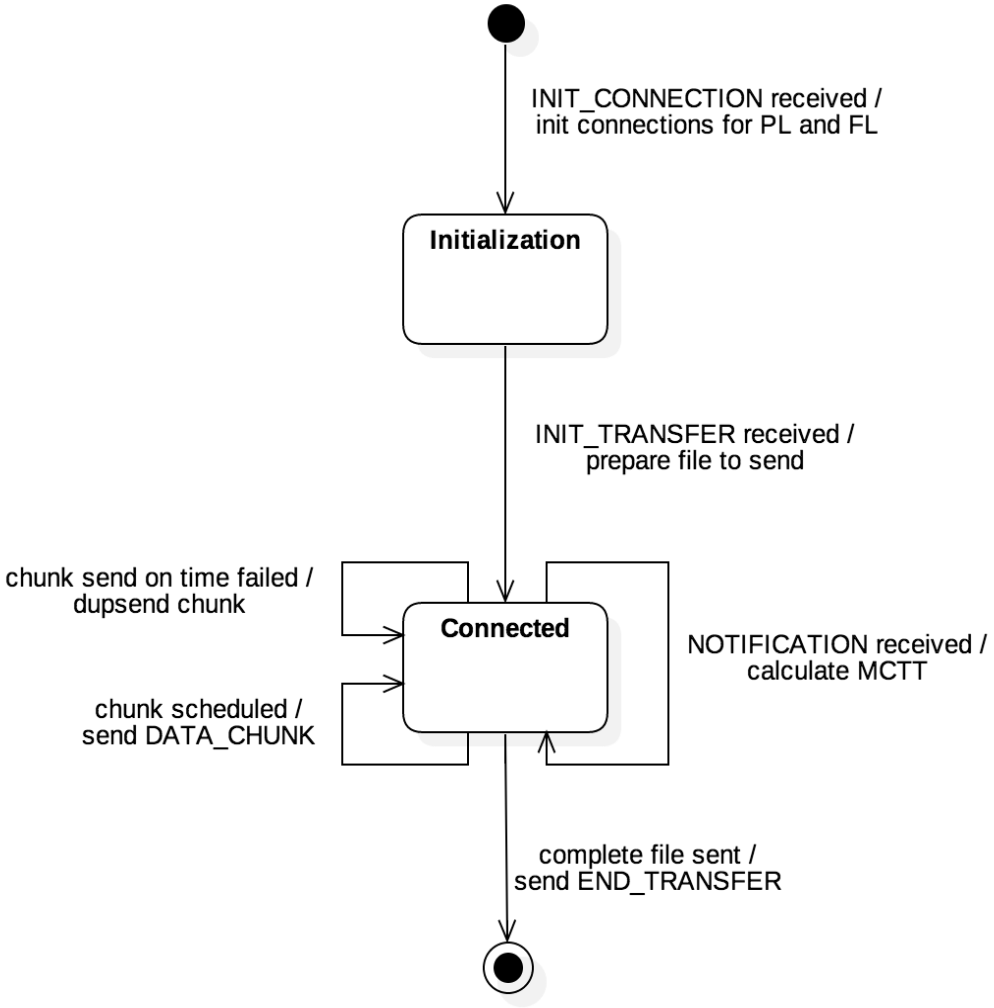

Figure 20 Tasks Handled by The Server

3.4.1. Scenario 1: FC Throughput Equal or Larger Than Chunk Usage Rate

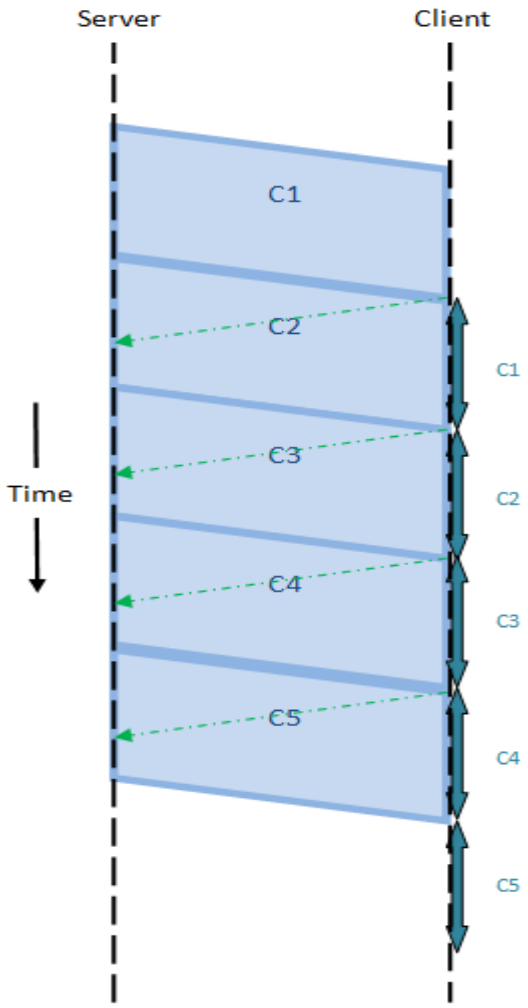

Figure 21 The Server Scenario 1

In this scenario, free connection's throughput is equal to scheduled chunk usage rate (CUR) or higher than it, as shown in Figure 21. One may think that; no special mechanism is needed in this scenario to be able to download the entire time sensitive file over free connection. On the contrary, this scenario contains and explicitly displays the first and the most basic mechanism of CATS system; how to fully utilize throughput of free connection.

3.4.1.1. Maximum Free Connection Utilization

Consider a scenario where free connection has a throughput of 1Mbit/s while time sensitive stored file to be streamed has a 1Mbit/s scheduled usage rate. In case of 100% utilization of free connection, it is possible to download all chunks over free connection

without any money cost by 0% usage of paid connection. However, this is a challenging situation because file to be sent is time sensitive and underlying network delays the transmission of chunks.

One of the simple methods to utilize free connection fully is continuously writing all chunks' bytes to the free connection's TCP socket. This method may be successful if the file to be sent isn't time sensitive and the goal is to minimize the complete file transmission time. However, in the context of this thesis, file to be sent is time sensitive and chunk *n* has higher priority than chunk *n+1, n+2*. In this scenario, despite their scheduled usage time has not come yet, if many chunks are sent back to back it is not possible to pull any of them back from TCP socket in case the free connection throughput is not sufficient to deliver chunks before their scheduled usage time.

Another simple method can be employed to utilize free connection while sending chunks is stop and wait to prevent congestion of underlying TCP connection. However, if the underlying free link has large jitter, stop and wait method will result in less utilization of free connection because of the wait period caused by network.

CATS proposes a free connection utilization method which avoids underlying network congestion while utilizing free connection throughput close to maximum even if network delay is high. CATS limits the maximum number of chunks to be sent over free connection at any time by limiting the free connection's TCP socket sender window size. The socket buffer size is set to twice the chunk size. That is, a maximum two chunks can be written to TCP socket of free connection. Therefore, maximum number of bytes in flight limited to two chunks. In the meantime, as long as the round trip delay is lower than the chunk transmission time, CATS can utilize free connection's throughput at a rate close to 100%. To be able to enable proper operation, chunk time in CATS is chosen as 1 second. Thus, as long as sum of network delays is less than 1 second, CATS can utilize the entire bandwidth of free connection.

3.4.2. Scenario 2: Free Connection Speed Lower Than Chunk Usage Rate

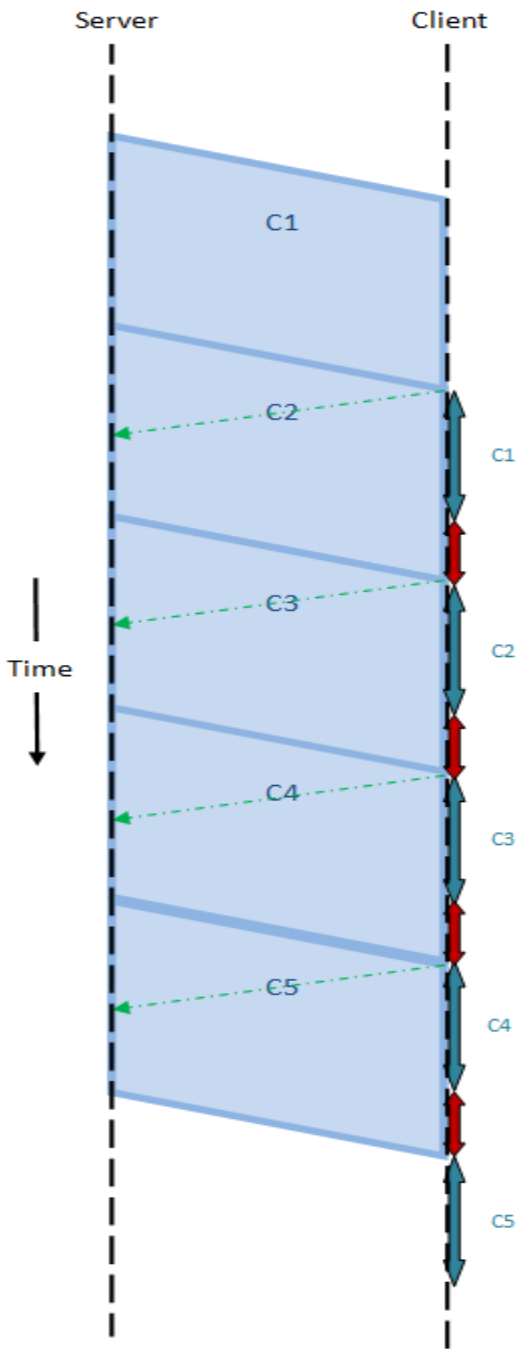

Figure 22 The Server Scenario 2

In the previous scenario, free connection speed was sufficient to stream all data over it. However, as it is visualized in the Figure 22, in this scenario, free connection throughput is lower than chunk usage rate and this situation results in client pause.

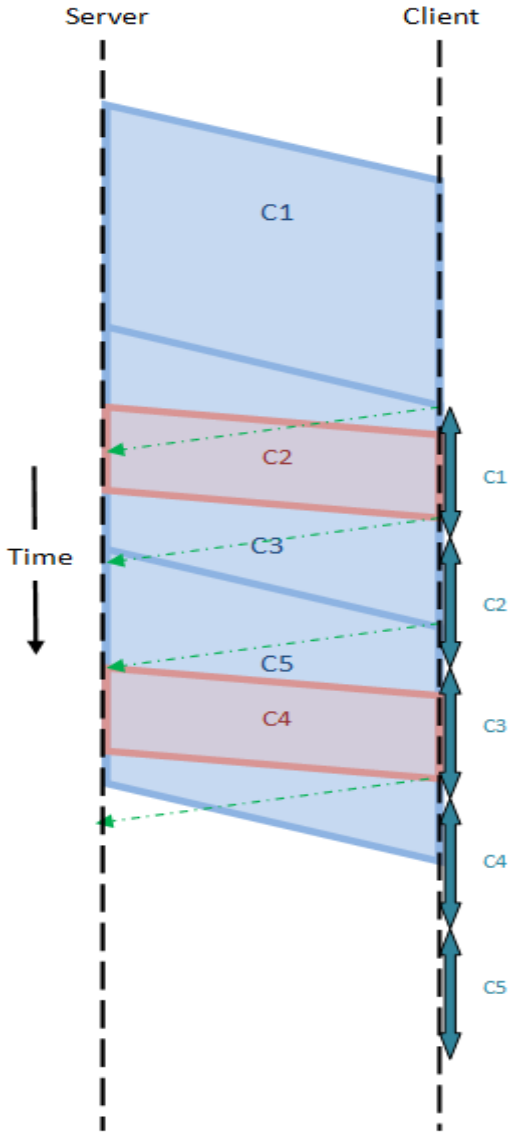

Figure 23 CATS Solution to Scenario 2

One of the main goals of CATS is minimizing the client pause and to be able to achieve this, the chunks which cannot arrive before their scheduled usage time should be downloaded over paid connection instead of free connection, as it is shown in Figure 23. Therefore, paid connection can be used for some chunks in this scenario. CATS decides which chunk should be sent over which connection to stream data in a way causing minimum client pause and minimum paid connection utilization. To this end, MCTT estimates are used. In this section, the mechanisms used by CATS for this purpose are explained in detail.

3.4.2.1. MCTT and SCTT Calculation

In CATS, MCTT (Measured Chunk Transfer Time) values are the only way to measure the throughput of underlying connection. MCTT calculation is made after receiving each notification packet as shown in Figure 24.

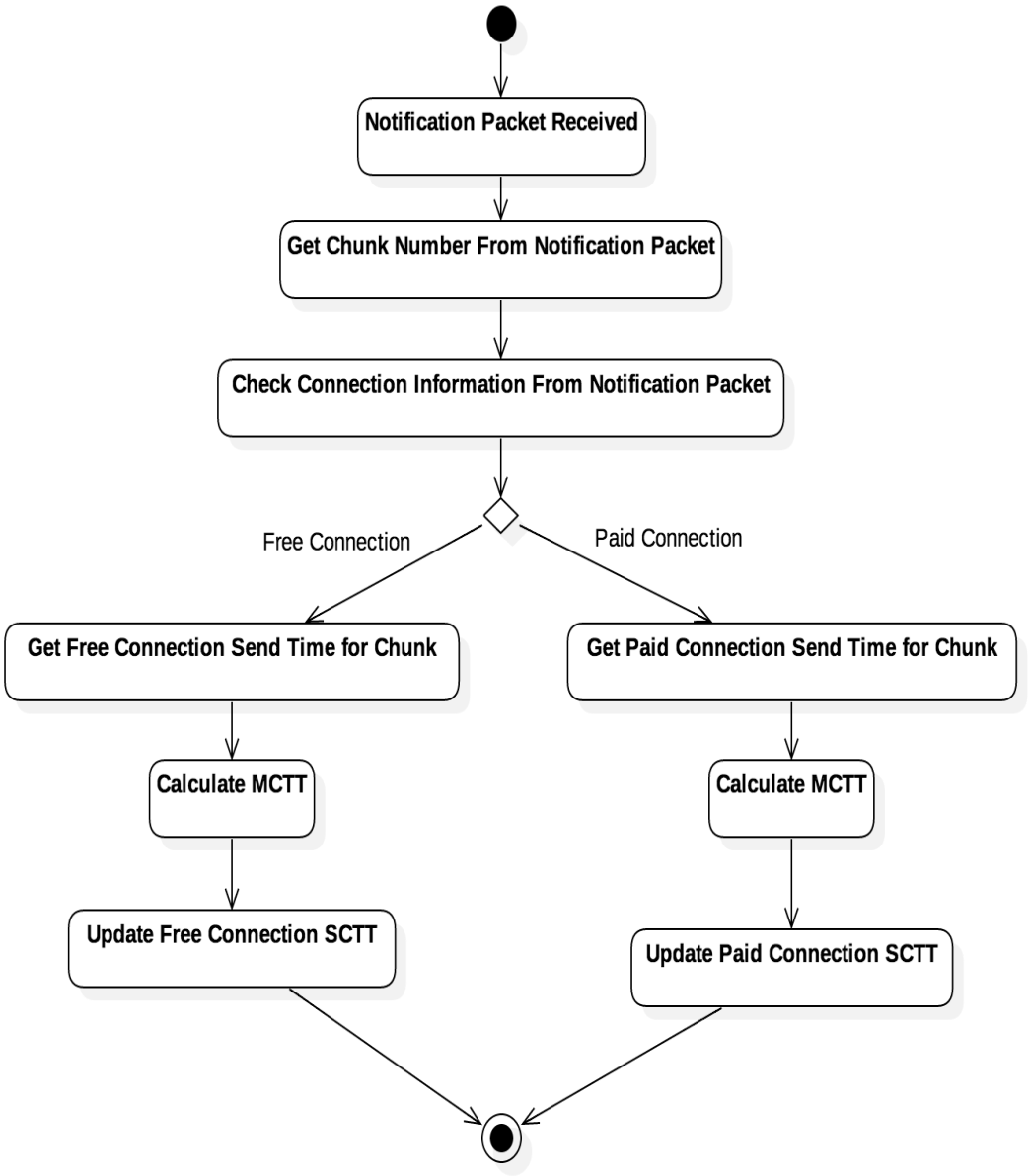

Figure 24 MCTT and SCTT Calculation Mechanism for The Server

Client gets and saves wall clock time when any chunk is sent. Chunks can be sent over either free or paid connection, or both, so system gets and saves the sending times for each chunk for each connection.

