Throughput and Fairness Considerations in Overlay Networks for Content Distribution

A Thesis
Presented to
The Academic Faculty

by

Pradnya Karbhari

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

College of Computing
Georgia Institute of Technology
December 2005

Throughput and Fairness Considerations in Overlay Networks for Content Distribution

Approved by:

Dr. Mostafa Ammar
Co-Advisor, College of Computing
Georgia Institute of Technology, Adviser

Dr. Ellen Zegura
Co-Advisor, College of Computing
Georgia Institute of Technology

Dr. Constantinos Dovrolis
College of Computing
Georgia Institute of Technology

Dr. Misha Rabinovich
Department of Computer Science
Case Western Reserve University

Dr. George Riley
School of Electrical and Computer Engineering
Georgia Institute of Technology

Date Approved: August, 18th 2005

To my parents,

Dr. Mrs. Anagha Karbhari and Mr. Ajit Karbhari.

ACKNOWLEDGEMENTS

I would like to thank my advisors, Dr. Mostafa Ammar and Dr. Ellen Zegura for their guidance, support, encouragement and patience throughout my Ph.D. They have encouraged me to keep going when I was in the right direction, pushed me a bit when I was getting lax, and helped me pull myself out of tight spots when I hit any roadblocks. I have learned a lot more from Mostafa than just that which pertains to academics and the doctoral degree. I thank him for everything that he has taught me, knowingly and unknowingly. Amongst other things, I have learnt a lot about presenting myself and my work from Ellen, and I thank her for everything.

Many thanks are due to Dr. Misha Rabinovich for being a great mentor for two summers during my internship at AT&T Labs- Research and for being on my thesis committee. I would also like to thank Dr. George Riley for collaborating with me and for being on my committee, as well as Dr. Constantinos Dovrolis for being on my committee and for all his feedback on my work.

Next, I would like to thank Matt Sanders and Dr. Russ Clark of OIT, and CNS folks including Peter Wan, Dan Forsyth and Karen Carter, for helping me set up some of the measurement experiments, as part of my research. All the staff at the College of Computing, and at Georgia Tech has been really helpful, and I thank them for their help.

My student life at Georgia Tech has been an awesome experience. It would have been very boring and depressing without the incessant support and entertainment of all my friends at Georgia Tech. First, I would like to thank all my labmates in the NTG group, including Richard, Shashi, Christos, Ruomei, Qi, Wenrui, Meng, Sridhar, Amogh, Manish, Ravi, Yarong and Abhishek for all the fruitful and fruitless discussions, the lunches, the dinners, the get-togethers and the camaraderie. Two labmates who deserve special mention are Richard and Amogh. Richard helped me learn the ropes in my initial years, served as a sounding board for numerous technical discussions and competed with me in the online

version of the game, “Settlers of Catan”. He has been an awesome labmate! Amogh has been a great labmate, a sincere co-author and a special friend, who was helped me face the ups and downs of a Ph.D. student’s life, especially during the latter half of my PhD. Thank you Amogh.

Amongst my non-labmate friends at Georgia Tech, special thanks are due to Andrew and Badri for their friendship and all the advice and encouragement throughout these years. Srini, Deepak, Aru and Karthik have been a great help and a source of “hajjar” fun and entertainment, especially during the later part of my Ph.D.

I will be forever thankful to all my roommates, including Harinee, Shalini, Prameela, Nikhila, Shilpa, Avanti, Nisarga and Nupur, over the course of my graduate life, who made me feel at home when I got back from a hard day’s work at school. In addition to my roommates, Sharvari, Amruta and Gayatri have been great company during the last year of my Ph.D. I would also like to thank Shalini and Ravi for being great hosts during my first year in Atlanta.

Volunteering for Vibha Inc. (formerly CRY Inc.), a non-profit organization for under-privileged children in India, has been a significant part of my non-academic life while at Georgia Tech. I made quite a few friends while volunteering for CRY/Vibha. Vivek, Durga, Shantanu, Sushant and Aravind, have been more than just “CRY friends”. It was a great experience to have volunteered with others in Vibha-Atlanta, including Vijay, Srinivas and Sumatiji.

Anagha, who has been a very good friend since pre-school (or Junior K.G.) has been a constant source of encouragement and support. All the phone calls filled with grief on one end and cheering up on the other end serve as reminders of the days when things were going wrong in either my Ph.D. or hers. I am glad that those days are over for both of us now; but every time I think of one of those days, I realize how lucky I am to know such a wonderful person. Thank you, Anagha! Arthi, Kiran, Miku, Mima, Rohit and Tejal have been great friends since my VJTI days. I deeply appreciate their friendship and encouragement all these years.

Three people who may not be aware of the inspiration that I have derived from them,

and who are a major factor in my decision to pursue a doctoral degree are: my cousin (Dada), and my two professors in VJTI, Dr. Prerana Rane and Dr. Lalita Deshpande. My heartfelt thanks to Dada for serving as a perfect role model for me throughout my career. I have seeked the advice of Dr. Rane and Dr. Deshpande on numerous occasions during and after my VJTI days, about pursuing a Ph.D. and about my career in general. I thank them all for their words of wisdom.

My greatest reward at Georgia Tech is Niket. It was here that I met Niket, it was here that I enjoyed his company, it was here that I had the pleasure of volunteering with him for Vibha, it was here that a wonderful friendship was formed, it was here that a beautiful relationship has started, and it will be from here that we will start a new phase of life together. Thanks a lot Niket, for everything, and for more that is to come in life.

My family, including my parents, sister, aunts, uncles, cousins, grandparents, and Donde ajji-aajoba have given me a lot that I am thankful for. Shruti has been a great source of entertainment, support and friendship throughout my life. What would I do without her during all those times when I was feeling terribly low, and a chat with her for even a few minutes about totally random things would serve as the right catalyst to lift my spirits. Thank you Shrutu, for being the best sister in this world.

My Ph.D. would have been absolutely impossible without the immense support, patience, understanding and encouragement of my parents. They have amazed me with their support in all my decisions in life. All these years of "When are you graduating?" questions on every phone call have finally paid off, and I am sure they are the happiest people on earth now. I am short of words to express my gratitude towards them for everything they have bestowed on me. Thank you Aai. Thank you Baba.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

The Internet has been designed as a best-effort network, which does not provide any additional service or control to applications using the network. Overlay networks, which form an application layer network on top of the underlying Internet, have emerged as popular means to provide specific services and greater control to applications. Overlay networks offer a wide range of services, including content distribution, multicast and multimedia streaming. In my thesis, I focus on overlay networks for content distribution, used by applications such as bulk data transfer, file sharing and web retrieval.

I first investigate the construction of such overlay networks by studying the bootstrapping functionality in an example network (the Gnutella peer-to-peer system). This study comprises the analysis and performance measurements of Gnutella servents and measurement of the GWebCache system that helps new peers find existing peers on the Gnutella network.

Next, I look at fairness issues due to the retrieval of data at a client in the form of multipoint-to-point sessions, formed due to the use of content distribution networks. A multipoint-to-point session comprises multiple connections from multiple servers to a single client over multiple paths, initiated to retrieve a single application-level object. I investigate fairness of rate allocation from a session point of view, and propose fairness definitions and algorithms to achieve these definitions.

Finally, I consider the problem of designing an overlay network for content distribution, which is fair to competing overlay networks, while maximizing the total end-to-end throughput of the data it carries. As a first step, I investigate this design problem for a single path in an Overlay-TCP network. I propose two schemes that dynamically provision the number of TCP connections on each hop of an Overlay-TCP path to maximize the end-to-end throughput using few extraneous connections. Next, I design an Overlay-TCP

network, with the secondary goal of overlay layer congestion or intra-overlay-network fairness. I propose a centralized algorithm for deciding the number of TCP connections to be used on each overlay hop, as well as three heuristics that are distributed in nature. In addition, I show that by varying the maximum number of connections allowed on overlay hops in each competing overlay network, one can vary the proportion of sharing between competing overlay networks.

CHAPTER I

INTRODUCTION

The Internet has penetrated the daily lives of millions of people through the use of a variety of applications such as the World Wide Web (WWW), email, file transfer and multimedia streaming. With increasing dependence of people on the Internet, end-users have started expecting more and more services from the Internet. The requirements of different applications vary significantly. Some applications, like web retrieval, expect that the requested data should start being delivered very quickly (low wait time); some others, like multimedia streaming, expect that the jitter in packet delivery, as well as the wait time should be low; and some others, like data transfer applications, require that the total data be delivered fast enough (high throughput).

The Internet, however, provides a best-effort service to the end-user. This is not good enough for many applications, especially if there exist more intelligent ways of using the network. Recently, *overlay networks* are gaining increasing focus commercially as well as in the research community for their ability to offer additional services, beyond those offered by the underlying native network (Internet). In addition, they offer greater flexibility and control over the use of the Internet. Overlay networks form an application-layer network on top of the underlying Internet with the aim of providing a particular service to the users, in order to meet their specific requirements. These services span a wide range, including content distribution [4], file sharing [32], multicast [56], quality of service [97], multimedia streaming [72], fault tolerance [3] and improved routing [89]. These overlay networks, which address the requirements of applications by explicitly setting up an infrastructure, are referred to as infrastructure-based overlay networks.

In this thesis, we focus on overlay networks for content distribution. Content distribution services are often used by applications such as bulk data transfer, file sharing and web retrieval. The primary requirement of these applications is that the time to retrieve and

use or view the requested data should be short, and the throughput of the data transfer should be high. In particular, we look at three problems that concern overlay networks for content distribution.

- Construction of overlay networks— We start by looking at the construction of overlay networks. In particular, we study the construction of the Gnutella [32] peer-to-peer network, which is an overlay network used for sharing of content between end-users.
- Fairness issues introduced due to the use of overlay networks for content distribution— The use of content distribution networks (CDNs) for faster retrieval of data has led to applications retrieving data over a session of multiple point-to-point connections from multiple servers to a single client. We investigate the fairness in allocation of resources between multiple such sessions.
- Design of an Overlay-TCP network— Finally, we design a single path in an overlay network, and then an entire overlay network, with the aim of maximizing the total throughput of the data carried on the overlay path or overlay network respectively, while maintaining fairness between competing overlay networks. Our design dynamically provisions each overlay hop with one or more TCP (Transmission Control Protocol) connections, as required.

We will elaborate on each one of the listed contributions in the rest of the chapter.

1.1 Construction of Overlay Networks

We first look at the problem of construction of overlay networks. In particular, we consider the process of construction of an example overlay network— the Gnutella [32] peer-to-peer system. Peer-to-peer networks, also referred to as file-sharing networks, are a type of overlay networks, that are typically formed for sharing content between users. The most important characteristic of these networks is that all peers in these networks can function as clients as well as servers, and can receive as well as send data. Peers using these networks search for the content that they are interested in, and download the content from peers that have the requested content. Any end-user can participate in these systems, thus causing the system

to be very diverse in terms of resource capabilities of peers. The system is also very dynamic, as peers join and leave the network quite frequently. The topology of these networks could either be structured, where peers maintain a particular structure while joining the network, or it could be unstructured. Unstructured peer-to-peer networks are partly infrastructure-based, unlike structured peer-to-peer networks and content distribution networks, and are therefore interesting from the point of view of overlay network construction. In our work, we focus on the question of how exactly do peers wanting to join an unstructured peer-to-peer system locate the peers that they should set up connections with. This process, referred to as the *bootstrapping process*, is the first step that any peer needs to execute before it can use the system. Unless this process is completed, a peer cannot participate in any search activity on the system. Also, the initial neighbor peers found during bootstrapping determine the new peer's location in the overall Gnutella topology, and ultimately its search and download performance. Hence, the bootstrapping process plays a significant role in the lifetime of a peer. It is essential that the bootstrapping time of a peer in the system be minimal, and that the neighbor peers that result from the bootstrapping process give a good search and download performance. In this thesis, we study the bootstrapping process in depth, and evaluate it by conducting a measurement-based study.

1.2 Multipoint-to-Point Session Fairness in Overlay Networks for Content Distribution

Next, we look at throughput and fairness issues resulting from the use of content distribution networks. Content distribution networks like Akamai [1], Digital Island [22] and Speedera [92] are infrastructure-based overlay networks that aim to expedite the reception of data by a client by setting up servers (or caches) that serve static content from locations topologically closer to clients. These caches are deployed across the Internet, and host static content for commercial websites like, say, cnn.com. Consider an example where a user requests the webpage www.cnn.com. The client application (user's browser in this case) sends a request to the origin server, which satisfies part of the request, and then redirects the client to one or more caches that are topologically closer to the user than the origin server. This

causes the requests from the users to be satisfied speedily. Effectively, applications that use these networks to retrieve data will experience higher throughputs and lower download times. Additionally, these content distribution networks significantly reduce the load on the origin servers, thus enabling them to support a larger client population. The result of using these content distribution networks is that a client retrieves data from multiple servers (origin server, cache server, image server, etc.) over connections set up with each server. Krishnamurthy et al. [54] have observed in a January 2001 study that the average number of CDN servers contacted by a client is somewhere between 3.4 and 10.3, with the median being around 6.

Proposals for content delivery of streaming media [4, 13, 72] take advantage of path diversity between multiple servers/peers and clients. These approaches use different forms of content coding (e.g. multiple description coding) to produce complementary descriptions of content, which are then served from multiple servers/peers to the same client. Companies like PeerGenius.com [73], CenterSpan.com [17], digitalfountain.com [23, 14] and parallel file-download applications such as those proposed in [82] start multiple connections to multiple servers in order to expedite the reception of data, thus improving user-perceived performance.

The conclusion we draw from these example applications is that data transfer is increasingly being performed over a set of point-to-point connections from multiple servers to a single client, and we can expect this trend to continue. We refer to such a set of multiple parallel point-to-point connections as a *multipoint-to-point session*. According to current per-connection rate allocation methods, such a session gets a higher total rate than competing sessions with fewer connections, thus leading to unfair sharing of bottleneck bandwidth. In summary, competition among content providers has motivated them to employ aggressive methods for content delivery in an attempt to give better delay and bandwidth performance than their competitors, which can potentially cause some unfairness toward other data transfers. In current literature, the problem of fair sharing between competing multipoint-to-point sessions, with each session comprising connections that use different paths to different servers and which may or may not share bottleneck links, has

not been studied. We explore this multipoint-to-point session fairness issue in depth in this thesis. We propose two session fairness definitions (*normalized rate session fairness* and *per-link session fairness*), and algorithms to achieve these definitions. We show that the proposed algorithms are indeed fair from a session point of view.

1.3 Design of an Overlay Network with Throughput and Fairness Considerations

Finally, we focus on the design of an overlay network. With the increasing popularity of overlay networks, we envision the simultaneous operation of multiple overlay networks on top of the native Internet. This co-existence of overlay networks implies that the same set of underlying resources (such as link bandwidth) will be used by these competing networks. This calls for mechanisms to enable fair sharing of underlying resources by competing overlay networks.

Different overlay networks proposed in practice operate using different protocols such as UDP (User Datagram Protocol) or IP (Internet Protocol) for forwarding data between overlay nodes. However, these protocols are not congestion responsive, and implement some ad-hoc form of congestion control, if at all. Recent proposals for the virtualization of networks [74] indicate the possible co-existence of multiple overlay networks in the future. These competing overlay networks that use UDP or IP tunnels can prove to be unfair when sharing the link bandwidth in the native network, a problem identified in [52]. One way to ensure fair sharing is to use one or more TCP [77] connections for forwarding data on every hop in every overlay network. In this thesis, we look at designing overlay networks that use TCP to forward data between overlay nodes, referred to as *Overlay-TCP networks*. Such networks have the desirable property that they ensure that multiple overlay networks will share native link bandwidths fairly, and that the Internet will not suffer due to lack of congestion control in the network. An additional goal in our design is to maximize the total end-to-end throughput of the data carried on the network.

1.3.1 Design of a Single Path in an Overlay-TCP Network

The design of the entire Overlay-TCP network involves enough challenges to warrant the design of a single path of overlay nodes in such a network, as a first step. We therefore start by focusing on a single path between a pair of source (ingress) and destination (egress) overlay nodes, that traverses multiple intermediate overlay nodes. We aim to optimize the throughput of the end-to-end transfer on the overlay path between the ingress and egress overlay nodes. We provision individual overlay hops by dynamically introducing multiple parallel TCP connections on each hop, with the aim of improving the total throughput on slower hops. Although we introduce multiple TCP connections on each overlay hop, our goal is to use as few connections on each hop as possible. The addition of a bound on the number of parallel TCP connections on each overlay hop ensures fairness between multiple overlay networks. We have proposed two schemes (*Isolated Rate Probing Scheme* and *Buffer Occupancy Scheme*) that assess the state of the overlay path and determine the number of TCP connections required on each overlay hop, without using too many extraneous connections. We refer to this architecture as the Adaptive Overlay-TCP Provisioning Architecture. We evaluate these schemes by implementing them on multiple overlay paths of 2-5 hops on PlanetLab [75], and show that they indeed improve the total throughput of the end-to-end data carried on the overlay path.

1.3.2 Design of an Overlay-TCP Network

Armed with our understanding of provisioning a single path in an Overlay-TCP network, we proceed to design an overlay network, keeping in mind the problem of fairness to other overlay networks. This involves significant challenges, in addition to those that we faced while designing a single path. These challenges include what we refer to as overlay-layer congestion, or *intra-overlay-network fairness*, wherein two data transfers that share a common overlay hop on their overlay paths should share the throughput of the TCP connection on that overlay hop fairly between themselves.

We propose a centralized algorithm to determine the minimum number of connections required on each overlay hop to maximize the total end-to-end throughput of data transfers

on all overlay paths in an overlay network, while maintaining fairness to other overlay networks, and maintaining intra-overlay-network fairness. We also propose three distributed heuristics that operate on varying amounts of information about the overlay network, and which approximate the allocation achieved by the centralized algorithm. We show that all four schemes indeed achieve the specified goal. In addition, we show that by varying the maximum number of connections allowed on overlay hops in each competing overlay network, we can vary the proportion of sharing between competing overlay networks.

1.4 Thesis Organization

The rest of the thesis is organized as follows. In Chapter 2, we present a review of the related work in this field. Chapter 3 looks at the construction of an example overlay network. In particular, it presents the investigation of the bootstrapping function in the Gnutella peer-to-peer network [46]. In Chapter 4, we focus on fairness issues in allocation of resources between competing multipoint-to-point sessions [49], that are caused due to the use of overlay networks for content distribution. Chapter 5 presents the Adaptive Overlay TCP Provisioning Architecture for a single path in an Overlay-TCP network [48], that aims to maximize the total end-to-end throughput of the data carried on the overlay path. Chapter 6 presents the design of an Overlay-TCP network [47] that aims to maximize the total throughput of the data carried on the network, while being fair to competing overlay networks. Chapter 7 concludes with a summary of the contributions of the thesis, and directions for future work in this area.

CHAPTER II

RELATED WORK

2.1 Background on Overlay Networks

Overlay networks are networks formed at the application layer, with the aim of offering services to applications, in addition to those offered by the underlying Internet. There has been significant research devoted to the design of overlay networks for specific applications. These applications span a wide range, some of which include the following. Content distribution [1, 4, 92] and multimedia streaming [4, 13, 72], have been popular applications of overlay networks, aimed at speeding up data delivery to clients. Since native multicast has faced deployment issues, overlay multicast [9, 20, 43, 56] has been proposed in order to provide the multicast service, although with the overhead of some redundancy in native paths. Overlay networks that provide quality of service [97] have been proposed to provide service guarantees beyond the best effort service provide by the Internet. Overlay networks that address fault tolerance [3] and improved routing [89] are aimed at giving applications more control over choosing the path for transferring data, based on the quality of the path in terms of outages, RTTs, packet losses etc. In our work, we focus on overlay networks for content distribution.

2.2 Proposed Modifications to Applications for Handling Applications' Demands

A major drawback of overlay networks is the significant investment necessary to setup such an infrastructure. Other means of achieving the service demands of applications, that do not face the infrastructure overhead, are implemented by the applications themselves. These solutions are specific to each application, depending on the applications' requirements, such as end-to-end throughput or latency. For example, for a bulk data transfer application, a simple strategy to improve throughput is to modify the transport protocol (say TCP [77]).

Such modifications include starting with a higher congestion window, or modifying the TCP slow start or not using a protocol that does congestion control at all and instead using UDP [76] instead. Although the use of the TFRC protocol [37] is a possibility, applications can be modified to not comply with the protocol. Some higher layer tricks include the use of persistent TCP connections for faster web transfers [61] over HTTP, or multiple parallel TCP connections [82] between the same client-server pair, using either the same path or multiple different paths. Some other modifications are discussed by Krishnamurthy et al. in [53]. However, due to the lack of control on applications, these solutions can be quite selfish and may prove to be unfair to other competing applications. In addition, they require a deeper knowledge of the protocols, and are difficult for end-users to accomplish. As stated earlier, in our work we will focus on the infrastructure-based solution i.e. the use of overlay networks for content distribution.

2.3 Peer-to-Peer Networks

Peer-to-peer networks have been studied extensively in the research community as well as commercially. As summarized in a survey of peer-to-peer content distribution technologies [95], peer-to-peer networks can be classified into structured and unstructured peer-to-peer networks depending on the type of topology of these networks.

Structured peer-to-peer networks maintain a particular structure when peers join or leave the system. The advantage of the structure of the system is that content can be located in the network very quickly. The primary disadvantage of this type of systems is that in a dynamic environment where peers join and leave the system at a very high rate, the overheads of maintaining the structure can be quite high. A high churn rate is quite typical for peer-to-peer systems, and hence can be a problem in structured peer-to-peer networks. Additionally, implementation of keyword-based search for locating content is difficult in these systems due to the way the system is structured. Some examples of structured peer-to-peer networks include Chord [96], CAN [81], Pastry [84], Tapestry [105], Oceanstore [55], PAST [85], and Splitstream [16], to name a few.

On the other hand, unstructured peer-to-peer networks do not maintain any structure when peers join or leave the system. The advantage with these networks is that the join/leave overhead is low. However, the search performance of peers in the system can be quite poor. The search queries are typically flooded throughout the system, with a time-to-live fiel in terms of number of peer-hops over which the query is forwarded. Although the query is expected to reach the peer(s) holding the requested content, this might not happen due to an expired time-to-live field, and the query might die before it reaches the appropriate peer. Hence there is a high probability that the search query might not get any response, even though the requested content might exist in the system. Thus, the topology and the neighborhood of a peer plays a significant role in a peer's search and download experience in such peer-to-peer systems. Some examples of unstructured peer-to-peer systems include Gnutella [32], Kazaa [51], Freenet [21], eDonkey [24], and BitTorrent [12]. In the first part of the thesis, we focus on the Gnutella peer-to-peer system, which is an unstructured peer-to-peer network.

Various studies have been undertaken with the aim of measurement based characterization [42, 58, 90] and modeling [27, 106] of different peer-to-peer systems such as BitTorrent, Kazaa and Gnutella. In particular, Saroiu et al. [88], Chu et al. [19] and Sripanidkulchai et al. [94] have studied the Gnutella network, with the aim of characterization of peers on the Gnutella network based on their uptimes, bottleneck bandwidths, latencies and other factors such as search popularity. Other studies such as the ones by Ng et al. [65] and Chawathe et al. [18] aim to improve a peer's search and download experience of peers. However, the bootstrapping process in unstructured peer-to-peer networks has received very little attention to date. In this thesis, we undertake a systematic measurement study of the current bootstrapping process in the Gnutella network.

2.4 Content Distribution Networks

Content distribution networks have been studied extensively in literature. Various studies [30, 44, 80] focus on the design of content distribution networks, a survey of a few of which can be found in [91]. Krishnamurthy et al. [54] and Johnson et al. [45] have compared

the performance of various commercially available CDNs. Biliris et al. [11] have proposed an architecture for interoperation of various content distribution networks and intelligent redirection of clients to other CDNs. In our work, we look at fairness issues resulting from the use of content distribution networks for data retrieval.

The problem of fairness of resource allocation to multiple connections between the same client and server pair has been studied previously. Proposals such as Webmux [28], Ensemble-TCP [25, 100], and Integrated Congestion Manager [6, 7] advocate the sharing of congestion information between the connections, in order to avoid synchronized losses due to congestion seen by all connections at the same bottleneck link. The Ensemble-TCP work differentiates between temporal sharing of information and spatial sharing of information. Spatial sharing implies the sharing of path information between connections that are active on the same path at the same point of time, as considered in [6, 7]. Temporal sharing implies sharing of path information between multiple connections started on the same path at different times. However, these connections should be started during a small enough time interval such that the network conditions would not have changed. Padmanabhan et al. [69, 71] present techniques for temporal sharing of information in their work.

In our work, we extend this problem to the case of multiple servers sending to a single client, with all connections active at the same time. The challenge in this case is that each connection in the session might traverse different paths and hence might have different bottleneck links. Thus, connections cannot share congestion information with each other, as proposed in the above papers.

Banchs [8] proposed the notion of user-fairness, in which every sender is considered as a single user in the network. This fairness scheme, referred to as *user fair queueing (UFQ)*, suggests that all connections started by a user (sender) should be considered as a single entity for rate allocation. This proposal moves away from the notion of per-connection fairness, towards session fairness from a sender point of view. This approach might put the client at a disadvantage because it might contact a server that has many other open connections, each one of them thus receiving a low rate. We compare the allocation achieved by this proposal with our session fair allocations. Our algorithms have a more fair allocation

from the client's point of view, compared to the user fairness approach.

In ATM research, there has been some work on multipoint-to-point session fairness. Fahmy et al. [26] give an informal definition for virtual-circuit based (VC-based) fairness, but without a formal definition or an algorithm to achieve the same. Moh and Chen [62] present an informal definition and an algorithm for multipoint-to-point multicast flow control. These approaches benefit from the ability of ATM switches to do complex processing and focus on intra-network merge point behavior. However, in the Internet we are limited in our ability to implement functionality in routers. We want the solution to be purely an end-to-end [87] solution.

Point-to-multipoint session fairness has been discussed in the multirate multicast context in the paper by Rubenstein et al. [86]. In the multirate multicast scenario, the bandwidth used by each session on any link is the *maximum* of the receiving rate of any of the downstream receivers. In our case however, the session rate on any link is the *sum of the total rates* of the upstream senders that are sending to a particular client. Hence we cannot directly use their approach to solve the multipoint-to-point session fairness problem.

2.5 Overlay-TCP Networks for Content Distribution

As mentioned earlier in this section, there has been significant research devoted to the design of overlay networks for specific applications. The co-existence of multiple such overlay networks is slowly getting some research focus. Peterson et al. [74] argue that network virtualization, and the co-existence of multiple overlay networks is a way to take architectural innovation forward. Deployment and management of multiple overlay networks has been addressed by studies such as the X-Bone architecture [101]. Keralapura et al. [52, 79] look at problems faced due to multiple overlay networks taking independent decisions in the event of route failures. However, the problem of sharing of native resources between multiple coexisting overlay networks has not been researched yet. In our work, we look at using TCP connections on each overlay hop in order to control the sharing of resources between multiple co-existing overlay networks.

Analysis of TCP connections on consecutive overlay hops has been undertaken in [98]

and [99] where the authors argue that back-to-back TCP connections between consecutive nodes on a path, are better than an end-to-end TCP connection in lossy environments. This is because the multiple TCP connections will have smaller RTTs and hence will recover from losses faster than an end-to-end TCP connection with a longer RTT. Lee et al. [57] suggest the use of TCP tunnels for avoiding congestion collapse in the Internet. He et al. [39] analyze congestion control mechanisms of the set of TCP connections between consecutive nodes, used for forwarding data on the Gnutella peer-to-peer network. All these papers employ schemes such as TCP backpressure, packet dropping, and explicit congestion notification to throttle the connections to the rate of the slowest hop. The end-to-end throughput in all these papers are limited by the rate on the slowest hop. None of these consider the option of using multiple TCP connections per overlay hop.

Amir et al. [2] employ TCP connections between overlay nodes in their architecture, and mention the possibility of multiple connections on each hop, but do not explore it further. The work on translucent proxying of TCP [83] proposes an architecture for intercepting TCP connections using proxies. They mention at various issues involved, including the buffer size and backpressure, but do not explore it further. Han et al. [36] look at data transfers using end-to-end TCP connections over an overlay network where multiple paths can be used for routing each TCP connection. They propose an algorithm to split the flow between each source-destination pair across multiple end-to-end TCP connections on multiple paths in a stable and optimal way. Note that they consider end-to-end TCP connections on the overlay and do not look at TCP connections on each overlay hop.

In the reliable multicast literature, a number of studies have explored the issue of end-to-end throughput using TCP connections on overlay hops. The ROMA architecture [56] places focus on reliability and propose increasing throughput using erasure resilient coding. Other research, such as that presented by Baccelli et al. [5] and Urvoy-Keller et al. [103] shows that the end-to-end throughput of an overlay path, and consequently the whole multicast tree is limited by the overlay hop with the minimum throughput.

The effect of using parallel TCP connections for transferring data to a single client from a single server on a single path, or from multiple servers on multiple paths has been studied

earlier [6, 25, 29, 64, 82]. However, consecutive parallel TCP connections on two or more hops have not been studied.

For the design of the Overlay-TCP network, as well as a single path in the overlay network, overlay nodes are required to estimate the TCP throughput that a single TCP connections will get on the overlay hop. In our work, we estimate this TCP throughput by either directly setting up a TCP connection, or by observing the socket buffer occupancy at the intermediate overlay node. Other possible methods include measuring the RTT and loss rate on each overlay hop and then using formula-based prediction, as described by Padhye et al. [68], or using history-based predictors, as discussed by He et al. [40].

CHAPTER III

CONSTRUCTION OF AN EXAMPLE OVERLAY NETWORK

3.1 Introduction

In this chapter, we investigate the process of construction of an example overlay network. In particular, we undertake a measurement study of the current bootstrapping process in the Gnutella network. As a reminder, the bootstrapping process refers to the function executed by a new peer that wishes to join the Gnutella network, during which it discovers other on-line peers and connects to them. This process is quite significant in the lifetime of a peer because unless this step is completed, a peer cannot take part in any of the search and download activities on the peer-to-peer network. Additionally, the initial neighbor peers that result from the bootstrapping process play an important role in the search and download experience of the peer.

We give the readers a quick historical perspective of the Gnutella network, as well as the bootstrapping process in the Gnutella network. In the initial version of Gnutella, all Gnutella peers, known as *servents* (due to their ability to function as *serv*ers as well as cli*ent*s), had equivalent functionalities in the network. The second version implemented the *Ultrapeer* [102] nodes, which had greater resources and capabilities than ordinary peers in the Gnutella network. These nodes are long-running, stable nodes with more bandwidth capabilities and therefore have the ability to support more incoming peer connections. They are equipped with the ability to cache the indices of normal or ordinary neighbor peers and respond to search requests on their behalf, thus reducing the load on these peers. This introduces a hierarchy in the current version of Gnutella, wherein the Ultrapeers form the core of the Gnutella network, and other ordinary peers are usually at the edge of the network.

Initially Gnutella users relied on word of mouth to determine the address of an on-line peer that would allow newly joining peers to tap into the network. The use of automated caching servers, as well as caching in the Gnutella servent itself, was introduced at a later time. As Gnutella gained in popularity after Napster was shut down, the caches ultimately became the pre-dominant bootstrapping technique [67]. Anecdotally, it has been observed that the switch from the use of word of mouth to the use of automated caches resulted in a significant change to the structure of the Gnutella network and a worsening of its performance [67].

Our investigation consists of three parts:

1. An analysis and performance comparison of the bootstrapping algorithms of four Gnutella servent[1] implementations: LimeWire [59], Mutella [63], Gtk-Gnutella [34] and Gnucleus [31].

2. A measurement-based characterization of the Gnutella Web Caching System [33] (GWebCaches), a primary component of bootstrapping algorithms.

3. A measurement-based analysis of the role of neighbor peers, resulting from different bootstrapping algorithms, in the search performance of a peer.

Based on our analysis of the data collected, we highlight below our four main findings about the current Gnutella bootstrapping system.

1. Although similar in the basic structure of the algorithm and the data structures used, the servent implementations differ in the details, with significant impact on their bootstrapping times, as seen in our measurements.

2. The neighbors of a peer play an important role in the search performance of the peer, thus pointing to the importance of the bootstrapping process.

3. An analysis of the request rates at different GWebCaches points to the disparity in traffic volume handled by these caches– some caches are very busy, and their host

[1]The implementations of Gnutella peers are referred to as *servents* because they function as *serv*ers and as cli*ents*. We use the terms *peers* and *servents* interchangeably.

Table 1: Messages Used in the GWebCache System

Argument	Cache Response
$ping$=1	*pong* message to servent
$urlfile$=1	list of caches
$hostfile$=1	list of online hosts
ip=<IPaddress>	host list is updated with IP address and port number
url=<URL of cache>	cache list is updated with URL
$statfile$=1	access statistics over last hour

and cache lists evolve much faster than some others. The load balancing goal of any distributed system is not really achieved in this system.

4. The GWebCache system is subject to significant misreporting of peer and cache availability. This is because the data reported in the updates to these caches is not validated by the caches. New peers wanting to join the system might thus waste time trying to connect to inactive GWebCaches or to off-line hosts returned by the GWebCaches.