As described in the communication between client and server section, notification packet sent by client contains two information; chunk number and the connection through which the chunk is sent. Thus, server can calculate time difference between sending time and notification receive time to determine MCTT value as shown in Figure 25.

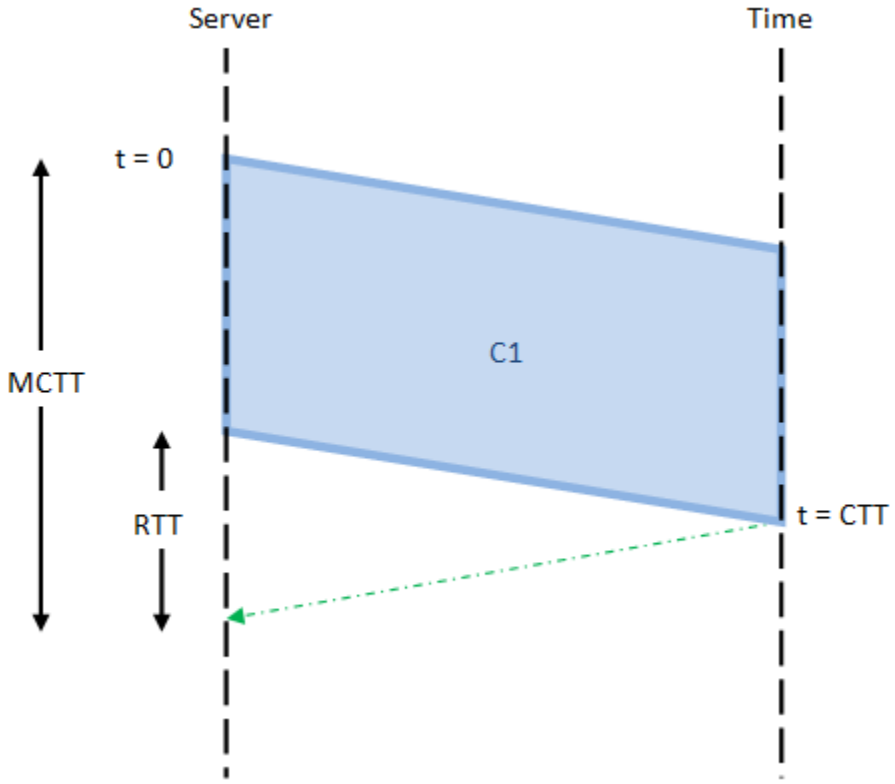

Figure 25 MCTT Calculation with CTT and RTT

MCTT contains both CTT (Chunk Transfer Time) and propagation delay for notifications packets as it is shown in the Figure 25. It is assumed that RTT (Round Trip Time) value is much smaller than MCTT and MCTT is calculated for each chunk as:

$$MCTT = CTT + RTT/2 \quad (3)$$

RTT value is ignored, so;

$$MCTT \cong CTT \quad (4)$$

MCTT values are rapidly changing due to changing network conditions and it is very challenging to make estimations depending on individual measurements. Smoother MCTT values are needed for CATS and this necessity brings us to Smoothed Chunk Transfer Time (SCTT) Calculation which is the exponentially weighted moving average of MCTT measurements. Jacobson's Algorithm [2] is employed for the calculation of SCTT as:

$$SCTT = \alpha * SCTT + (1 - \alpha) * MCTT \quad (5)$$

where α value is used as 0.875 and it is obtained with heuristic experimental results.

3.4.2.2. Chunk Scheduler

Chunk scheduler is the most important part of CATS. In this section, the chunk scheduler utilized in the proposed algorithm will be explained in detail.

Chunk scheduler uses two different tag windows to schedule chunks. Both windows are maintained by the chunk scheduler and the contents are used by chunk sender threads. Both windows have sliding window structure and region of interest contains information for the chunks which are sent but not notified or not sent yet. As chunks are sent and notified, windows slides and new region of interest contains new chunk numbers which are sent but not notified or not sent yet.

1	2	3	4	5	6	7	8	9	10

Figure 26 An Example Portion of Sliding Window and Region of Interest

In Figure 26, an example portion of sliding window is visualized. Numbers represent the chunk numbers, and grayed region is the current region of interest for the sliding window. This means that, chunk numbers with 1, 2 and 3 have been sent and notified, so algorithm has nothing to do with these chunks. Chunks 9 and 10 are out of the scope for the time being and thus they are not scheduled and don't contain any tagging information yet. Chunks 4, 5, 6, 7 and 8 are in the region of interest of the chunk scheduler.

The connection tag window contains the connection type information, that is, which chunk will be sent over which connection. As connection tag window slides and new untagged chunks are included in the region of interest, algorithm makes estimations for these chunks. If the chunk will be able to deliver to the client before its scheduled usage time when it is sent over free connection, it is tagged as free connection, otherwise it is tagged as paid connection.

The other window is the command tag window and it includes the information whether the chunk in the window must be sent immediately or the transmission must be delayed. Sender threads read the command tag window and if "wait" command is set for any

chunk, it will not send the chunk immediately, otherwise if “send” command is tagged, related chunk is sent immediately if the underlying socket is available.

Region of interest of both tag windows slide concurrently and same chunk numbers are included in the region of interest of both connection and command tag windows. If any chunk number is tagged or retagged in one of the tag windows, the same chunk number is also tagged or retagged in the other tag window.

Sizes of region of interest for tag windows are initially set to 5 and this value is chosen after several experiments conducted. Size can’t be smaller than 5 but it can increase when needed. The only condition checked to increase size is the chunk count which is tagged as free connection in the region of interest of the connection tag window. It is aimed to utilize free connection throughput at a rate close to 100% during file streaming and if any of the chunks is not tagged as free connection in the region of interest of connection tag window, sender threads cannot send any chunk over free connection, so this results in wasting the resources of free connection. So, window size is increased up to a point at least one the chunk is tagged as free connection.

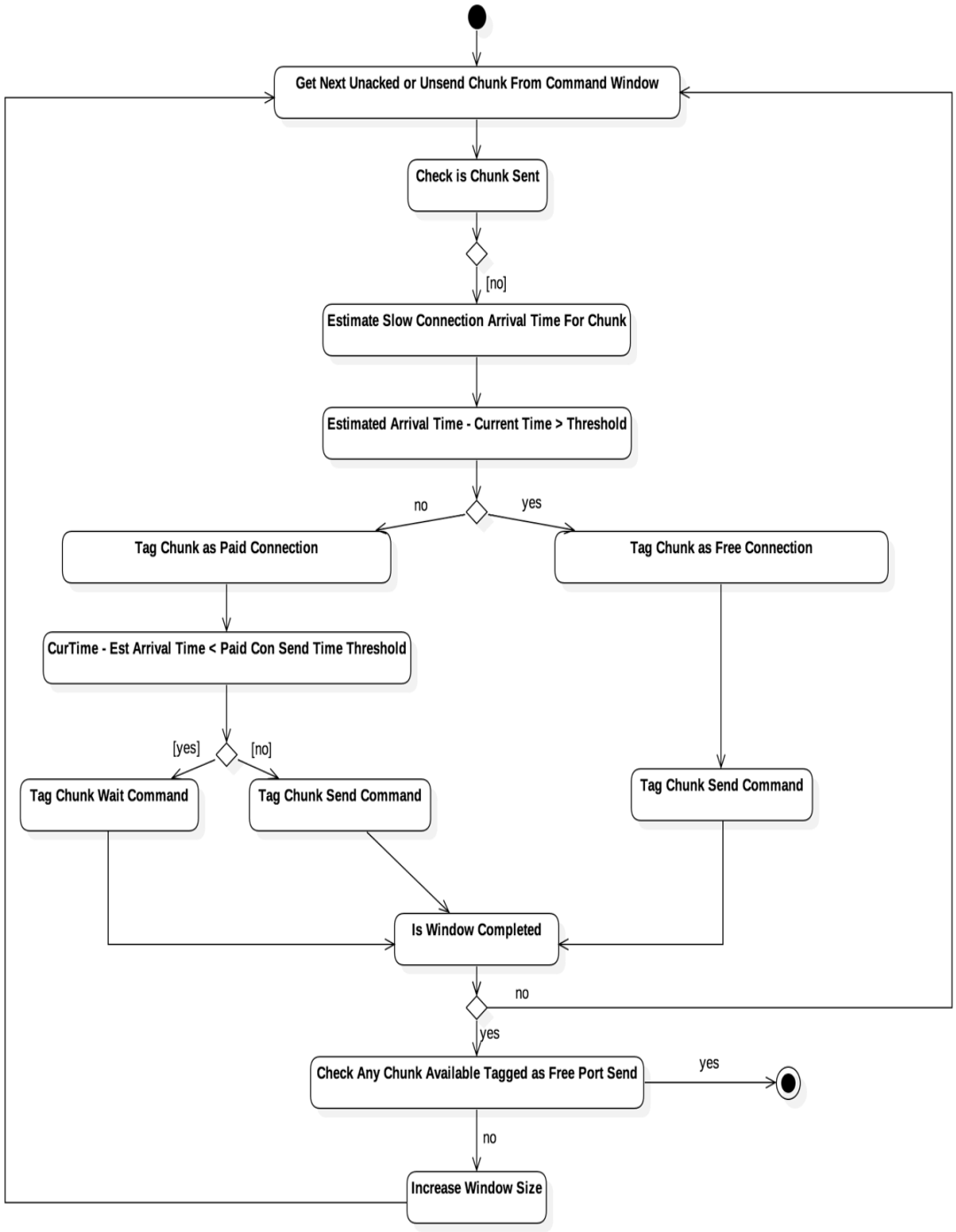

Figure 27 Chunk Scheduler Window Tagging Procedure

In Figure 27, activity diagram for chunk scheduler tagging operation is presented. At each iteration, chunk scheduler reads chunk connection and command tag windows and makes estimations for the chunks included in the region of interest of the sliding window. Chunk scheduler periodically reads chunk command and connection windows and tags each one again and again as the network conditions change.

Figure 28 Time T Connection Window

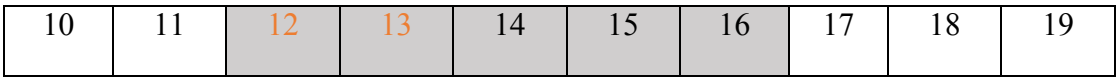

Figure 29 Time T Command Window

Figure 30 Time T + 1 Connection Window

Figure 31 Time T + 1 Command Window

In Figure 28-31, an example sequential tagging case is delineated. Shaded region is the region of interest for all windows. In Figure 28 and Figure 29, information included in command and connection tag windows on Time T are represented by some color coding. In Figure 30 and Figure 31, information included in command and connection tag windows on time T + 1 are illustrated by different colors. Blue color in the connection tag windows represents the free connection tag, while red color represents the paid connection tag. Orange color in the command tag windows represents the chunk sent but not notified tag, while black color represents chunk is not sent yet tag. In Time T, chunks 12 and 13 are sent over free connection but their notification packet is not received yet. Chunks 14, 15 and 16 are tagged as to be sent over free connection after the estimations of chunk scheduler but they are not sent yet. In time T + 1, chunk 12 is notified, both command and connection tag windows slide and new chunk number 17 is in the region of interest. As it is mentioned before, chunks are not tagged only once, all chunks in the region of interest are tagged periodically to be able to adapt to the rapidly changing network conditions. For example, chunk 14 is tagged as free connection at time T. But at Time T + 1, new notification packet received for chunk 12 and SCTT value is updated. Therefore, with the updated SCTT value connection tag information for chunk 14 is altered. At Time T, scheduler estimated chunk 14 as can be sent over

free connection as it can arrive before its scheduled usage time. However, after receiving a new notification for chunk 12 and updating SCTT with this notification, sending chunk 14 from free connection would result client pause, therefore chunk 14 is retagged as to be sent over paid connection.

As it is shown in Figure 27, chunk scheduler makes estimations for the chunks which are in the region of interest of tag windows and in the light of these estimations it tags command and connection tag windows. Whether a chunk will reach to the client before its scheduled usage time or not is the only deciding factor to tag chunk connection tag window. At each iteration, for each chunk to be tagged in the region of interest, scheduler employs the following formula:

$$Estimated\ Usage\ Time = UsageTimeOffset + Chunk\ Number * Chunk\ Time\ Period \quad (6)$$

Client sends time synchronization packets to the server as explained in the client section. Estimated scheduled usage time for each chunk is calculated with the usage time offset value which received with the time synchronization packets sent by client. Server needs to calculate and know exact scheduled usage time for each chunk to be able to decide the connection which chunk will be sent over. According to the formula given above, chunk usage time offset value (Equation 2) received from the client ensures that the server calculates exact scheduled usage time for each chunk.

$$USCTT = \alpha * SCTT + (1 - \alpha) * TEMCTT \quad (7)$$

$$Estimated\ Arrival = Current\ Time + USCTT \quad (8)$$

$$Difference = Esimated\ Usage\ Time - Estimated\ Arrival \quad (9)$$

To be able to decide the connection which a chunk will be sent over, estimated usage time for each chunk must be compared to estimated arrival time as it is given in equation 9. Estimated arrival time is calculated with the updated version of the smoothed chunk transfer time (SCTT) value; updated smoothed chunk transfer time (USCTT). SCTT is calculated with MCTT values and it has the information of all past MCTT values but it doesn't have the information of time elapsed MCTT (TEMCTT) values of chunks which are sent but not notified yet. USCTT is calculated by the algorithm only at the start of each window tagging iteration. USCTT value is calculated with the combination of SCTT value of free connection and not notified but sent chunk's current Time Elapsed MCTT values (TEMCTT). Not notified but sent chunks don't have calculated MCTT value yet but their current TEMCTT value can be calculated with the difference between current time and their send time. If TEMCTT value of not notified but sent chunks is greater than the calculated SCTT, in this case USCTT includes TEMCTT value too. This feature added to the algorithm to be able to cope with sudden, huge deterioration of underlying network. For example, at time T, free connection may be completely closed or its throughput may be ten times lower than the throughput of time T – 1 because of

underlying network deterioration. At this point, algorithm may not get notification packets for free connection for a long time because of any of the sent chunks are not transmitted. Therefore algorithm assumes that SCTT value is the same as before and keeps sending chunks from free connection with the estimations based on this obsolete SCTT value. In this case, free connection is utilized inefficiently, chunks which will never be able to reach before their scheduled usage time will be sent over it and free connection will be more congested. However, with TEMCTT value calculation feature, even if underlying network for free connection is as bad as to cannot transmit any sent chunks, algorithm can continuously update its SCTT with TEMCTT values before tagging any chunk. In the experimental results section, it will be represented that, sudden bitrate changes in free connection doesn't affect the outputs of the algorithm, this feature is one of the most important reasons of this consequence.

Difference between estimated playout time and estimated arrival is calculated as it is given in equations 7 to 9. If calculated difference value is greater than the predetermined threshold value, chunk is tagged as free connection, otherwise chunk is tagged as paid connection. At this point, predetermined threshold value is used to the cope with rapid changes of underlying network. Threshold value is calculated as:

$$FC\ Tag\ Threshold = Chunk\ Time\ Period * 4 \quad (10)$$

CATS aims to minimum usage of paid connection. Because of this reason, if any chunk is tagged as paid connection in the connection tag window, this chunk is not sent immediately, due to possible throughput increase in free connection which makes sending this chunk over free connection possible. It kept waiting before send until the last critical time threshold value is reached. For example, if it is estimated that chunk n can't reach before its scheduled usage time if it is sent over free connection so it is tagged as paid connection in the connection tag window. In the meantime, if it is sent immediately over paid connection, it will be received by client way before its scheduled usage time. In such scenarios, chunk's send command is tagged as "wait". Because in the context of this thesis study it is considered that, throughput of free connection is very variable. Throughput of free connection may increase in time and this can be detected with the receipt of notification packets of chunk n-1, n-2 and sending chunk n before its scheduled usage time over free connection may be possible. Therefore, chunks which are tagged as paid connection, are kept waiting before sending until a threshold value is reached, in case of free connection throughput increases and chunks can be retagged as free connection. Thus, paid connection usage can be minimized.

Paid connection send command tag threshold value calculation formula is:

$$PC\ Send\ Command\ Threshold = \frac{Chunk\ TimePeriod}{2} \quad (11)$$

If the difference between estimated arrival and estimated chunk usage time is smaller than threshold, chunk command tag window is tagged as “send”, otherwise chunk command tag window is tagged as “wait”.

In the Figure 27, it is stated that, chunks which tagged as free connection in the connection tag window are always tagged as “send” in the command tag window. Because CATS aims maximum utilization of free connection.

3.4.2.3. Chunk Sender

In this section, working mechanism of chunk sender threads, which read the output of chunk scheduler and send chunks to the client, are explained.

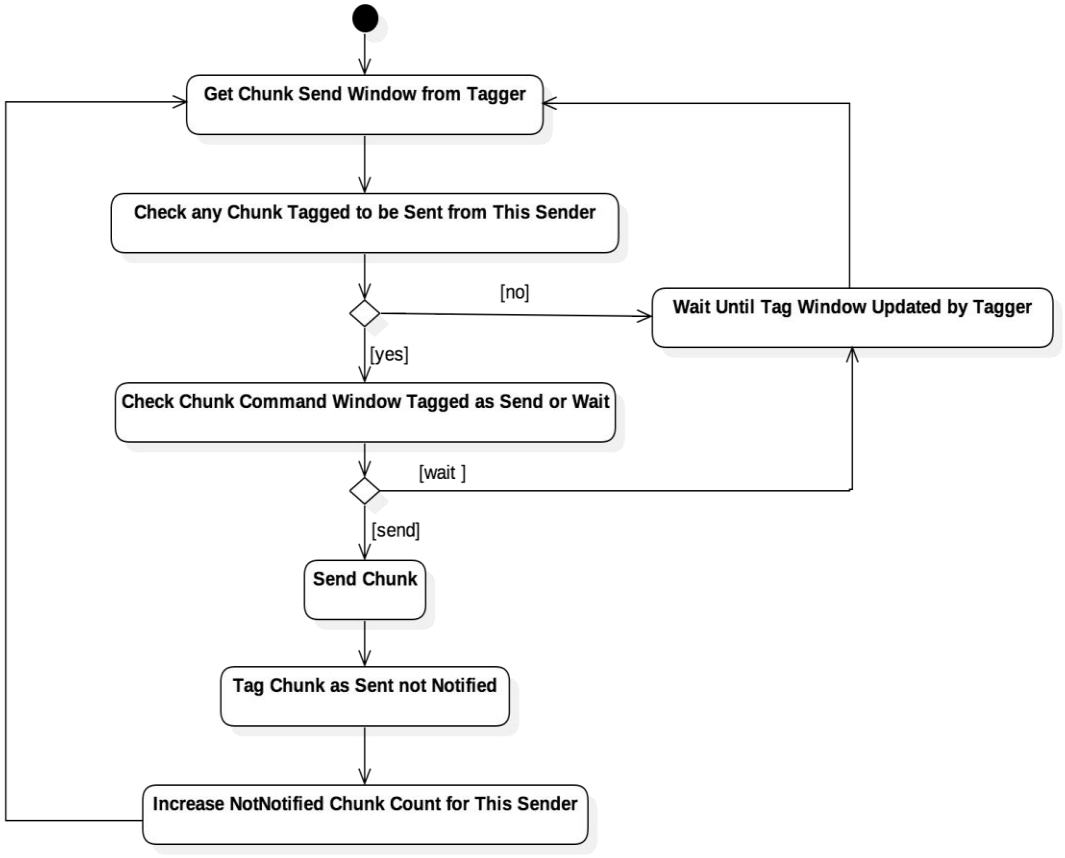

Figure 32 Chunk Sender Threads Working Mechanism

In Figure 32, working mechanism of sender threads are illustrated. In CATS, each connection, free and paid connections, have their individual threads. Both threads are identical, each of them just belonging to one of the connections. Threads read outputs of chunk scheduler that are the tagged connection and command tag windows. If any chunk

is tagged in the connection tag window belongs to thread's connection, command window is checked, and if value of chunk's command tag is "send", chunk is written to the connection's TCP socket and sent from server to client.

3.4.3. Scenario 3: Sudden Free Connection Throughput Decrease

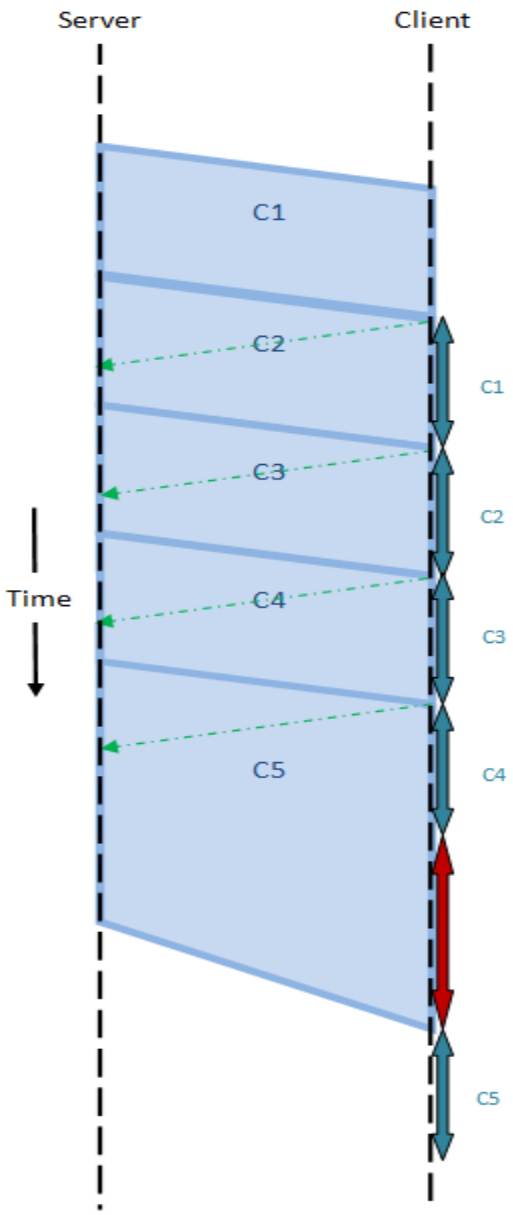

Figure 33 The Server Scenario 3

As it can be seen in Figure 33, sudden throughput decrease on free connection may cause client pause. In some cases, any chunk may be scheduled and sent over free

connection, but because of sudden throughput decrease on free connection, that chunk may not arrive before its scheduled usage time and this will result with the client pause. In the Figure 33 it is shown that, chunk 1, 2, 3 and 4 are scheduled as to be sent over free connection and all of them arrived before their scheduled usage time. Chunk 5 is also scheduled to be sent over free connection and it is estimated as will arrive before its scheduled usage time. However, because of the reasons like packet loss, sudden connection throughput decrease, arrival of chunk 5 takes much more time than estimated and this caused with the client pause. CATS include the duplicate chunk send mechanism to cope with such situations.

3.4.3.1. Duplicate Chunk Send Tagging Mechanism

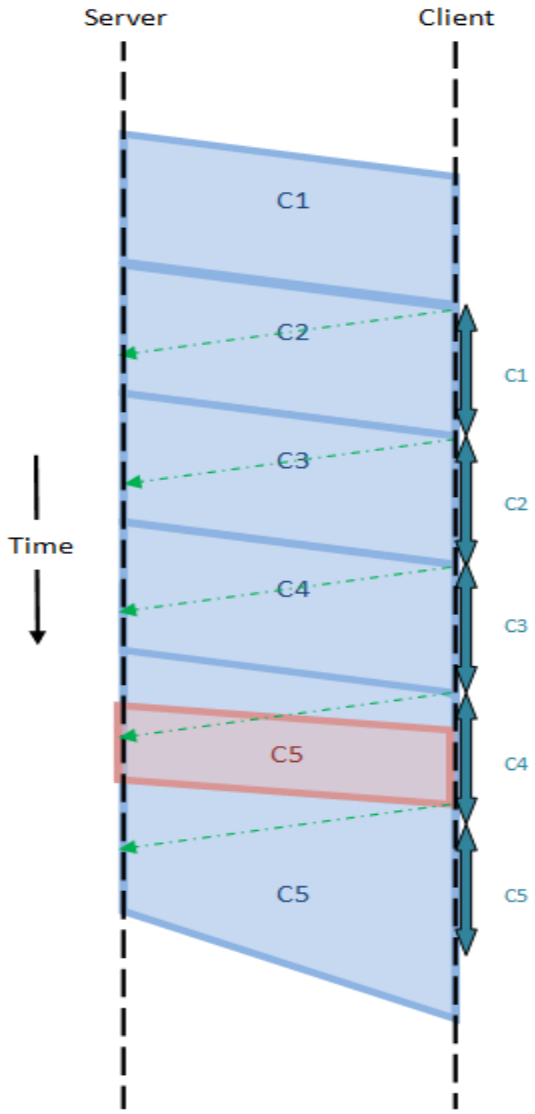

Figure 34 Duplicate Chunk Send Solution for Scenario 3

One of the main goals of CATS is maximum utilization of the free connection with an efficiency ratio near 100% which means that, each chunk should be sent only once over one connection. Downloading a chunk twice over both free and paid connection is an undesirable case and accurate scheduling is aimed to prevent this undesirable consequence. However, in some cases, underlying free connection's conditions may change rapidly because of packet loss, sudden throughput decrease therefore, scheduled and sent over free connection chunks may not arrive before their scheduled usage. So in such cases, chunks which are sent but then estimated as not will arrive on time, are re-sent over paid connection. As illustrated in Figure 34, chunk 5 is sent over free

connection, but its arrival takes much more time than estimated and server cannot receive its notification packet before expected time. Thus, to prevent client pause it is duplicate sent over paid connection.

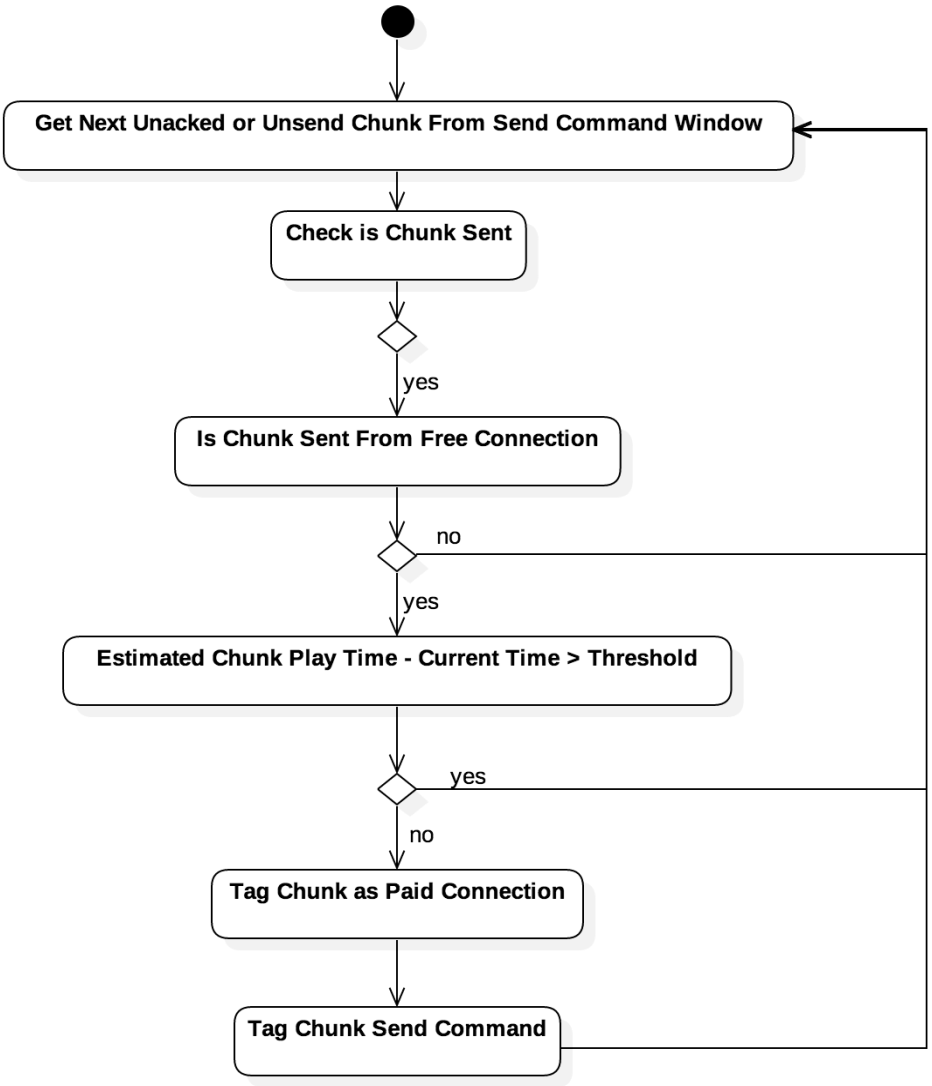

Figure 35 Duplicate Chunk Send Tagging Mechanism

In Figure 35, duplicate chunk send tagging algorithm is illustrated. Duplicate chunk send tagging covers the chunks which are sent over free connection, but not notified before their scheduled usage time. These chunks are sent over paid connection once more. Sent but not notified chunks are still in the scope of tagging windows. Hence, chunk scheduler continuously checks these chunks in case of duplicate send is needed to prevent client pause. At each iteration, chunk scheduler calculates difference between estimated scheduled usage time of chunk and current time for the chunks which are sent

over free connection and not notified yet. If calculated difference value is smaller than the duplicate send threshold value, chunk is retagged to be duplicate send over paid connection and sent immediately to avoid client pause. The threshold value which is obtained with heuristic experimental results is defined as below;

$$Duplicate\ Send\ Threshold = PCSCTT + \frac{Chunk\ Time\ Period}{2} \quad (12)$$

3.5. Implementation of CATS

It is aimed to implement CATS in a platform independent way so that any client application can easily utilize it. Java Programming Language is a portable object oriented language that lets application developers "write once, run anywhere". Hence, the Java programming language is used to create a prototype implementation of CATS. In this section, implementation details for two fundamental parts; client and server are explained by means of UML Class Diagrams.