The rest of the chapter is structured as follows. In Section 3.2, we give an overview of the bootstrapping process in different Gnutella servents, with special focus on the GWebCache system. We discuss the performance of the different servents with respect to their bootstrapping times in Section 3.3, and the role of the resulting neighbor peers in the search performance, in Section 3.4. In Section 3.5 we discuss the performance of the GWebCache system. In Section 3.6, we summarize our findings and discuss future work.

3.2 Bootstrapping Process in the Gnutella Network

In this section, we describe the bootstrapping process in the Gnutella servents we analyzed, and the functioning of the GWebCache system.

3.2.1 Gnutella Web Caching (GWebCache) System

A peer intending to join the Gnutella network requires the IP addresses of online peers in the network. Currently, the GWebCache system functions as a distributed repository for maintaining this information. Peers can query the caches in this system to get a list of online

peers to connect to. In the first execution of a particular Gnutella servent, the only means to locate other online peers is the GWebCache system. In successive executions, individual servent implementations try approaches apart from the GWebCaches, such as maintaining local lists of hosts seen during their earlier runs. We first discuss the GWebCache system, as it is an important component of the bootstrapping functionality, and is essential in the understanding of the servent bootstrapping algorithms.

The GWebCache system [33] is a network of voluntarily-operated caches that maintain a list of online peers accepting incoming connections. When a new peer wants to join the Gnutella network, it can retrieve the host list from one or more of these GWebCaches. The GWebCaches also maintain a list of other caches in the system. Typically each cache maintains a list of 10 other caches and 20 hosts that are currently accepting incoming connections.

The peers in the Gnutella network are responsible for keeping the information in these caches up-to-date; the caches do not communicate with each other at any time. A host accepting incoming connections is supposed to update the caches with its IP address and port number, and with information about some other GWebCache that it believes is alive. The GWebCaches maintain the host and cache lists as first-in-first-out lists.

Table 1 lists the messages sent by a client using the GWebCache protocol. An HTTP request of the form "URL?*argument*" is sent to the webserver at which the cache is located. The caches respond as shown in the table. Note that the GWebCaches do not maintain any information about the online hosts, other than their IP addresses and port numbers. Some additional information such as whether the peer is an *ultrapeer* or a *leaf node* and optional details such as its bandwidth, its current online time in the system and the number of files it shares might have helped new peers make more intelligent decisions in selecting initial neighbors.

3.2.2 Bootstrapping Algorithms of Gnutella Servent Implementations

In this section, we discuss the bootstrapping algorithms of the Gnutella servents that we compared, and point out the differences between them. We analyzed Limewire v2.9 [59],

1. Initialize the following data structures in memory by reading the corresponding files from disk—
   - list of caches
   - list of known hosts
   - list of permanent hosts
   - list of ultrapeer hosts (except in Gtk-Gnutella)
2. Depending on mode (ultrapeer/normal), determine the minimum number of connections to be maintained.
3. Try to establish the minimum number of connections to peers in the order:
   - In LimeWire and Gnucleus, try to connect to ultrapeers.
   - Try to connect to any host in the known hosts and permanent hosts lists.
   - If the servent is still not connected, request the host list from a GWebCache (multiple GWebCaches in LimeWire) and try to connect to these hosts.
4. Periodically, a connection watchdog checks whether the minimum num of connections (step 2) are alive. If not, try to establish a new connection (step 3).
5. Periodically update a cache with its own IP address and URL of another cache (for LimeWire and Mutella, this is done only if in ultrapeer mode)
6. On shutdown, write the different files to disk, for retrieval on next startup.

Figure 1: Generic Bootstrapping Algorithm in Gnutella Servents

Gtk-Gnutella v0.91.1[34], Mutella v0.4.3 [63] and Gnucleus v1.8.6.0 [31]. All these versions support retrieval from and updates to the GWebCache system. The bootstrapping processes in the four servents are similar in their use of the GWebCache system and the local caching of hosts.

The data structures maintained by these servents include a list of known GWebCaches, which is periodically populated with addresses of new GWebCaches. Servents also maintain lists of *known* hosts and *permanent* hosts, the definitions of which differ slightly in different servents. Informally, permanent hosts are hosts that the servent managed to contact in current and previous runs. Some servents also maintain a separate list of ultrapeers. Ultrapeers [102] are hosts with high bandwidth and CPU power, and long uptimes and are not firewalled. Normal or leaf nodes have lower CPU and bandwidth capabilities and typically

Table 2: Differences in Implementation of Different Gnutella Servents

Characteristic	Limewire	Mutella	Gtk-Gnutella	Gnucleus
Is an Ultrapeer list maintained?	Yes	Yes	No	Yes
Are Ultrapeers given priority when connecting?	Yes	No	No	Yes
Are host & cache lists prioritized by age?	Yes	No	Yes	No
What modes send updates to GWebCaches?	Ultrapeer mode	Ultrapeer mode	Any mode	Any mode
Number of hardcoded caches	181	3	3	2

connect to Ultrapeers.

Figure 1 outlines the generic bootstrapping algorithm, and Table 2 summarizes the differences in the implementations, as discussed below.

1. Limewire and Gnucleus maintain a list of ultrapeers and give priority to hosts in this list during connection initiation. Since ultrapeers have relatively long uptimes and the capability to support more incoming connections, prioritizing these peers during connection initiation increases the probability of successfully connecting to a peer. Although Mutella also maintains a list of ultrapeers, this information is not used during bootstrapping. When establishing a connection, it picks a host at random from the lists of ultrapeers, known hosts and permanent hosts. Gtk-Gnutella does not distinguish between ultrapeers and normal peers thus performing relatively poorly in terms of its bootstrapping time, as will be seen in Section 3.3.

2. LimeWire and Gtk-Gnutella prioritize their host and cache lists by age. This enables them to act on more recent (and hence more accurate) information.

3. Although all four servents we examined support the GWebCache system for retrieving information, LimeWire and Mutella support updates to the GWebCaches only in the ultrapeer mode. This is better for the system because the probability of ultrapeers accepting incoming connections is higher than when in the leaf mode. Gtk-Gnutella and Gnucleus update the GWebCaches even in the leaf mode. This is not good for the system as the limit on number of connections is low in leaf mode, and they might

not be accepting any more incoming connections by the time a peer tries to connect to them.

4. The Gnutella servents have a set of hardcoded caches, which are used during the very first run of the servent, before any other information about caches or hosts is known. As seen in Table 2, compared to other servents LimeWire has a very high number of hardcoded caches (181), 135 of which were active when we tried to ping them at the Gnutella level.

In the next section, we will discuss the effects of these differences in bootstrapping algorithms on the performance of different servent implementations.

3.3 Measurement of Bootstrapping Performance of Servent

In this section, we compare the performance of the servents considered in our study, based on their *bootstrapping times*. We define the bootstrapping time of a servent as the time between the start of the servent and the initial establishment of the first *stable* Gnutella-level connection. A "Gnutella-level" connection is established after the Gnutella handshake messages are exchanged between the two connecting peers. We say that a connection is *stable* if it is maintained for at least *threshold* seconds.

3.3.1 Measurement Methodology

We modified the sourcecode of the three Linux-based servents (LimeWire, Gtk-Gnutella and Mutella) to log the times at which the application was started and shut down, and when a Gnutella-level connection was established and terminated. For the Windows-based servent (Gnucleus), we used Windump [104] to collect packet traces, and then analyzed them to determine the connection times.

We started the Linux-based servents once every hour, synchronously at two locations, at a university campus on a Fast Ethernet Link and at a residence on a DSL link to a local ISP. We started Gnucleus once every three hours at the university location only. Each servent was allowed to run for 15 minutes, after which it was shut down. In the following section we

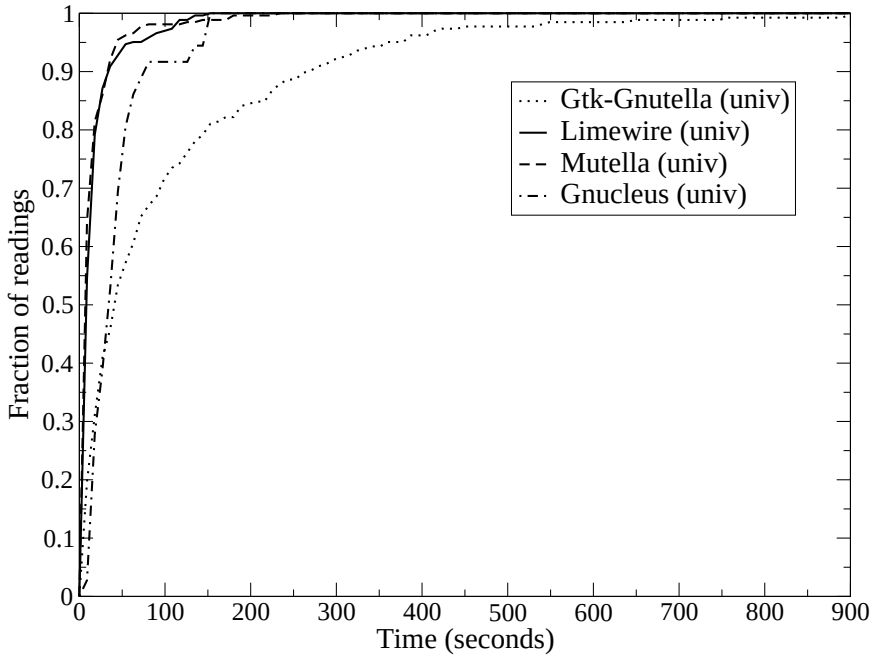


Figure 2: CDF of Bootstrapping Times of Servents at University

analyze the bootstrapping times measured during an 11-day experiment. One limitation of our study is that both locations in our experiments have high bandwidth access links. We did not run any experiments at a slower access link.

3.3.2 Performance Measurements

Figure 2 shows the cumulative distribution function of the bootstrapping times of the four servents at the university location. In this graph we set *threshold* to 120 seconds. We analyzed the bootstrapping times with different values for *threshold* and observed similar results. The graphs for the bootstrapping times of servents on the DSL link are similar.

The most striking observation is that Gtk-Gnutella performs much worse than Mutella and LimeWire. We conjecture that this is due to the fact that Gtk-Gnutella does not differentiate between ultrapeers and normal peers. Also, once it selects a particular GWebCache to contact for retrieving the host list, it uses it for 8 consecutive updates or retrievals. In Section 3.5, we will see that cache quality varies; thus, maintaining a poor choice of cache can affect performance. Gnucleus also performs worse than Mutella and LimeWire, but better than Gtk-Gnutella. This is probably because the GWebCache list and the different

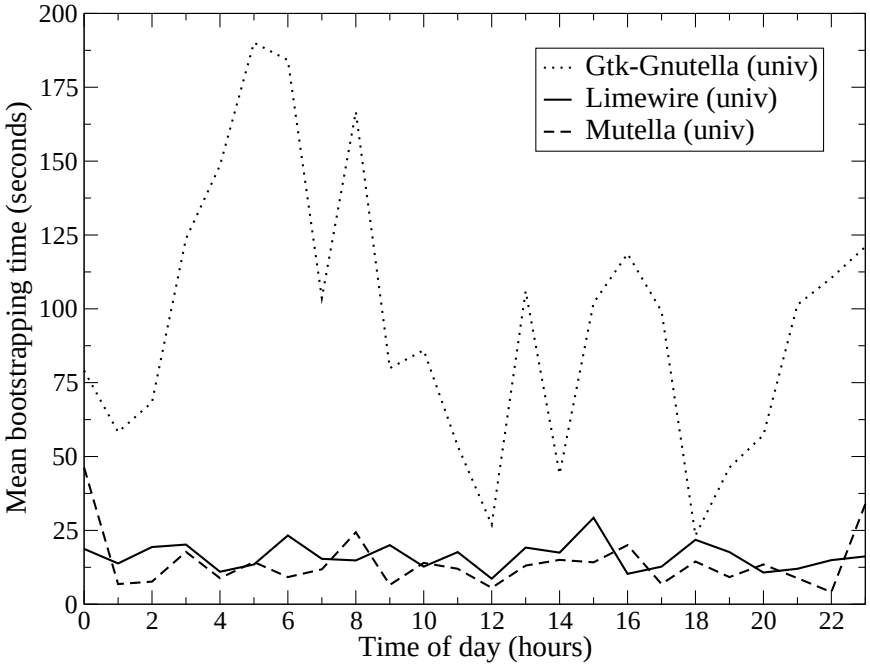


Figure 3: Mean Bootstrapping Times of Servents at Different Times of the Day

host lists are not prioritized by age in the Gnucleus implementation.

Figure 3 shows the mean bootstrapping times for the three Linux-based servents at the university location for different times of the day. LimeWire and Mutella perform almost the same throughout the day. Gtk-Gnutella, which does not differentiate between ultrapeers and normal peers performs similar to LimeWire and Mutella around noon or late afternoon, when there are more normal peers online in the system. Early in the morning, with very few normal peers around, Gtk-Gnutella shows a higher mean bootstrapping time. This highlights the importance of ultrapeer awareness on the part of a Gnutella servent.

Although we started multiple instances of Gnutella servents on the same local area network, none of our peers were able to discover each other in any one of our experiments over two weeks. This highlights the lack of Internet location awareness in the GWebCache system and in the local host list of the servents.

In the next section, we discuss the importance of neighbor peers (resulting from the bootstrapping process) in the search performance of peers.

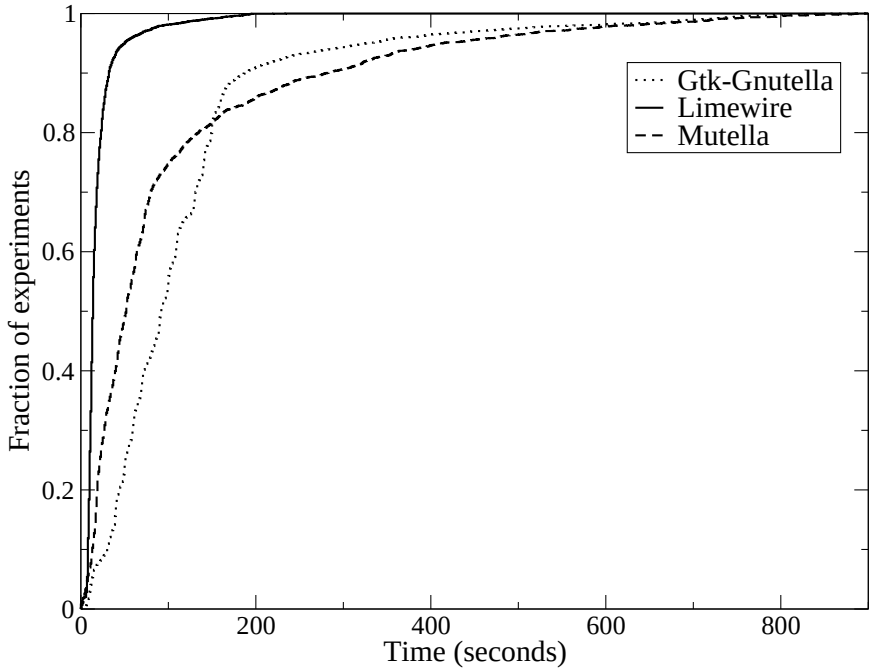


Figure 4: CDF of Time to Receive First Query Response at Servent

3.4 Importance of Neighbor Peers

A peer gets access to the peer-to-peer network through its directly connected neighbors. The peers that these neighbors have access to, within an N-hop radius (usually N=7), comprise the *neighborhood* of the peer. All query messages originated by the peer will be forwarded to this neighborhood. The number of peers in this neighborhood, the types of files shared, the number of files shared amongst all these peers will reflect on the search performance of a peer.

We studied the effect of neighbors on search performance of LimeWire, Mutella and Gtk-Gnutella for the the 15 most popular search queries [19]. The performance metric we considered is the time to get the first response, which is the time-lag from when the servent is bootstrapped and issues the query, to the time when it gets the first response. Figure 4 shows the CDF of this response time for the top 15 queries issued by that servent during any experiment.

We found that there is usually a significant variation in the time to get the initial response. Limewire performs the best, primarily because during bootstrapping it prioritizes

ultrapeers, who usually have access to a larger neighborhood. Mutella and Gtk-Gnutella perform worse, and take more than 5 minutes to give a result, in about 10% of the experiments.

We conclude that for a good search experience it is very important to have a set of good neighbors that provide access to many peers, sharing many files.

3.5 Performance of the GWebCache System

We analyzed the performance of the GWebCache system with reference to the properties of a good distributed caching infrastructure (e.g., sufficient total system capacity, good load balancing, reliability of cached items, and physical or topological proximity of cached items served). With this goal in mind, we performed a measurement study of the system at two levels, globally and locally.

3.5.1 Global GWebCache System Performance

We studied the global GWebCache system by periodically crawling all the caches. We sent requests in the format shown in Table 1 to each active cache, according to the information required. We collected multiple traces over a five month period (Apr-Sept 2003), with the goal of answering the following questions.

1. How many GWebCaches does the system comprise of? How many of the reported caches are active at any time?

We retrieved the *cache list* every 30 minutes, starting with a seed GWebCache and crawled the caches returned, until we had polled all reachable caches. We also determined the number of active GWebCaches by sending Gnutella *ping* messages to the crawled list.

Although we found URLs of 1638 unique GWebCaches in 5 months, only one-fourth of them (403) were active at any time, and at most 220 of them were active during a single poll. This is quite a low number of reachable GWebCaches, potentially serving about 100000 active hosts[2] at any time on the Gnutella network, only 10% of which accept incoming connections. This indicates that the GWebCache system might get overloaded.

[2]As shown by the Limewire [59] hostcount, during the period of our study.

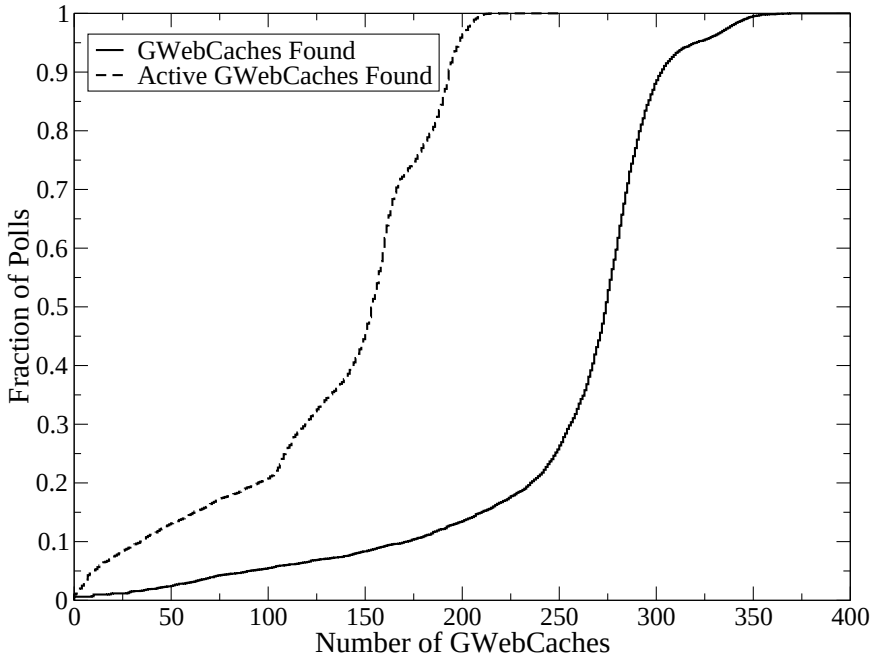


Figure 5: CDF of Active Caches in the GWebCache System

Figure 5 shows the CDF of the number of GWebCaches found during each of our polls, and the number of caches which were actually active (i.e. responded to out Gnutella-level *ping* messages). Most of the time, only about 160 caches out of 280, or about 60% were active. This is because the GWebCache system does not have any means of deleting an introduced cache. Since peers update caches with URLs of caches they know of (probably even cached from previous runs), without necessarily knowing whether they are alive or not, it is quite likely that inactive caches are reported for a long time.

2. What are the access patterns for different requests (cache list, host list, and updates) at different GWebCaches? What are the differences in access patterns across different GWebCaches and in the whole system?

We retrieved the *statistics file* every hour from each active GWebCache. The statistics file gives the number of update requests and total requests for cache and host lists the GWebCache received within the last hour.

Figure 6 shows the CDF of the mean update rates to the cache and host lists (determined by analyzing lists retrieved in consecutive polls) at a single GWebCache. About 80% of the GWebCaches get cache list update rates of 10 per hour or less, while a few caches receive

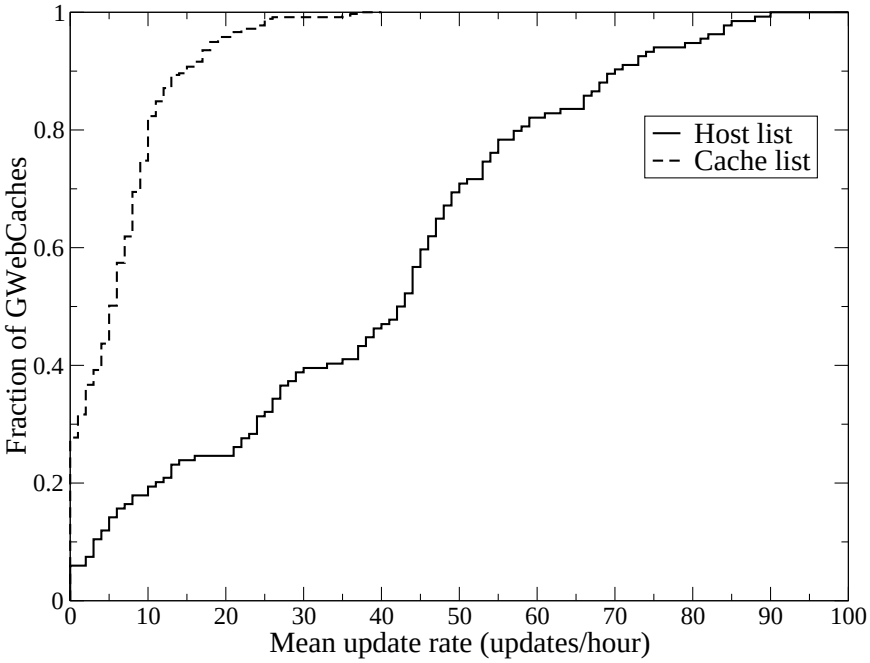


Figure 6: CDF of Mean Update Rate for Host and Cache Lists at a Single GWebCache

very high update rates, upto 40 updates per hour. About 60% GWebCaches receive host list update rates of less than 1 per minute, whereas others receive update rates almost twice as much.

Similarly, Figure 7 shows the CDF of the mean and maximum total request rates (as reported by the statistics files) at a single GWebCache. These include requests for the host and cache lists and updates to both lists from peers. About 90% of the GWebCaches receive an average request rate of 3000 per hour or less, whereas some caches receive extremely high loads of the order of 20000 requests per hour on an average, with a maximum of 30000 requests per hour.

This points to the disparity in the type of GWebCaches in the system. Some caches are very busy, with their lists evolving faster and receiving high request rates, whereas others are relatively lightly loaded. The servents we studied have some hardcoded GWebCaches, indicating that the request rates to these caches could be very high. This suggests poor load balancing in the GWebCache system.

3. How does the host list at a single GWebCache and at all GWebCaches evolve? What percentage of new hosts added to the GWebCaches are unique?

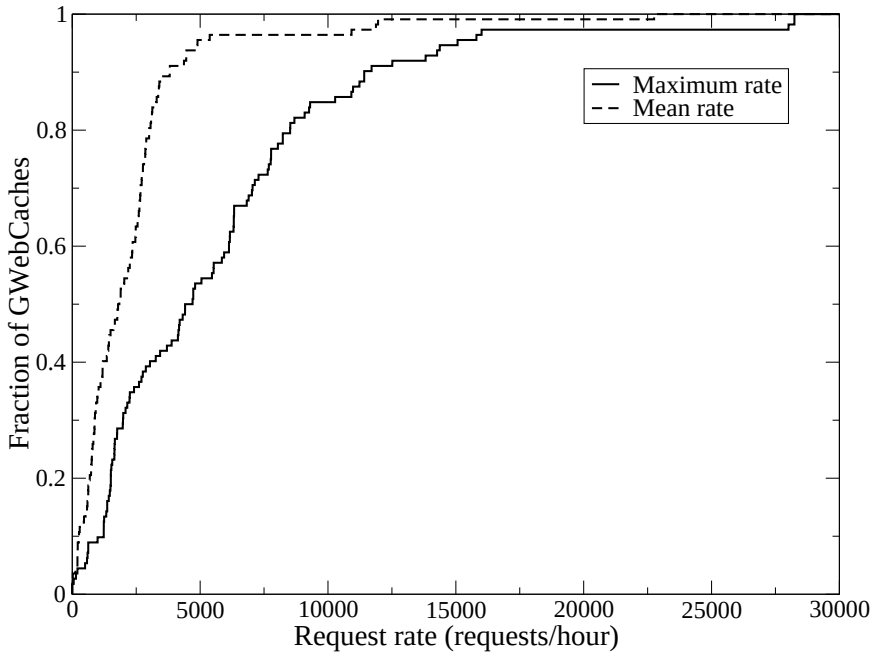


Figure 7: CDF of Request Rate for Host and Cache Lists at a Single GWebCache

We retrieved the *host list* from the active GWebCaches every 5 minutes, and studied its evolution at a particular cache and in the whole system. As expected, the host list evolves much faster than the cache list in any GWebCache. During a 15-day period in our study, we saw over 300000 unique IP address:port combinations in all GWebCaches.

Figure 8 shows the CDF of the host update rates at all GWebCaches in the system. The rightmost line shows the CDF of the host updates received at all GWebCaches in the system. The dotted line shows the CDF of the host updates with unique IP address:port combination at each cache. The leftmost curve with the dashed line shows the CDF of the unique IP address:port combination seen in the whole system. The average rate for unique IP address:port updates at a particular GWebCache is lower than the actual update rate at that cache. The update rate for IP address:port, unique throughout the system is much lower, almost by a factor of 10. This suggests that the same hosts (presumably ultrapeers) update the GWebCaches frequently with their IP addresses, leading to a high replication rate of the same addresses in multiple caches.

4. In the host list returned by the GWebCaches, how many hosts are alive, how many are accepting Gnutella-level connections, and how many are ultrapeers?

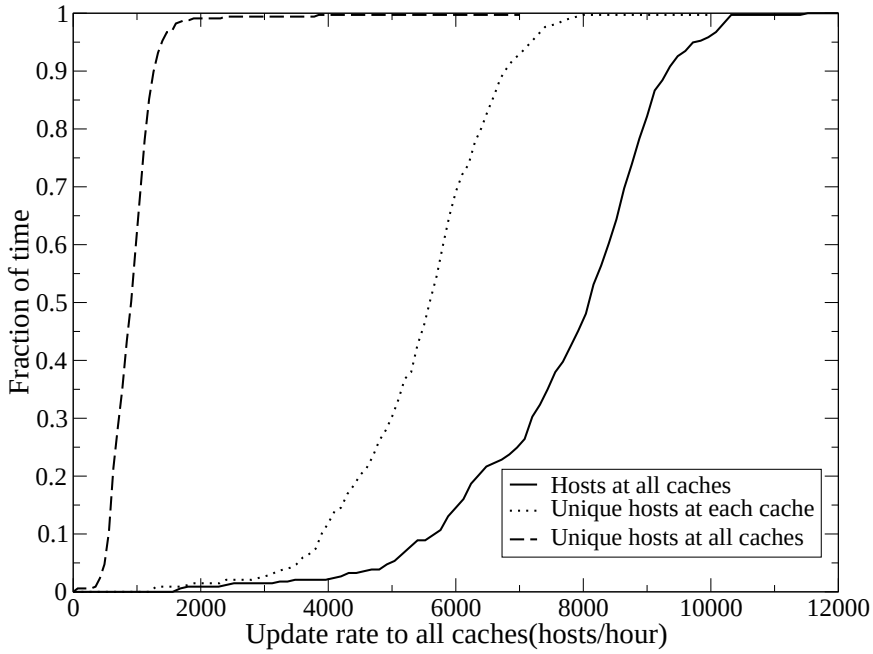


Figure 8: CDF of Host List Update Rate at all GWebCaches

We sent Gnutella-level connect messages to the hosts in the host lists returned by the GWebCaches. If a TCP connection was established, we determined that the host was alive. If a Gnutella-level connection was established, we determined that the host was accepting incoming connections. Out of the hosts that responded with the proper *pong* response, we determined whether the host was an ultrapeer or not, using a field *X-Ultrapeer: True/False* in the response.

When we tried connecting to the hosts in the host lists retrieved, on an average we found 50% peers online, 16% peers accepting incoming Gnutella-level connections, and 14% ultrapeers. This shows that a surprisingly low number of peers indicated in the GWebCaches are actually accepting incoming connections. This could be a cause for the high bootstrapping times of servents in some cases, where peers waste time trying to connect to off-line hosts returned by the GWebCaches. The reliability of content served by the GWebCache system is therefore questionable.

Our measurement methodology has several limitations. Since we polled the GWebCaches starting with a seed GWebCache, we will miss caches in any disconnected components of the GWebCache system. Also, between the times we retrieved the list and tried

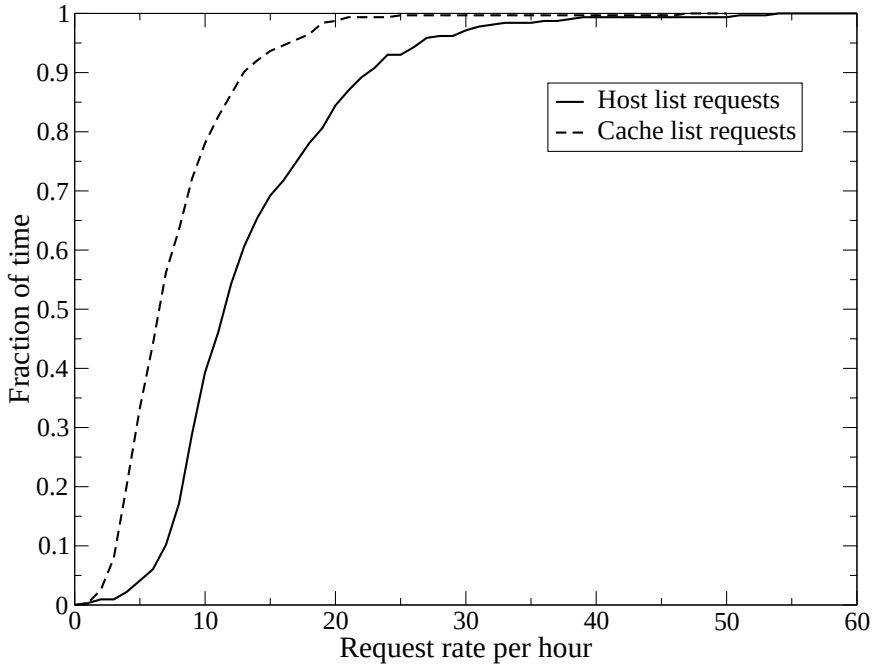


Figure 9: CDF of Request Rates for Host and Cache Lists at Local GWebCache

connecting to the peer, the peer could have gone offline. We assume that the information returned by the GWebCaches during our polls is valid (i.e., the GWebCaches are not misconfigured or misbehaving).

3.5.2 Experience of a Local GWebCache

We set up a GWebCache locally by modifying a publicly available PHP script for the GWebCache v0.7.5 [33] to log request arrivals, and advertised it to the global caching infrastructure.

We introduced the GWebCache into the system using a feature in the Gnucleus servent. The servent first ensures that the GWebCache is alive, using the Gnutella *ping-pong* message exchange. It then iterates through its list of known GWebCaches and sends an update request to each cache, with the URL of the new cache. It also adds the new cache to its local list of GWebCaches. In our polls, we found that 17 GWebCaches pointed to our cache immediately after the advertisement, after which the number stabilized to about 4 or 5 GWebCaches pointing to ours every hour.

Figure 9 shows the CDF of requests for the host and cache lists at the local GWebCache.

Our cache received a request rate of about 15-20 per hour for the host list and about 5-10 per hour for the cache list. It received update rates of about 7 per hour to the host list and 4 per hour to the cache list. Comparing these rates to those of other GWebCaches seen earlier, we can see that our local cache is used less frequently than the other GWebCaches.

One short-coming of the GWebCache system is that once a GWebCache is in the system, it is quite difficult to remove it from the system. This is because peers store the URLs of GWebCaches they have seen in the past. Even after a GWebCache is taken down, one of the peers that has cached its URL might end up sending a request. After we terminated our GWebCache, we saw requests to the cache in the webserver logs for quite a long time.

3.6 Summary

In summary, our study highlights the importance of understanding the performance of the bootstrapping function as an integral part of a peer-to-peer system. We find that (1) Although all servents implement a similar structure for the bootstrapping algorithm, there is considerable variation among various implementations, that correlates to their bootstrapping performance. (2) The neighbors of a peer play an important role in the search performance of the peer. Hence it is important that the bootstrapping process results in good neighbors. (3) Even though the GWebCache system is designed to operate as a truly distributed caching system in keeping with the peer-to-peer system philosophy, it actually operates more like a centralized infrastructure function, with some GWebCaches handling a large volume of requests while others are idle. (4) The GWebCache system is subject to significant misreporting of peer and GWebCache availability due to stale data and absence of validity checks.