3.5.1. The Client

The Client of CATS is responsible for tasks like; system initialization, chunk receiving, cost calculation, sending notification packets and time synchronization. All of these tasks are performed with client and its auxiliary Java classes.

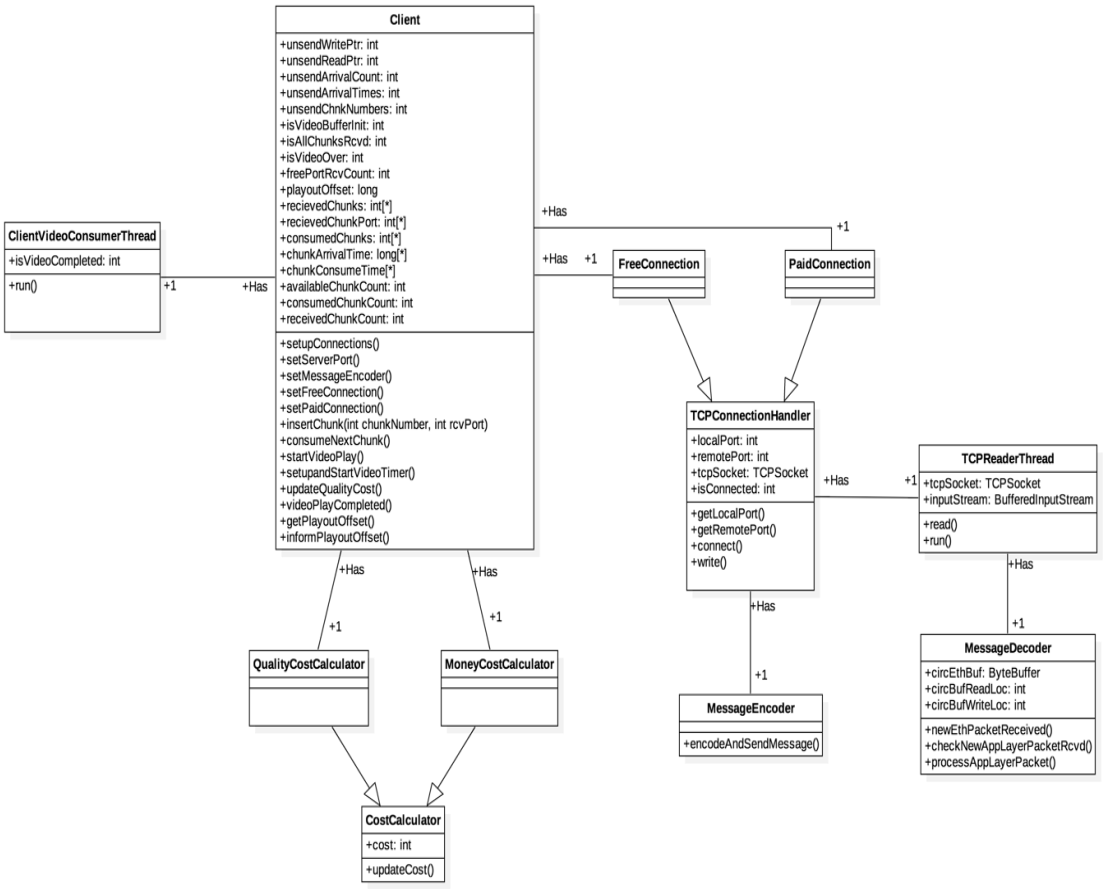

Figure 36 UML Class Diagram for the CATS Client

In Figure 36, UML Class Diagram for the Client is given. Client class is the main class for client side implementation. Client class performs its tasks by help of other helper classes. One of these classes is client file user thread. CATS currently supports the use case scenario of time sensitive data streaming. To be able to simulate a chunk user, client user thread class is created. This class continuously consumes received chunks in a time ordered manner.

CATS operates and makes real time decisions during time sensitive file streaming. Because of this reason, optimization of the implementation is very crucial. One of the most time consuming tasks for proposed system is TCP Socket operations. Excessive numbers of bytes are read, written and interpreted during the system operation. All of these socket operations are assigned to the free connection and the paid connection classes for each connection. The free connection and the paid connection classes are identical in terms of implementation and they are inherited from TCP Connection Handler parent class. TCP Connection Handler class is responsible for handling TCP socket connection, reading, writing bytes from socket etc. All of these socket operations are handled also with the help of other auxiliary classes. One of these classes is TCP

Reader Thread class. This class continuously polls related socket and checks if there are any incoming bytes. Socket polling and reading operation is implemented with a non-blocking method. TCP Reader thread polls socket during a predefined timeout value, if it receives as many bytes as predefined value, it polls again the related socket otherwise it waits during the predefined timeout value. Thus, underlying processor is not occupied continuously, if there are available bytes, they are read from the socket otherwise processing power released during the timeout value for any other tasks.

TCP Reader Thread forwards read bytes to Message Decoder class. Message Decoder class is responsible for interpreting the contents of the received packets. It has a circular buffer structure and continuously puts bytes received from TCP Reader Thread to its circular buffer. If bytes are interpreted as a complete command, length and data application layer message, it transmits these messages to the client class.

Client also responsible for sending packets to the server and these messages also must be in the correct format. Message Encoder is the class which is created to handle this task. Message Encoder encodes messages to be sent in the correct format before sending to the server.

Money and quality cost calculators are the measurement units for system success rate. These classes will not be placed in the final application, but for system development and verify phases, they are utilized as output creator of system success rate.

3.5.2. The Server

Server of CATS is responsible for the tasks like chunk scheduling, chunk sending etc. In this section, classes implementing the server side are explained.

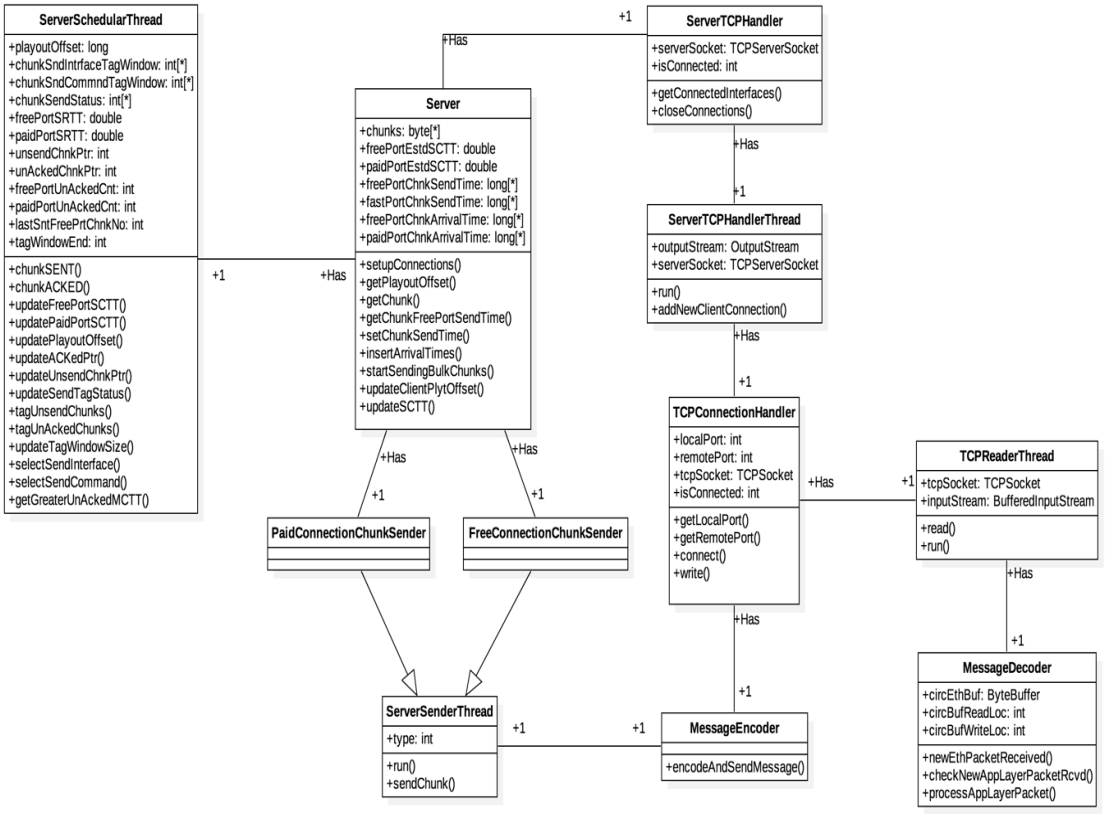

Figure 37 UML Class Diagram for the CATS Server

In Figure 37, server class diagram is given. Server class is the main class for implementation of the server. Server class executes tasks of server side in the proposed algorithm by organizing other supporting classes. Chunk scheduling is handled in Server Scheduler Thread class which inherits Java Thread class and it continuously runs during the time sensitive file streaming operation. In CATS, chunks are continuously tagged to be able to keep track of changing network environments. Hence, chunk scheduler logic part is implemented as an individual thread and it continuously polls network conditions and tags, retags chunks' send interface and command according to network conditions.

Server has chunk sender threads for each individual connection; free and paid connections. As it can be seen from Figure 37, server class has Paid Connection Chunk Sender and Free Connection Chunk Sender classes. Both classes are identical and they inherit Server Sender Thread parent class. Only difference between Paid Connection Chunk Sender and Free Connection Chunk Sender is type variable. Type variable is set due to distinguish between paid connection and free connection. If type is set as free connection sender, class sends chunks tagged as to be sent over free connection, otherwise class sends chunks tagged as to be sent over paid connection.

Server TCP Handler class handles the incoming connections to the server. It continuously polls for incoming TCP socket connection requests and for each established connection, it creates a TCP Connection Handler to maintain that connection during the operation. TCP Connection Handler, Message Encoder, Message Decoder and TCP Reader Thread classes works same as described in the client implementation section.

CHAPTER 4

EXPERIMENTAL RESULTS

In the context of this thesis study, experiments are carried out by using the prototype implementation of CATS to evaluate the performance of CATS under various conditions. In this section, results of experiments realized on different setups with different parameters to test different aspects of CATS are explained. The prototype is implemented by using the Java programming language. Hence, in these experiments, client and server implementations have been executed and tested on various operating systems.

CATS tested under many different network conditions and Dummynet [24] is employed to simulate different network conditions along the experiments. Dummynet is a live network emulation tool, originally designed for applications including bandwidth management. It simulates/enforces queue and bandwidth limitations, delays, packet losses, and multipath effects.

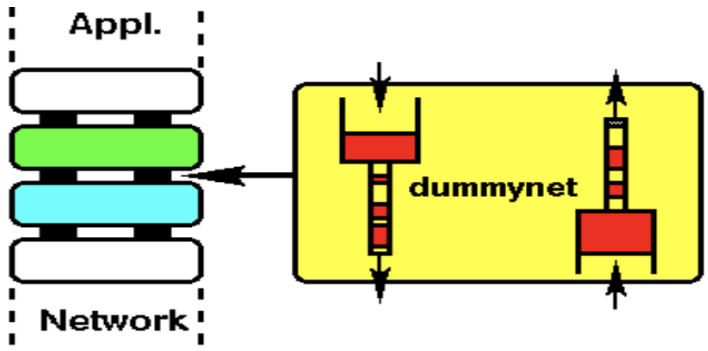

Figure 38 Dummynet As Network Emulation Tool [25]

Dummynet runs within different operating systems; FreeBSD, OSX, Linux and Windows and works by intercepting selected traffic on its way through the network stack, as it is shown in the Figure 38, and passing packets to objects called pipes which

implemented as set of queues, a scheduler, and a link all with configurable features. Features of Dummynet like bandwidth limitation, packet loss and delay are employed for the experiments of CATS.

Quality Cost, Free Connection TCP Throughput (FCTT) and Money Cost are the metrics to evaluate system success on each setup. Quality Cost corresponds to the total measured client pause duration in milliseconds. Money Cost is the number of chunks downloaded over paid connection. Free Connection TCP Throughput stands for the average throughput of the free connection during the streaming. It is calculated as:

- FCTT: Free Connection TCP Throughput
- LT = Wall Clock Time of Latest Chunk Downloaded Over Free Connection
- FT = Wall Clock Time of First Chunk Downloaded Over Free Connection
- CC = Chunk Count Downloaded Over Free Connection
- MBS = Mbit Size for Each Chunk

$$\text{FCTT} = \frac{CC * MBS}{LT - FT} \quad (13)$$

4.1. Setup 1

In this setup, CATS is tested in an isolated network in which the client and the server are connected via an access point. The purpose of testing CATS in a closed network is to analyze the appropriateness of the proposed fundamental mechanisms without any external interference.

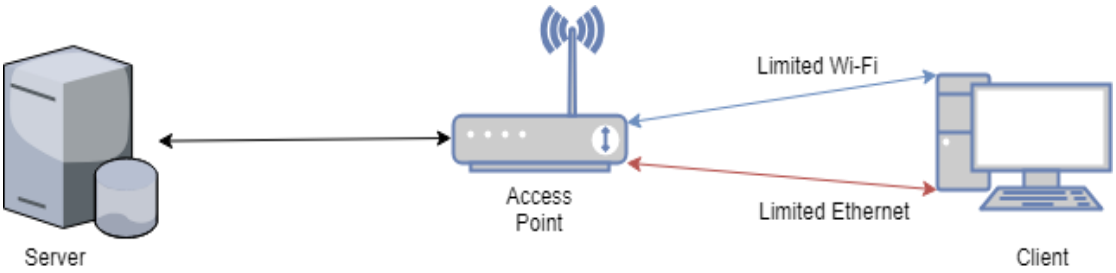

Figure 39 Experimental Setup 1

Client implementation is executed on OSX operating system, while server implementation is executed on Windows 7 operating system along the experiments done on setup 1. Both computers are located at the same physical location as it is shown in Figure 39. The client and the server connected to each other over 10 Gbit/s wireless access point. The client connected to the access point over two different links which are

Ethernet and Wi-Fi links while the server connected to access point over Ethernet link. Ethernet connection simulates the paid connection while Wi-Fi connection simulates the free connection. Ethernet and Wi-Fi connection throughputs of the client are limited to predefined values by using Dummynet to be able to analyze outputs of different experimental scenarios. Two different experiment scenarios are realized at this setup, and results of both will be analyzed in detail.