CHAPTER IV

MULTIPOINT-TO-POINT SESSION FAIRNESS IN OVERLAY NETWORKS FOR CONTENT DISTRIBUTION

4.1 Introduction

In this chapter, we will look at fairness issues that arise due to the use of overlay networks for content distribution.

In Chapter 1, we have presented a number of example applications which indicate that data transfers are increasingly being performed over a set of point-to-point connections from multiple servers to a single client. We refer to such a set of connections as a *multipoint-to-point session*.

The long-standing max-min fair [10] rate allocation strategy, which proposes a fair allocation for competing connections, is based on a network with point-to-point connections. It allocates rates to connections, considering each connection independently. In the current Internet, this strategy favors sessions with more connections because a session with multiple connections from multiple servers to a single client will be given a higher total rate than a session with a single connection between one of the servers and the client.

For example, if two connections belonging to a single session are bottlenecked at the same link, the max-min fair rate allocation strategy says that each connection should get an equal share at that link. Thus, the session as a whole will receive twice as much bandwidth as any other single-connection session bottlenecked at that link. One may view this as being "unfair" from the point of view of a session with fewer connections. In this work, we explore the possibility of treating sessions "equally", irrespective of the number of connections they comprise[1].

[1]The appropriate use of our mechanisms is a matter of policy.

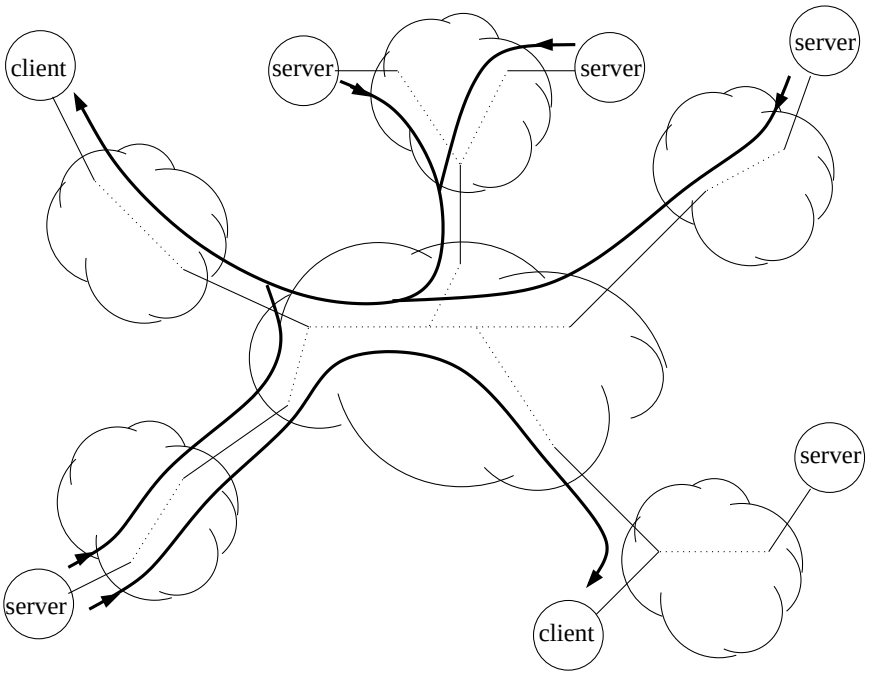


Figure 10: Multipoint-to-Point Sessions in the Internet

In general, the data path of a multipoint-to-point session forms a tree with data flow from the leaves to the root, as shown in Figure 10. The whole network will thus have a set of senders and a set of receivers, with each session comprising some of these senders and a single receiver[2]. Each connection in the session might have different individual bottlenecks. The data path of a session might thus have multiple bottlenecks and sharing of these bottlenecks at a session level, as opposed to the current connection-level sharing, is a challenging problem.

Our goal in this chapter is to explore this issue of fairness of rate allocations from a session perspective. This problem has been alluded to as an open problem by Balakrishnan et al. [7] and Padmanabhan [70]. In particular, we look at *multipoint-to-point sessions*, which are defined as *a set of point-to-point connections started from multiple servers to a client in order to transfer an application-level object.* We explore answers to the questions:

- What are "reasonable" definitions for session fairness?

[2]For simplicity, we assume a network with only point-to-point and multipoint-to-point connections. We expect that point-to-multipoint connections can be accommodated as a set of point-to-point connections.

- How do connection-fair allocations differ from session fair allocations?

We use Raj Jain's fairness index [78], variance of session rates, and mean, minimum and maximum session rates as quantitative metrics to compare rate allocations. We show that the fairness index for session rates improves with the session fair allocations, while maintaining or improving overall utilization, in comparison with the connection fair allocation. Also, the variance and minimum of the session rates achieved with the session fair allocations are lower than those with the connection fair allocation.

The outline of the chapter is as follows. We elaborate on the problem statement in Section 4.2. We then formalize the definitions of session fair allocations, and give algorithms to achieve the same in Section 4.3. The evaluation results comparing the session fair algorithms with each other and with the original connection fair algorithm are presented in Section 4.4. Implementation issues are briefly outlined in Section 4.5, and we conclude in Section 4.6.

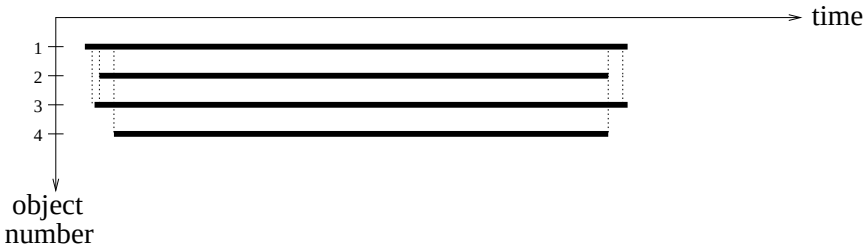
4.2 Discussion of Problem Statement

This section provides more detail regarding the problem of multipoint-to-point session fairness, and describes informally the proposed fairness definitions.

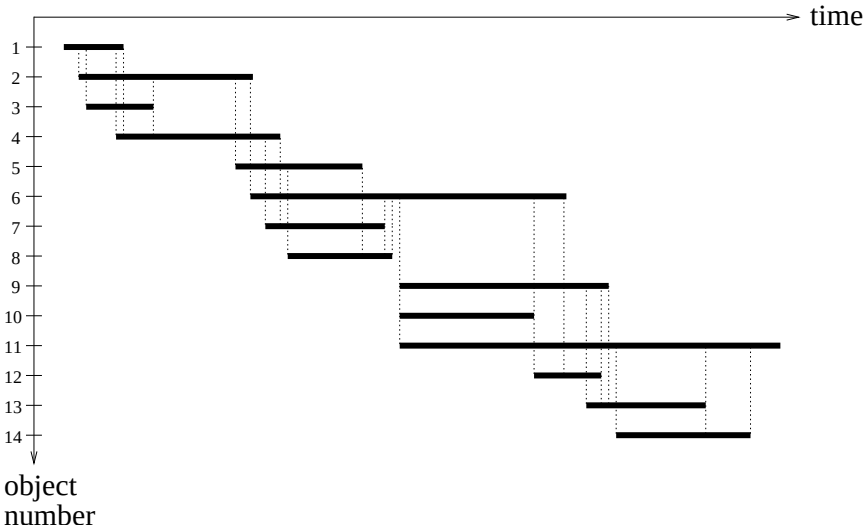
4.2.1 Static and Dynamic Sessions

We begin by differentiating between static and dynamic sessions. *Static sessions* are multipoint-to-point sessions comprising connections which, for the purposes of rate allocation, start and terminate at approximately the same time. On the other hand, connections in *dynamic sessions* start and terminate at different times. Thus, the composition of a dynamic session varies significantly over time. Clearly, the designation of static or dynamic must be based on the temporal granularity of resource allocation.

A typical static session is shown in Figure 11(a). The length of each horizontal line denotes the duration that the connection was active. The Multiple Description Streaming Media with Content Delivery Networks approach [4], the parallel-access approach in Rodriguez et al. [82], and retrieval by companies like PeerGenius.com [73], CenterSpan.com



(a) Connections in a static multipoint-to-point session



(b) Connections in a dynamic multipoint-to-point session

Figure 11: Static and Dynamic Multipoint-to-Point Sessions

[17], digitalfountain.com [23], are examples of applications using static sessions.

Retrieval of webpages, on the other hand, is typically an example of a dynamic session. Figure 11(b) from Liston et al. [60] shows a typical object retrieval with a number of connections (possibly to different servers) starting and ending at different points in time.

In this work, we focus on static sessions. Static sessions are important in their own right, given the number of near-simultaneous parallel download applications that are increasingly under development. Further, the definitions and algorithms proposed here can be used as building blocks to solve the session fairness problem for dynamic sessions[3].

4.2.2 Inter-Session and Intra-Session Fairness

The notion of session fairness has two components—

- *Inter-session fairness* refers to the fairness of the total session rates with reference to each other.
- *Intra-session fairness* refers to the division of the session rate amongst the connections constituting the session.

Inter-session fairness advocates fair sharing of network resources between independent sessions on intersecting paths. To this end, we propose the notions of *normalized rate session fairness* and *per-link session fairness*, the details of which are presented in Section 4.3.

Normalized rate session fairness is based on the notion of *weight of a connection*, which is used to express, in some sense, the number of connections in the session. Each connection in a session is assigned a weight, such that the total weight of the connections in a session is 1. The rate allocation for each connection at every link in the network is based on the weight of the connection. The constraint on the total weight of the connections in a session ensures that the session rates are fair with reference to each other. In fact, if all connections in all sessions traverse the same bottleneck link, all sessions will get an equal session rate.

[3]Dynamic sessions may require additional or alternative strategies, particularly if the connections are highly dynamic.

Per-link session fairness, on the other hand, looks at each link in the network and tries to ensure that each session shares the capacity of the link in a fair manner. The capacity of every link in the network is divided equally between the sessions traversing that link. The sessions divide the rate assigned to them at that link amongst the connections belonging to that session, that traverse the link[4].

Intra-session fairness options allow the client in each session to decide for itself as to how each connection in the session should be treated, in terms of rate allocation. An allocation is said to be *"source and session fair"* if each connection in the session is treated equally. That is, the rate allocation within the session is fair from the point of view of the sources in the session. If each connection in a session is given a rate allocation in proportion to the amount of data retrieved over that connection, the allocation is said to be *"data and session fair"*. This allocation has the appeal of allowing sources with more data to operate at higher speeds than sources with less data. If each connection in a session is given a rate allocation based on its path in the network (i.e. based on sharing of links between connections in the session), the allocation is said to be *"path and session fair"*. For simplicity, we describe algorithms that achieve "source and session fairness".

The normalized rate session fair algorithm is very flexible and can be easily extended to incorporate intra-session fairness definitions, based on the clients' choices, by adjusting the weights of the connections. It is also amenable to a distributed end-to-end implementation. On the other hand, it is difficult to extend the per-link session fair algorithm to incorporate the intra-session fairness definitions. This is because the algorithm has implicit weights embedded for each connection. There is no way for the client to specify (say with the weight of the connection) that it wants otherwise, in terms of rate allocation within the session, at any link.

In the next section, we discuss these definitions and algorithms in detail.

[4]We conjecture that per-link session fairness is equivalent to normalized rate session fairness for some globally-coordinated assignment of weights. However, such an assignment is sufficiently complex that it makes sense to describe per-link session fairness separately.

4.3 Definitions and Algorithms

In this section we formalize the definitions for different fairness criteria for rate allocation to multipoint-to-point sessions. For comparison, we define the connection max-min fairness criterion. We also present algorithms to achieve the allocations defined, and discuss properties of these allocations with respect to their session rates. The notations used are given in Table 3[5].

Table 3: Notations Used in Multipoint-to-Point Session Fairness Definitions and Algorithms

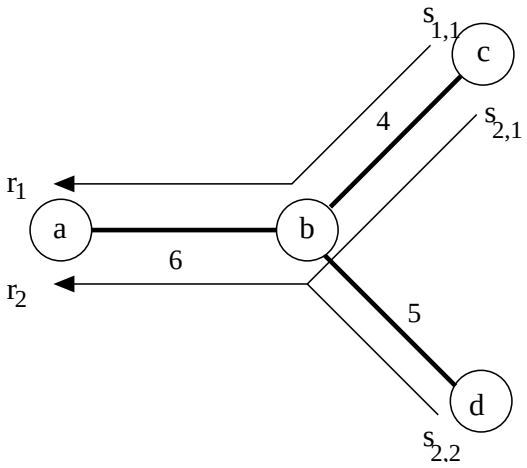
l_j	jth link in the network
C_j	capacity of link l_j
c_j	residual capacity of link l_j
S_i	ith session in the network
m	number of sessions in the network
k_i	number of senders in session S_i
$s_{i,k}$	kth sender in session S_i
r_i	ith receiver in the network
$R_{i,j}$	set of receivers in S_i whose data path traverses l_j
R_j	set of receivers in all sessions whose data path traverses l_j
$a_{i,k}$	data rate of sender $s_{i,k}$
$\overline{a}_{i,k}$	data rate of sender $s_{i,k}$ using an alternate allocation
$w_{i,k}$	weight of sender $s_{i,k}$
$z_{i,k}$	normalized rate for sender $s_{i,k}$
$b_{i,k,j}$	link rate for kth connection in session S_i on l_j
$b_{i,j}$	link rate for session S_i on l_j
b_j	link rate for all sessions on l_j
a_i	data rate for session i (or session rate of S_i)
X, Y	set of unassigned receivers
$c_{i,j}$	residual capacity for session i on link l_j
$u_{i,j}$	number of unassigned connections in session i on link l_j
u_j	number of unassigned sessions on link l_j

4.3.1 Connection Fairness (Max-Min Fairness)

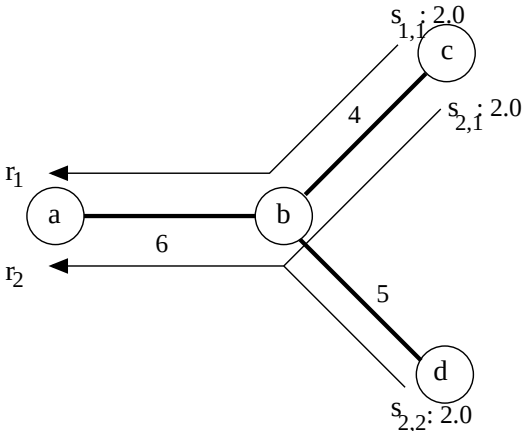
We first present the traditional connection max-min fairness definition, for comparison with our session fairness definitions.

Definition: An allocation of sender rates $a_{i,k}$ is said to be *connection fair* [10] if it is feasible and for any alternative feasible allocation of sender rates where $\overline{a}_{i,k} > a_{i,k}$, there is

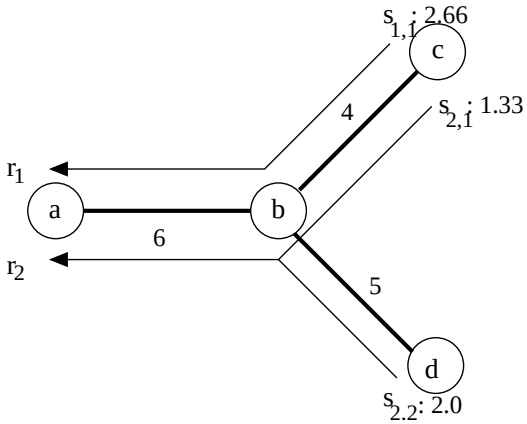
[5]For consistency, we adopt a notation similar to that used in Rubenstein et al. [86].



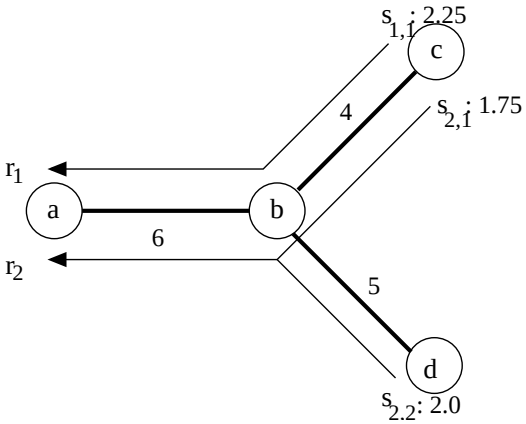
(a) Session Layout



(b) Connection fair allocation (S_1 : 2.0, S_2 : 4.0)



(c) Normalized rate session fair allocation (S_1 : 2.66, S_2 : 3.33)



(d) Per-link session fair allocation (S_1 : 2.25, S_2 : 3.75)

Figure 12: Example to Illustrate Multipoint-to-Point Session Fair Algorithms and Resulting Allocations: (a)Example Network, (b)Connection Fair, (c)Normalized Rate Session Fair, (d)Per-Link Session Fair

some other sender $s_{i',k'} \neq s_{i,k}$ such that $a_{i,k} \geq a_{i',k'} > \overline{a}_{i',k'}$.

An allocation is said to be *feasible* if each sender $s_{i,k}$ is assigned a sending rate $a_{i,k}$, subject to the constraint

$$\forall j : \{\sum a_{i,k} : s_{i,k} \in R_j\} \leq C_j$$

Informally, the rate of a connection at its bottleneck link is greater than or equal to the rate of any other connection at that link. The feasibility condition ensures that the link is not loaded beyond its capacity. The algorithm for achieving this allocation is given in Bertsekas and Gallager [10].

Example allocation: We describe the working of the algorithms with the help of an example. Figure 12(a) shows the topology considered. Session S_1 consists of sender $s_{1,1}$ sending to receiver r_1. Session S_2 consists of two senders, $s_{2,1}$ and $s_{2,2}$, sending to r_2. The capacities of the links are as shown in Figure 12(a).

Figure 12(b) shows the connection fair rate allocation. Link a-b is the bottleneck link for all three connections, and hence each connection gets a rate of 2 units. Session S_1 gets a total rate of 2 units, and session S_2 gets a total rate of 4 units.

Properties of the Allocation: We discuss properties of the allocation from a session point of view. These properties are in addition to the normal connection rate properties.

- If all connections in all sessions have the same bottleneck link, the session rates will be in proportion to the number of connections in the session.
- Two connections between the same sender and receiver, belonging to different sessions will be bottlenecked at the same link and will receive equal rates.

4.3.2 Normalized Rate Session Fairness (NRSF)

We now present the two session fair definitions and discuss them in detail. We also present algorithms to achieve the allocations and then discuss some properties of the allocations.

Definition: An allocation of sender rates $a_{i,k}$ is said to be *normalized rate session fair* if it is feasible and for any alternative feasible allocation of sender rates where the normalized rate $\overline{z}_{i,k} > z_{i,k}$, there is some other sender $s_{i',k'} \neq s_{i,k}$ such that $z_{i,k} \geq z_{i',k'} > \overline{z}_{i',k'}$.

The *normalized rate of a connection* is defined as

$$z_{i,k} = \frac{a_{i,k}}{w_{i,k}},$$

where $w_{i,k}$ is the *weight of the connection*, subject to the constraints:

$$\sum_k w_{i,k} = 1$$
$$w_{i,k} \leq 1$$

Informally, the normalized rate of a connection at its bottleneck link is greater than or equal to the normalized rate of any other connection at that link. The first constraint on the weights ensures that the set of connections in each session behaves as at most one connection throughout the data path of the session, and hence as at most one connection on each link that it traverses. The rate assigned to each connection at its bottleneck link will be in proportion to its weight[6]. The second constraint ensures that no connection behaves as more than one connection on any link.

The session fairness definition can be extended to an *r-parallel connection fairness* definition if each session behaves as at most r connections throughout the data path of the session. This approach lies between the connection-fair and the session fair rate allocation approaches, with $r = 1$ corresponding to the session fair approach and $r =$ number of connections corresponding to the connection-fair approach. r-parallel connection fairness can be achieved by defining the *normalized rate of a connection* as

$$z_{i,k} = \frac{a_{i,k}}{w_{i,k}},$$

subject to the constraints:

$$\sum_k w_{i,k} \leq r$$
$$w_{i,k} \leq 1$$

The second constraint ensures that no connection behaves as more than one connection on any link.

[6]Moh and Chen [62] also use the concept of weights for each connection, but their definition and algorithm differ from ours.

Algorithm: We first assign weights to each connection according to the intra-session fairness approach that the client wants, as described in Section 4.2. For "source and session fairness", we assign equal weights to each connection, for "data and session fairness", we assign weights to each connection in proportion to the size of data retrieved from that server, and for "path and session fairness", we assign weights to each connection according to the sharing of the data path of the session. For simplicity, we describe the normalized rate session fair algorithm with all clients using "source and session fairness".

In every iteration, we compute the normalized rate attainable on every link and saturate the link with the minimum normalized rate. The connections which get saturated as a result are bottlenecked at that link. Thus, after every iteration, at least one link will be saturated, and the algorithm will terminate when all connections have a bottleneck link. We expect the session rates to be fair with respect to each other because we have assigned weights subject to the constraint:

$$\sum_k w_{i,k} = 1$$

In fact, the weight of each connection gives an idea about the number of other connections constituting the session it belongs to.

Figure 13 presents the algorithm for determining the normalized rate session fair allocation. We assume that each connection has a weight $w_{i,k}$. In step 1, we initialize the unsaturated sender set (senders in a session with no bottleneck link) to all the sender-receiver pairs in the topology. Each connection is assigned a rate $a_{i,k}$ of 0. In step 3, x_j is calculated as the ratio of the residual capacity of the link to the sum of the weights of all unsaturated connections traversing that link, that do not have a bottleneck yet. In step 4, we compute the minimum of all these x_j's. In step 5, the rate for all unsaturated connections is incremented by its weight, times the minimum value, thus saturating at least one link. The residual capacity on all links along the path from the senders in set X to the receivers is then decremented by that amount. In step 6, all senders that are saturated as a result are removed from set X. We repeat steps 2 through 7 until all receivers have a bottleneck link. The rate allocation thus determined is the required normalized rate session

1. $X = \{s_{i,k} : i = 1..m, k = 1..k_i\}$;
   $\forall j, c_j = C_j$;
   $\forall s_{i,k}, a_{i,k} = 0$

2. while $(|X| > 0)$

3. $\forall j, x_j = \frac{c_j}{\sum_{s_{i,k} \in R_j} w_{i,k}}$

4. $x_{min} = min\{x_j\}$

5. $\forall s_{i,k} \in X$:
   $a_{i,k}$ += $w_{i,k} * x_{min}$
   $\forall l_j$ along the path from $s_{i,k}$ to r_i
   c_j −= $w_{i,k} * x_{min}$

6. $\forall j$, if $(c_j == 0)$[7]
   $X = X - \{s_{i,k} \in R_j\}$

7. repeat from step 2

Figure 13: Algorithm to Determine the Normalized Rate Session Fair Allocation

fair allocation.

Example allocation: Figure 12(c) shows the allocation achieved by the normalized rate session fair algorithm. The connection in session S_1 is assigned a weight of 1.0, and the two connections in session S_2 are each assigned a weight of 0.5. In the first iteration, the total weight on link a-b is 2.0, hence a weight of 1.0 would correspond to a rate of 3 units. The total weight on link b-c is 1.5, hence a weight of 1.0 corresponds to a rate of $4/1.5 = 2.66$ units. The total weight on link b-d is 0.5, hence a weight of 1.0 corresponds to a rate of 10 units. The minimum normalized rate is 2.66, hence the algorithm will saturate link b-c first. $s_{1,1}$, which has a weight of 1.0, therefore gets a rate of 2.66 units, and $s_{2,1}$, which has a weight of 0.5, gets a rate of 1.33 units. In the next iteration, link a-b is saturated, and $s_{2,2}$ gets a rate of 2.0 units. Thus, session S_1 gets a total rate of 2.66 units and session S_2 gets a rate of 3.33 units.

Properties of the Allocation: We present some properties of the allocation that follow in a relatively straightforward manner from the definition of normalized rate session fairness.

- If all sessions have an equal number of connections, they will share bandwidth as in

the connection max-min fair allocation. This is because, the weights of all connections in all the sessions will be equal, and they will share the capacity on bottleneck links equally.

- If all connections in all sessions traverse the same bottleneck link, the rates allocated to each session will be equal because we have imposed the constraint:

$$\sum_k w_{i,k} = 1$$

on each session. Each session will therefore be treated as a single connection on that link. The connections within a session will share this session rate.

- All connections (possibly from different sessions) bottlenecked at the same link will have the same normalized rate:

$$z_{i,k} = \frac{a_{i,k}}{w_{i,k}}$$

- Two connections between the same sender and receiver (assuming they follow the same path), belonging to different sessions with different number of connections (and hence having different weights, to be more precise), will be bottlenecked at the same link and will receive different rates, in proportion to their weights. However, their normalized rates will be the same.

4.3.3 Per-Link Session Fairness (PLSF)

Definition: An allocation of sender rates $a_{i,k}$ is said to be *per-link session fair* if it is feasible and for any alternative feasible allocation of sender rates $\overline{a}_{i,k}$, where $\overline{b}_{i,j} > b_{i,j}$ for session S_i at some link l_j, there is some other session $S_{i'} \neq S_i$ such that $b_{i,j} \geq b_{i',j} > \overline{b}_{i',j}$.

Informally, this means that the rate of a connection can be increased only by decreasing the rate of a session with an already lower or equal rate on a link at which the higher rate session, which comprises the connection, is bottlenecked. This also ensures that each session behaves like at most one connection on each link that it traverses. The basic idea here is that the capacity of the bottleneck link is shared equally by all the sessions traversing that

link (and not by the individual connections). In other words, we try to achieve session fairness on every link.

The above definition does not lead to a unique allocation. This is because the per-link session fairness definition advocates the sharing of the session rate at each link in a fair manner. The session then splits this rate amongst the connections that traverse that link. This leads to residual capacity on some links. The manner in which this residual capacity is shared amongst the sessions can lead to multiple allocations which satisfy the above definition. We present an algorithm which will lead to one such allocation.

Algorithm: In this algorithm, we make the distinction between *virtual bottleneck links* and *physical bottleneck links*. Each link is considered to be a set of virtual links, one for each session on that link. In every iteration, at least one virtual link is saturated. Thus, at the end of one run of the algorithm, each connection has a virtual bottleneck link. But it might be the case that a connection does not have a physical bottleneck link on its path, and hence it can send at a higher rate. We therefore reassign the available capacities amongst all sessions without physical bottleneck links and reiterate through the algorithm. We do this until each connection has a physical bottleneck link on its path.

Figure 14 presents the algorithm to compute the per-link session fair allocation. Set Y is the set of receivers without a physical bottleneck link. Set X is the set of receivers without a virtual bottleneck link in the current iteration. In step 5, each session is assigned the rate it is allowed to use on that link. This is the ratio of the residual capacity of that link to the number of sessions traversing that link. In step 7, each session then determines the rate of each connection it comprises as the ratio of the above rate to the number of unsaturated connections constituting that session on that link. In step 8, we determine the minimum of these rates, and increment the rates of all unsaturated connections by that amount. We thus saturate at least one virtual link in this step, and remove these connections from consideration in this iteration. If any physical edge has a residual capacity of zero, we remove those connections from consideration permanently. We repeat steps 6 through 10 until all connections are saturated by a virtual link. We then reassign the set X to the set Y and continue steps 3 to 11 until each connection has a physical bottleneck link. The

1. $\forall j, c_j = C_j$;
   $\forall s_{i,k}, a_{i,k} = 0$
2. $Y = \{s_{i,k} : i = 1..m, k = 1..k_i\}$
3. while ($|Y| \ > \ 0$)
4. $X = Y$
   u_j = number of sessions on link l_j
   $u_{i,j}$ = number of connections in session i on link l_j
5. $\forall i, j$, if ($u_j \neq 0$)
   $c_{i,j} = \frac{c_j}{u_j}$
6. while ($|X| \ > \ 0$)
7. $x_{i,j} = \frac{c_{i,j}}{u_{i,j}}$
8. $x_{min} = min\{x_{i,j}\}$
9. $\forall s_{i,k} \in X$:
   $a_{i,k}$ += x_{min}
   $\forall l_q$ along the path from $s_{i,k}$ to r_i
   $c_{i,q}$ −= x_{min}
   c_q −= x_{min}
   if ($c_{i,q}$ == 0)[8]
   $X = X - \{s_{i,k} \in R_{i,q}\}$
   $\forall l_q$ on the path from these $s_{i,k}$'s to r_i,
   $u_{i,q}$−−
   if ($u_{i,q} == 0$)
   u_q−−
   if ($c_j == 0$)
   $Y = Y - \{s_{i,k} \in R_j\}$
10. repeat from step 6
11. repeat from step 3

Figure 14: Algorithm to Determine the Per-Link Session Fair Allocation

resulting rate allocation is the per-link session fair allocation.

Example allocation: Figure 12(d) shows the allocation achieved by the per-link session fair algorithm. On link a-b, sessions S_1 and S_2 can each use a rate of at most 3 units. Session S_2 can therefore use at most 1.5 units for each connection, assuming that it shares the rate equally between the connections in the session. On link b-c, sessions S_1 and S_2 can each use at most 2 units. On link b-d, session S_2 can use at most 5 units. Since 1.5 units is the minimum rate that can be used by a connection in this scenario, the virtual link for session S_2 on link a-b is saturated in this iteration, and $s_{2,1}$ and $s_{2,2}$ each get a rate of 1.5 units. These two connections are then removed from contention on all the links that they traverse. In the next step, $s_{1,1}$ is assigned 0.5 units more, thus saturating its virtual link on b-c. Note that we did not reallocate the extra 0.5 units not used by session S_2 on link b-c, because we can reuse it in successive iterations. In the next run, we therefore reconsider all senders that are not saturated by a physical link. All three senders satisfy this condition—the residual capacity on link b-c is 0.5 units, that on link a-b is 1 unit, and that on link b-d is 3.5 units. $s_{1,1}$ and $s_{2,1}$ get 0.25 units each, in this round, thus saturating physical link b-c. $s_{2,2}$ also gets 0.25 units, thus saturating the virtual link for session S_2 on link a-b, but the physical link is not yet saturated. In the next round, the only connection in contention is $s_{2,2}$, and it gets the residual 0.25 units, thus saturating link a-b. Session S_1 therefore gets a rate of 2.25 units under the per-link session fair allocation, and session S_2 gets a total rate of 3.75 units.

Properties of the Allocation: The properties presented are for the allocation in general, and not just for the allocation achieved by this particular algorithm. They follow in a relatively straightforward manner from the definition of per-link session fairness.

- If all connections in all sessions traverse the same bottleneck link, the rates allocated to each session will be equal by definition. That is, each session will be treated as a single connection on that link. The connections within a session will share this session rate.

- Two connections between the same sender and receiver, belonging to different sessions

will be bottlenecked at the same link if they are part of sessions with the same number of connections at the bottleneck link, and they will receive the same rates. If the connections belong to sessions with different number of connections on that link, they will receive different rates.

To summarize this section, we have proposed two definitions for session fairness, and presented algorithms to achieve those allocations. We have also discussed some properties of these allocations.

4.4 Evaluation Results

This section describes an evaluation of the proposed session fair algorithms using simulations of a wide area network. Our goal is to evaluate the advantages offered by the proposed algorithms in an environment like the Internet, with data paths of sessions intersecting arbitrarily, and to ensure that the session fair allocations do not just degrade to the existing connection fair allocation. We compare the two session fair algorithms with each other, and with the original connection fair algorithm. We also compare these algorithms with the user fair queueing algorithm presented by Banchs [8].

4.4.1 Evaluation Model

We implemented the four algorithms (normalized rate session fair, per-link session fair, connection fair, and user fair queuing) and computed the allocations achieved for various session configurations in the topologies discussed below.

We constructed transit-stub topologies of 100 nodes and 600 nodes using GT-ITM [15]. A number of sessions were then simulated on top of this topology, with varying percentages of clients and servers. These sessions comprised 1, 4 or 15 connections from multiple servers to a single client. We allowed for a single client to have multiple co-located sessions.

Based on a study by Krishnamurthy et al. [54], we varied the percentages of 1, 4 and 15-connection sessions, such that the average number of servers contacted by a client is between 4 and 9. We varied client and server percentages and their locations in the network, computed the allocations with the four algorithms and plotted the performance

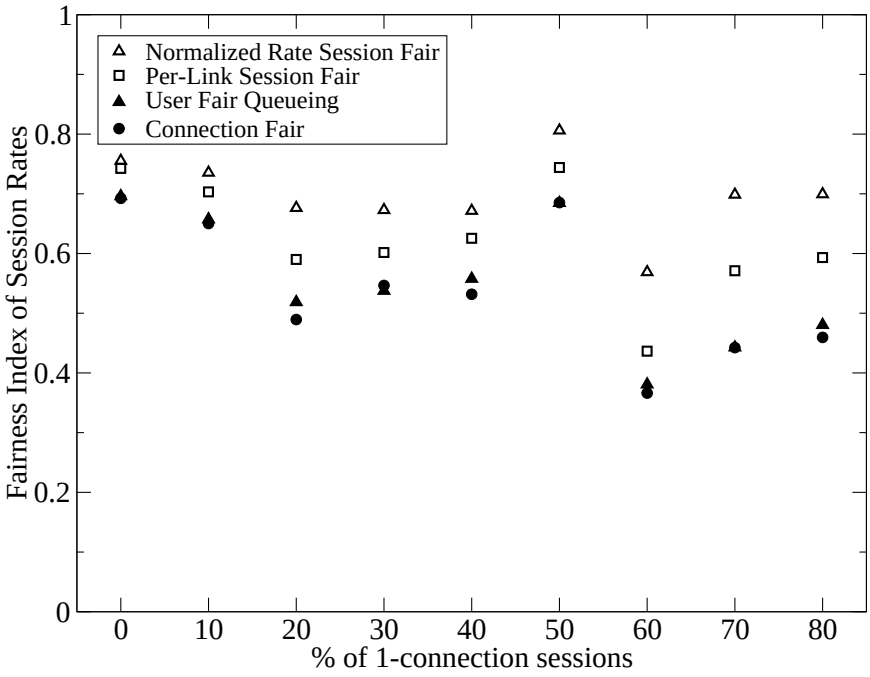


Figure 15: Fairness Index for Session Rates

metrics discussed in Section 4.4.2. A typical plot for the performance metrics is analyzed later in this section.

4.4.2 Performance Metrics

The performance metrics used for comparison of the algorithms, and their significance are discussed below.

- *Raj Jain's fairness index*: This is a commonly accepted fairness performance measure, and is defined as:

$$f(x_1, x_2, \ldots x_n) = \frac{(\sum_{i=1}^{n} x_i)^2}{n \sum_{i=1}^{n} {x_i}^2}$$

  It accounts for variability in the x_i's of all the users. The index assumes a range of values between 0 and 1. A fairness index of 1 implies an equal distribution of x_i's. The lower the value of the fairness index, the more unfair the distribution is. For calculation of the fairness index for session rates, the x_i's are the total data rates of each session. For calculation of the fairness index for connection rates, the x_i's are the rates allocated to each connection by the algorithm.

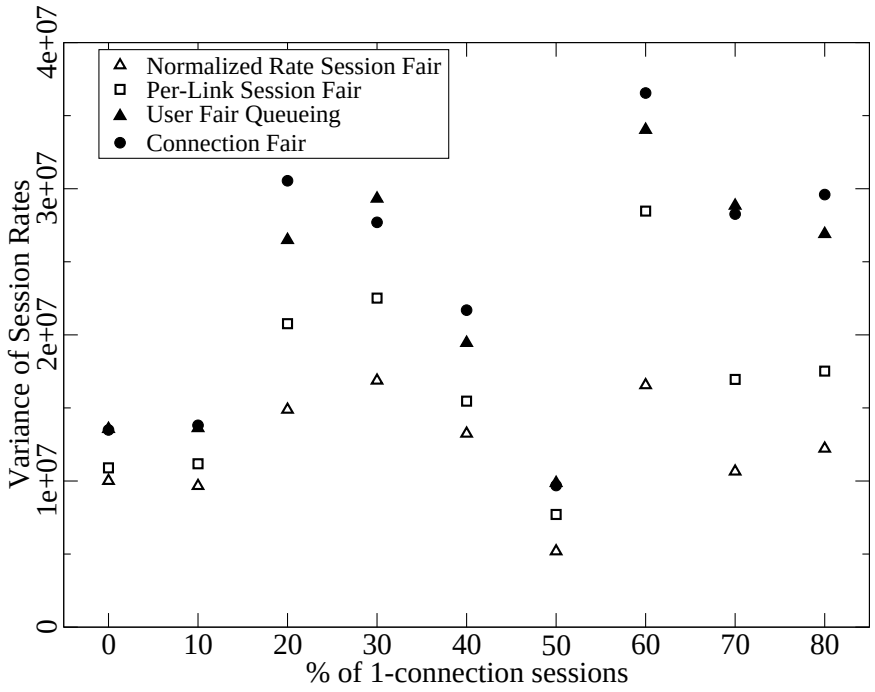


Figure 16: Variance of Session Rates

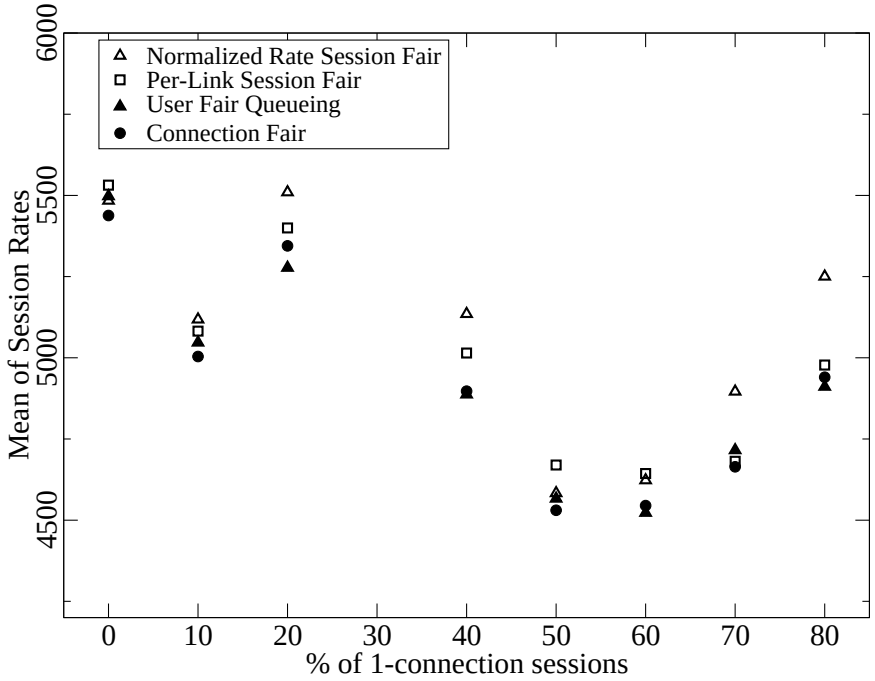


Figure 17: Mean of Session Rates

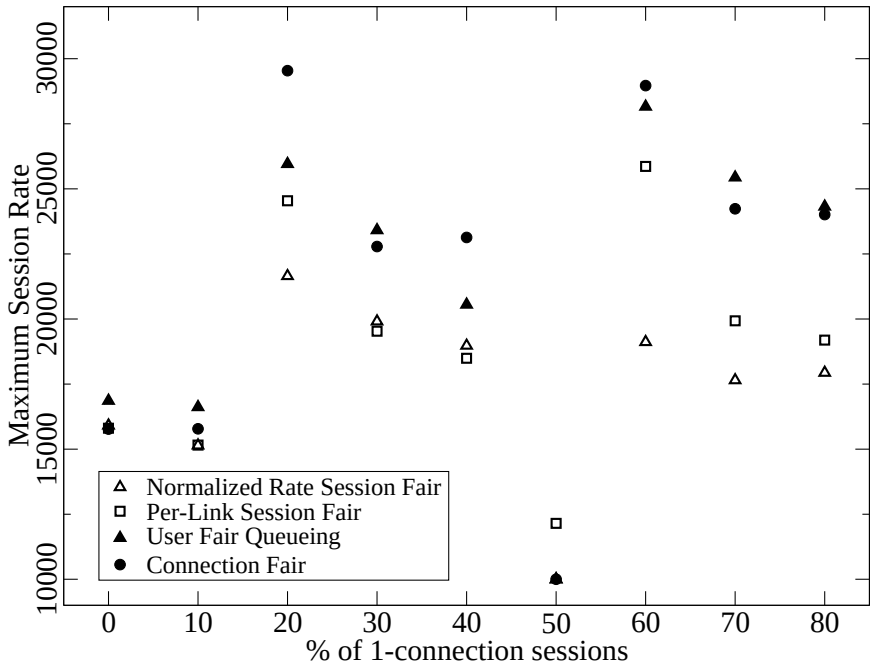


Figure 18: Maximum of Session Rates

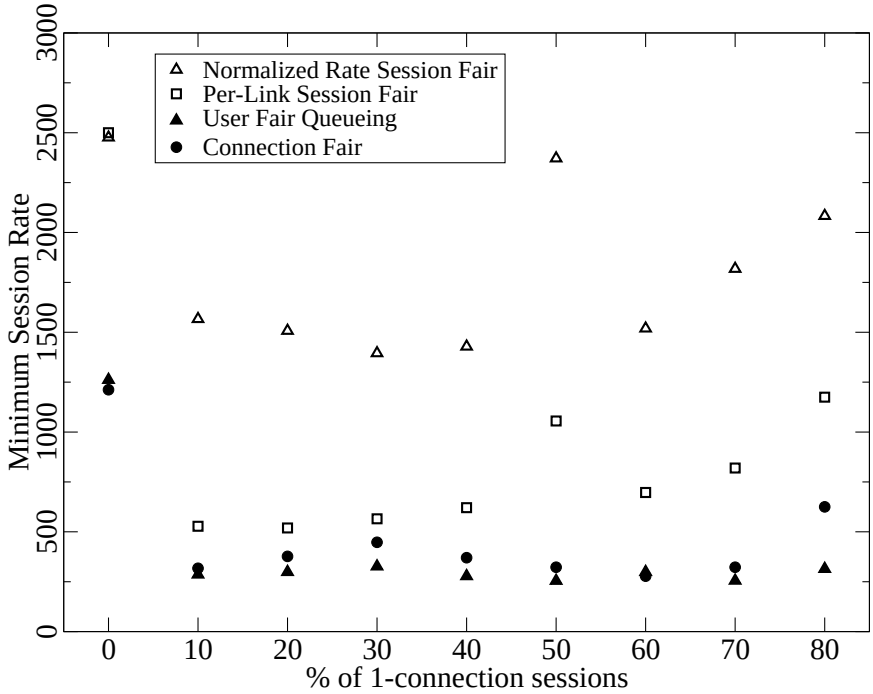


Figure 19: Minimum of Session Rates

- *Variance* of session rates: A higher variance in session rates implies that the session rates are unevenly distributed across all sessions. Hence, a lower variance indicates a more fair distribution.

- *Mean* session rate: A higher mean session rate indicates a higher utilization of available bandwidth resources.

- *Minimum* and *Maximum* session rates: The minimum and maximum session rates are just two extreme points in the distribution. Since the session with the lowest rate is least satisfied, a fair algorithm would try to increase the minimum session rate. Also, a fair algorithm would try to redistribute the excess rate in the session with the maximum rate amongst sessions with lower total rates. Hence, an algorithm with a higher minimum session rate and a lower maximum session rate can be considered more fair.

4.4.3 Comparison of Session Fair Algorithms with the Connection Fair Algorithm

Figures 15, 16, 17, 18 and 19 show typical plots of the performance metrics discussed in the previous section. These plots are for a 100-node topology with 30 servers and 20 clients. 20% sessions have 15 connections, and we vary the percentage of 1-connection and 4-connection sessions between 0% and 80%. The x-axis for each graph is the percentage of sessions with 1 connection, and therefore, the percentage of sessions with 4 connections is $(80 - x)\%$.

From Figure 15, it is clear that the fairness indices for the normalized rate session fair algorithm are higher than the per-link session fair and connection fair algorithms. It thus achieves the most fair allocation of session rates for this configuration.

The variances in session rates, as seen from Figure 16, are much higher with the connection fair algorithm than the per-link session fair algorithm and the normalized rate session fair algorithm. In some cases, the connection fair algorithm has a variance as much as two or three times that of the normalized rate session fair algorithm, thus giving a rather unfair allocation.

Although the mean session rates, as seen in Figure 17, are almost the same for all three

algorithms, in most cases the normalized rate session fair algorithm achieves a higher mean than the connection fair algorithm and the per-link session fair algorithm. Note that a higher mean session rate implies a higher overall bandwidth utilization in the network.

As seen in Figure 19, the minimum session rate achieved by the normalized rate session fair algorithm is always greater than that achieved by the per-link session fair algorithm, which is again greater than that achieved by the connection fair algorithm. On the other hand, as seen from Figure 18, which plots the maximum session rate, the connection fair algorithm has a higher maximum session rate most of the times. The session fair algorithms try to decrease the maximum rate achieved by any session, and redistribute that rate amongst sessions with lower rates, thus increasing the minimum rate. Thus, the normalized rate session fair algorithm is more fair than the connection fair algorithm.

Thus, we can conclude that the session rates achieved by the session fair algorithms are more fair than those achieved by the connection fair algorithm, while maintaining or increasing overall bandwidth utilization.

4.4.4 Comparison of Session Fair Algorithms with the User Fair Queueing (UFQ) Algorithm

In the normalized rate session fair algorithm, we need to be careful when we assign weights to the connections in each session. Our algorithm assigns weights in a session at the client or receiver end, subject to the condition $\sum_k w_{i,k} = 1$. However, if the weights are distributed amongst all connections at the sender, as suggested in the user fair queueing algorithm [8], the session-rate distribution may not be fair.

In Figures 15, 16, 17, 18 and 19 the fourth algorithm plotted is the user fair queueing algorithm, which is basically the normalized rate session fair algorithm, with weights assigned at the server. As can be seen, the minimum session rate achieved by this algorithm is often lower than even that achieved by the connection fair algorithm. The variance is almost as high as that with the connection fair algorithm, or sometimes higher, and the mean session rate (and hence utilization) is lower than that with the connection fair algorithm at times. This is because, in user fair queueing, if a sender has many clients (N), the weight assigned to each connection is $1/N$. Effectively, some multipoint-to-point session might get

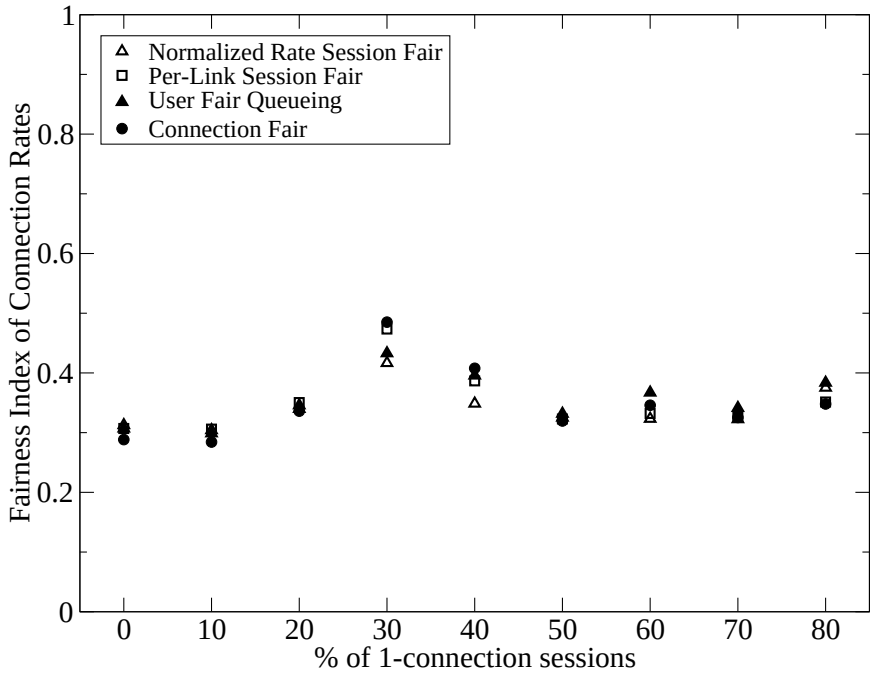


Figure 20: Fairness Index for Connection Rates

a total weight of more than 1, and if two such sessions share a bottleneck link for all their connections, one session will get a higher share than the other session. The normalized rate session fair algorithm performs better than the user fair queueing algorithm in all cases.

4.4.5 Comparison of Connection Rate Allocations Using Different Algorithms

The improved session fairness does not come without a penalty. The performance metrics for the *connection rates* are shown in Figures 20, 21, 22, 23 and 24. The x-axis shows varying percentages of 1-connection sessions in the network. On the y-axes we plot minimum, maximum, variance and mean of the connection rates in the network. It is clear that the minimum connection rates for the session fair algorithms are lower than those for the connection-fair algorithm and the variances are higher. This is the tradeoff seen due to our proposal of achieving fairness at the session level, as opposed to fairness at the connection level.

Thus, we conclude that the session fair algorithms, while achieving the same or higher total utilization as the connection fair algorithm, distribute the rate allocations more evenly among sessions.

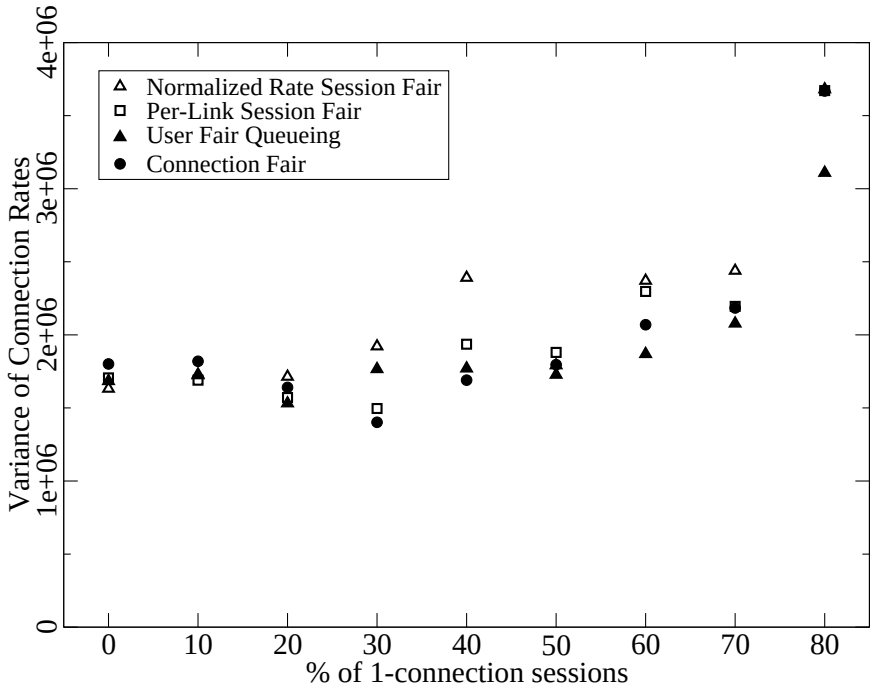


Figure 21: Variance of Connection Rates

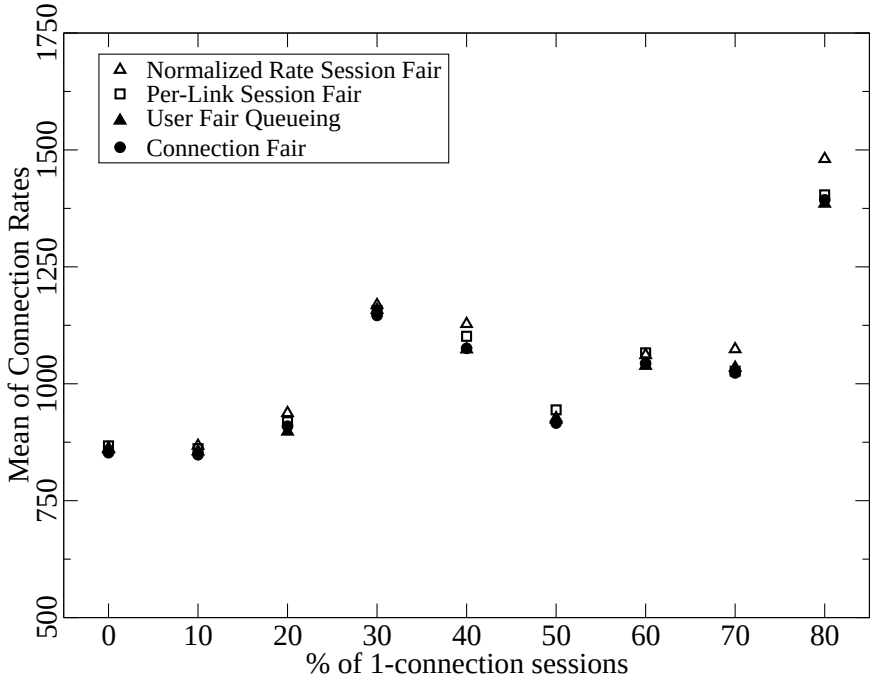


Figure 22: Mean of Connection Rates

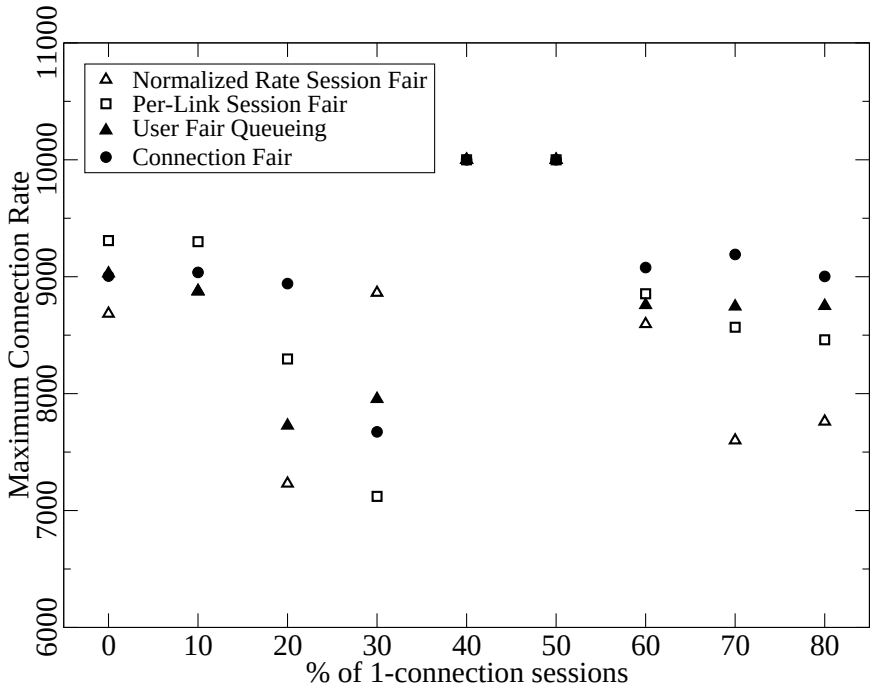


Figure 23: Maximum of Connection Rates

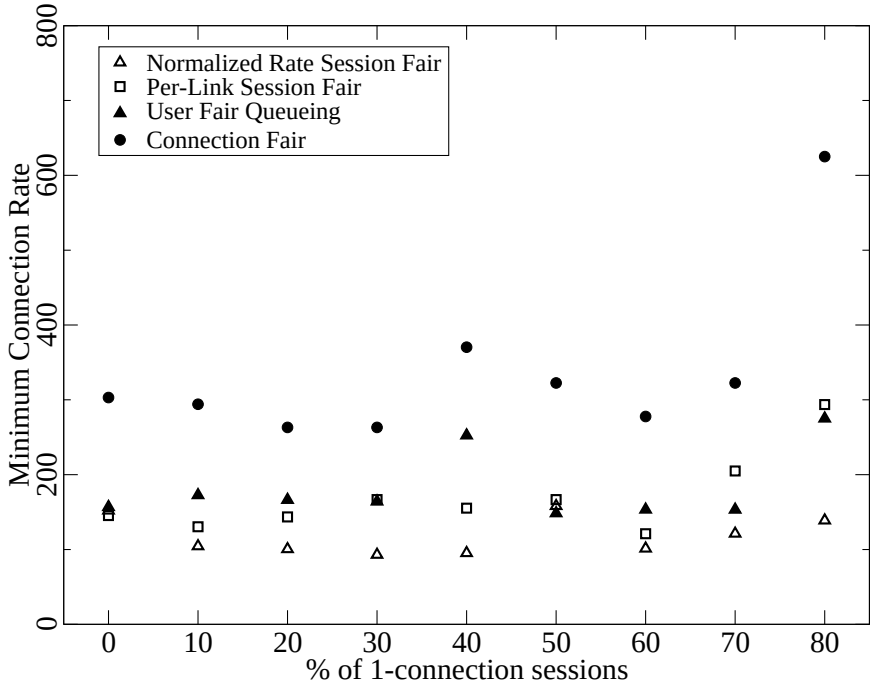


Figure 24: Minimum of Connection Rates

4.5 Implementation Issues

We present some preliminary thoughts on the challenges involved in implementing the session fair algorithms proposed. Both the algorithms assume global knowledge of the residual capacities of each link, and information about all connections in all sessions. Thus, these algorithms are inherently centralized and do not scale well. Modifications to these algorithms are necessary in order to implement them in a distributed and scalable manner.

The three issues involved in the distributed implementation of the session fair algorithms at the client end are as follows:

- Identification of the transport connections belonging to the same session.
- Communication between connections within the same session about the existence of other connections in the session.
- Estimation of the sending rate of each connection, in accordance with the session fair algorithms, and then achieving that rate.

The issue of session identification of a connection, and communication between connections in a session can be implemented explicitly by each application. Alternatively, the Session Control Protocol [93] or the Session Initiation Protocol [38] can be extended to identify the connections belonging to a session. The Integrated Congestion Management architecture [6] has the ability to communicate information between different connections, though clearly the appropriate type and use of information must be modified to meet our needs.

More work is needed to develop techniques for estimating the correct sending rate and incorporating rate adjustment into existing protocols. Over here, we need to distinguish between the two proposed session fair algorithms.

The normalized rate session fair algorithm is more amenable to a distributed end-to-end implementation. This is because, once a client knows the connections that comprise its multipoint-to-point session, it can assign weights to each session according to the specified intra-session fairness scheme. The question is, how do we use this *weight* information inside

the network in order to achieve the normalized rate session fair allocation. We will present two possible solutions to the above problem later in this section.

The per-link session fair algorithm, on the other hand, appears to require a more complex implementation, because of the need to compute the session rate at every link. In this section, we present a possible solution to the problem using support from intermediate overlay nodes in an overlay network.

4.5.1 Implementation of Normalized Rate Session Fair Algorithm

Two possible solutions to implementing a distributed version of the normalized rate session fair algorithm are presented in this section. One solution uses the eXplicit Control Protocol (XCP [50]), and another solution uses ideas from the design of an Overlay-TCP network, presented in Chapter 6.

- *Using eXplicit Control Protocol (XCP):* The eXplicit Control Protocol, developed by Katabi et al. [50] decouples utilization control from fairness control in the network. XCP routers implement an efficiency controller, which maximizes link utilization while minimizing drop rate and persistent queues, and a fairness controller, which apportions the feedback to individual packets to achieve fairness. We are interested in a modified version of the fairness controller, described in the paper, which provides differential bandwidth allocation. This idea is based on a *price* associated with each flow. The network allocates bandwidth so that competing flows on the same bottleneck link will achieve a throughput in proportion to their prices.

  In the normalized rate session fairness implementation, once a client computes the weight of each connection, it can assign the corresponding weight as the price of that flow, and attach the price to each packet in the network. Recall that the weights are normalized such that each multipoint-to-point session gets a total weight of 1 unit. Hence, the price will be assigned accordingly, and will ensure that all sessions get a total rate allocation that is normalized rate session fair.

- *Using Overlay-TCP Networks:* The *intra-overlay network fairness* concept presented

in Section 6.3.2 defines max-min fair sharing amongst overlay flows in the same overlay network. In our design of an Overlay-TCP network, we implement this fairness between overlay flows by requiring the overlay nodes to track the number of overlay flows forwarded on each overlay hop, and then reading from the socket buffers of the TCP connections on each overlay hop in proportion to the number of flows on that hop.

This can be modified to implement the normalized rate session fair algorithm by assigning weights to each overlay flow as before, assuming that each connection in a session is a single overlay flow. Each packet in the overlay network carries the weight assigned to the overlay flow by the client. The overlay nodes are required to track the total weight of the overlay flows on each overlay hop and then read data from the socket buffers in proportion to the total weight of the overlay flows comprising that hop. If all nodes implement this strategy, the resulting allocation should be normalized rate session fair.

4.5.2 Implementation of Per-Link Session Fair Algorithm

The solution we present for the distributed implementation of the per-link session fair algorithm derives ideas from the design of an Overlay-TCP network, similar to the previous discussion. The per-link session fair algorithm does not use the concept of weights of each connection. Instead, it requires each packet to carry a globally unique session identifier, which indicates the session that the packet belongs to. This session identifier can be assigned by the client, assuming that each connection in a session is a single overlay flow. Each overlay node will then need to keep track of the number of sessions on each overlay hop, and then read from the socket buffers in proportion to this number. Thus, all overlay flows belonging to the same session will be identified as a single unit on each overlay hop. This will ensure that at each overlay hop, all sessions traversing that hop get an equal share.

A detailed investigation of these options, with the goal of designing and building a workable system for session fair rate control, is a potential direction for future work in this

area.

4.6 Summary

In the current Internet, many applications start sessions with multiple connections to multiple servers in order to expedite the reception of data. Such aggressive behavior can be viewed as unfair sharing of available bandwidth. This has been our motivation to propose the notion of session fairness when allocating rates to connections. In particular, we looked at *static multipoint-to-point sessions* which comprise multiple connections from multiple senders to a single client, starting and terminating at approximately the same time. We explored the session fairness space and proposed and evaluated two definitions and algorithms to achieve the definitions. The *normalized rate session fair* algorithm achieves a higher level of session fairness, while achieving the same or higher network utilization, as compared to the connection fair algorithm. The *per-link session fair* algorithm also performs better than the connection fair algorithm, but not as well as the normalized rate session fair algorithm. We discussed some of the issues involved in implementing these algorithms and presented some preliminary thoughts on options for implementation. The normalized rate session fair algorithm appears to be easier to implement than the per-link session fair algorithm.

CHAPTER V

OPTIMIZING END-TO-END THROUGHPUT FOR DATA TRANSFERS ON AN OVERLAY-TCP PATH

5.1 Introduction

In this chapter, we focus on a single path in an Overlay-TCP network, and describe the proposed Adaptive Overlay-TCP Provisioning Architecture that dynamically provisions the number of TCP connections on each overlay hop on the overlay path.

Figure 25 shows an example of an Overlay-TCP network. Consider a native network comprising some native nodes and native links between these nodes. A subset of these native nodes function as overlay nodes, with overlay hops between these nodes. Senders and receivers, referred to as *overlay users*, are not necessarily part of the overlay network. Sources send data to an *ingress overlay node*, which detects the transfer, say by looking at headers, as suggested by the proxy mechanism in [74]. The ingress overlay node then transfers the data over the Overlay-TCP network to the *egress overlay node*. This node demultiplexes the data and delivers it to the appropriate receiver. We refer to the data transfer between a pair of ingress and egress overlay nodes as an *overlay flow*. Each overlay hop carries data over one or more TCP connections. We assume the existence of a separate routing infrastructure that determines the overlay path from a sender S to a receiver R.

Now consider a single path in such an Overlay-TCP network, as shown in Figure 26, which is explained in detail in Section 5.2. In order to optimize the end-to-end throughput of the overlay path, the main design parameter is we consider the number of TCP connections on each overlay hop in the path. To understand the need for such a design, we define the *isolated rate* of an overlay hop as the throughput that a single long-lived continuously backlogged TCP connection would achieve on that overlay hop. Now consider an Overlay-TCP path in which a single TCP connection is used on each hop. Clearly, the isolated rates

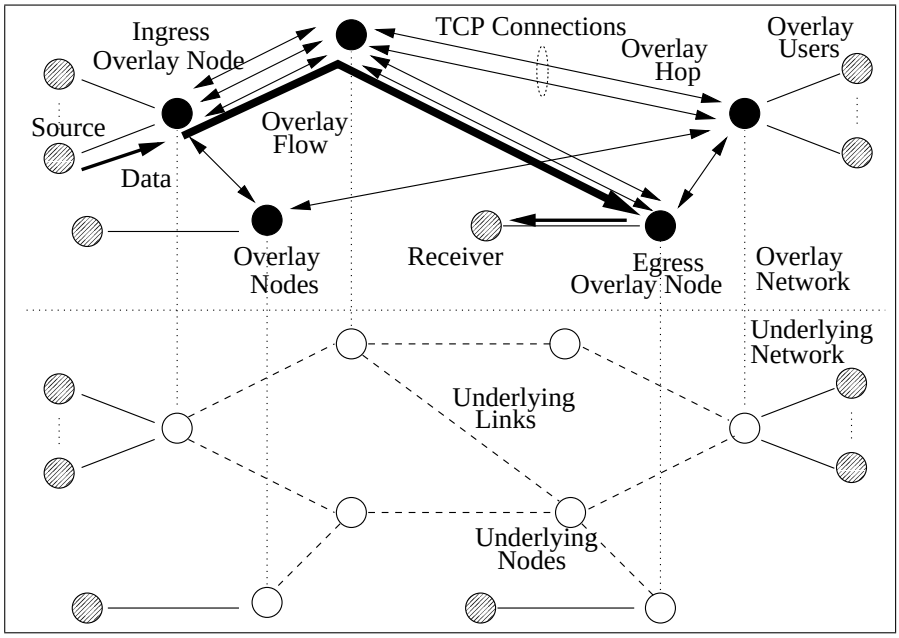


Figure 25: Overlay-TCP Network

on the overlay hops are usually not equal, and vary over time. The end-to-end throughput on this path is therefore limited by the minimum of the isolated rates on the individual overlay hops.