4.1.1. Experiment Scenario 1

Ten different tests are performed in this scenario. Maximum throughput of free and paid connections which are simulated with Wi-Fi and Ethernet connections respectively, are limited to constant bit rates for each test. The same throughput limits are used for each test while different chunk usage rates (CUR) are employed.

4.1.1.1. Test Parameters

- Limited Free Connection Throughput: 3Mbit/s
- Limited Paid Connection Throughput: 10Mbit/s
- File Time Period: 10 Minutes
- Each Chunk Time Period: 1 Second
- Total Chunk Count: 600
- CUR: Different for Each Test, from 1Mbit/s to 6Mbit/s

4.1.1.2. Results

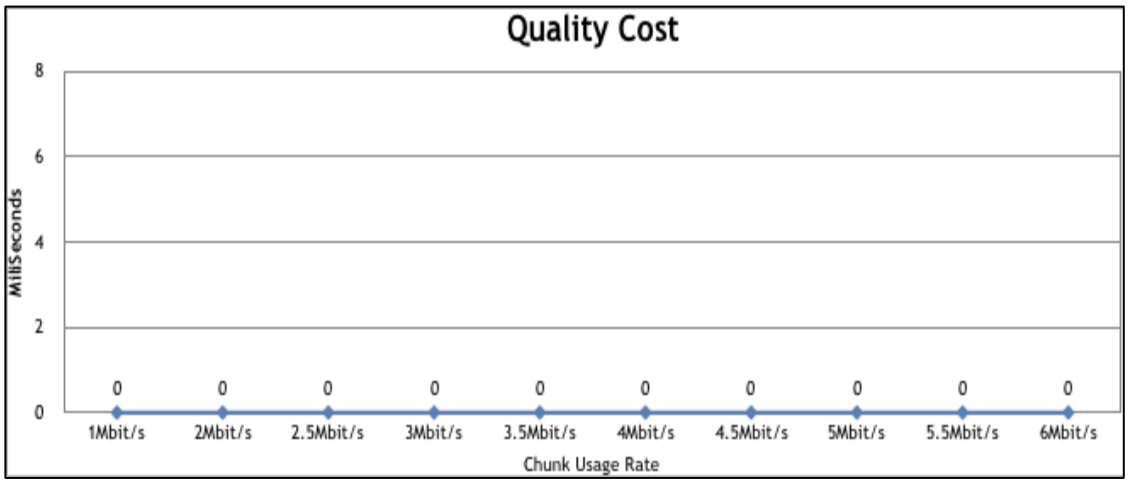

Figure 40 Quality Cost for Experiment Scenario 1

Ten different 10minute time sensitive file streaming tests with CURs; 1, 2, 2.5, 3, 3.5, 4, 4.5, 5, 5.5 and 6Mbit/s are carried out and quality cost results of them given in Figure

40. Total client pause for each test is measured as 0 milliseconds. One of the main goals of CATS is the minimum client pause and it is achieved for each 10minute streaming tests which all have different CURs. In this experiment scenario, client's free connection's maximum throughput is limited to 3Mbit/s, while paid connection's maximum throughput is limited to 10Mbit/s. It is an expected result to measure 0 millisecond client pause for the tests which have CUR lower than 3Mbit/s, since free connection's throughput is sufficient to download all chunks before their scheduled usage time. However, downloading all chunks over free connection before their scheduled usage time for the tests that have CUR higher than 3Mbit/s is not possible. As the results of test reveals, to be able to prevent client pause, some of the chunks are downloaded over paid connection in the tests which have higher CUR than free connection throughput.

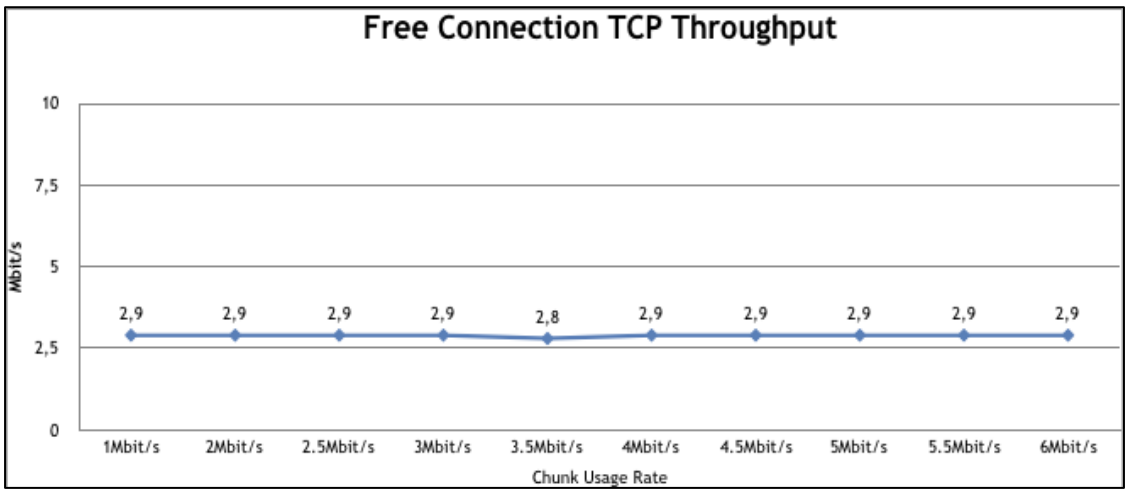

Figure 41 Free Connection TCP Throughput for Experiment Scenario 1

The client and the server directly connected to the 10Gbit/s access point and their maximum throughputs are limited via Dummynet in this setup. One of the main goals of CATS is maximum utilization of free connection. Therefore, in the result of each test, measured free connection TCP throughput is expected to be close to the maximum throughput limitation value, 3Mbit/s. Free connection TCP throughputs (FCTT) of each test are given in Figure 41 and according to the results, FCTT values for each test are close to the limitation value, 3Mbit/s. Therefore, it would be true to say, CATS achieves one of its main goals; maximum free connection utilization.

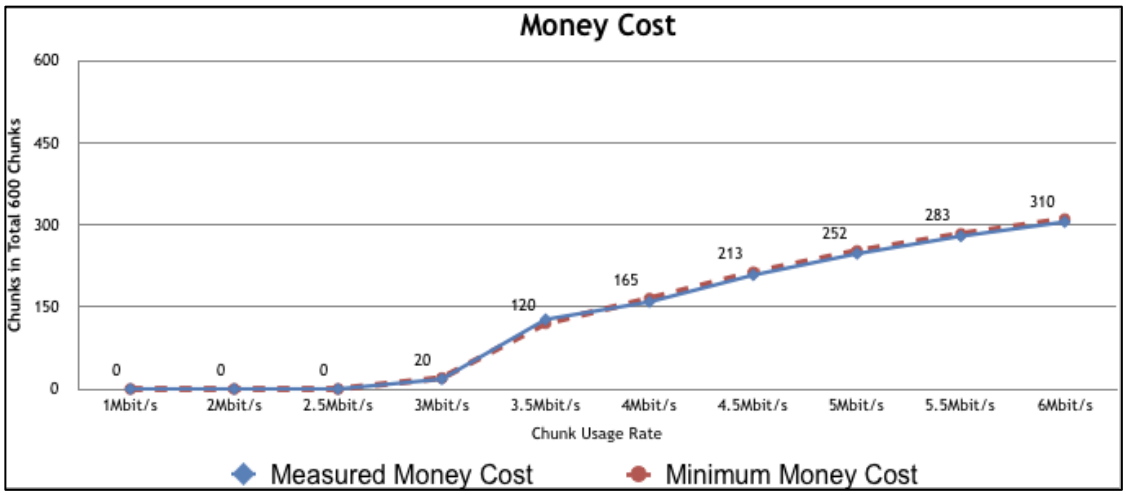

Figure 42 Money Cost for Experiment Scenario 1

Fully utilizing free connection bandwidth is not enough for complete success scenario since efficient and accurate chunk scheduling is also one of the main goals of CATS. If scheduling isn't accurate which also means that free connection utilization is not efficient, chunks sent over free connection will not arrive before their scheduled usage time, in this case, these chunks will be duplicate sent over paid connection so this results with higher money cost than expected. Therefore, to be able to evaluate the efficiency of free connection utilization the possible minimum money cost is calculated and compared with the measured one. Minimum money cost can be calculated as:

$$\text{Minimum } Money\ Cost = \frac{CUR - FCMTT}{CUR} * Total\ Chunk\ Count \quad (14)$$

Measured and calculated money cost results for the tests are given in Figure 42. Blue line represents the measured money cost while dashed red line represents the calculated money cost. As it can be seen from Figure 42, in each ten test, measured and minimum money cost results are very close, approximately the same. Minimum money cost represents the chunk count that must be downloaded over paid connection because of inadequacy of free connection throughput when compared to CUR. Therefore, obtaining close measured and minimum money cost results means that CATS utilizes free connection with a maximum rate of efficiency and it sends the chunks over paid connection only when it is not possible to send them over free connection without client pause.

Measured money cost result is zero for the tests which have chunk usage rate lower than the limited free connection throughput, 3Mbit/s. As chunk usage rate increases and exceeds the limited value of free connection throughput, CATS downloads some of the chunks over paid connection to be able to prevent the client pause. Closeness of

measured and minimum money cost for each test is the most important indicator for the success of CATS. Minimum money cost value reaches measured money cost value in some test results as it can be seen from Figure 42. This difference is caused because of initial chunk buffering as it is explained in the proposed algorithm section. Initial buffer chunk count is predefined as 10 for each test.

Firstly, quality cost results are evaluated for each test and it is observed that the client didn't pause at any of the tests as it is aimed. Secondly, free connection throughput which is limited with a constant value is compared with the measured free connection TCP throughput and it is observed that CATS utilizes free connection close to 100%. Finally, money cost result is evaluated. By combining the results of this test scenario, it would be correct to say that, CATS achieves its goals; streaming time sensitive file in a cost aware manner without any client pause.

4.1.2. Experiment Scenario 2

In this experiment scenario, all tests are performed in the setup 1. In the first experiment scenario, limited free and paid connection throughputs remained constant for each test while chunk usage rate is changed. Exact opposite scenario will be performed in this section, limited free and paid connection throughputs will be changed in each test while chunk usage rate will be constant for each test.

4.1.2.1. Test Parameters

- Limited Free Connection Throughput: Different for Each Test, 1Mbit/s to 6Mbit/s
- Limited Paid Connection Throughput: 10Mbit/s
- File Time Period: 10 Minutes
- Chunk Time Period: 1 Second
- Total Chunk Count: 600
- CUR: 5Mbit/s

4.1.2.2. Test Results

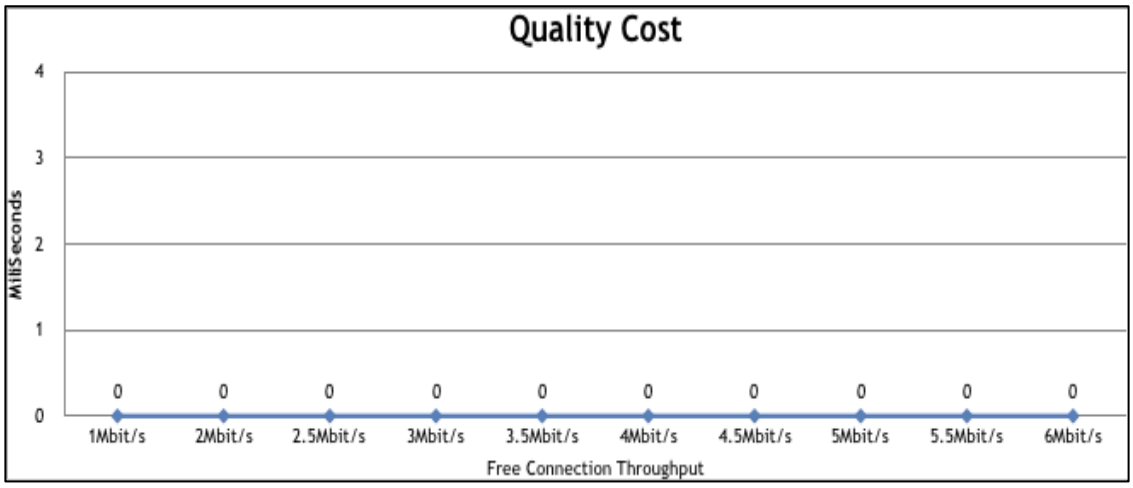

Figure 43 Quality Cost for Experiment Scenario 2

As it can be seen from the Figure 43, client didn't pause at any of the ten tests. CATS downloaded all of the chunks before their scheduled usage time for each test.

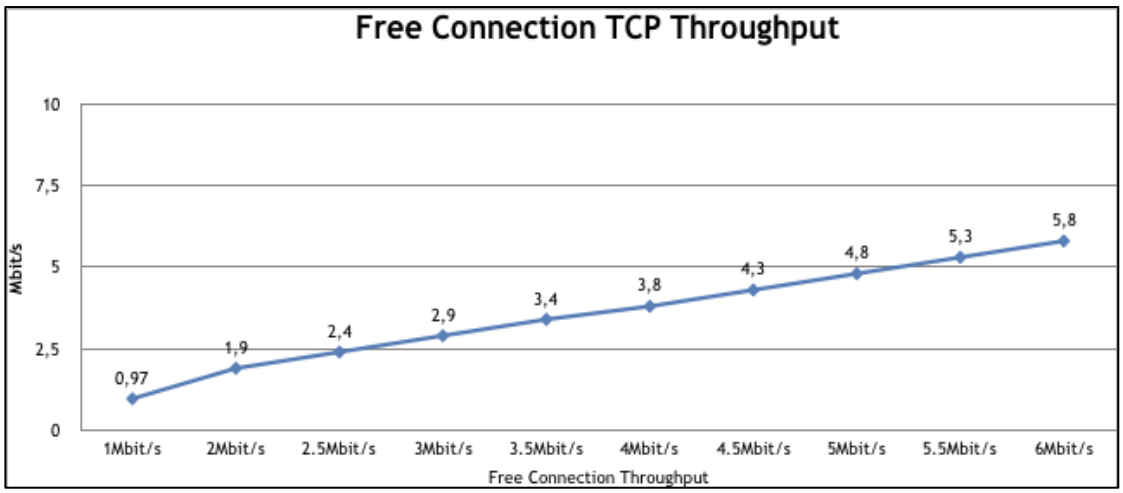

Figure 44 Free Connection TCP Throughput for Experiment Scenario1

In the previous experimental scenario, free connection throughput is limited with same value, 3Mbit/s, for each test and FCTT could be evaluated only with this constant limitation value. However, in this test scenario, because of different free connection throughput limitations employed at each test, measured free connection TCP throughput is evaluated for different limitation values. As it is explained before, client and server

directly connected in local network, over access point, and there is not any external network interference. Therefore, measuring FCTT values close to the limitation value of free connection throughput is expected. FCTT values for this experiment scenario are given in Figure 44. Free connection throughput is limited with different values at each test; 1Mbit/s, 2Mbit/s, 2.5Mbit/s, 3Mbit/s, 3.5Mbit/s, 4Mbit/s, 4.5Mbit/s, 5Mbit/s, 5.5Mbit/s and 6Mbit/s. As it can be seen from Figure 44, FCTT values are close to the limited free connection throughputs for each test. So, it would be true to say that CATS utilizes free connection close to the possible maximum for each different test which employs different throughput limitation values.