This end-to-end throughput can be improved by introducing multiple parallel connections on the slower overlay hops to increase their rate to match that of the faster overlay hops. This might be considered an aggressive measure, and unfair to connections competing on the bottleneck native link. However, the addition of a bound on the number of parallel TCP connections can provide effective control on this unfairness and equalize treatment among multiple overlays. Also, we are using TCP connections which are congestion responsive. Finally, although we intend to use parallel connections on overlay hops, we aim to maximize the throughput using as few connections as possible.

The improvement in path throughput achieved with the addition of multiple parallel connections comes at the expense of reordering at the receiver. We observed that the reordering index [1] increases with increasing number of parallel connections between two nodes.

[1] We define the *packet reordering index* as the root mean square error of the received packet order compared to the initial sending order.

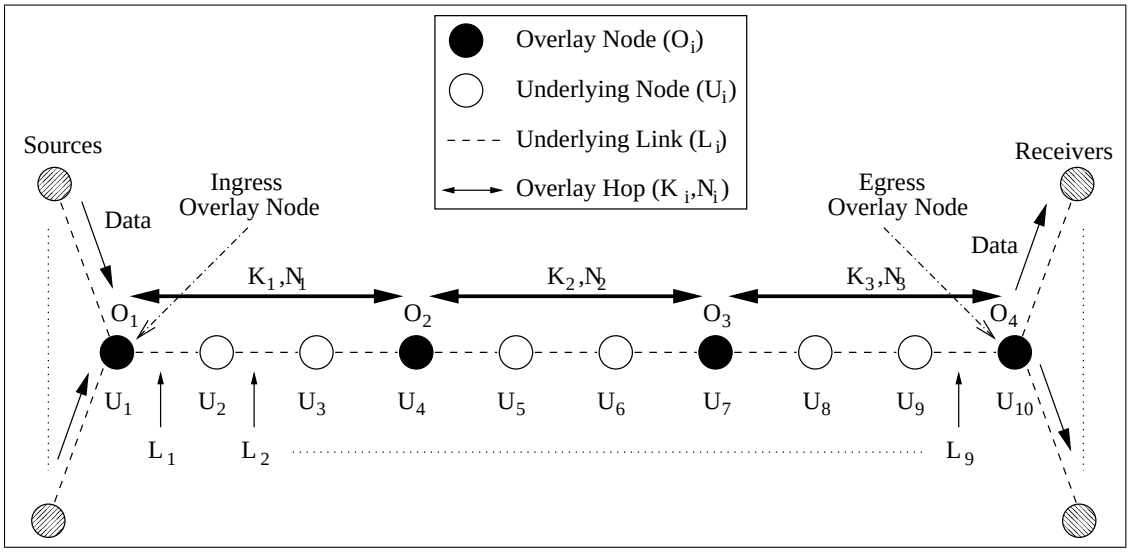


Figure 26: Model of an Overlay-TCP Path

The receiver has to buffer and reorder all data sent over multiple parallel connections, thus adding some complexity. The reordering of packets therefore serves as an added incentive to keep the number of parallel connections to a minimum.

We first show that using more than one connection on some overlay hops can indeed increase the throughput on overlay paths. We then propose two schemes, the *buffer occupancy scheme* and the *isolated rate probing scheme* that assess the path state and dynamically introduce and remove TCP connections on individual overlay hops. We evaluate these schemes on a set of PlanetLab [75] nodes and show that our schemes significantly increase the end-to-end throughput with as few extraneous connections as possible. We discuss the appropriate parameters to be used so as to be less aggressive while maximizing the end-to-end throughput.

The rest of the chapter is structured as follows. We formulate the problem in Section 5.2 and present a case study of a 2-hop overlay path in Section 5.3. We discuss the proposed Adaptive Overlay-TCP Provisioning architecture in Section 5.4, outline the proposed schemes in Section 5.5, and evaluate their performance in Section 5.6. Section 5.7 summarizes the conclusions of the current work and presents our directions for future work.

Table 4: Model of an Overlay-TCP Path

System Parameters	
m	Number of underlying nodes on the path
n	Number of overlay nodes on the path
System Variables	
U_i	i^{th} underlying node on the path, $i = 1..m$
O_i	i^{th} overlay node on the path, $i = 1..n$
L_i	i^{th} underlying link on the path, $i = 1..m-1$
K_i	i^{th} overlay hop on the path, $i = 1..n-1$
N_i	Number of TCP connections on i^{th} overlay hop on the path, $i = 1..n-1$
R_i	Average isolated rate on i^{th} overlay hop on the path, $i = 1..n-1$

5.2 Problem Statement

Our model for an overlay path is shown in Figure 26. The parameters and variables for the overlay path are listed in Table 5. Consider a path of m nodes, each node denoted by U_i in the underlying network. These nodes are connected by $m-1$ links, each denoted by L_i. Let n of the m underlying nodes function as overlay nodes, where $n \leq m$. The overlay nodes, denoted by O_i, form an overlay path of $n-1$ hops, each denoted by K_i, with N_i TCP connections on each hop. The overlay path comprises one or more links and nodes in the underlying network.

We consider one-way data transfer on this path. A set of sources send data to a set of receivers. The senders and receivers are not part of the overlay network. Sources send data to an ingress overlay node, which detects the transfer, say by looking at headers, and then transfer the data over the Overlay-TCP network, as suggested by the proxy mechanism in [74]. The egress overlay node demultiplexes the data and delivers it to the appropriate receiver. We assume that the aggregate incoming data at the ingress overlay node keeps the connection continuously backlogged. The overlay path is assumed to carry background traffic.

Given such an overlay path, with varying background traffic, our goal in this work is to dynamically determine and provision the number of TCP connections necessary on each overlay hop in order to maximize the end-to-end throughput. We refer to this approach as

5.3 Case Study: Two-Hop Overlay Path

We start by studying the behavior of a 2-hop overlay path with a single connection on each hop. At the intermediate overlay node, the socket buffer for the incoming TCP connection forwards the data packets to an application layer buffer. This buffer then forwards the packets to the socket buffer for the outgoing TCP connection. See Figure 28 for more details. The outgoing connection is therefore forced to depend on the incoming connection for supplying the data to be sent downstream, thus affecting the end-to-end throughput as seen below.

Recall that we define the *isolated rate* of an overlay hop as the throughput that a single long-lived continuously backlogged TCP connection would achieve on that overlay hop. The isolated rates of the incoming hop (R_1) and the outgoing hop (R_2) at any overlay node will generally be unequal. We define the *degree of mismatch,* M ($M \geq 1$) at an overlay node as $M = \frac{max(R_1,R_2)}{min(R_1,R_2)}$.

1. *If* $R_1 < R_2$, the data comes in at the intermediate node at a rate slower than what can be forwarded on the outgoing hop. The outgoing connection does not have enough data to operate at the isolated rate (R_2) and is limited by the data rate of the incoming connection. Also, since the outgoing connection is not continuously backlogged any more, it might repeatedly go into the slow start phase, thus slowing it even more.

2. *If* $R_1 > R_2$, the data comes in at the intermediate node at a rate faster than what the outgoing connection can keep up with. Although the outgoing connection can now operate at the maximum rate (R_2), the intermediate node has to buffer the excess data. Once the buffers at the intermediate node are full, the incoming connection has to throttle back. It can now send data only after the buffers have space to hold more data. The incoming connection is now bounded by the data rate of the outgoing connection. Also, depending on the time to drain the buffer at the intermediate node, the incoming connection might go into repeated slow-starts, thus lowering the effective throughput even more.

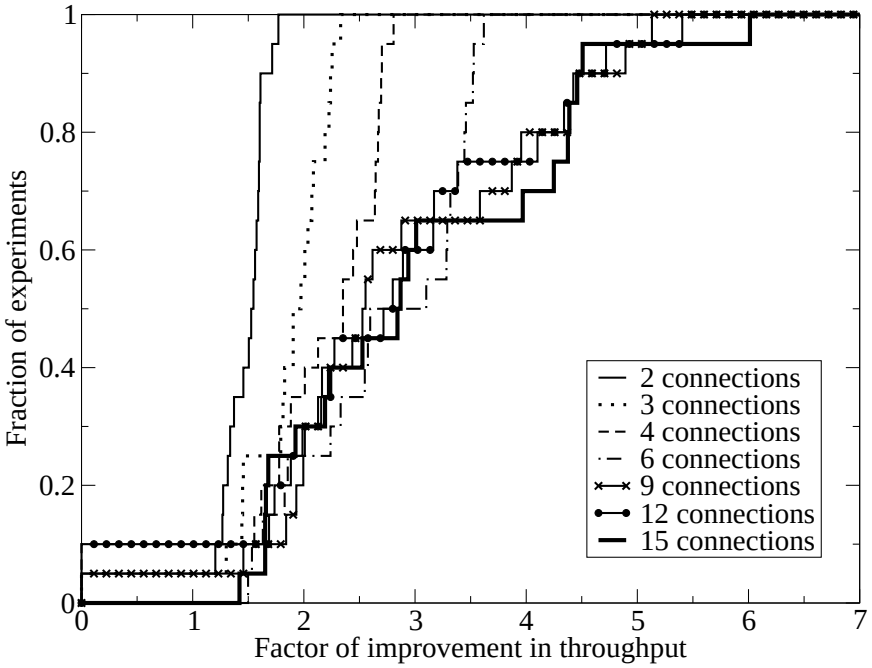


Figure 27: Factor of Improvement in Throughput Using Multiple Connections: PlanetLab Experiments

3. *If* $R_1 = R_2$, the average connection rates over some time period T are equal. However, at any instant, each TCP connection will be in the slow start or congestion avoidance phase. These phases need to be synchronized in order to get the maximum effective rate.

Effectively, the end-to-end throughput on a 2-hop overlay path with a single TCP connection on each hop is bounded by the throughput of the slower connection along the path. In an attempt to improve the end-to-end throughput, we evaluate the effect of using multiple TCP connections on the slower overlay hop.

We set up 2-hop overlay paths on a set of PlanetLab [75] nodes and determined the isolated rates on each overlay hop by performing a 50 MByte transfer on a TCP connection between consecutive overlay nodes. We then started overlay TCP transfers with one connection on the faster hop and multiple (2-15) connections on the slower hop.

Figure 27 shows a sample of the results we obtained on an overlay path where the PlanetLab nodes at Duke University, Georgia Institute of Technology and University of

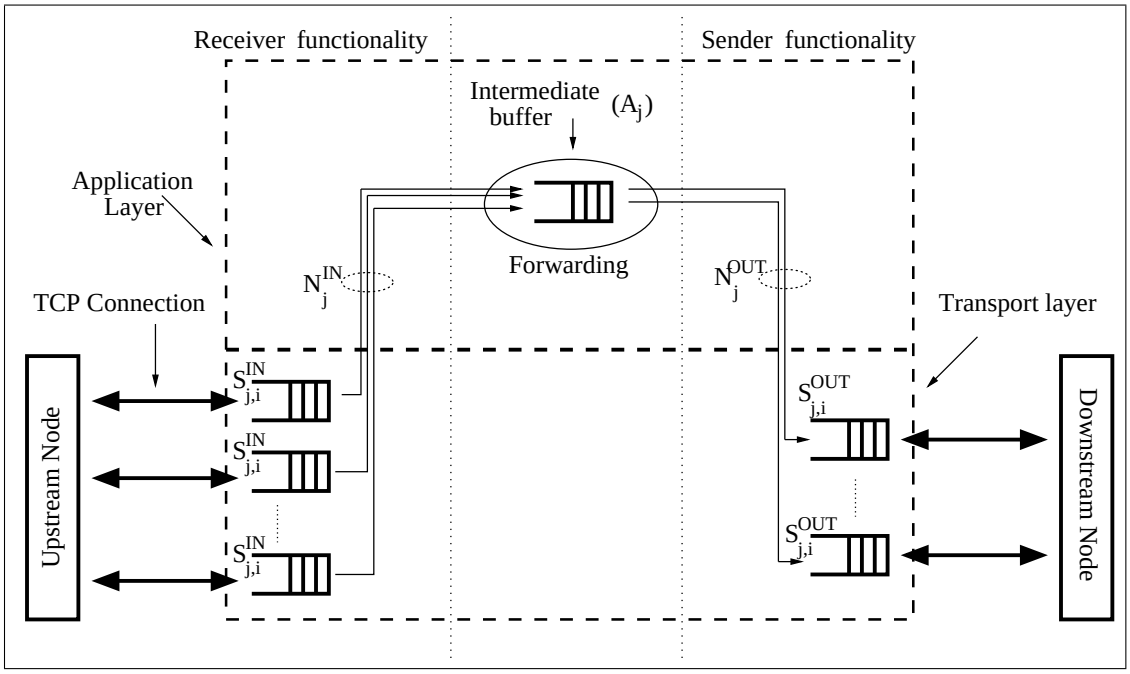


Figure 28: Model for an Intermediate Overlay Node O_j in an Overlay-TCP Path

Arizona were used as ingress, intermediate and egress overlay nodes respectively. On this 2-hop path, the isolated rate from Duke University to Georgia Tech was higher than that between Georgia Tech and University of Arizona (i.e. $R_1 > R_2$). The figure plots the CDF of the factor of improvement, defined as the ratio of the end-to-end throughput using multiple connections on an overlay hop, to the that using a single connection on each overlay hop. We see that the factor of improvement increases with more connections being added downstream. However, the effect of adding new connections decreases as the number of connections increase. We observed similar results for the case where $R_1 < R_2$ when using multiple connections on the incoming overlay hop.

We conclude from these experiments that adding multiple TCP connections on a slower overlay hop does indeed improve end-to-end throughput. However, there is a threshold beyond which adding new connections does not have a significant effect on the throughput.

5.4 The Adaptive Overlay-TCP Provisioning Architecture

We can generalize the above discussion to an n-hop overlay path. The effective end-to-end throughput of a data transfer on this path, with a single TCP connection on each hop

Table 5: Adaptive Overlay-TCP Provisioning: Parameters and Variables

Parameters	
N_{max}	Maximum number of TCP connections between consecutive overlay nodes
A_j	Size of application layer buffer
$S_{j,i}^{IN}$	Size of socket buffer for each incoming connection
$S_{j,i}^{OUT}$	Size of socket buffer for each outgoing connection
Variables at Overlay Node O_j	
M_j	Degree of mismatch between upstream and downstream isolated rates
N_j^{IN}	Current number of incoming TCP connections
N_j^{OUT}	Current number of outgoing TCP connections
$B_{j,i}^{IN}$	Current occupancy of socket buffer for each incoming connection
$B_{j,i}^{OUT}$	Current occupancy of socket buffer for each outgoing connection

will be limited by the slowest TCP connection on the path. We can however improve the throughput by using multiple TCP connections on slower overlay hops.

The architecture for a system to provide *Adaptive Overlay-TCP Provisioning* is based on the model of an overlay node O_j, shown in Figure 28. The dashed box represents the internal structure of the overlay node, which is split into two layers— the Application Layer, and the Transport Layer. A pair of consecutive overlay nodes are allowed to communicate over a maximum of N_{max} TCP connections in the overlay network. N_{max} is a system-wide parameter. At any point of time, the overlay node O_j has N_j^{IN} active TCP connections with the upstream node, and N_j^{OUT} active TCP connections with the downstream node. These connections read or write data into the transport layer socket buffers, each with a maximum capacity of $S_{j,i}^{IN}$ and $S_{j,i}^{OUT}$ for the incoming and outgoing TCP connections respectively. All incoming connections forward the packets from the socket buffers to the application buffer, of capacity A_j. Effectively, each overlay node has a buffering capacity of $\sum_{i=1}^{N_{max}} S_{j,i}^{IN} + \sum_{i=1}^{N_{max}} S_{j,i}^{OUT} + A_j$. At any point of time, the incoming and outgoing socket buffer occupancy for connection i of overlay node j, is denoted by $B_{j,i}^{IN}$ and $B_{j,i}^{OUT}$ respectively.

The Adaptive Overlay-TCP Provisioning architecture consists of three components: the network condition evaluation module, the decision algorithm, and the connection setup and maintenance module.

1. *Network Condition Evaluation Module:*

   This module evaluates the network conditions under which the TCP connections will be operating. This can be done by measuring multiple quantities. A direct approach is to periodically probe the isolated rates on each overlay hop by performing a data transfer. An indirect method is to measure the buffer occupancy of the incoming TCP connections, which give an indication of the relative isolated rates.

2. *The Decision Algorithm:*

   Based on the measured quantities, the decision algorithm decides whether multiple connections are required on any hop on the overlay path. If yes, it also gives either the number of connections required on each hop, or a decision to increase or decrease the number of connections.

3. *Connection Setup and Maintenance:*

   Since the number of upstream and downstream connections varies over time, we have two options to implement these in practice. First, we can keep N_{max} connections active at all times. At any point of time, we use N_j^{IN} and N_j^{OUT} connections, as computed by the decision algorithm, to send data. Another option is to start a new connection and tear down an existing one every time the decision algorithm alters either N_j^{IN} or N_j^{OUT}. However, there is a connection setup overhead associated with this option, and hence we use the first option in our solution.

Once the number of incoming and outgoing connections have been decided and implemented, we cannot assume that those are the optimal values for the rest of the time of data transfer, since the network conditions might change, especially for longer data transfers. Hence, in our schemes, we periodically evaluate the network conditions and repeat the decision process.

5.5 Proposed Schemes

Based on the two quantities measured by the Network Condition Evaluation Module, we propose two schemes and corresponding decision algorithms to determine the number of

connections required on each overlay hop.

5.5.1 Direct Measurement: Isolated Rate Probing Scheme

In this scheme, we directly measure the isolated rates on each overlay hop by periodically performing a 250 KByte data transfer over a TCP connection between the overlay nodes at the ends of each hop. The advantage of this scheme is that it is relatively accurate and gives a good estimate of the values of the isolated rates. However, it places additional load on the path of the data transfer. We can use the lightweight alternative of measuring the loss probability and round-trip time for each overlay hop, and estimating the TCP throughput using the TCP equation [68] to give an estimate of the isolated rates. Alternatively, we can use some of the other schemes mentioned in [40] for predicting TCP throughput. However, in this work we do not implement these schemes.

```
1.   if (R1 > R2) // faster incoming link
2.       M = R1/R2 // calculate degree of mismatch
3.       if (M < Nmax)
4.           N_j^OUT = int(M) // assign higher number of outgoing connections
5.       elseif (M ≥ Nmax)
6.           N_j^OUT = Nmax // do not exceed maximum connection count
7.       endif
8.   elseif (R2 > R1) // faster outgoing link
9.       M = R2/R1 // calculate degree of mismatch
10.      if (M < Nmax)
11.          N_j^IN = int(M) // assign higher number of incoming connections
12.      elseif (M ≥ Nmax)
13.          N_j^IN = Nmax // do not exceed maximum connection count
14.      endif
15.  endif
```

Figure 29: Decision Algorithm Using Direct Measurement of Isolated Rates

2-*Hop Overlay Path:* Figure 29 shows the decision algorithm at overlay node O_j for determining the location and number of new connection(s) on a 2-hop overlay path, with isolated rates R_1 and R_2 on incoming and outgoing overlay hops respectively. If R_1 is greater, the algorithm computes the degree of mismatch (M) in Line 2 as the ratio of R_1 to R_2 and decides in Line 4 that the number of outgoing connections required is equal to M. Similarly, if R_2 is greater, then the degree of mismatch (M) is calculated in Line 9 and assigned to the number of incoming connections in Line 11. Lines 5 and 12 ensure that the maximum connection limit is not exceeded.

Multi-Hop Overlay Path: We now extend the algorithm to the multihop case. We compute M_i^{max}, the maximum degree of mismatch for each overlay hop along the whole path as the ratio of the maximum isolated rate (R_{max}) on the whole overlay path, to the isolated rate (R_i) of each overlay hop on the whole path. This ratio gives the minimum number of connections necessary on the overlay hop to increase the total rate on that hop to the rate of the hop with the maximum isolated rate. With the limit on the number of connections on any hop, the number of downstream connections required at each overlay node is given by $N_i^{OUT} = min(\frac{R_{max}}{R_i}, N_{max})$.

In order to implement the above algorithm, we require the ingress overlay node to send its upstream and downstream isolated rates to the downstream overlay node. This node appends its own downstream isolated rate and forwards it to the next overlay node, and so on until the vector of isolated rates reaches the egress overlay node on the path. This node then computes the N_i^{OUT} values for each overlay hop and sends the vector back along the overlay path.

5.5.2 Indirect Measurement: Intermediate Buffer Occupancy Scheme

The isolated rate probing scheme can incur a significant overhead for determining isolated rates and signaling among the overlay nodes. We now describe a scheme based on local observation of an overlay node's buffers. To illustrate this scheme, consider the occupancy of the incoming socket buffers at an intermediate overlay node on a 2-hop overlay path. Our intuition indicates that when the isolated rate on the incoming hop is greater than that on

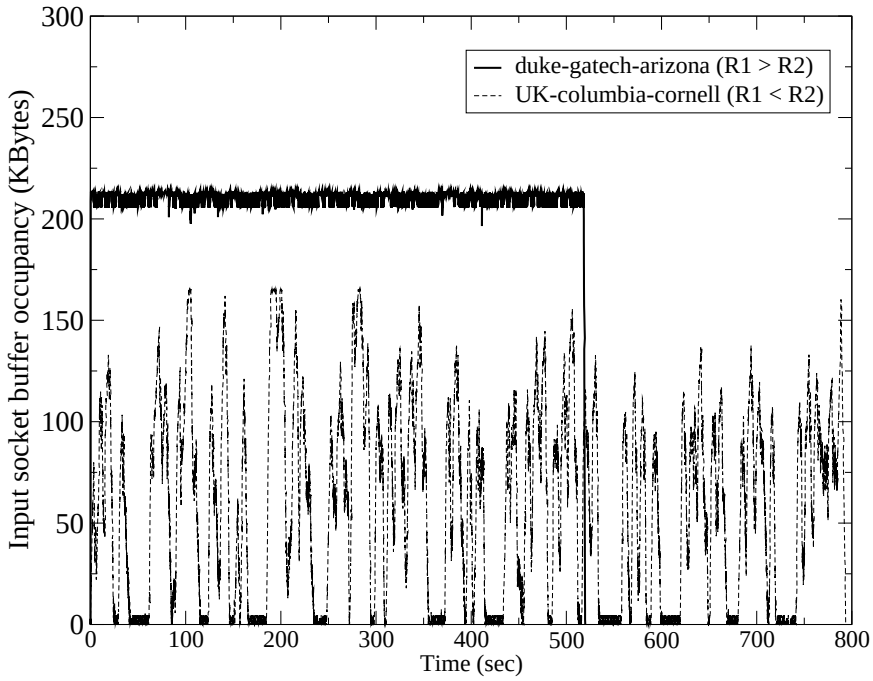


Figure 30: Buffer Occupancy at Intermediate Overlay Node: PlanetLab Experiments

the outgoing hop (i.e. $R_1 > R_2$), the buffer should be relatively full most of the time, whereas, if $R_1 < R_2$, the buffer should be relatively empty. We validated this by observing the incoming socket buffer occupancy over time for several 2-hop overlay paths on a set of PlanetLab nodes. Figure 30 shows the buffer occupancy over time for two overlay paths, each with 2 overlay hops. On one sample path, the PlanetLab nodes at Duke University, Georgia Institute of Technology and University of Arizona were used as ingress, intermediate and egress nodes respectively. The isolated rate on the incoming overlay hop was higher than that on the outgoing hop (i.e. $R_1 > R_2$), and hence, the buffer occupancy was a steady 200 KB. Similarly, on an overlay path between University of Cambridge, Columbia University and Cornell University, where $R_2 > R_1$, the buffer occupancy is less than 10 KB most of the time, with intermediate bursts. The intermediate bursts are due to the lack of synchronization of the phases of the TCP connections on the upstream and downstream hops.

For the general multi-hop case, Figure 31 shows the decision algorithm at an overlay node to determine whether it needs to either increase or decrease the number of connections

on either the incoming or outgoing hop. This algorithm operates at each overlay node independently. The algorithm uses a buffer occupancy estimator, $\hat{B}$, based on an exponentially weighted moving average of periodic samples of instantaneous buffer occupancy as follows, $\hat{B} = \gamma * \hat{B} + (1-\gamma) * \hat{B}_{sample}$, where $0 < \gamma < 1$. Let α and β be the low and high watermarks respectively, below which we consider the buffer at the intermediate node to be underfull, and above which we consider the buffer to be overfull. These parameters are the fraction of the total buffer space available at the receiving end of the incoming TCP connection at the intermediate node.

```
1.   if (B̂ > β * B) // overfull buffer
2.       if (N_j^IN > N_j^OUT) // too many incoming connections
3.           N_j^IN − −
4.       elseif (N_j^IN ≤ N_j^OUT) // not enough outgoing connections
5.           if (N_j^OUT < N_max)
6.               N_j^OUT + +
7.       endif
8.   elseif (B̂ < α * B) // underfull buffer
9.       if (N_j^IN ≥ N_j^OUT) // not enough incoming connections
10.          if (N_j^IN < N_max)
11.              N_j^IN + +
12.      elseif (N_j^IN < N_j^OUT) // too many outgoing connections
13.          N_j^OUT − −
14.      endif
15.  else
16.      Do nothing
17.  endif
```

Figure 31: Decision Algorithm Using Estimates of Buffer Occupancy

If the current buffer occupancy is higher than the high watermark, the algorithm checks

whether the number of incoming (N_j^{IN}) or outgoing connections (N_j^{OUT}) is greater. If there are too many incoming connections (line 2), the outgoing link might not be able to keep up, hence we decrease one incoming connection. On the other hand, if N_j^{IN} is less than N_j^{OUT} (line 4), the outgoing connections cannot keep up even though there are very few incoming incoming connections. In this case, we increase the number of outgoing connections. Note that line 5 ensures that the number of outgoing connections do not exceed N_{max} at any point of time. Similarly, Lines 8-14 handle the case when the buffer is underfull. If the current buffer occupancy is within the acceptable range ($\alpha * B \leq \hat{B} \leq \beta * B$), we do not take any action.

If the algorithm at an intermediate overlay node decides to change the number of outgoing connections, it can be implemented immediately. On the other hand, if the number of incoming connections needs to be changed, the overlay node notifies its upstream node accordingly. The upstream node then changes its number of outgoing connections if its own buffer occupancy estimates indicate the same. However, the ingress node implements whatever the downstream node tells it, because it does not do any buffer estimation itself.

The advantage of this scheme is that we can measure the conditions in the network without introducing additional traffic. Also, we can do the estimation while the data transfer is in progress. The disadvantage is that we cannot determine the exact values of the isolated rates, and hence we do not know the exact number of connections required. We need to continuously monitor and increase or decrease either the number of upstream or downstream connections by one, re-estimate the buffer occupancy, and repeat the decision process.

5.6 Performance Evaluation

In this section, we present a set of experimental results, based on experiments on PlanetLab [75] nodes. We discuss the measurement methodology, the parameters for the schemes, and then the performance results.

5.6.1 Measurement Methodology

We experimented with multiple overlay paths of 2-5 hops, the results for which are presented below. We selected 3-6 overlay nodes for each overlay path from the set of available

PlanetLab [75] nodes. For the isolated rate probing scheme, we first started a set of control nodes that performed a 250 KByte transfer every 30 seconds to estimate the current value of R_i. We then start parallel transfers of 100 MB each between the following nodes.

1. Direct TCP transfer between all sets of consecutive overlay nodes. These are used to monitor the isolated rates on each overlay hop.

2. Overlay-TCP transfer from ingress overlay node to egress overlay node with a single TCP connection on each overlay hop. We compare the throughput of data transfers using our schemes to this benchmark case, which would be the one used in an Overlay-TCP without multiple connections.

3. Overlay-TCP transfer from ingress overlay node to egress overlay node using our Adaptive Overlay-TCP Provisioning schemes to maintain the appropriate number of parallel connections between overlay nodes.

5.6.2 Parameters Used in the Schemes

The parameters involved in the implementation are listed below. We performed a number of experiments with different values of each parameter and determined the best values as discussed below.

- *Socket Buffer Size:*

  Our experiments with the TCP socket buffer size varying from minimum (8 KB) to maximum (256 KB) showed that the end-to-end throughput of a data transfer over an Overlay-TCP path with a single connection on each hop is higher with the socket buffer size and the receiver advertised window set to the maximum value. This is because the socket buffers can absorb intermittent burst in either the upstream or downstream connections.

- *EWMA parameter (γ):*

  γ is the fractional weight of the previous buffer estimate in the exponentially weighted moving average estimator for estimating the current buffer occupancy of the incoming

TCP connections. We experimented with values of γ from 0.05 to 0.95, and concluded that a value between the range 0.7 to 0.95 follows the trend well enough and filters out intermediate bursts in the buffer occupancy. In our experiments we used $\gamma = 0.85$.

- *Estimation and Decision Algorithm Time Interval:*

  In our experiments, we estimate the buffer occupancy every 200 ms. We run the decision algorithm every 30 seconds to determine whether the network conditions have changed enough to either increase or decrease the number of outgoing or incoming connections.

- *Buffer Occupancy Thresholds (α and β):*

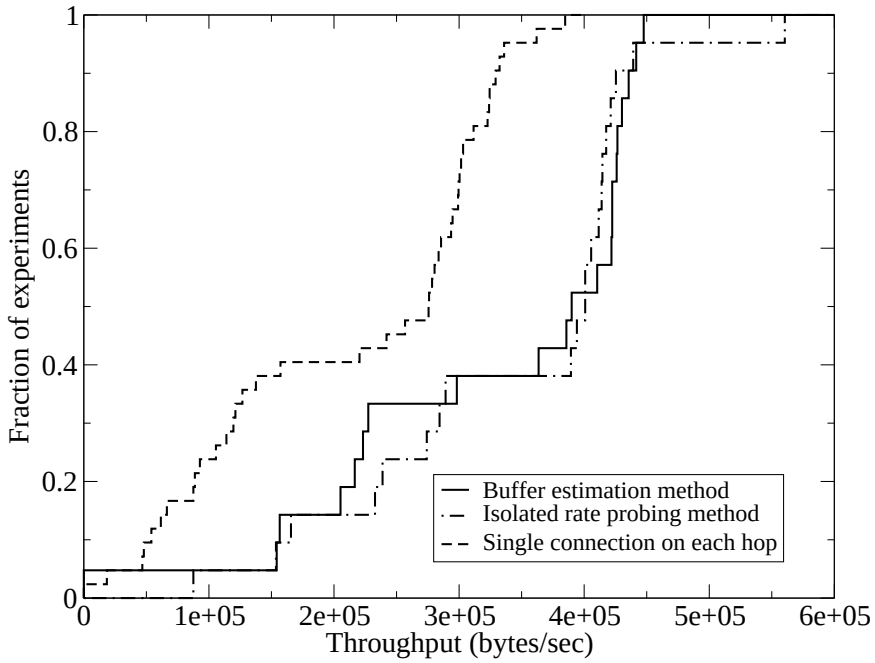
  The buffer occupancy thresholds determine how aggressive the scheme is in using more parallel connections. The lower threshold is set to 0.1, below which we consider the socket buffers to be underfull. In order to set the higher threshold, we observed from our experiments on 2-hop paths that if β is too low (0.15), a large number of connections are added and removed on the downstream hop, even when $R_1 < R_2$. On the other hand, if β is too high (0.8), the addition of connections is very slow, even when the downstream overlay hop is the bottleneck. In our experiments, we use $\beta = 0.4$.

- *Maximum connections on each overlay hop (N_{max}):*

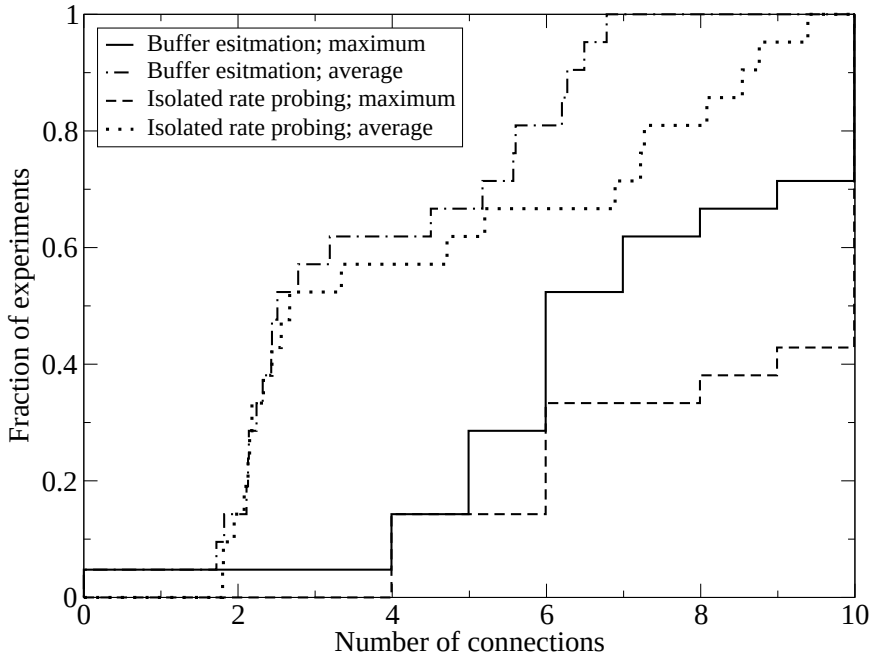
  We use $N_{max} = 10$ in our experiments, based on results shown in Section 5.3.

5.6.3 Evaluation for a Two-Hop Overlay Path

We performed experiments, as described in Section 5.6.1, using the parameters listed above on a variety of 2-hop overlay paths On each path, we performed 10-25 experiments. We present representative results for an overlay path with Cornell University, Columbia University and Stanford University as the ingress, intermediate and egress overlay nodes respectively. Figure 32(a) shows a CDF of the end-to-end throughput achieved using the proposed schemes and that achieved using a single connection on each overlay hop. The two schemes are quite similar in their performance, with the isolated rate probing scheme



(a) End-to-End Throughput



(b) Number of Connections

Figure 32: Evaluation of a 2-Hop Overlay Path: (a) End-to-End Throughput, (b) Number of Connections

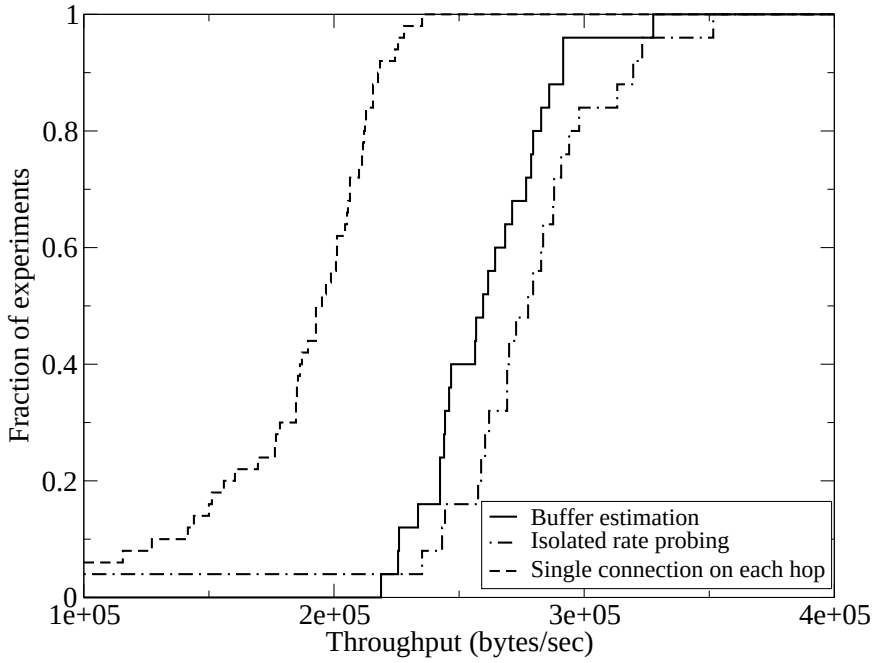
giving a slightly higher throughput. However, we also need to determine how aggressive the two schemes are, by looking at the number of connections started by each one. Figure 32(b) shows a CDF of the average and maximum number of connections started by each scheme. The isolated rate probing scheme is more agressive, and uses a higher number of maximum as well as average connections. The buffer estimation scheme is more conservative and effectively achieves lower end-to-end throughput. We also observed that an increase in degree of mismatch causes a higher factor of improvement in end-to-end throughput, using either of the two schemes, as compared to a single TCP connection on each overlay hop.

5.6.4 Evaluation for a Multihop Overlay Path

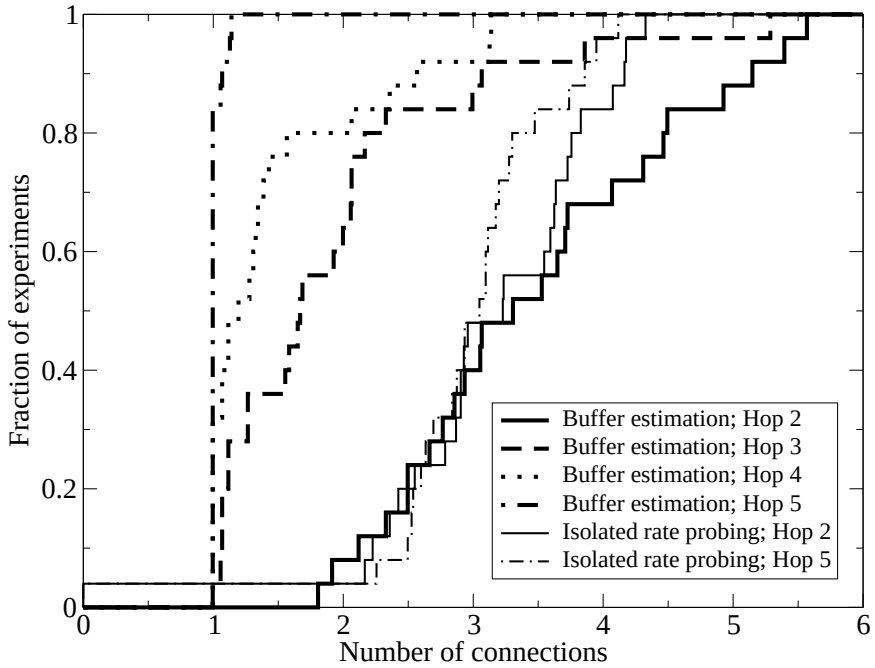
We evaluated the two schemes on overlay paths of 3-5 hops. Figure 33(a) shows the CDF of the throughput achieved using the two schemes, and the throughput achieved using a single TCP connection on each overlay hop, for a sample 5 hop overlay path between University of Massachusetts, New York University, Georgia Institute of Technology, University of Texas, University of Arizona and University of California at Los Angeles. Clearly, the two proposed schemes achieve better throughput than using a single TCP connection on each overlay hop. The isolated rate probing scheme achieves higher throughput than the buffer estimation scheme. Figure 33(b) shows the CDF of the number of connections started by each scheme on different overlay hops. The isolated rate probing scheme starts 2 connections on an average on hops 1, 3 and 4, and the buffer estimation scheme starts 1 connection on hop 1. On the bottleneck hop (hop 2), the buffer estimation scheme starts more connections than the isolated rate probing scheme. On the rest of the overlay hops, the isolated rate probing scheme is again more aggressive than the buffer estimation scheme.

5.7 Summary

The end-to-end throughput of data transfers in overlay networks that carry data over one or more TCP connections between consecutive overlay nodes is limited by the minimum of the TCP throughputs achievable on each overlay hop. In this work, we focus on a path in an Overlay TCP network with the aim of maximizing the end-to-end throughput by using multiple parallel connections on one or more overlay hops. We show that the



(a) End-to-End Throughput



(b) Number of Connections

Figure 33: Evaluation of a 5-Hop Overlay Path (a) End-to-End Throughput, (b) Number of Connections

use of multiple parallel connections on some overlay hops does indeed increase the end-to-end throughput. We have proposed two schemes that assess the network conditions and dynamically increase or decrease the number of connections used on each overlay hop. We show through experiments on PlanetLab nodes that both schemes significantly improve performance while keeping the number of extraneous connections to a minimum.

The overlay path design was just a first step toward maximizing the throughput of an Overlay-TCP network. Next, we look at the design of a network of overlay nodes with the aim of maximizing the throughput of data carried by the overlay network as a whole.

CHAPTER VI

DESIGN OF AN OVERLAY-TCP NETWORK WITH THROUGHPUT AND FAIRNESS CONSIDERATIONS

6.1 Introduction

In this chapter, we present the design of an Overlay-TCP network that maximizes the total end-to-end throughput of the data carried on the overlay network, while maintaining fairness between competing overlay networks.

At the outset, we would like to remind the readers that in this chapter, we consider *infrastructure* overlay networks that provide some service to applications. Such networks have a stable topology with a set of overlay nodes that provide the functionality necessary for the offered service, and specified routing between overlay nodes. They carry aggregated data from multiple sources to multiple destinations, which are external to the network. We are *not* looking at peer-to-peer file-sharing networks that support voluntary participation of nodes and thus have highly dynamic topologies. We also stress that we are primarily concerned with sharing between competing overlay networks as we are looking at scenarios where the overlay traffic accounts for a majority of the traffic (say 70-80%) on the native network.

To illustrate the potential unfairness in sharing native resources between competing overlay networks, consider a simple example, shown in Figure 34. In this scenario, two overlay networks carry data from multiple ingress overlay nodes to multiple egress overlay nodes. A data transfer between a pair of ingress and egress overlay nodes is referred to as an *overlay flow*. The two overlay networks compete for the bottleneck bandwidth on native link I-J, with network 1 carrying four flows and network 2 carrying one flow on that link. As one might expect, simulations using ns2 [66] show that when both networks forward data over UDP or IP tunnels, they share the native link bandwidth in the ratio 4:1. While

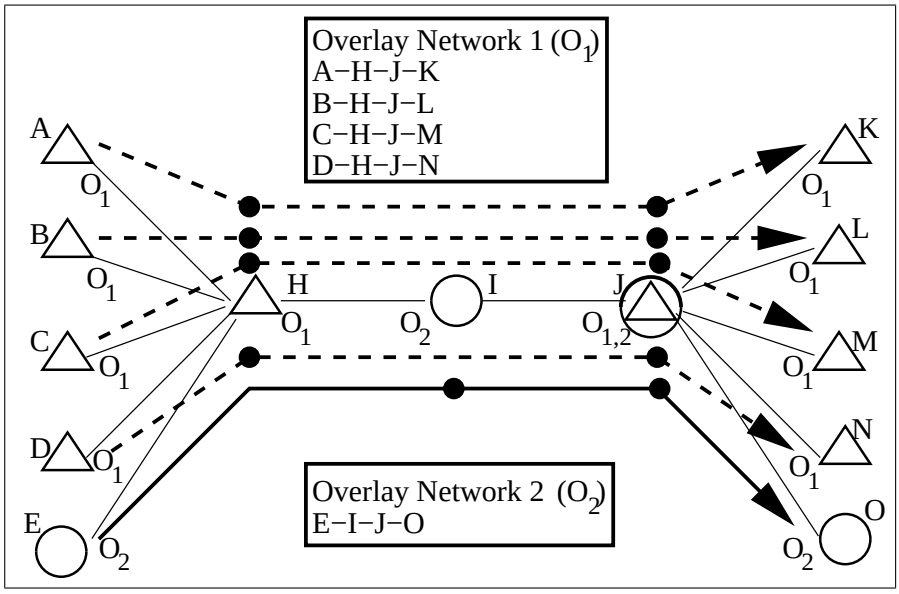


Figure 34: Example: Sharing Between Multiple Overlay Networks

this sharing behavior is not necessarily unfair, it is strictly a function of the end-to-end traffic offered by the two overlay networks, and it is not controllable. We refer the readers to Section 6.2 for more details on this example.

In our work, we investigate the use of TCP connections for building overlay networks, as a mechanism to control the sharing of native resources by competing overlay networks. Within the constraint of fairness to competing overlay networks, our secondary aim is to maximize the total throughput of the data carried by each overlay network, We use TCP connections because it implements congestion control mechanisms, which help with fair sharing. The specific sharing, of course, depends on factors such as the round trip times of the TCP connections and the loss probability on the native links. In addition, the use of TCP connections enables us to control the proportional sharing between multiple overlay networks, as will be seen later.

Given TCP connections on overlay hops, we face the question of exactly how to use TCP connections to build overlay networks. There are three possibilities:

1. Use a single ingress-to-egress TCP connection for each overlay flow in each overlay network.

2. Use a single TCP connection on each overlay hop in each overlay network.

3. Use multiple TCP connections on each overlay hop in each overlay network, with a limit on the maximum number of TCP connections allowed on each hop.

Solution (1) is quite straightforward to implement. However, it does not allow control over the total load offered by an overlay network, and can therefore lead to unfair sharing of native link bandwidth between competing overlay networks. Using this solution for the example shown in Figure 34, overlay network 1, with more overlay flows on the bottleneck link, will get a higher total throughput than network 2.

Solutions (2) and (3) provide more control over the sharing of resources between competing overlay networks. Both solutions require aggregation of the data to be transferred on each overlay hop. Solution (2) forwards the aggregated data on each overlay hop using a single TCP connection, whereas solution (3) uses multiple TCP connections. In solution (3), by changing the number of TCP connections used on each overlay hop, we can control the total TCP throughput achieved on the hop. This helps us maximize the total throughput of the overlay network, as well as control the sharing between multiple overlay networks. The total end-to-end throughput achieved in solution (3) is higher than that in solution (2).

In this chapter, we focus on the design of an *Overlay-TCP network* that implements solution (3) at the application layer. An example of such an Overlay-TCP network is shown in Figure 25, the terminology of which is explained in Section 6.3. The primary challenge in this design is to determine the number of TCP connections required on each overlay hop. In addition, the aggregation of data on each overlay hop introduces congestion at the overlay layer. This leads to fairness issues between overlay flows in the same overlay network, referred to as *intra-overlay-network fairness*, which we handle in our work.

In summary, our goal in this work is to design an overlay network that (1) shares resources with competing overlay networks in a fair and controllable manner, (2) maximizes the total end-to-end throughput of data carried on the network, (3) uses as few TCP connections as possible on each overlay hop, and (4) maintains intra-overlay-network fairness.

We develop a centralized algorithm to compute the number of TCP connections required on each overlay hop to achieve the above goal. We also propose three heuristics that are distributed in nature, and which aim to achieve the allocation specified by the

centralized algorithm. Using simulations, we demonstrate that by varying the maximum number of TCP connections on each overlay network, we can control the sharing of native resources between competing overlay networks. We also show that all four schemes achieve a significantly higher throughput than the scheme where each overlay hop uses a single TCP connection. The total throughput achieved improves when more knowledge about the overlay network is available to the heuristic.

Compared to the provisioning architecture presented in Chapter 5, the challenges posed when designing an *overlay network* are quite different. In addition to the problem of fair sharing between multiple overlay networks, the aggregation of multiple overlay flows at each overlay node introduces the problem of congestion at the overlay layer. We address both problems in this work. In summary, in our work, we develop mechanisms for controlling the sharing of native resources between competing overlay networks. The appropriate use of these mechanisms is a matter of policy.

The outline of the chapter is as follows. We start with a simple example to illustrate the sharing of native resources between competing overlay networks in Section 6.2, and elaborate on the design goals of this work in Section 6.3. We develop a centralized algorithm and three heuristics that meet these goals in Section 6.4. In Section 6.5, we outline some implementation details, and present simulations results in Section 6.6. Finally, we summarize the conclusions of our work in Section 6.7.

6.2 Example: Sharing Between Multiple Overlay Networks

In this Section, we present a simple example that illustrates the potential for unfair sharing of native resources between competing overlay networks when no mechanisms are used.

Figure 34 shows the topology used in the example. Nodes in overlay network 1 (O_1) are shown with triangles, and those in overlay network 2 (O_2) are shown with circles. With overlay flows as shown in the figure, native links H-I-J are shared by both overlay networks. Bottleneck link I-J, with a capacity of 1Mbps, is shared by overlay flows A—K, B—L, C—M, D—N in O_1 and E—O in O_2. All flows have the same round-trip time. We also introduce some background TCP traffic on the bottleneck link.

Table 6: Example of Sharing Between Multiple Overlay Networks Using Different Methods: Total Throughput Achieved

Method	Overlay Network 1 (bytes/sec)	Overlay Network 2 (bytes/sec)	Total (bytes/sec)	Ingress-Egress Loss
UDP	99313	25702	125015	95%
End-to-End TCP	91000	16720	107720	0%
Hop-by-Hop Single TCP	29936	28948	58884	0%
Hop-by-Hop Multiple 4/4 TCP	55466	65341	120807	0%
Hop-by-Hop Multiple 10/4 TCP	88749	32778	121527	0%

We present results for ns2 simulations of this scenario. Table 6 shows the total throughput achieved by each overlay network in bytes/sec and the total throughput in the whole system. The four flows in O_1 share the total throughput achieved by the overlay network equally, as their round trip times are the same. The methods compared are:

- UDP— The simple case where overlay nodes forward data to each neighbor without using any congestion control mechanisms.
- End-to-End TCP— Each overlay flow starts an end-to-end TCP connection.
- Hop-by-Hop Single TCP— Each overlay hop uses a single TCP connection to forward data.
- Hop-by-Hop Multiple N_1/N_2 TCP— Each overlay hop uses multiple TCP connections to forward data, with a maximum of N_1 or N_2 connections allowed on each overlay hop, for overlay networks 1 and 2 respectively. We use our proposed centralized algorithm to determine the number of TCP connections on each overlay hop.

As expected, comparing the total throughput achieved by each overlay network, we observe that when using UDP as well as end-to-end TCP, overlay network 1 achieves 4 times as much total throughput as overlay network 2. The overlay networks therefore share the bottleneck link bandwidth in proportion to the number of overlay flows carried by each network on that link. This sharing is strictly a function of the end-to-end offered traffic on both overlay networks, and is not controllable. For instance, a 3:1 sharing is not achievable

in this case. The UDP flows get a higher throughput than the end-to-end TCP flows due to the fact that the UDP flows totally block out the background TCP flows. Also, the UDP flows suffer a loss rate of almost 95%, as they are all competing with each other and sending data at a high rate. The end-to-end TCP flows, on the other hand share the bottleneck link bandwidth fairly with the background traffic.

On the other hand, when using TCP connections on each overlay hop, the two overlay networks share the bottleneck link bandwidth in more equal measure. Using a single TCP connection on each overlay hop gives a 1:1 sharing, with the least total throughput. This sharing can be controlled and total throughput can be improved by varying the maximum number of TCP connections allowed on each overlay hop for each overlay network. As seen in the last two rows in the Table, the sharing can be changed from 1:1 to 3:1 by changing the maximum number of TCP connections from 4:4 to 10:4 for O_1 and O_2 respectively. In addition, the total throughput in the whole network is maximum using our schemes. The end-to-end TCP scheme gives the least throughput because each TCP connection has a larger RTT, and loss recovery is done end-to-end, rather than on each overlay hop, thus lowering the end-to-end throughput. This has been observed by others as well [98, 99].

In summary, this simple experiment demonstrates (1) the potential unfairness in sharing of native bottleneck bandwidth between multiple overlay networks when using UDP or IP tunnels to forward data, (2) the ability to control the sharing between competing overlay networks by varying the maximum number of TCP connections allowed on each overlay hop in each overlay network, and (3) the ability to improve total throughput in the system using our proposed schemes.

6.3 Design Goals

In this section, we focus on the design of an Overlay-TCP network that is fair to competing overlay networks. We outline the model of an Overlay-TCP network, present the challenges faced in the design, and then summarize our design goals.

Table 7: Model of an Overlay Node in an Overlay-TCP Network

System Parameters	
m	Number of nodes in native network
n	Number of nodes in overlay network
l	Number of links in native network
h	Number of links in overlay network
f	Number of overlay flows in overlay network
N_{max}	Maximum number of TCP connections on each overlay hop
System Variables	
UN_i	i^{th} node in native network, $i = 1..m$
O_i	i^{th} node in overlay network, $i = 1..n$
L_i	i^{th} native link, $i = 1..l$
K_i	i^{th} overlay hop, $i = 1..h$
N_i	Number of TCP connections on i^{th} overlay hop, $i = 1..h$
R_i	Average isolated rate on i^{th} overlay hop, $i = 1..h$
T_{i,N_i}	Total throughput on i^{th} overlay hop using N_i TCP connections, $i = 1..h$

6.3.1 Model of an Overlay-TCP Network

As a reminder, we describe below the model of an Overlay-TCP network that we consider in our work. Table 7 summarizes the notation used in the chapter. Consider a native network comprising m nodes and l links between these nodes. A subset ($n \leq m$) of these nodes function as overlay nodes, with h overlay hops between these nodes. Each overlay hop i carries data over N_i TCP connections. We define the *isolated rate* (R_i) of an overlay hop as the throughput of a single TCP connection on that hop. We refer to such a network as an *Overlay-TCP Network*. An example Overlay-TCP Network is shown in Figure 25.

Senders and receivers, referred to as *overlay users*, are not necessarily part of the overlay network. Sources send data to an *ingress overlay node*, which detects the transfer, say by looking at headers, as suggested by the proxy mechanism in [74]. The ingress overlay node then transfers the data over the Overlay-TCP network to the *egress overlay node*. This node demultiplexes the data and delivers it to the appropriate receiver. We refer to the data transfer between a pair of ingress and egress overlay nodes as an *overlay flow*. We assume that the routing between overlay nodes is given, and each overlay node forwards data according to the forwarding table available.

The model of an overlay node in this chapter has been modified, compared to the model

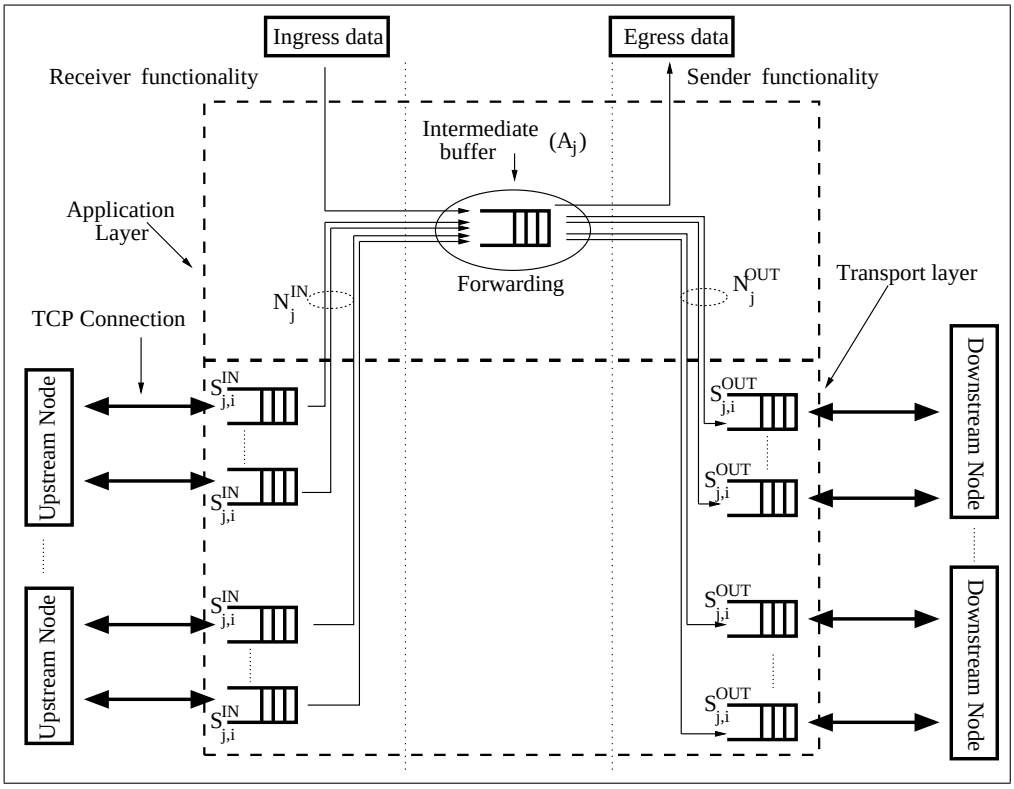


Figure 35: Model of an Intermediate Overlay Node in an Overlay-TCP Network

shown in Figure 28 in Chapter 5 to account for multiple upstream and downstream overlay hops in an overlay network. The modified model for an overlay node is shown in Figure 35. The dashed box represents the internal structure of the overlay node. The upper half is the application layer, whereas the lower half is the transport layer. The part to the left of the first vertical dotted line is the *receiver functionality* of the overlay node, and the part to the right of the second dotted line is the *sender functionality*. Each overlay node has multiple incoming TCP connections with each upstream node. The overlay node reads data from the socket buffers of these TCP connections in the form of packets and forwards it to the application-layer buffer. The *forwarding* module looks at the (application/overlay) header of the packet, determines the destination of the packet and looks up the forwarding table to determine the next overlay node to forward the packet to. It forwards the packet to one of the TCP connections to that overlay node. Note that all overlay nodes now function as routers. In addition, at each overlay node, there can be new ingress data that is added to the overlay network and some egress data that exits the network.

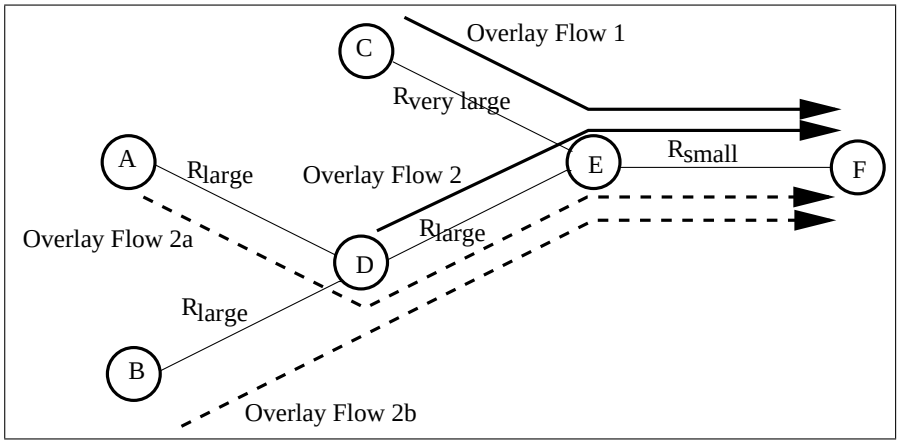


Figure 36: Example: Intra-Overlay-Network Fairness

6.3.2 Challenge: Intra-Overlay-Network Fairness

We elaborate on the problem of congestion at the overlay layer, introduced due to the aggregation of overlay flows carried on an overlay hop. To simplify the discussion, we consider overlay hops with a single TCP connection in the following section. However, the arguments hold for overlay hops with multiple TCP connections as well.

As mentioned in Chapter 5, if an overlay flow is the only flow traversing an overlay network, its end-to-end throughput is limited by the minimum isolated rate of the overlay hops on its path. The isolated rate of each overlay hop depends on the characteristics of the underlying native path. However, when multiple overlay flows are traversing an Overlay-TCP network, the flows that share an overlay hop will need to share the isolated rate on that overlay hop. This can lead to overlay-layer congestion due to a number of flows being routed through the same overlay hop. As a result, we face the *intra-overlay-network fairness* problem, as discussed below.

Consider overlay nodes arranged in the topology shown in Figure 36, with the isolated rate on each overlay hop as shown. Consider 2 overlay flows, C—F (flow 1) and D—F (flow 2), such that node E forwards the data received on overlay hops C-E and D-E onto overlay hop E-F. Ideally, the isolated rate on overlay hop E-F should be split equally between the two overlay flows. This can be done if node E alternately reads data from the incoming socket buffers of TCP connections on hops C-E and D-E. If there is no data available on the

socket buffer of D-E, the read can be kept non-blocking, and E can then read and forward data from C-E.

Now consider a scenario in which 3 overlay flows, C—F (flow 1), A—F (flow 2a) and B—F (flow 2b) exist in the network. The data from hops A-D and B-D is aggregated at node D and sent on hop D-E over one TCP connection. At node E, data from hops C-E and D-E is aggregated and sent over hop E-F. Now, if node E reads alternately from the socket buffers of connections C-E and D-E to send onto connection E-F, the isolated rate on overlay hop E-F will be shared in the ratio 2:1:1 between overlay flows C—F, A—F and B—F respectively. This can be considered unfair to flows A-F and B-F. Ideally all three flows should get an equal (one third) share of the isolated rate on E-F. We refer to this problem as the *intra-overlay-network fairness* problem.

This problem can be solved by keeping track of all the (ingress, egress) pairs, corresponding to overlay flows, that traverse each incoming overlay hop at each overlay node. Overlay nodes can then read from the socket buffer of the TCP connection on each overlay hop in proportion to the number of overlay flows carried on that hop. In the above example, overlay node E should determine that TCP connection D-E comprises two overlay flows (A—F and B—F) and that TCP connection C-E comprises just one overlay flow (C—F). Then, node E should read from TCP connection D-E twice before reading from TCP connection C-E. Node D should also read alternately from A-D and B-D, thus ensuring that all three overlay flows get an equal (one-third) share of the isolated rate on E-F. We formalize this fairness notion as follows.

Definition: An allocation of overlay flow rates is said to be *intra-overlay-network fair* if it is feasible and for any alternative feasible allocation of overlay flow rates where a particular overlay flow gets a higher rate in the new allocation as compared to the old allocation, there exists some other overlay flow with an already lower rate than the first flow, which gets an even lower rate in the new allocation. This definition is the max-min[10] fairness definition applied to the sharing of TCP throughput on each overlay hop amongst flows at the overlay layer.

To summarize, any provisioning scheme for an Overlay-TCP network should be able to

handle slow overlay hops due to the native path, as well as congestion at the overlay layer due to multiple overlay flows sharing the overlay hops.

6.3.3 Summary of Design Goals

In summary, our goal in this work is to design an Overlay-TCP network that:

1. shares native resources with competing overlay networks in a fair and controllable manner
2. maximizes the total end-to-end throughput of all data transfers on the overlay network
3. uses as few TCP connections as possible on each overlay hop, with a maximum of N_{max} connections
4. maintains fairness between overlay flows in the same overlay network.

6.4 Design Details

In this section, we focus on the design of a *single* Overlay-TCP network that meets our goals. We start by trying to determine in a centralized manner, the number of TCP connections required on each overlay hop, and the rate allocation for all flows in the overlay network, that will achieve our goals.

Assume that we are given the isolated rate on each overlay hop in the network and that these isolated rates are static for some period of time[1]. Considering the capacity of each overlay hop to be equal to the isolated rate on that hop, we can determine the rate for each overlay flow using the max-min fairness algorithm [10]. Now, in order to increase the total end-to-end throughput in the overlay network, we need to increase the total throughput achieved on the bottleneck overlay hops. The problem is to determine the bottleneck overlay hop that requires more TCP connections, and the number of new connections required.

We first present a heuristic to determine the total TCP throughput on each overlay

[1]We will address the issue of varying isolated rates in Section 6.5.

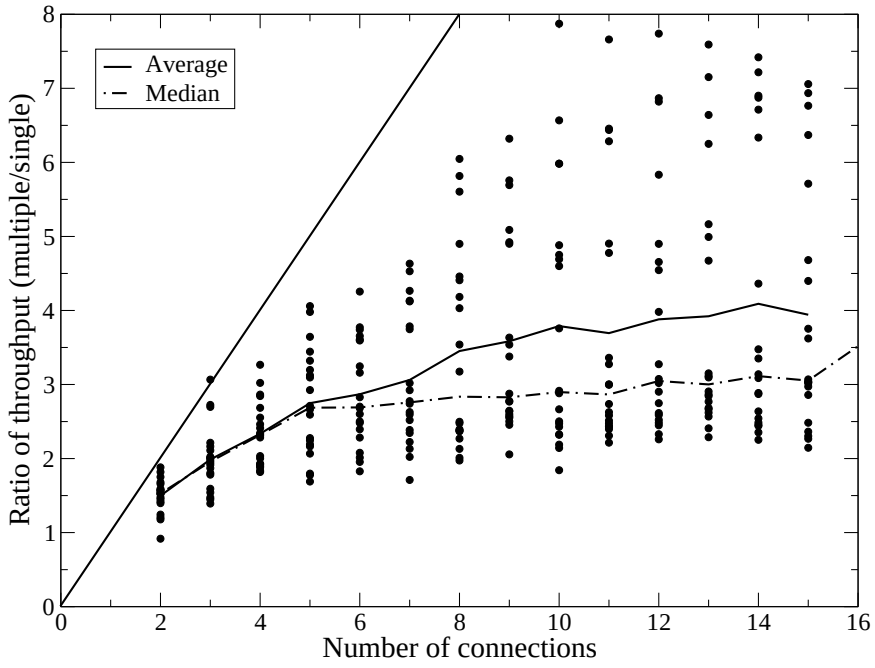


Figure 37: Improvement in Throughput Using Multiple TCP Connections: PlanetLab Experiment Results

hop using multiple TCP connections. We then develop a centralized algorithm that simultaneously determines the overlay hop that requires more TCP connections, as well as the required number of new connections. Finally, propose three distributed heuristics to achieve this allocation, with varying amount of knowledge of the network.

6.4.1 Effect of Increasing the Number of TCP Connections on Overlay Hops

Consider an overlay hop with an isolated rate of R. A naive assumption would conclude that using N TCP connections on the overlay hop would give a total throughput of N times R, and that one can increase the number of TCP connections indefinitely until the capacity of the bottleneck link on the native path of the overlay hop is reached. However, in practice, we are limited by two factors. The first is N_{max}, the system-wide limit on the number of TCP connections on each overlay hop in the network.