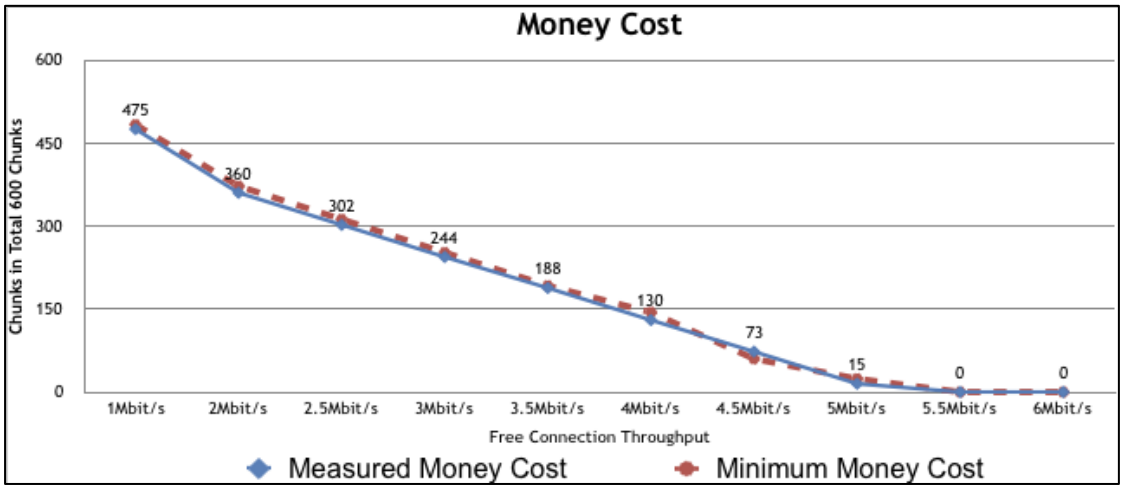

Figure 45 Money Cost for Experiment Scenario 2

In this experimental scenario, CUR is fixed at 5Mbit/s for each test and free connection throughput limit is changed for each test. Money cost result is expected as 0, for the tests which have free connection throughput higher than 5Mbit/s while money cost is expected more than 0 for the tests which have free connection throughput lower than 5Mbit/s. Money cost result is increasing with decreasing free connection throughput and it is zero for the tests which have free connection throughput higher than 5Mbit/s, as it can be seen from Figure 45. Minimum and measured money cost results are very close and this means the efficient utilization of free connection.

4.2. Setup 2

In this setup, the client and the server are connected to each other over the Internet and performance of CATS evaluated under variable network conditions.

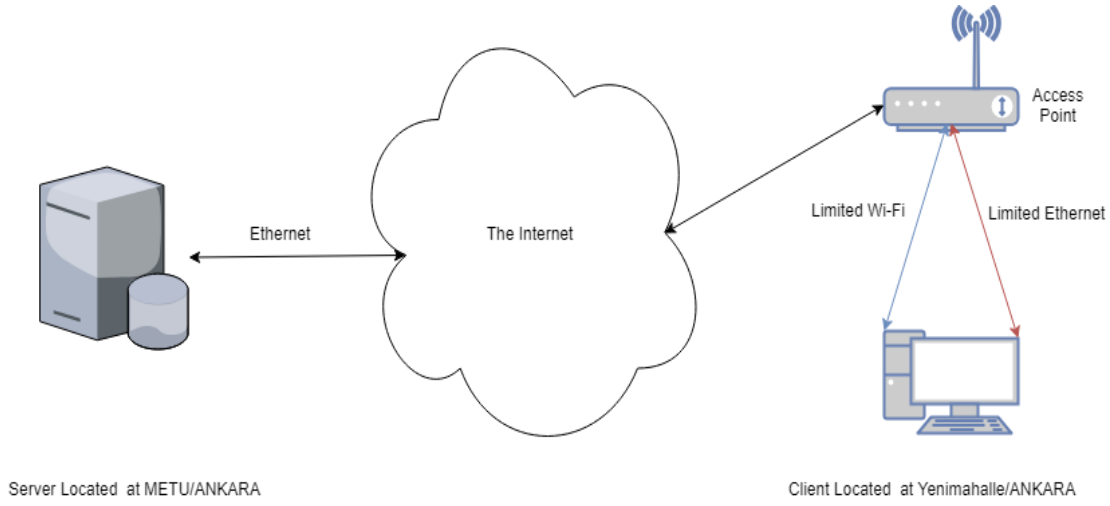

Figure 46 Experimental Setup 2

The client and the server are located at different physical locations in this setup as shown in Figure 46. The client is located at Yenimahalle, ANKARA and it is connected to access point, which is also at the same physical location with the client, over two different connections; Wi-Fi and Ethernet. Wi-Fi connection simulates free connection while Ethernet connection simulates paid connection. Throughputs of both connections are limited to the predefined limitation values at each test performed in this setup. The server is located at METU, ANKARA and connected to the access network via Ethernet. Client application is executed on OSX, while server application is executed on Ubuntu 16.04. As stated before, along the experiments, both client and server applications are executed on different platforms to be able to evaluate the platform independency.

Main difference between setup 1 and setup 2 is the variability of the underlying network. All the conditions are fixed for each test done in setup 1 because of local network employed in it. On the other hand, in setup 2, the client and the server connected to each other over the Internet so, propagation delay, queuing delay, packet losses are all effected the transfer time of each data packet.

Ten different 10minute time sensitive stored file streaming tests are performed on this setup. Paid and free connections of client is limited to a constant value for each test while CUR is changed at each test.

4.2.1. Test Parameters

- Limited Free Connection Throughput: 3Mbit/s
- Limited Paid Connection Throughput: 10Mbit/s

- File Time Period: 10 Minutes
- Chunk Time Period: 1 Second
- Total Chunk Count: 600
- CUR: Different for Each Test, 1Mbit/s to 6Mbit/ s

4.2.2. Results

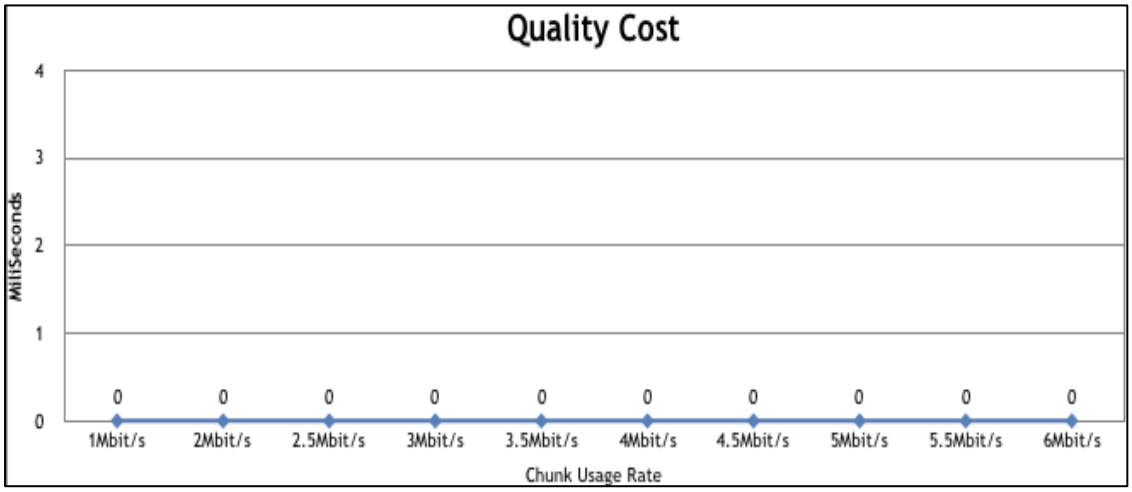

Figure 47 Quality Cost for Setup 2

Quality cost results obtained from the tests performed are given in Figure 12. The client didn't pause at any of the 10minute time sensitive file streaming tests which all have different CURs.

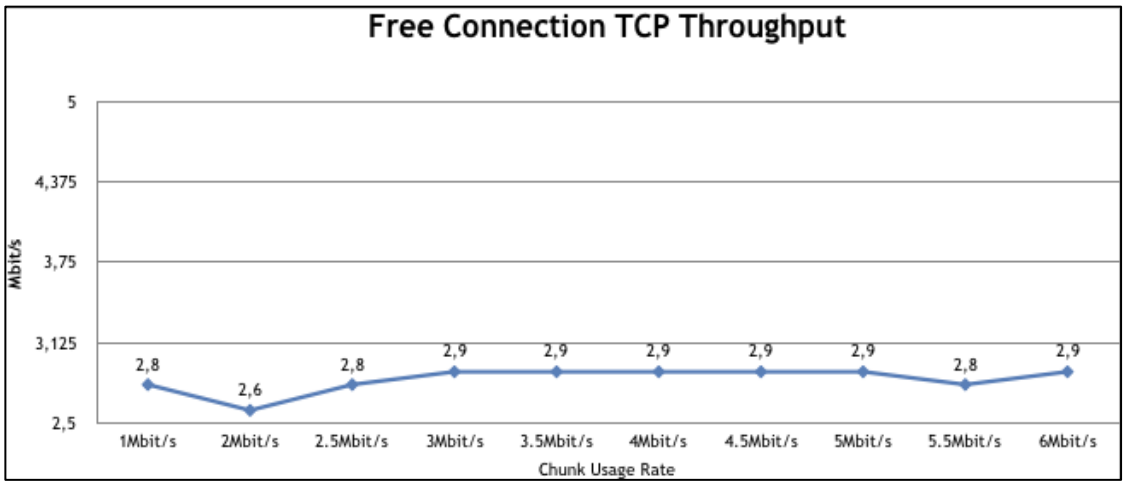

Figure 48 Free Connection TCP Throughput for Setup 2

In setup 1, free connection utilization evaluation is done in a local network environment which has relatively stable network conditions. However, in setup 2, even though free connection throughput is limited artificially, the server and the client are connected over internet and all network delays like propagation delay, queuing delay and packet losses are added to the characteristic of free connection. In CATS, more than one chunks are written to the free connection's TCP socket to be able to eliminate the effect of network delays. In Figure 48, measured free connection TCP throughputs are given for each test. Even if setup 1 and setup 2 have completely different network conditions, CATS can utilize free connection at a maximum rate in setup 2. Free connection throughput is limited to 3Mbit/s while CUR is changed for all ten different tests as it is shown in Figure 48. As it can be seen from the FCTT results, CATS utilized the free connection with a value close to 100% for each test. When these results compared with the ones in setup 1, it is possible to say that, CATS eliminates effects of network delays like propagation and queuing, the FCTT values are close to the actual limited free connection throughput values.

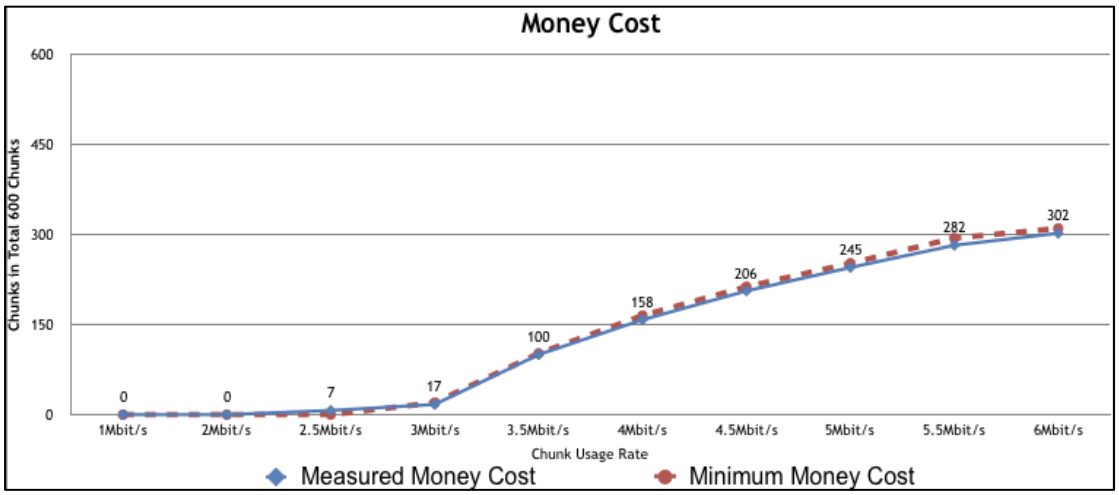

Figure 49 Money Cost for Setup 2

In Figure 49, money cost result for each test performed in setup 2 is given. Measured money cost results are approximately the same with the minimum money cost results. Experiment scenario 1 which is performed on setup 1 is similar to the tests are done in this setup. However, chunk scheduling is a bit more challenging for this setup because of the client and the server are connected over internet and factors like packet losses and delays make it difficult to estimate accurately. However, as it can be seen from Figure 49, money cost results obtained in this setup is very close to the ones obtained in the setup 1.

4.3. Setup 3

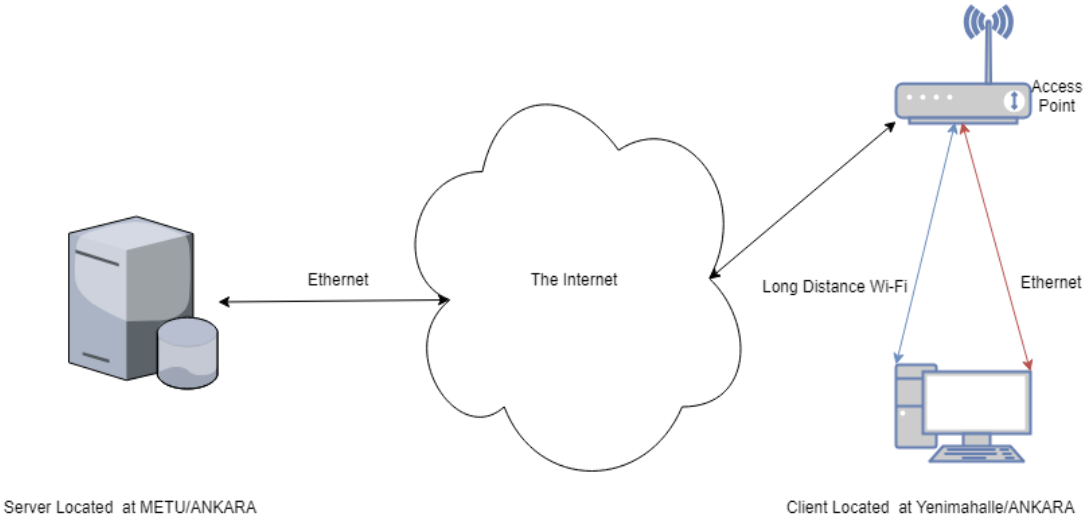

Figure 50 Experimental Setup 3

In setup 3, the client and the server are connected to each other over the Internet and they are at different physical locations as shown in Figure 50. Wi-Fi connection simulates free connection while Ethernet connection simulates paid connection in setup 3. The client host computer placed quite away from the wireless access point to deteriorate the conditions as much as possible for free connection. Therefore, free connection throughput decreased and became significantly variable. Paid connection which is simulated with Ethernet connection connected to the access point with Ethernet cable so, its throughput remains less variable. Client application is executed on Windows 7 operating system while server application is executed on Ubuntu 16.04 during the tests performed in setup 3.