The other factor is that adding a new TCP connection does not necessarily translate to the addition of the throughput of one TCP connection on that overlay hop. To verify this, we performed numerous experiments on different overlay hops on PlanetLab [75] nodes. In Figure 37, we present a sample of the results obtained for an overlay hop between

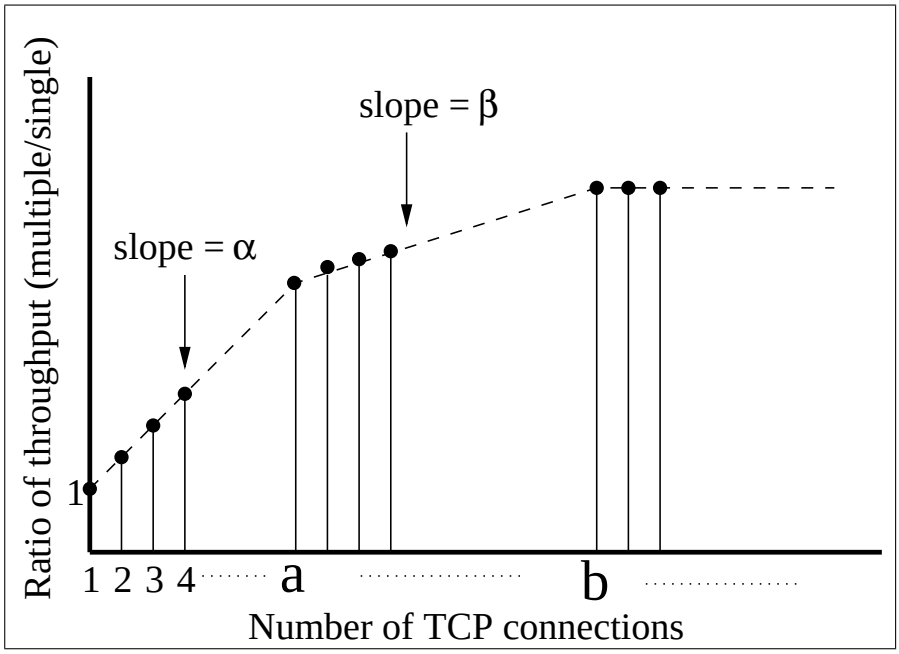


Figure 38: Model for Improvement in Throughput Using Multiple TCP Connections

Columbia University and Stanford University. The Y axis shows the ratio of throughput using multiple TCP connections to the throughput using a single TCP connection (isolated rate). We observe that as more connections are added, the throughput does not increase indefinitely. Also, the increase in total throughput is less than linear with the number of TCP connections.

We generalize the above observations to arrive at a model for the improvement in throughput using multiple parallel TCP connections, as shown in Figure 38. Let the isolated rate (throughput of a single TCP connection) on an overlay hop be denoted by $R = T_1$. When the number of TCP connections on the overlay hop is N, let the total throughput achieved be T_N. Note that N can only take non-zero integral values. We compute the improvement in total throughput as the ratio $\frac{T_N}{T_1}$. Initially, adding up to a connections, results in an improvement with a slope of α. When more than a connections are started, the slope decreases to β, and eventually, after b parallel TCP connections have been started, the ratio stays constant. Hence, increasing the number of TCP connections beyond b is futile. This pattern has been studied in literature [35], and was confirmed by our experiments on PlanetLab nodes. Using this model, the throughput of N parallel TCP connections can be

given by:

$$T_N = \begin{cases} T_1(1+\alpha(N-1)) & \forall N \leq a \\ T_1(1+\alpha(a-1)+\beta(N-a)) & \forall a < N \leq b \\ T_1(1+\alpha(a-1)+\beta(b-a)) & \forall b < N \end{cases}$$

The parameters specified in the above model (i.e. a, b, α, β) will be different for different overlay hops, depending on the native path characteristics such as the round-trip time, the loss probability and the background traffic. For example, on some hops, new TCP connections might not help. In this case, we assume $\alpha = 0, \beta = 0$ and $a = 1, b = 1$. In the remainder of the chapter, these parameters for overlay hop i are referred to as R_i, a_i, b_i, α_i and β_i, and the total throughput using N_i TCP connections is denoted by T_{i,N_i}. We present techniques to estimate these parameters in Section 6.5.

To conclude this discussion, we use the above model in our algorithm to determine the number of TCP connections required on each overlay hop.

6.4.2 Centralized Algorithm

Figure 39 presents a centralized algorithm that computes the minimum number of TCP connections required on each overlay hop to give the maximum total end-to-end throughput, while maintaining intra-overlay-network fairness. The algorithm assumes complete knowledge of the network, which includes the isolated rate ($R_i = T_{i,1}$), the TCP model parameters (a_i, b_i, α_i, β_i), and the number of overlay flows U_i carried on each overlay hop i.

The algorithm assumes that the capacity of each overlay hop is equal to the total TCP throughput using N_i connections on the hop (step 5). The algorithm successively improves the allocation by determining the bottleneck overlay hop in terms of throughput per overlay flow (steps 6-7), and if possible, increasing the number of TCP connections on that hop (step 17) to improve total the throughput. The intra-overlay-network fairness property is maintained at all times. When the algorithm terminates, it produces an allocation such that every overlay flow has at least one overlay hop with either N_{max} (maximum allowed) or b_i (maximum useful) number of TCP connections. This hop is the bottleneck overlay

1. $\forall i$, $N_i = 1$
2. $\forall i$, U_i = Number of unsaturated overlay flows on overlay hop i
3. $X = \{i\}$, $i = 1..h$ (Set of all overlay hops)
4. $Z = \{j\}$, $j = 1..f$ (Set of all unsaturated overlay flows)
5. $\forall i \in X$, Compute $C_i = T_{i,N_i}$
6. $\forall i \in X, F_i = \frac{C_i}{U_i}$
7. $F_{min} = min\{F_i\}$
8. Choose i with $F_i = F_{min}$ and min N_i
9. If $((N_i == b_i)$ or $(N_i == N_{max}))$
10. $\quad \forall j \in X, C_j - = U_j * F_{min}$
11. $\quad U_j - -$ on entire path for each overlay flow on overlay hop i
12. $\quad$ If $(C_j == 0)$
13. $\quad\quad X = X - \{j\}$
14. $\quad Z = Z - \{j\}$ for each overlay flow on hop i
15. $\quad$ if $Z = \{\}$, break
16. $\quad$ Repeat from step 6
17. else Increase N_i
18. Repeat from step 2
19. Compute final max-min fair allocation for overlay flows

Figure 39: Centralized Decision Algorithm

hop for that overlay flow. The final step computes the final max-min fair allocation for all overlay flows using the algorithm in [10].

As the algorithm has full knowledge of the overlay network, in some sense, the allocation specified by this algorithm is the best achievable allocation using as few TCP connections as possible. However, in practice, the actual rates achieved by each overlay flow will be subject to overheads due to the use of TCP connections on each overlay hop. These overheads include synchronization of phases of TCP connections on different overlay hops, the head-of-line blocking problem (described in Section 6.5), and the variation in isolated rate due to background traffic.

6.4.3 Proposed Heuristics

The above algorithm incurs a significant overhead due to control data sent between the overlay nodes and a control node, which computes the centralized allocation. In an attempt to devise a more distributed algorithm, we develop three heuristics to compute and control the number of TCP connections on the direct upstream and downstream overlay hops. These heuristics are implemented at each overlay node and vary in the amount of knowledge about the network, available to the node when making a decision.

1) Upstream and Downstream Tree Knowledge (Tree-Based):

To start with, we assume that every overlay node has knowledge of the necessary parameters and measured values of all overlay hops on the upstream and downstream trees formed by the overlay flows that are traversing this particular overlay node. Each node computes the allocation achieved for all the component overlay flows using a modified version of the centralized algorithm. This algorithm, shown in Figure 40, assumes that the overlay node can only control the TCP connections on the upstream and the downstream overlay hops. If one of these overlay hops is the bottleneck for any of the overlay flows traversing this overlay node, the algorithm increases the number of TCP connections on that hop (steps 18-19). This continues until one of the following two termination conditions is reached. The first is that none of the immediate upstream or downstream overlay hops is the bottleneck for the considered overlay flows. The second is that the upstream or downstream bottleneck

1. $\forall i$, $N_i = 1$
2. $\forall i$, U_i = Number of unsaturated overlay flows on overlay hop i
3. $X = \{$Set of overlay hops in upstream and downstream trees$\}$
4. $Y = \{$Set of direct upstream and downstream overlay hops$\}$
5. $Z = \{j\}$, $j = 1..f$ (Set of all unsaturated overlay flows)
6. $\forall i \in X$, Compute $C_i = T_{i,N_i}$
7. $\forall i \in X, F_i = \frac{C_i}{U_i}$
8. $F_{min} = min\{F_i\}$
9. Choose i with $F_i = F_{min}$ and min N_i
10. If $((N_i == b_i)$ or $(N_i == N_{max})$ or $(i \notin Y))$
11. $\quad \forall j \in X, C_j - = U_j * F_{min}$
12. $\quad U_j - -$ on entire path for each overlay flow on overlay hop i
13. $\quad$ If $(C_j == 0)$
14. $\quad\quad X = X - \{j\}$
15. $\quad Z = Z - \{j\}$ for each overlay flow on hop i
16. $\quad$ if $Z = \{\}$, break
17. $\quad$ Repeat from step 7
18. else if $(i \in Y)$
19. $\quad$ Increase N_i
20. Repeat from step 2
21. Compute final max-min fair allocation for overlay flows

Figure 40: Upstream and Downstream Tree Knowledge Algorithm

overlay hops have reached either N_{max} (maximum allowed) or b_i (maximum useful) number of TCP connections.

2) 1-Hop Neighborhood Knowledge:

In this case, an overlay node has knowledge about only the upstream and downstream overlay hops, i.e., a very small part of the trees. This information is relatively easy to gather, and is very much toward the zero-knowledge end of the spectrum. We then use the above algorithm (Figure 40), again assuming that we can control the number of TCP connections on any of the upstream or downstream overlay hops.

3) Zero Knowledge:

At the other end of the spectrum, the zero knowledge algorithm makes a decision purely based on local observations at an overlay node. We consider decisions based on the buffer occupancy level of the TCP socket buffers of the incoming connections, an idea proposed in Chapter 5. Figure 41 shows the corresponding algorithm, applied to the overlay network case for every incoming overlay hop at every overlay node in the network. The algorithm uses a buffer occupancy estimator, $\hat{B}$, based on an exponentially weighted moving average of periodic samples of instantaneous buffer occupancy as follows, $\hat{B} = \gamma * \hat{B} + (1-\gamma) * \hat{B}_{sample}$, where $0 < \gamma < 1$. We compare the estimated buffer occupancy to some low (w_L) and high (w_H) watermarks, below which we consider the buffer at the intermediate overlay node to be underfull, and above which we consider the buffer to be overfull. N_i^{IN} refers to the number of incoming connections on the current hop being considered, at overlay node i, and $N_{i,j}^{OUT}$ refers to the number of outgoing connections from node i to node j.

A high buffer occupancy (step 1) could mean that either the incoming overlay hop is too fast or at least one of the output hops for the flows on the incoming overlay hop is too slow. Hence we either decrease the number of connections on the incoming hop (step 3) because it is too fast, or increase the number on the outgoing hop with the minimum number of TCP connections (step 5-6). On the other hand, a low buffer occupancy (step 8) could imply that either the incoming hop is too slow or the outgoing hops for all overlay flows comprising the incoming hop are too fast. Hence, in this case, we need to either increase the number of TCP connections (step 11) on the incoming overlay hop, or decrease the number

1. if ($\hat{B} > w_H * B$) // overfull buffer
2. if ($N_i^{IN} > max\{N_{i,j}^{OUT}\}$) // too many incoming connections
3. $N_i^{IN} - -$
4. elseif ($N_i^{IN} \leq max\{N_{i,j}^{OUT}\}$) // not enough outgoing connections
5. if ($min\{N_{i,j}^{OUT}\} < N_{max}$)
6. $N_{i,j}^{OUT} + +$
7. endif
8. elseif ($\hat{B} < w_L * B$) // underfull buffer
9. if ($N_i^{IN} \geq max\{N_{i,j}^{OUT}\}$) // not enough incoming connections
10. if ($N_i^{IN} < N_{max}$)
11. $N_i^{IN} + +$
12. elseif ($N_i^{IN} < max\{N_{i,j}^{OUT}\}$) // too many outgoing connections
13. $N_{i,j}^{OUT} - -$
14. endif
15. else
16. Do nothing
17. endif

Figure 41: Decision Algorithm using Estimates of Buffer Occupancy

(step 12-13) on the outgoing overlay hop with the maximum number of TCP connections that the incoming overlay hop forwards data to.

6.5 Implementation Details

All algorithms proposed above require a number of quantities to be measured and parameters to be estimated. In this section, we present techniques for estimating the required quantities.

6.5.1 Network Conditions and Parameter Estimation

Each overlay node evaluates the network conditions under which the TCP connections operate, on its upstream and downstream overlay hops. In our work, we measure the isolated rate either directly by performing a data transfer at the start of the experiment, or indirectly by periodically measuring the buffer occupancy of the incoming TCP connections. Alternatively, some of the techniques discussed by He et al. [40] can be used for this estimation.

The estimation of the parameters (a_i, b_i, α_i and β_i) is more difficult. One way to estimate this is an incremental one, where the overlay node keeps adding a TCP connection and determines the parameters implicitly. Another way is to implement a separate estimation module, located at each overlay node to explicitly perform the experiments required to determine the above parameters. In Section 6.6, we evaluate the sensitivity of the allocations computed by the centralized algorithm to incorrect estimation of the parameters. Based on this analysis, we can determine how often and how accurate the parameters need to be, in order to achieve a reasonable allocation. In our simulations, we estimate these parameters explicitly using a separate estimation module.

The number of overlay flows transiting through an overlay hop can be determined by looking at the overlay header of each incoming packet and determining the (ingress, egress) pair, that identifies each overlay flow.

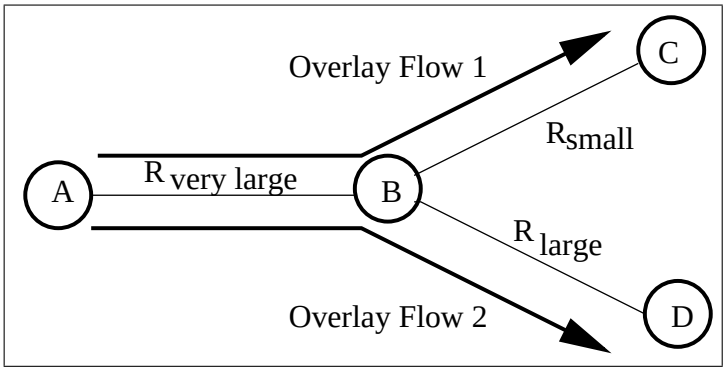


Figure 42: Example: Head-of-Line-Blocking

6.5.2 Communication Between Overlay Nodes

The parameters and values measured above need to be communicated by each overlay node to either a control node or to other nodes in the overlay network, as required by each algorithm.

In the *centralized algorithm*, all overlay nodes send all information to a designated control node. This node computes the number of connections required on each overlay hop using the centralized algorithm, and sends a response back to each overlay node. Although inefficient, this algorithm might be a feasible option for smaller overlay networks.

In the *upstream and downstream tree knowledge algorithm*, all ingress overlay nodes send the information gathered to the downstream neighbors, which forward it to their downstream neighbors and so on until each node receives information about its complete upstream tree. The egress overlay nodes similarly send the information to their upstream neighbors and so on until each node receives information about its complete downstream tree. Each overlay node can then compute the number of TCP connections on the upstream and downstream overlay hops.

The *1-hop neighborhood knowledge algorithm* requires the overlay nodes to send the information to just the upstream and downstream neighbors, whereas the *zero knowledge algorithm* does not require any communication between overlay nodes.

6.5.3 Refinements to Handle Head-of-Line Blocking

One problem that we face when using TCP connections on each overlay hop is what we refer to as the *head-of-line blocking* problem. Consider a scenario where an incoming overlay hop carries two overlay flows, which are forwarded onto two different outgoing overlay hops, all of which use a single TCP connection, as shown in Figure 42. Assume that the incoming hop has a much higher isolated rate than the two outgoing hops, and that one of the outgoing hops has a much lower isolated rate than the other outgoing hop. At node B, since the TCP connection on link B-C is very slow, it will eventually fill up the socket buffer. The next read call on the TCP connection A-B that returns a packet to be forwarded to B-C will block until the outgoing buffer has more space. Now, even if the TCP connection on B-D has the capacity to forward data, it will be unable to do so as there is no new data being read from TCP connection A-B. In this case, the slow connection on B-C is affecting the throughput of the overlay flow A-B-D. The addition of an application layer buffer to buffer some of the data being sent from A-B to B-C does not help because the buffer will eventually fill up and lead to the same problem.

This problem can be handled in multiple ways. The simplest policy would be to set up at least one incoming TCP connection for every input-output pair of overlay hops. This can lead to too many TCP connections on each overlay hop, similar to the end-to-end TCP approach. A more sophisticated way to do this would be to add TCP connections on demand when an overlay node/hop observes this kind of stalling behavior. At each overlay node, we need a mapping from the TCP connections on each incoming overlay hop to the TCP connections on each outgoing overlay hop. This mapping is a function of the relative rates that the overlay flows will get on their respective outgoing overlay hops.

If such a mapping does not help (due to the limit on the number of TCP connections), we can introduce overlay-layer losses by dropping packets that cannot be forwarded to the corresponding outgoing TCP connections (because the outgoing socket buffers are full) at the moment that they were read from the incoming TCP connection. This will increase the delivered throughput in the entire overlay network, at the cost of losses introduced on the slow overlay flows. These losses can be limited by dropping packets only up to a certain

allowed loss percentage for each overlay hop.

In our schemes, the use of multiple TCP connections mitigates this problem to some extent. In addition, when we read packets that cannot be forwarded to the corresponding overlay hop, we append them to the end of the application buffer from which they are read. This allows the next packet in the buffer to be read. The limitation of this approach is that eventually the application buffer will be filled up with packets that cannot be forwarded.

6.6 Simulation Results

In this section, we present the evaluation of our design using ns2 [66] simulations. We extended ns2 with enhancements for the FullTcpAgent [41] and implemented our proposed design using TCP tunnels with socket buffers on each overlay hop. We evaluated the four algorithms with respect to different performance metrics and also looked at the sharing between multiple overlay networks using our algorithms.

6.6.1 Simulation Setup and Performance Metrics

We first describe our simulation setup. We performed a number of different simulations, and show a sample of these in this section. For the following simulations, we set up a 10-overlay node topology on a 100-node native topology. These nodes were connected using 15 randomly chosen overlay hops. We did not implement a complete graph on the overlay topology. Each overlay hop is bidirectional, and uses one or more TCP connections for each direction, as determined by our algorithms. We added bottleneck links arbitrarily in the native network, and introduced a number of background TCP connections between arbitrary pairs of native nodes.

We then started a number overlay flows, again chosen at random on this overlay topology. In addition to the four schemes proposed in our work (centralized, tree-based, 1-hop neighborhood knowledge and buffer estimation), we present results for the single TCP connection scheme, as well as the NMax TCP scheme where all overlay hops carrying overlay flows in the network always use the maximum allowed number of connections. The single TCP scheme and the NMax TCP scheme represent two ends of the spectrum of the number of TCP connections used in the overlay network, and our schemes fall somewhere in

Table 8: Comparison of Schemes: Overlay-TCP Network with 20 Overlay Flows

Scheme	Total Throughput (bytes/sec)	Num of TCP Connections	Throughput/ Connection (bytes/sec)
Centralized	848980	80	10612
Tree-Based	802860	80	10036
1-hop Neighborhood	779140	96	8116
Buffer Estimation	527160	52	10138
Single TCP	398600	30	13287
NMax TCP	779140	96	8116

between.

The performance metrics used include the mean throughput of the overlay flows and the total throughput in the overlay network. We also show the number of TCP connections used by each algorithm[2], and compute the average efficiency of each TCP connection in the resulting allocation, as the ratio of the total throughput in the overlay network to the total number of TCP connections used. A higher mean or total throughput is desirable, while the number of TCP connections should be low. The efficiency of the TCP connections should preferably be high.

6.6.2 Single Overlay Network: Comparison of Schemes

We first compare the schemes for a sample simulation, as described above, to illustrate their performance. Table 8 shows the performance metrics for a simulation of 20 flows in the overlay network, with a maximum of 4 TCP connections allowed on each overlay hop. The primary observation is that all the proposed schemes give a better total throughput than the single-TCP connection scheme. The centralized scheme which has complete knowledge of the network uses fewer connections than the maximum (NMax TCP scheme) and gives a higher total throughput. As a result, it gives a much higher efficiency in terms of throughput per TCP connection in the system, as compared to the NMax TCP scheme.

The tree-based scheme starts almost the same number of connections as the centralized scheme, primarily because each overlay node gathers information about all overlay hops

[2]The number of TCP connections varies over time for the *buffer estimation scheme*, hence we show the maximum number of TCP connections used.

that are on the paths of the overlay flows traversing this node. It thus achieves a total throughput comparable to that of the centralized scheme. The 1-hop neighborhood scheme has limited knowledge, thus starting too many connections, and almost approaches the NMax TCP scheme in terms of number of connections and total throughput achieved. The buffer estimation scheme is more conservative due to very limited knowledge and indirect estimation of the isolated rate. It therefore uses very few extraneous connections and performs slightly better than the single TCP scheme.

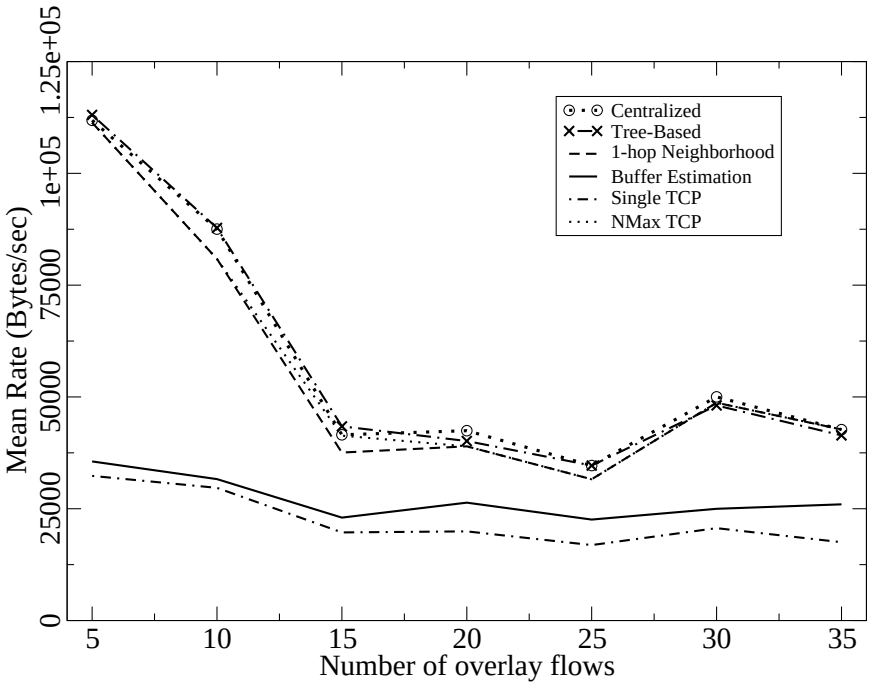
It should be clear that the single TCP connection scheme uses the minimum number of total TCP connections (30 i.e. one connection on each of the 15 overlay hops in each direction), whereas NMax TCP scheme uses the maximum total number of TCP connections. Although the single TCP scheme gives a high efficiency of TCP connection usage, it gives a low total throughput.

We now present an evaluation of the intra-overlay network fairness goal in our design of an Overlay-TCP network. In the above simulations, we computed the total throughput achieved on each overlay hop as the sum of the throughput of all the TCP connections comprising that hop. We then used the max-min fairness algorithm [10] to compute the rate that each overlay flow should get, with the capacity of each overlay hop equal to the total TCP throughput on that hop. We then compared this rate allocation vector with the vector of the end-to-end throughput of each overlay flow. For all the schemes considered, the rate vectors differed by less than 1%, on an average for each overlay flow. This shows that our schemes do indeed achieve an intra-overlay network fair allocation.

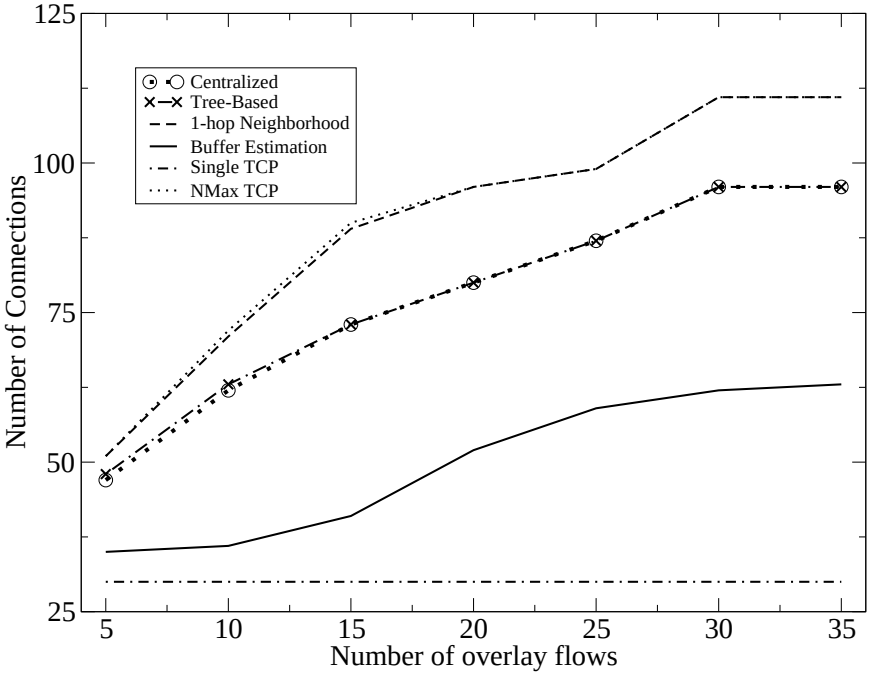
6.6.3 Single Overlay Network: Effect of Load on Overlay Network

Figure 43 presents the results of simulations using the different schemes for varying number of flows in the overlay network, which corresponds to varying load in the overlay network. In all cases, we maintain the same amount of background traffic. The X-axes show the different number of overlay flows that we considered in the network, and the Y-axes show the different performance metrics.

With increasing number of overlay flows in the network, the mean throughput of the



(a) Average Throughput



(b) Number of TCP Connections

Figure 43: Comparison of Schemes: Varying Number of Overlay Flows: (a) Average Throughput of Overlay Flows, (b) Number of TCP Connections Used by the Scheme

Table 9: Sharing Between 2 Overlay Networks: Equal N_{max}

Scheme	Total Throughput (bytes/sec)	Num of TCP Connections	Throughput/ Connection (bytes/sec)
Overlay Network 1: $N_{max} = 4$			
Centralized	1067210	59	18088
Tree-Based	1078320	55	19606
1-hop Neighborhood	987840	71	13913
Buffer Estimation	450770	36	12521
Single TCP	451610	30	15054
NMax TCP	909030	72	12625
Overlay Network 2: $N_{max} = 4$			
Centralized	741790	59	12573
Tree-Based	833630	56	14886
1-hop Neighborhood	872540	70	12465
Buffer Estimation	441360	34	12981
Single TCP	419600	30	13987
NMax TCP	913200	72	12683

overlay flows decreases for all algorithms. The centralized algorithm and the tree-based algorithm perform the best even with varying load in the network. They give the maximum total throughput using very few TCP connections. Although the single-TCP scheme and the buffer estimation scheme use fewer TCP connections, they give a low mean throughput as well. The buffer estimation scheme uses a few extraneous connections and performs slightly better than the single TCP scheme.

6.6.4 Sharing Between Two Overlay Networks

In this section, we demonstrate the sharing between 2 overlay networks, and our ability to control this sharing. We add an additional overlay network on top of the underlying topology, exactly the same way as described in Section 6.6.1. Tables 9 and 10 present the performance metrics for an ns2 simulation of this scenario.

Table 9, in which both overlay networks use the same value for the maximum number of TCP connections on each overlay hop in the network, shows that overlay network 1 achieves an almost equal, but slightly higher mean throughput than overlay network 2 using all schemes. Although the number of TCP connections started on both overlay networks is the same in all schemes, it does not imply that both overlay networks get the same total

Table 10: Sharing Between 2 Overlay Networks: Unequal N_{max}

Scheme	Total Throughput (bytes/sec)	Num of TCP Connections	Throughput/ Connection (bytes/sec)
Overlay Network 1: $N_{max} = 4$			
Centralized	650150	60	10846
Tree-Based	785840	56	14033
1-hop Neighborhood	563590	71	7948
Buffer Estimation	453280	36	12591
NMax TCP	576980	72	8014
Overlay Network 2: $N_{max} = 10$			
Centralized	1238850	122	10155
Tree-Based	1191840	106	11244
1-hop Neighborhood	1166990	151	7738
Buffer Estimation	441060	42	10501
NMax TCP	1144600	156	7347

throughput. This is because these connections are started on different overlay hops with different bottleneck links for each network.

Table 10, presents results for the case where the value of N_{max} differs for the two networks. Overlay network 1 uses a maximum of 4 TCP connections on each overlay hop, while overlay network 2 uses a maximum of 10 connections. In this case, overlay network 2 gets almost twice the mean throughput of overlay network 1 using most schemes. This shows that by varying the value of N_{max}, we can control the ratio in which 2 overlay networks share the native link bandwidth. The buffer estimation scheme is quite conservative, and therefore does not use as many connections as other schemes, and therefore does not give the unequal sharing that other schemes do. More aggressive parameters for this scheme, on overlay network 2, might result in a 1:2 sharing as desired in this case.

6.6.5 Sharing Between Multiple Overlay Networks

We now look at the sharing between multiple (> 2) overlay networks. In the ns2 simulation setup described in Section 6.6.1 we added multiple overlay networks, again, as described in Section 6.6.1. We present results for a scenario with 5 competing overlay networks that share the same set of underlying resources in the native network.

Figure 6.6.5 shows the total throughput achieved by each of the 5 competing overlay

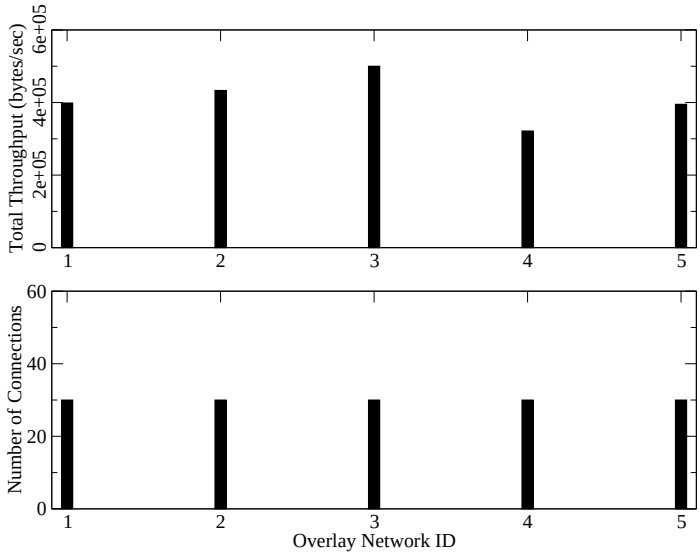
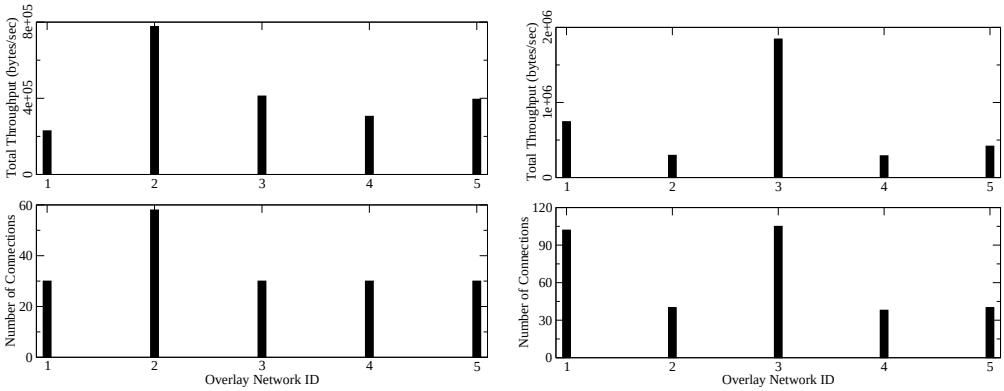


Figure 44: Sharing Between Multiple Overlay Networks: All Overlay Networks use 1 Connection on Each Overlay Hop

networks, and the total number of TCP connections used, when using a single TCP connection on each overlay hop. The total number of TCP connections is the same for all overlay networks. The total throughput is also almost the same for all overlay networks. The throughput for all networks is not exactly equal as different overlay networks share different links in the native network.