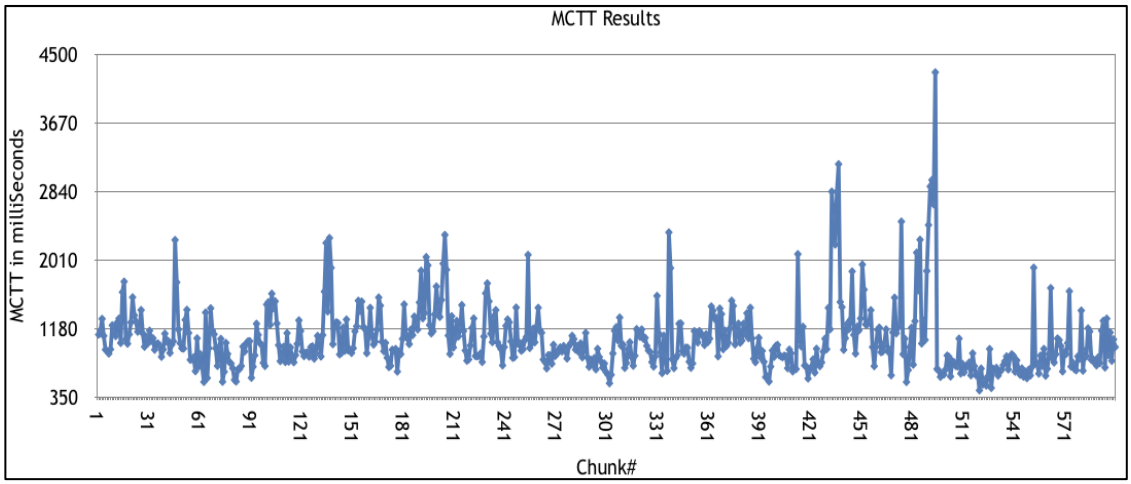

Figure 51 MCTT Results for 600 Chunks

MCTT values, which are obtained from one of the 10minute time sensitive file streaming tests performed in this setup, are given in Figure 51. Obtained MCTT values are variable between 350 milliseconds to 4200 milliseconds. In the previous setups, CATS tested with more stable free connection throughput. But in setup 3, as it can be seen from the MCTT values given in the above Figure 51, throughput of long distance placed Wi-Fi which is simulating free connection is quite variable. Therefore, in setup 3, CATS exposed to quite variable free connection throughput and results are obtained under these conditions.

4.3.1. Test Parameters

- Free Connection Throughput: Unlimited, Long Distance Placed Wi-Fi
- Paid Connection Throughput: Unlimited, Cable Connected
- File Time Period: 10 Minutes
- Chunk Time Period: 1 Second
- Total Chunk Count: 600
- CUR: Variable for Each Test, 1Mbit/s to 9Mbit/s

4.3.2. Test Results

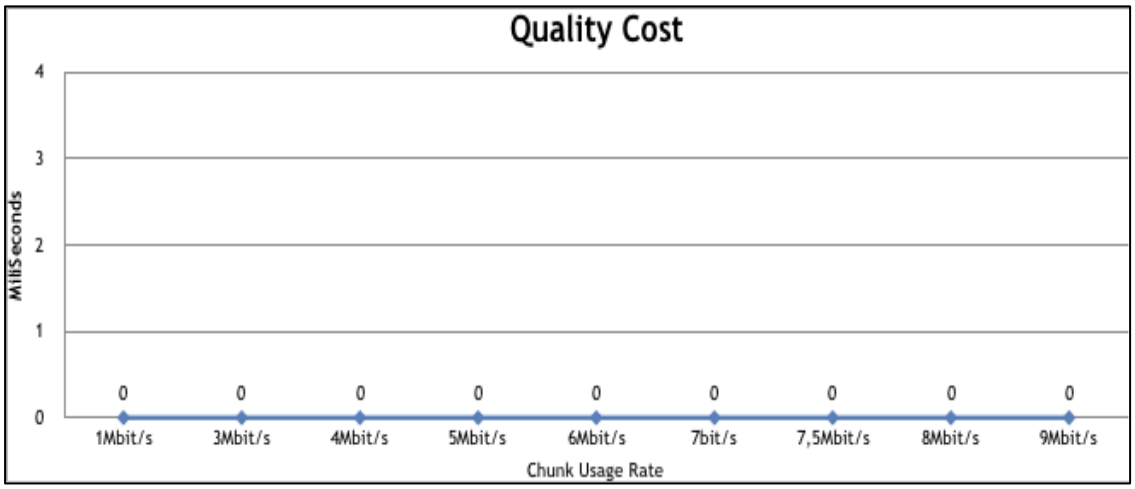

Figure 52 Quality Cost for Setup 3

During the tests realized in Setup 3, 10-minute time sensitive file streaming tests are carried out with different CURs and their quality cost results are given in Figure 52. The

client didn't pause in any of the tests. As it is given in the Figure 51, during these tests, free connection throughput is quite variable. Even though network is excessively variable for free connection, CATS scheduler estimated underlying network accurately and all the chunks could be downloaded before their scheduled usage times over free and paid connections. As it is explained in the proposed algorithm section, CATS has duplicate chunk send feature and even if sudden changes occur in free connection throughput, CATS duplicate send chunks over paid connection and thanks to this mechanism, chunk arrives before its scheduled usage time and client pause is prevented as it can be seen from the quality results given in the above Figure 52.

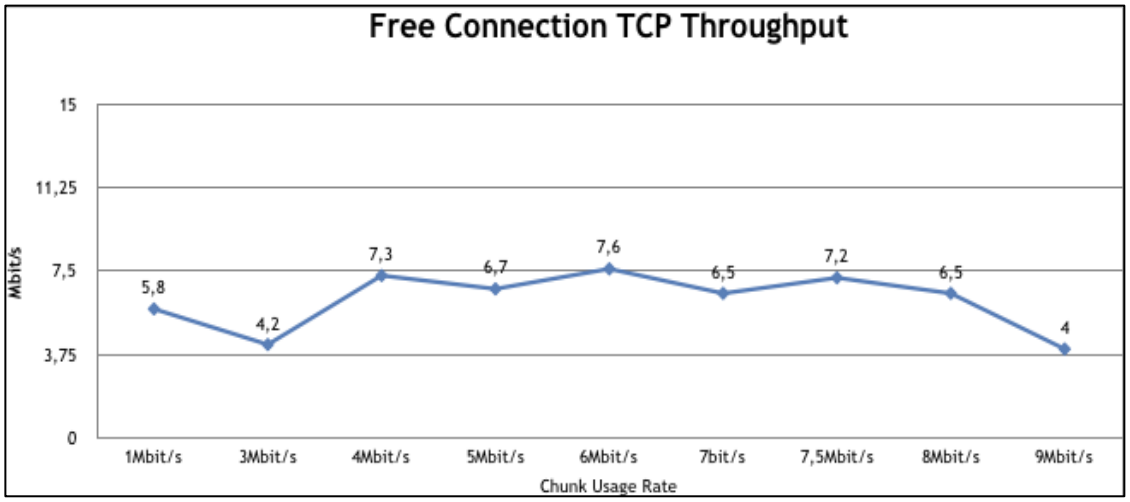

Figure 53 Free Connection TCP Throughput for Setup 3

Free connection throughput results for the tests are given in the above Figure 53. FCTT results are comparable with limited free connection throughputs for the test setups 1 and 2. However, in this setup, any throughput limitation isn't employed and the entire throughput results are obtained with the long distance placed Wi-Fi connection which simulates free connection. So, different FCTT values are obtained at each test.

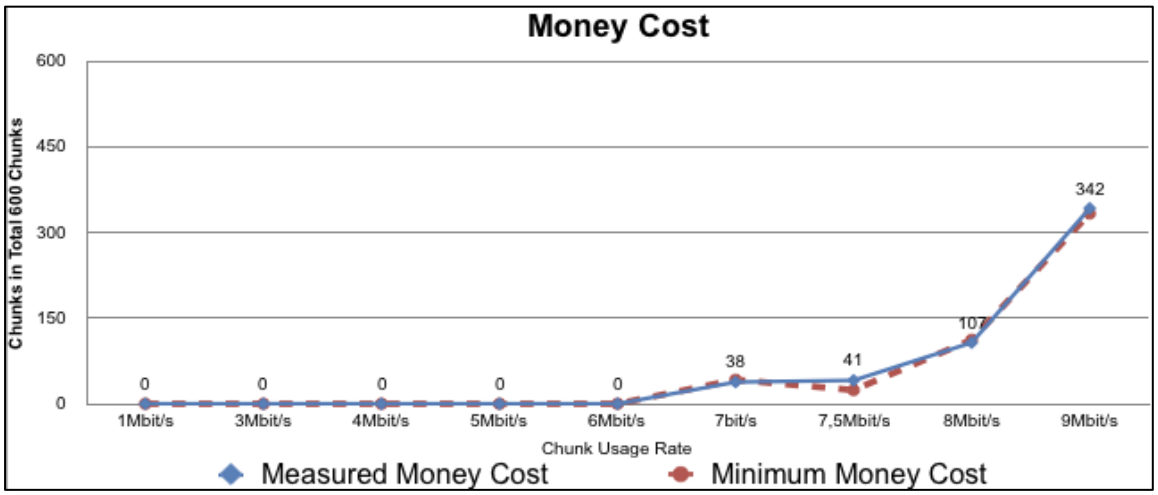

Figure 54 Money Cost for Setup 3

As it is given in Figure 54, quite variable MCTT values are obtained for the chunks downloaded over free connection during the tests performed in Setup 3. Even though free connection's throughput is quite variable, client didn't pause at any of these nine tests. One may think, most of the chunks are duplicate sent over paid connection when free connection throughput changed suddenly and client pause is prevented only with this method. However, using only this method would result with higher money cost and this would contradict with one of the main goals of proposed CATS; minimum paid connection utilization. Money Cost results are given in Figure 54 and as it can be seen from the results, measured and minimum money cost results are very close to each other. This result is obtained thanks to the sending chunks over free connection with predefined offset value method described in the proposed algorithm section. Chunk Scheduler sends chunks over free connection in advance with a security offset and by doing so, even if free connection throughput changes suddenly, chunks downloaded over free connection can arrive before their scheduled usage time. Thus method results with efficient usage of free connection, which causes measured money cost is close to minimum money cost as it can be seen from Money Cost results given in Figure 54.

4.4. Setup 4

In the previous 3 setups, paid connection is simulated with Ethernet connection because of monetary cost of cellular connections. In this setup, paid connection directly connected to the cellular 4G/LTE connection as it can be seen from Figure 55.

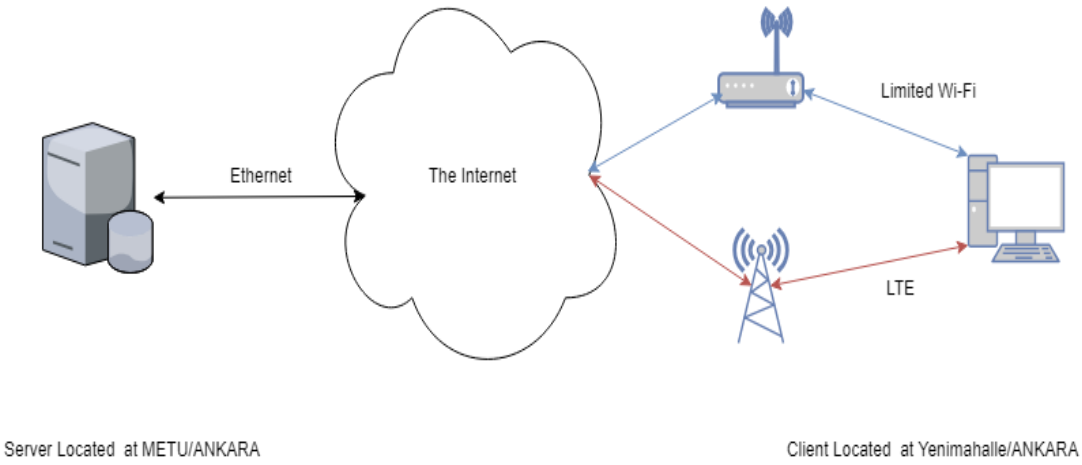

Figure 55 Setup 4 Environment

During tests performed in setup 3, free connection throughput is limited to 3Mbit/s and CUR is changed for each test. Client application is executed on OSX while server application is executed on Ubuntu 16.04.

4.4.1. Test Parameters

- Limited Free Connection Throughput: 3Mbit/s
- Limited Paid Connection Throughput: 10Mbit/s
- File Time Period: 1 Minute
- Chunk Time Period: 1 Second
- Total Chunk Count: 60
- CUR: Different for Each Test, 1Mbit/s to 5Mbit/s

4.4.2. Results

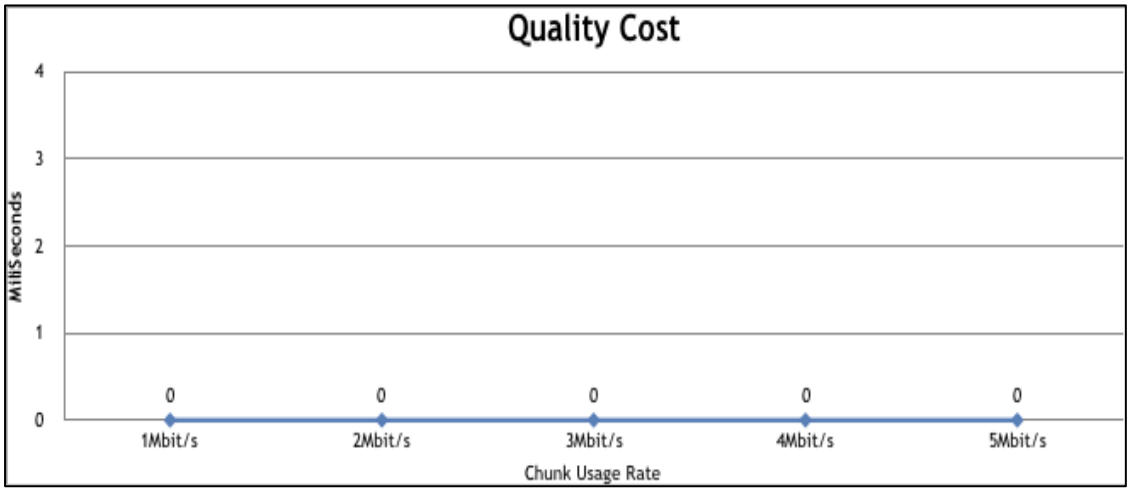

Figure 56 Quality Cost for Setup 4

Quality cost results for Setup 4 are given in the Figure 56 and client didn't pause in any of the tests. CATS downloaded all the chunks over two different connections, before their scheduled time.

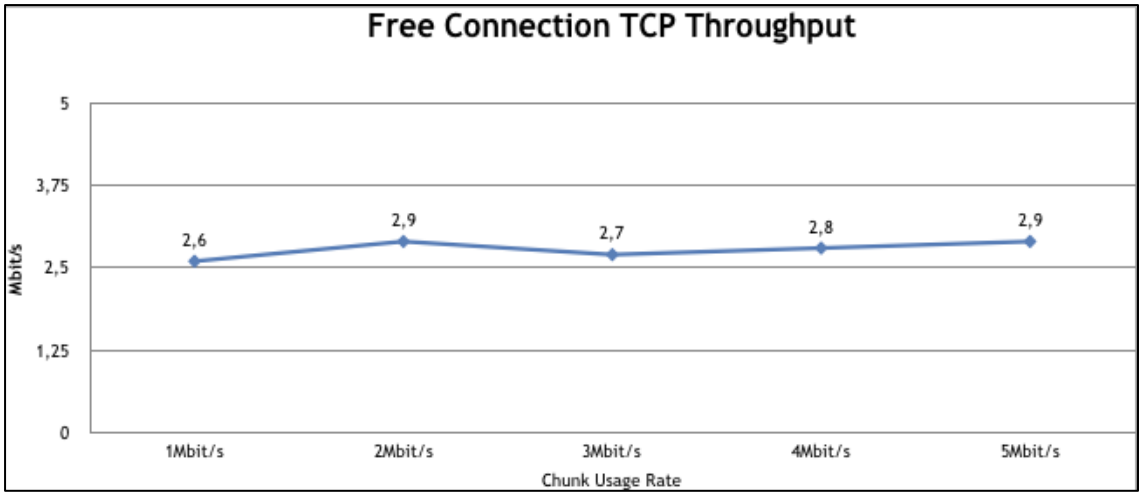

Figure 57 Free Connection TCP Throughput for Setup 4

Free connection throughput is limited to 3Mbit/s for each test and as it can be seen from the given free connection TCP Throughput results placed in Figure 57, free connection utilized with a value close to 3Mbit/s for all the tests.

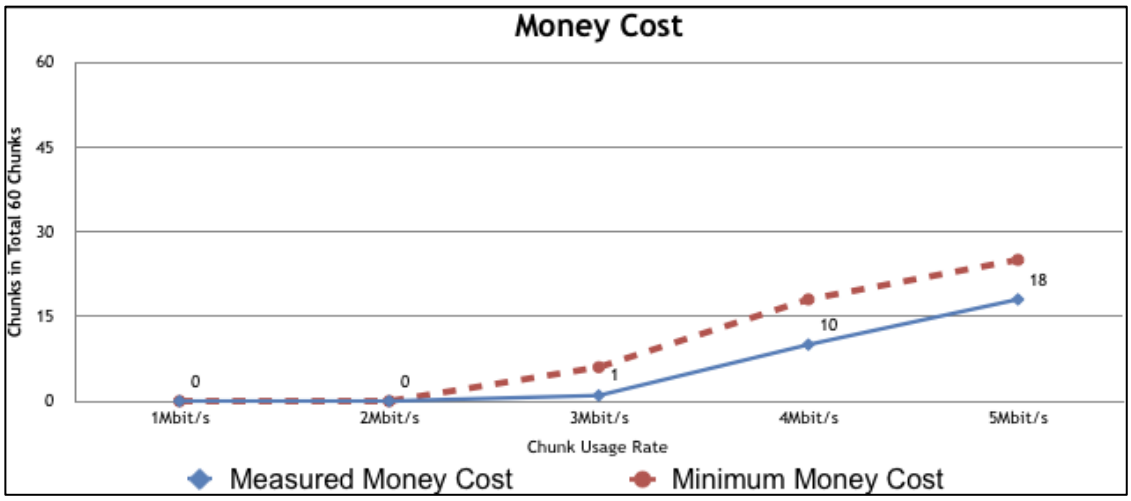

Figure 58 Money Cost for Setup 4

Money Cost results for Setup 4 is given in Figure 58. Measured and minimum money cost results are close to each other also in each test performed in setup 4. The reason why minimum money cost result is larger than the measured money cost result is the initial chunk buffering as stated before. In this setup, file duration is shorter, so chunk count is lesser than other setups, so this difference is more visible.

4.5. Discussions

CATS tested under different network conditions to evaluate its success rate. At each experimental setup, different network conditions are simulated and many tests are performed. The quality cost result for each test performed at each setup is zero. Thus, it would be true to deduce that CATS achieves one of its main goals; minimum client pause.

Free connection TCP throughput value is calculated after each test and this value is compared with actual bandwidth limitation value. It is observed that, CATS utilizes free connection throughput close to the maximum at each test performed different experimental setups. So, CATS achieves another goal of its; maximum free connection utilization.

The money cost is calculated after completion of each test and it compared with the possible minimum money cost which can be obtained with efficient usage of free connection resources. At each test, it is observed that, the money cost and the minimum money cost values are very close because of efficient utilization of free connection. That is, CATS successfully achieves final goal of it; minimum paid connection usage.

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1. Conclusion

Bandwidth aggregation, that is using more than one access network simultaneously to achieve higher throughput, is being utilized by many applications to enhance the quality of service. This thesis proposes a cost aware bandwidth aggregation method called CATS that provides higher throughput by bandwidth aggregation while minimizing the monetary cost associated with data transfer.

CATS focuses on transferring constant bit-rate time sensitive data from a server to a client over two separate access networks called paid link and free link. The client establishes two separate TCP connections, paid connection and free connection, to the server through paid link and free link, respectively. The server divides the file into fixed size chunks and sends chunks to the client over free and paid connections. Then the client starts consuming received chunks while receiving subsequent chunks from the server. The primary objective of CATS is to stream file chunks to the client in a cost aware manner without causing significant quality of service degradation. To this end, CATS maximizes the utilization of the free connection, minimizes the utilization of the paid connection, and ensures timely delivery of chunks to receiver.

Apart from similar studies, CATS mainly focuses on the monetary cost and client experience trade off. Most of the existing bandwidth aggregation methods focus on maximizing the throughput and solving the problems such as head of line blocking and out of order delivery problems associated with the utilization of multiple connections. Only a few studies target cost aware bandwidth aggregation. For instance, in [22] authors propose such a cost aware connection pooling method. However, the proposed method focus on joint optimization of file transfer completion time and monetary cost rather than the transfer of constant bit-rate time sensitive data. Split-layer SVC encoding is employed with the cost effective scheduler for video streaming applications in [23], but proposed scheduler aims to minimize the monetary cost only for the enhancement layers of encoded video file.

In the scope of this thesis, CATS is implemented using the Java programming language. Hence, the current implementation of CATS is available on platforms including desktop as well as mobile devices that have Java virtual machine installed. The performance of CATS under different network conditions is evaluated by several experiments carried out on various test setups. Experimental results show that the proposed method successfully provides timely delivery of time sensitive data from a server to a client with the minimum possible monetary cost. In the light of the experimental results, it is possible to conclude that the proposed method is suitable for the time sensitive streaming applications that require high bandwidth that may not be provided by a single connection. Therefore, the user experience can be enhanced while minimizing the monetary cost.

5.2. Future Work

CATS designed and implemented as an individual application which delivers time sensitive file from a server to a client. Current CATS implementation is specifically designed for the real-time file streaming. However, it can easily be extended to support real-time interactive multimedia applications such as video conferencing. The implementation of CATS as a library that can be used by network applications is left as a future work. Moreover, the primary mechanisms introduced by CATS can be used to create a scheduler to have a cost aware MPTCP Linux kernel implementation of MPTCP [6].

CATS employs EWMA [26] to estimate the delivery time of chunks. Furthermore, it is possible to take into account delay variation as it is done in TCP timeout mechanism.

In this thesis, we employ TCP as transport layer protocol. It is possible to use UDP [27] as transport layer protocol by implementing a reliability mechanism for CATS. We can have more control over actual data transfer by creating smaller chunks and delivering these chunks via UDP.

Video file streaming is one of the use-cases supported by the current implementation of CATS. The client may request to change video playout rate while streaming video, however current version of CATS doesn't support change in chunk usage rate (video playout rate). Thus, developing an additional mechanism to notify the server about the client video playout change request and maintain chunk delivery according to the new video playout rate is considered as a future work for making CATS more useful for video streaming use-case.

As a future work, utilization of more than two connections and assigning different cost values to each connection according to the client preference is considered.

In this thesis, only monetary cost is considered as a cost. However energy efficiency can be taken into account as well as cost.

REFERENCES

[1] "Nielsen's Law of Internet Bandwidth," *Nielsen Norman Group*. [Online]. Available: https://www.nngroup.com/articles/law-of-bandwidth/. [Accessed: 27-May-2018].

[2] J. Postel, "Transmission Control Protocol," *IETF Tools*, 1981. [Online]. Available: https://tools.ietf.org/html/rfc793. [Accessed: 26-May-2018].

[3] Ford, Raiciu, and Handley, "TCP Extensions for Multipath Operation with Multiple Addresses," *» RFC Editor*, Jan-2013. [Online]. Available: https://www.rfc-editor.org/rfc/rfc6824.txt. [Accessed: 26-May-2018].

[4] *IETF*. [Online]. Available: https://www.ietf.org/. [Accessed: 19-Jun-2018].

[5] "About Multipath TCP (MPTCP)," MPTCP. [Online]. Available: https://www.tessares.net/technology/mptcp/. [Accessed: 27-May-2018].

[6] "Welcome to the Linux Kernel MultiPath TCP project," *MultiPath TCP - Linux Kernel implementation : Main - Home Page browse*. [Online]. Available: https://www.multipath-tcp.org/. [Accessed: 26-May-2018].

[7] C. Paasch, S. Ferlin, O. Alay, and O. Bonaventure, "Experimental evaluation of multipath TCP schedulers," *Proceedings of the 2014 ACM SIGCOMM workshop on Capacity sharing workshop - CSWS 14*, 2014.

[8] J. Hwang and J. Yoo, "Packet scheduling for Multipath TCP," *2015 Seventh International Conference on Ubiquitous and Future Networks*, 2015.

[9] F. Yang, P. Amer, and N. Ekiz, "A Scheduler for Multipath TCP," *2013 22nd International Conference on Computer Communication and Networks (ICCCN)*, 2013.

[10] S. Ferlin, O. Alay, O. Mehani, and R. Boreli, "BLEST: Blocking estimation-based MPTCP scheduler for heterogeneous networks," *2016 IFIP Networking Conference (IFIP Networking) and Workshops*, 2016.

[11] T. D. Schepper, J. Struye, E. Zeljkovic, S. Latre, and J. Famaey, “Software-defined multipath-TCP for smart mobile devices,” *2017 13th International Conference on Network and Service Management (CNSM)*, 2017.

[12] E. Haleplidis, “Software-Defined Networking (SDN) : Layers and Architecture Terminology,” *IETF Tools*, Jan-2015. [Online]. Available: https://tools.ietf.org/html/rfc7426. [Accessed: 29-May-2018].

[13] H. A. Kim, B.-H. Oh, and J. Lee, “Improvement of MPTCP Performance in heterogeneous network using packet scheduling mechanism,” *2012 18th Asia-Pacific Conference on Communications (APCC)*, 2012.

[14] Y.-S. Lim, E. M. Nahum, D. Towsley, and R. J. Gibbens, “ECF: An MPTCP Path Scheduler to Manage Heterogeneous Paths,” *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies - CoNEXT 17*, 2017.

[15] F. Yang, Q. Wang, and P. D. Amer, “Out-of-Order Transmission for In-Order Arrival Scheduling for Multipath TCP,” *2014 28th International Conference on Advanced Information Networking and Applications Workshops*, 2014.

[16] Q. Peng, M. Chen, A. Walid, and S. Low, “Energy efficient multipath TCP for mobile devices,” *Proceedings of the 15th ACM international symposium on Mobile ad hoc networking and computing - MobiHoc 14*, 2014.

[17] C. Pluntke, L. Eggert, and N. Kiukkonen, “Saving mobile device energy with multipath TCP,” *Proceedings of the sixth international workshop on MobiArch - MobiArch 11*, 2011.

[18] R. Stewart, “Stream Control Transmission Protocol,” *IETF Tools*, Sep-2007. [Online]. Available: https://tools.ietf.org/html/rfc4960. [Accessed: 26-May-2018].

[19] R. Stewart, “Stream Schedulers and User Message Interleaving for the Stream Control Transmission Protocol,” *IETF Tools*, Nov-2017. [Online]. Available: https://tools.ietf.org/html/rfc8260#section-3.1. [Accessed: 26-May-2018].

[20] Katsaros, Dianati, and Mehrdad, “A cost-effective SCTP extension for hybrid vehicular networks,” *Surrey Research Insight Open Access*, 22-Jun-2017. [Online]. Available: http://epubs.surrey.ac.uk/813945/. [Accessed: 26-May-2018].

[21] J. Lee, K. Lee, C. Han, T. Kim, and S. Chong, “Resource-Efficient Mobile Multimedia Streaming With Adaptive Network Selection,” *IEEE Transactions on Multimedia*, vol. 18, no. 12, pp. 2517–2527, 2016.

[22] W. Lee, J. Koo, S. Choi, and Y. Park, "E$PA: Energy, usage ($), and performance-aware LTE-WiFi adaptive activation scheme for smartphones," *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*, 2014.

[23] S. Moon, J. Yoo, and S. Kim, "Exploiting Adaptive Multi-interface Selection to Improve QoS and Cost-Efficiency of Mobile Video Streaming," *2015 IEEE International Conference on Mobile Services*, 2015.

[24] L. Rizzo, "Dummynet a simple approach to the evaluation of network protocols," *ACM Computer Communication Review*, vol. 27, pp. 31–31, Jan. 1997.

[25] "The Dummynet Project," info.iet.unipi.it. [Online]. Available: http://info.iet.unipi.it/~luigi/dummynet/. [Accessed: 27-May-2018].

[26] "Exponential Weighted Moving Average," IETF. [Online]. Available: https://www.ietf.org/proceedings/43/slides/pim-sharma-98dec/tsld017.htm. [Accessed: 19-Jun-2018].

[27] "User Datagram Protocol," IETF Tools. [Online]. Available: https://tools.ietf.org/html/rfc768. [Accessed: 19-Jun-2018].