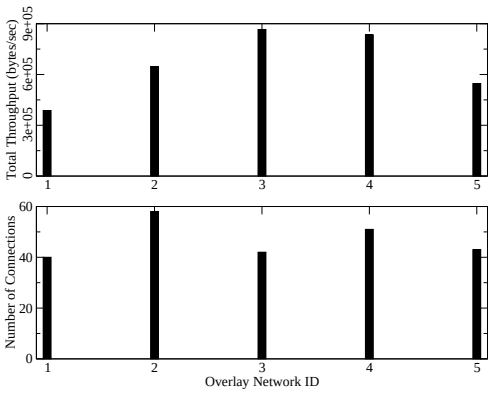
Figure 6.6.5 shows the total throughput and the total number of TCP connections used, when using the centralized algorithm, with different values of N_{max} for different overlay networks. In each subfigure, the lower graph shows the number of TCP connections used by each overlay network, and the upper graph shows the total throughput achieved by each overlay network. We observe that in all graphs, the proportion of total throughput roughly follows the proportion of the values of N_{max} for different overlay networks, and hence the total number of TCP connections used in the network. By setting the value of N_{max} to be very high, or very low, an overlay network can be forced to achieve a very high or a very low total throughput.

The main point we have tried to illustrate here is that by setting different values of the maximum number of connections allowed on each overlay hop for different overlay networks, we can vary the proportion of sharing between different networks. Thus, our schemes serve

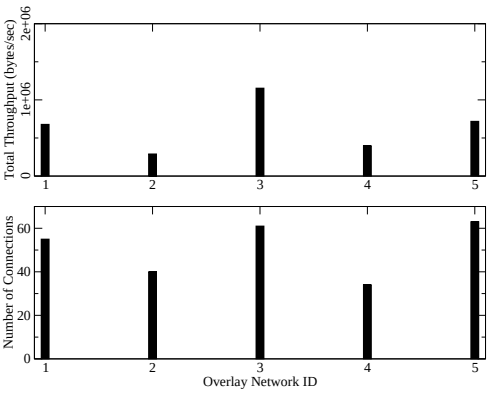


(a) N_{max} = 1,4,1,1,1

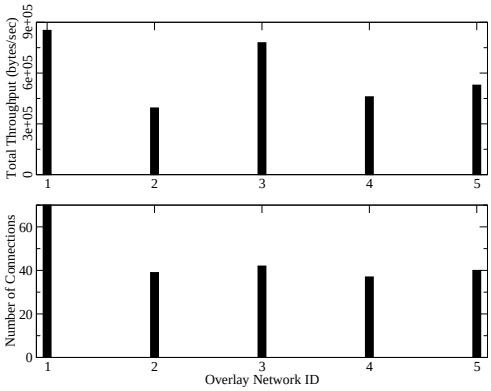
(b) N_{max} = 10,2,10,2,2



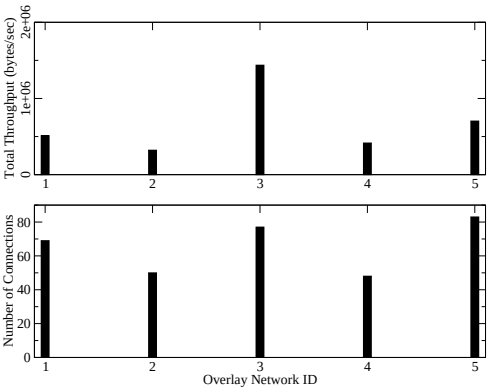
(c) N_{max} = 2,4,2,4,2



(d) N_{max} = 4,2,4,2,4



(e) N_{max} = 6,2,2,2,2



(f) N_{max} = 6,3,6,3,6

Figure 45: Sharing Between Multiple Overlay Networks: Centralized Algorithm with Different values of N_{max} for Different Overlay Networks

as mechanisms to achieve different definitions of fairness of resource allocation between different overlay networks.

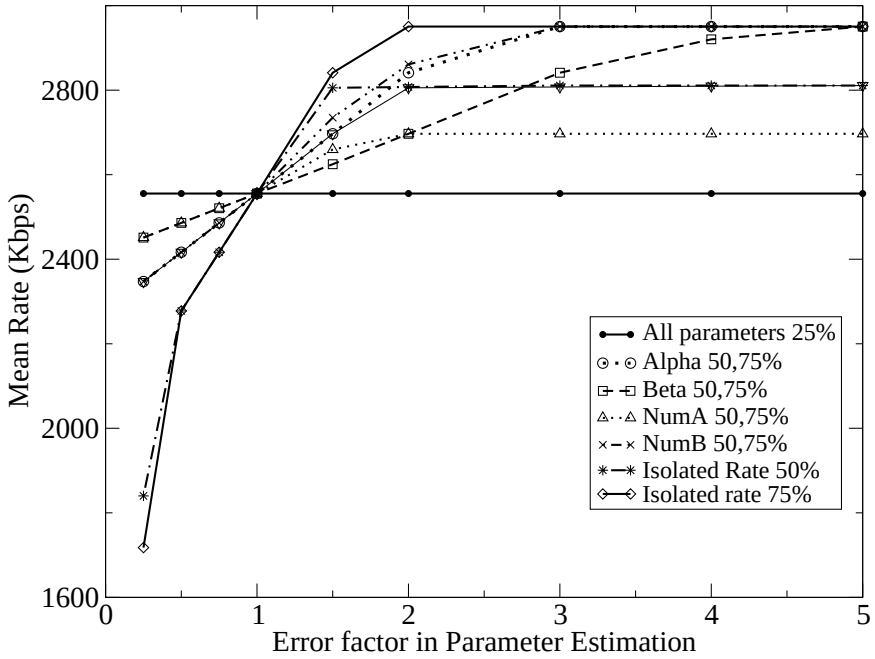
6.6.6 Sensitivity to Parameter Estimation

We now evaluate the sensitivity of the allocation achieved using the centralized algorithm to correctly predicting the parameters, $R_i, \alpha_i, \beta_i, a_i, b_i$. We vary these parameters by either under-estimating or over-estimating them on either the bottleneck links, or the non-bottleneck links, or all used links. We also vary the percentages of links in the network with either under-estimated or over-estimated parameters.

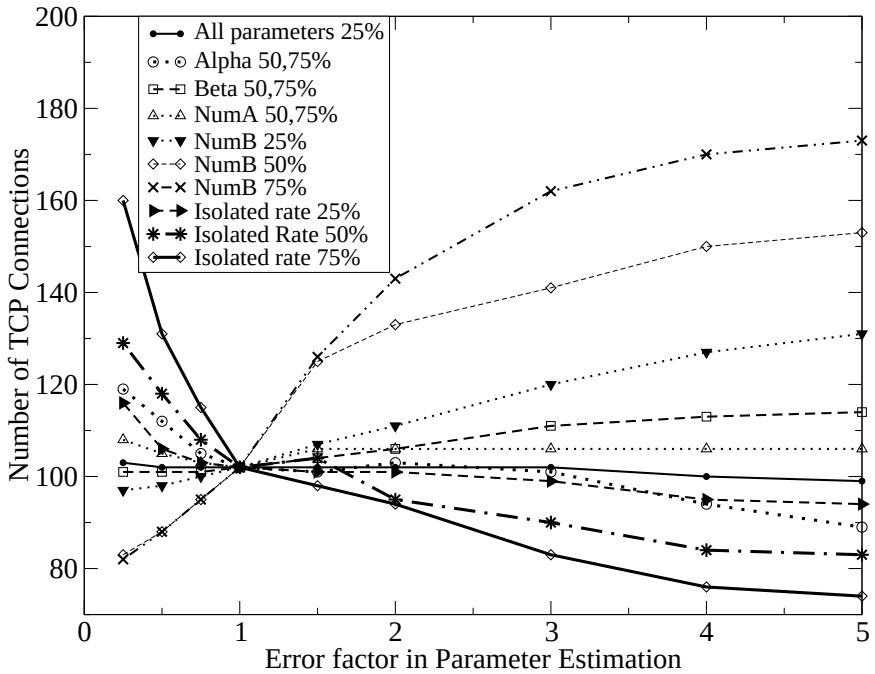
Figures 46 and 47 show various performance metrics for the allocation computed, when the error in estimation of parameters is on all used links. The X-axes in all graphs show the error in estimating each parameter. All errors less than 1 imply under-estimation, and errors greater than 1 imply over-estimation. An error of 1 indicates correct estimation, which is the reference point for comparison. The Y-axes show the average, variance and minimum of the computed rate allocation for all overlay flows in the network. We also show the total number of TCP connections used in each allocation.

The general trends indicate that an over-estimation of parameters causes an over-estimation of the mean and minimum; and an under-estimation of the variance and the number of TCP connections. On the other hand, an under-estimation of parameters causes an under-estimation of the mean and minimum; and an over-estimation of the variance and the number of TCP connections. We observe that an error in estimation of the parameters on 25% of the links does not affect the performance metrics much. An error in estimating the isolated rate affects the performance metrics the most and the value of a_i affects the metrics the least. The values of b_i and α_i affect the metrics significantly as well. In particular, an over-estimation of b_i causes significant over-estimation of the number of TCP connections. The maximum rate amongst all overlay flows does not vary with an error in estimation.

The graphs for error in parameter estimation on just bottleneck links look similar, but the effect is not as pronounced as in the all used links case. The effect of error in estimation

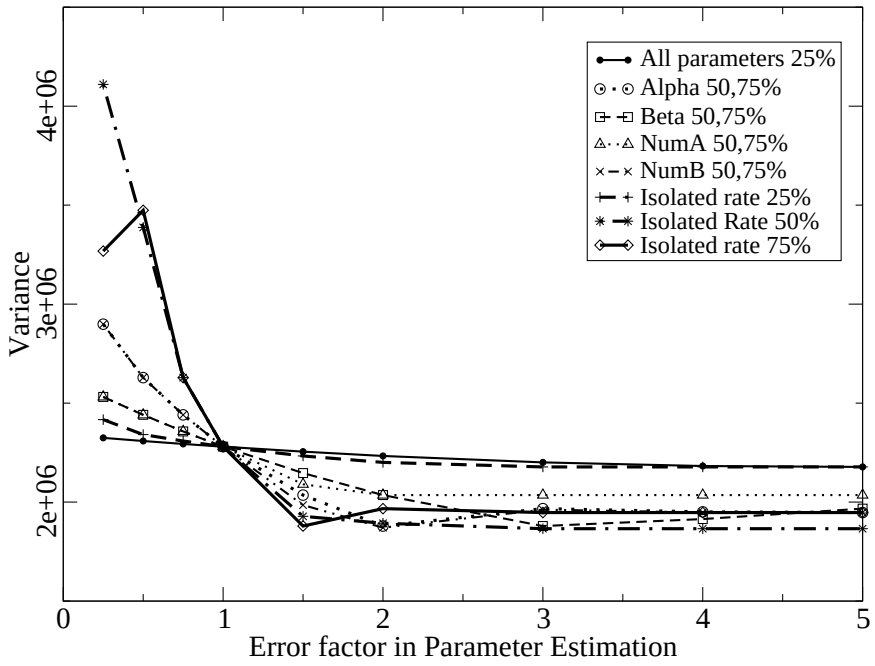


(a) Sensitivity: Mean Rate

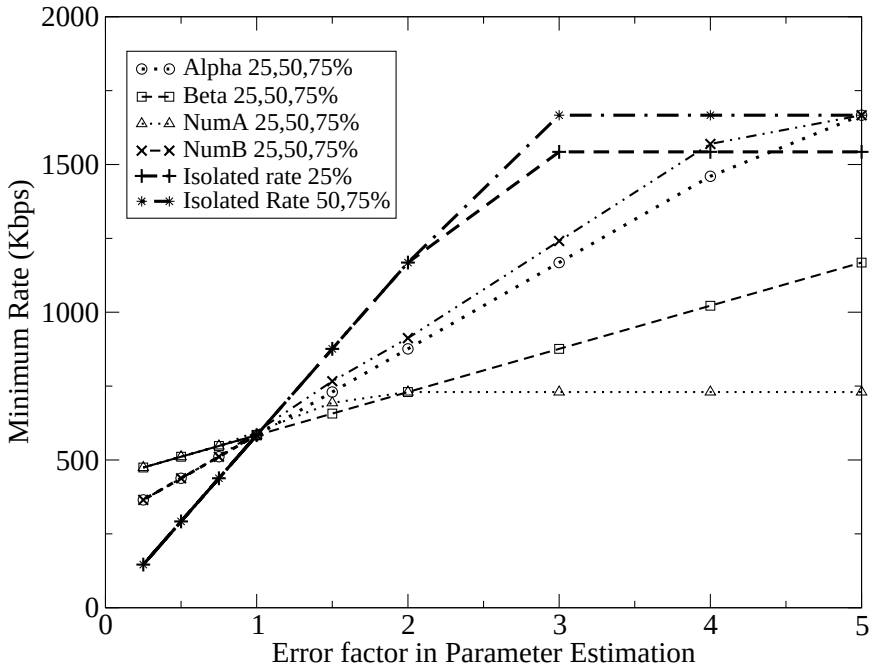


(b) Sensitivity: Number of TCP Connections

Figure 46: Sensitivity to Error in Parameter Estimation on a Fraction of All Used Links: (a) Mean of all Flow Rates, (b) Number of TCP Connections Specified by the Algorithm



(a) Sensitivity: Variance in Rates



(b) Sensitivity: Minimum Rate

Figure 47: Sensitivity to Error in Parameter Estimation on a Fraction of All Used Links: (a) Variance of All Flow Rates, (b) Minimum of All Flow Rates

of parameters on non-bottleneck links is not significant.

These graphs suggest that it is a good idea to periodically estimate the value of the isolated rate (R_i) and the values of b_i and α_i should be as accurate as possible. In our simulations we estimated the values at the start of the experiment.

6.7 Summary

In this chapter, we have shown the need for mechanisms that ensure fairness between multiple competing overlay networks that share the same set of resources in the underlying Internet. We look at the use of TCP connections for building overlay networks, as a mechanism to achieve fair sharing. We have designed an *Overlay-TCP network* that uses one or more TCP connections on each overlay hop. Our design controls the number of TCP connections used on each overlay hop in order to maximize the total throughput achieved by the network, while sharing native resources fairly with competing overlay networks. We show that this sharing can be controlled by varying the maximum number of TCP connections allowed on each overlay hop for different networks. Our design also ensures fairness between multiple overlay flows in the same overlay network. Our proposed centralized algorithm and the three heuristics that operate in a distributed manner achieve a significantly higher throughput than the scheme where each overlay hop uses a single TCP connection. In addition, our schemes use fewer TCP connections than the scheme where all hops use the maximum allowed number of TCP connections because some extraneous connections on faster overlay hops are deemed unnecessary.

Our work will help overlay network designers architect their network to share resources with competing overlay networks in a fair manner, where the definition of fairness would be a policy decision. In addition, the network designers will be able to maximize the total throughput of the data transferred on their overlay network, using as few TCP connections as possible. Also, the overlay flows within the same overlay network will be able to share the resources available to the overlay network in a manner similar to max-min fairness.

CHAPTER VII

CONTRIBUTIONS AND FUTURE WORK

7.1 Research Summary

To summarize, the contributions of this thesis are as follows:

- *Bootstrapping in Gnutella: A Measurement Study*

  The first part of the thesis investigates the bootstrapping function of the Gnutella peer-to-peer network. This study is the first to investigate the bootstrapping functionality in Gnutella. We have presented the common thread in the bootstrapping functionality of four popular Gnutella servents, as well as pointed out the differences. We have shown that the neighbor peers found during bootstrapping have a significant impact on the search performance of a peer. We have also carried out a measurement-based study of the GWebCache system, a primary component of the bootstrapping functionality in Gnutella.

- *Multipoint-to-Point Session Fairness in the Internet*

  The second part of the thesis investigates fairness issues involved in the use of static *multipoint-to-point sessions* while transferring data over connections from multiple servers to a single client. We have proposed two fairness definitions that are multipoint-to-point session fair. We have also proposed algorithms to achieve these definitions in a centralized manner.

- *Optimizing End-to-End Throughput for Data Transfers on an Overlay-TCP Path*

  This part of the thesis investigates issues in the design of a single path in an Overlay-TCP network. We have proposed the Adaptive Overlay-TCP Provisioning approach, in which the intermediate overlay nodes assess the state of the network path and dynamically determine the number of TCP connections needed on each overlay hop. The buffer occupancy estimation scheme proposed as a part of this provisioning is a

novelty in this context.

- *Design of an Overlay-TCP Network with Throughput and Fairness Considerations*

  This was the first work that looked into issues involved in the design of an Overlay-TCP network, with the option of using multiple parallel TCP connections on each overlay hop. The primary aim in designing this network was to enable competing overlay networks to share the same set of native resources in a fair and controllable manner. At the same time, we aimed at maximizing the total throughput of data carried on the overlay network, using as few TCP connections as possible. Our design also addressed the problem introduced due to overlay layer congestion, referred to as intra-overlay-network fairness. We showed that all four schemes indeed achieve the specified goal. In addition, we showed that by varying the maximum number of connections allowed on overlay hops in each competing overlay network, one can vary the proportion of sharing between competing overlay networks.

7.2 Future Directions

Finally, we will now present some directions for future research in this area.

- *Distributed Implementation of the Algorithms to achieve Multipoint-to-Point Session Fair Allocations*

  In Chapter 4, we have presented centralized algorithms that achieve the proposed session-fairness definitions. We have also presented some preliminary thoughts on the implementation of these algorithms in a distributed manner. The design of protocols to implement these algorithms, and a detailed investigation thereof is a direction for future research in this area. The use of the eXplicit Control Protocol (XCP [50]) for implementing the normalized rate session fair algorithm and the use of our design of an Overlay-TCP network to implement the per-link session fair algorithm, as well as the normalized rate session fair algorithm are suggested starting points to think in this direction.

- *Extensions to the Design of Overlay-TCP Networks*

The Overlay-TCP network design can be investigated further in multiple ways.

The use of multiple parallel TCP connections on each overlay hop can be modified to incorporate proposals for sharing congestion information between the TCP connections. As an example, the Integrated Congestion Management architecture [6] can be used to share congestion information between the TCP connections on each overlay hop. Alternatively, the parallel TCP connections can use multiple paths between the same pair of end-nodes of an overlay hop.

This idea can be further generalized to using multiple overlay paths, between two overlay nodes. This can be designed as a hybrid Overlay-TCP-UDP network, where overlay nodes can either read the data from the incoming TCP connections and forward it to the outgoing TCP connections, or they can read incoming UDP data and do UDP or IP forwarding to the outgoing hop.

In the design of the Overlay-TCP network, we have shown that the sharing between multiple overlay networks can be controlled. However, the exact values of the maximum number of TCP connections that can be allowed on each overlay hop of each overlay network, in order to achieve a specific ratio of sharing is left to future work. This is a difficult problem, as it is a function of the competing traffic at the overlay layer, as well as the non-overlay traffic in the native network, and the properties of the native links, including the propagation delay and the capacity.

In this thesis, the implementation and evaluation of the design of a single path in an Overlay-TCP network has been done on PlanetLab nodes. However, the evaluation of the design of the Overlay-TCP network has been done using ns2 simulations. The Overlay-TCP network design can be implemented on Planetlab nodes and evaluated, as a follow-up of this thesis.

REFERENCES

[1] "Akamai." http://www.akamai.com/, December 2002.

[2] Amir, Y. and Danilov, C., "Reliable communication in overlay networks," in *International Conference on Dependable Systems and Networks*, June 2003.

[3] Anderson, D., Balakrishnan, H., Kaaoshek, F., and Morris, R., "Resilient overlay networks," in *Proceedings of the 18th Symposium on Operating Systems Principles*, October 2001.

[4] Apostolopoulos, J., Wong, T., Wee, S., and Tan, D., "On Multiple Description Streaming with Content Delivery Networks," in *Proceedings of IEEE INFOCOM*, June 2002.

[5] Baccelli, F., Chaintreau, A., Liu, Z., Riabov, A., and Sahu, S., "Scalability of reliable group communication using overlays," in *Proceedings of IEEE INFOCOM*, April 2004.

[6] Balakrishnan, H., Rahul, H., and Seshan, S., "An Integrated Congestion Management Architecture for Internet Hosts," in *Proceedings of ACM Sigcomm*, September 1999.

[7] Balakrishnan, H. and Seshan, S., "The Congestion Manager," Request for Comments 3124, Internet Engineering Task Force, June 2001.

[8] Banchs, A., "User Fair Queueing: Fair Allocation of Bandwidth for Users," in *Proceedings of IEEE INFOCOM*, June 2002.

[9] Banerjee, S., Bhattacharjee, S., and Komareddy, C., "Scalable application layer multicast," in *Proceedings of ACM SIGCOMM*, August 2002.

[10] Bertsekas, D. and Gallager, R., *Data Networks*. Englewood Cliffs, NJ, Prentice-Hall, 1992.

[11] Biliris, A., Cranor, C., Douglis, F., Rabinovich, M., Sibal, S., Spatscheck, O., and Sturm, W., "CDN brokering," in *Proceedings of the Sixth International Workshop on Web Caching and Content Distribution*, June 2001.

[12] "BitTorrent." http://www.bittorrent.com, July 2005.

[13] Byers, J., Considine, J., Mitzenmacher, M., and Rost, S., "Informed Content Delivery Across Adaptive Overlay Networks," in *Proceedings of ACM Sigcomm*, August 2002.

[14] Byers, J. W., Luby, M., Mitzenmacher, M., and Rege, A., "A digital fountain approach to reliable distribution of bulk data," in *Proceedings of ACM SIGCOMM*, pp. 56–67, September 1998.

[15] Calvert, K., Doar, M., and Zegura, E., "Modeling Internet Topology," *IEEE Communications Magazine*, June 1997.

[16] Castro, M., Druschel, P., Kermarrec, A.-M., Nandi, A., Rowstron, A., and Singh, A., "Splitstream: High-bandwidth multicast in a cooperative environment," in *19th ACM Symposium on Operating Systems Principles (SOSP)*, October 2003.

[17] "Centerspan." http://www.centerspan.com/, December 2002.

[18] Chawathe, Y., Ratnasamy, S., L.Breslau, Lanham, N., and Shenker, S., "Making Gnutella-like P2P Systems Scalable," in *SIGCOMM*, August 2003.

[19] Chu, J., Labonte, K., and Levine, B., "Availability and locality measurements of peer-to-peer file systems," in *Proceedings of ITCom: Scalability and TrafficControl in IP Networks*, 2002.

[20] Chu, Y., Rao, S. G., Seshan, S., and Zhang, H., "A case for end system multicast," *IEEE Journal on Selected Areas in Communication, Special Issue on Networking Support for Multicast*, 2002.

[21] Clarke, I., "Freenet." http://freenet.sourceforge.org, July 2005.

[22] "Digital island." http://www.digisle.net/, December 2002.

[23] "Digital fountain." http://www.digitalfountain.com/, December 2002.

[24] "eDonkey." http://www.edonkey.com, July 2005.

[25] Eggert, L., Heidemann, J., and Touch, J., "Effects of Ensemble-TCP," *ACM Computer Communication Review*, vol. 30, pp. 15–29, January 2000.

[26] Fahmy, S., Jain, R., Goyal, R., and Vandalore, B., "Fairness for ABR multipoint-to-point connections," in *Proceedings of SPIE Symposium on Voice, Video and Data Communications,*, November 1998.

[27] Ge, Z., Fiegueiredo, D. R., Jaiswal, S., Kurose, J., and Towsley, D., "Modeling peer-peer file sharing systems," in *Proceedings of IEEE INFOCOM*, April 2003.

[28] Gettys, J. and Nielsen, H., "The WEBMUX protocol, Internet Draft (work in progress)." http://www.w3.org/Protocols/MUX/WD-mux-980722.html, August 1998.

[29] Gkantsidis, C., Ammar, M., and Zegura, E., "On the effect of large-scale deployment of parallel downloading," in *Proceedings of the Third IEEE Workshop on Internet Applications*, June 2003.

[30] "Globule." http://www.globule.org/, July 2005.

[31] "Gnucleus." http://www.gnucleus.net/, December 2003.

[32] "Gnutella." http://gnutella.wego.com/, December 2003.

[33] "Gnutella Web Caching System." http://www.gnucleus.com/gwebcache/, December 2003.

[34] "Gtk-Gnutella." http://gtk-gnutella.sourceforge.net/, December 2003.

[35] Hacker, T., Athey, B., and Noble, B., "The End-to-End Performance Effects of Parallel TCP Sockets on a Lossy Wide-Area Network," in *International Parallel and Distributed Processing Symposium (IPDPS)*, April 2002.

[36] Han, H., Shakkottai, S., Hollot, C. V., Srikant, R., and Towsley, D., "Overlay tcp for multi-path routing and congestion control," in *IMA Workshop on Measurement and Modeling of the Internet*, January 2004.

[37] Handley, M., Floyd, S., Padhye, J., and Widmer, J., "TCP Friendly Rate Control (TFRC): Protocol Specification," Request for Comments 3448, Internet Engineering Task Force, January 2003.

[38] Handley, M., Schulzrinne, H., Schooler, E., and Rosenberg, J., "SIP: Session Initiation Protocol," Request for Comments 2543, Internet Engineering Task Force, March 1999.

[39] He, Q. and Ammar, M., "Congestion control and message loss in gnutella networks," in *Multimedia Computing and Networking*, January 2003.

[40] He, Q., Dovrolis, C., and Ammar, M., "On the predictability of large transfer tcp throughput," in *Proceedings of ACM SIGCOMM*, August 2005.

[41] He, Q., "Enhancement to the FullTcpAgent in ns2." http://www.cc.gatech.edu/computing/compass/gnutella/tcp.html, July 2005.

[42] Izal, M., Urvoy-Keller, G., Biersack, E., Felber, P., Hamra, A. A., and Garces-Erice, L., "Dissecting BitTorrent: Five Months in a Torrent's Lifetime," in *Proceedings of the 5th Passive and Active Measurement Workshop*, April 2004.

[43] Jannotti, J., Gifford, D. K., Johnson, K. L., Kaashoek, M. F., and O'Toole, Jr., J. W., "Overcast: Reliable multicasting with an overlay network," in *Proc. Symposium on Operating System Design and Implementation (OSDI)*, pp. 197–212, October 2000.

[44] J.E. van der Merwe and P. Gausman and C.D. Cranor and R. Akhmarov, "Design, Implementation and Operation of a Large Enterprise Content Distribution Network," in *Proceedings of the Eighth International Workshop on Web Content Caching and Distribution*, September 2003.

[45] Johnson, K. L., Carr, J. F., Day, M. S., and Kaashoek, M. F., "The measured performance of content distribution networks," in *Proceedings of the Fifth International Web Caching and Content Delivery Workshop*, May 2000.

[46] Karbhari, P., Ammar, M., Dhamdhere, A., Raj, H., Riley, G., and Zegura, E., "Bootstrapping in Gnutella: A Measurement Study," in *Proceedings of the 5th Passive and Active Measurement Workshop*, April 2004.

[47] Karbhari, P., Ammar, M., and Zegura, E., "Design of an Overlay-TCP Network with Throughput and Fairness Considerations," in *Submitted*, July 2005.

[48] Karbhari, P., Ammar, M., and Zegura, E., "Optimizing End-to-End Throughput for Data Transfers on an Overlay-TCP Path," in *Proceedings of IFIP Networking Conference*, May 2005.

[49] Karbhari, P., Zegura, E., and Ammar, M., "Multipoint-to-Point Session Fairness in the Internet," in *Proceedings of IEEE INFOCOM*, April 2003.

[50] Katabi, D., Handley, M., and Rohrs, C., "Congestion control for high bandwidth-delay product networks," in *Proceedings of ACM SIGCOMM*, August 2002.

[51] "Kazaa." http://www.kazaa.com, July 2005.

[52] Keralapura, R., Taft, N., Chuah, C., and Iannaccone, G., "Can ISPs take the heat from Overlay Networks?," in *Third Workshop on Hot Topics in Networks (HotNets)*, November 2004.

[53] Krishnamurthy, B. and Rexford, J., *Web Protocols and Practice*. Addison Wesley, 2001.

[54] Krishnamurthy, B., Wills, C., and Zhang, Y., "On the Use and Performance of Content Distribution Networks," in *Proceedings of ACM SIGCOMM Internet Measurement Workshop*, November 2001.

[55] Kubiatowicz, J., Bindel, D., Chen, Y., Czerwinski, S., Eaton, P., Geels, D., Gummadi, R., Rhea, S., Weatherspoon, H., Weimer, W., Wells, C., and Zhao, B., "Oceanstore: An architecture for global-scale persistent storage," in *Proceedings of the Ninth international Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, November 2000.

[56] Kwon, G. and Byers, J., "Roma: Reliable overlay multicast with loosely coupled tcp connections," in *Proceedings of IEEE INFOCOM*, April 2004.

[57] Lee, B., Balan, R., Jacob, L., Seah, W., and Ananda, A., "Avoiding Congestion Collapse on the Internet using TCP Tunnels," *Computer Networks*, vol. 39, no. 5, pp. 207–219, 2002.

[58] Leibowitz, N., Ripeanu, M., and Wierzbicki, A., "Deconstructing the kazaa network," in *Proceedings of the Third IEEE Workshop on Internet Applications*, June 2003.

[59] "LimeWire." http://www.limewire.com, December 2003.

[60] Liston, R. and Zegura, E., "Using a Proxy to Measure Client-Side Web Performance," in *Proceedings of the Sixth International Workshop on Web Caching and Content Distribution*, June 2001.

[61] Mogul, J., "The Case for Persistent-Connection HTTP," in *Proceedings of ACM Sigcomm*, August 1995.

[62] Moh, M. and Chen, Y., "Design and Evaluation of Multipoint-to-Point Multicast Flow Control," in *Proceedings of SPIE Symposium on Voice, Video and Data Communications,*, November 1998.

[63] "Mutella." http://mutella.sourceforge.net/, December 2003.

[64] Myers, A., Dinda, P., and Zhang, H., "Performance Characteristics of Mirror Servers on the Internet," in *Proceedings of IEEE INFOCOM*, March 1999.

[65] Ng, T. E., Chu, Y., Rao, S., Sripanidkulchai, K., and Zhang, H., "Measurement-based optimization techniques for bandwidth-demanding peer-to-peer systems," in *Proceedings of IEEE INFOCOM*, April 2003.

[66] "ns2- The Network Simulator." http://www.isi.edu/nsnam/ns, July 2005.

[67] Oram, A., *Peer-To-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly, 2001.

[68] Padhye, J., Firoiu, V., Towsley, D., and Kurose, J., "Modeling TCP Throughput: A Simple Model and its Empirical Validation," in *Proceedings of ACM Sigcomm*, September 1998.

[69] Padmanabhan, V., "Addressing the Challenges of Web Data Transport," September 1998.

[70] Padmanabhan, V., "Coordinating Congestion Management and Bandwidth Sharing for Heterogenous Data Streams," in *Proceedings of NOSSDAV*, June 1999.

[71] Padmanabhan, V. and Katz, R., "TCP fast start: a technique for speeding up web transfers," in *IEEE Globecom Internet Mini-Conference*, November 1998.

[72] Padmanabhan, V., Wang, H., Chou, P., and Sripanidkulchai, K., "Distributing Streaming Media Content Using Cooperative Networking," in *Proceedings of NOSSDAV*, May 2002.

[73] "Peergenius." http://www.peergenius.com/, December 2002.

[74] Peterson, L., Shenker, S., and Turner, J., "Overcoming the Internet Impasse through Virtualization," in *Third Workshop on Hot Topics in Networks (HotNets)*, November 2004.

[75] "PlanetLab." http://www.planet-lab.org/, November 2004.

[76] Postel, J., "User Datagram Protocol," Request for Comments 768, Internet Engineering Task Force, August 1980.

[77] Postel, J., "Transmission Control Protocol," Request for Comments 793, Internet Engineering Task Force, September 1981.

[78] R. Jain, *The Art of Computer Systems Performance Analysis*. John Wiley and Sons Inc., 1991.

[79] R. Keralapura and C. Chuah and N. Taft and G. Iannaccone, "Can coexisting overlays inadvertently step on each other?," in *Proceedings of IEEE International Conference on Network Protocols (ICNP)*, November 2005.

[80] Rabinovich, M. and Aggarwal, A., "RaDaR: a scalable architecture for a global Web hosting service," *Computer Networks*, vol. 31, no. 11–16, pp. 1545–1561, 1999.

[81] Ratnasamy, S., Francis, P., Handley, M., Karp, R., and Shenker, S., "A scalable content addressable network," in *Proceedings of ACM Sigcomm*, August 2001.

[82] Rodriguez, P., Kirpal, A., and Biersack, E., "Parallel-Access for Mirror Sites in the Internet," in *Proceedings of IEEE INFOCOM*, March 2000.

[83] Rodriguez, P., Sibal, S., and Spatscheck, O., "TPOT: Translucent Proxying for TCP," in *Proceedings of the Fifth International Web Caching and Content Delivery Workshop*, May 2000.

[84] Rowstron, A. and Druschel, P., "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," in *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pp. 329–350, November 2001.

[85] Rowstron, A. and Druschel, P., "Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility," in *18th ACM Symposium on Operating Systems Principles (SOSP)*, pp. 188–201, October 2001.

[86] Rubenstein, D., Kurose, J., and Towsley, D., "The Impact of Multicast Layering on Network Fairness," in *Proceedings of ACM Sigcomm*, September 1999.

[87] Saltzer, J., Reed., D., and Clark, D., "End-to-End Arguments in System Design," *ACM Transactions on Computer Systems*, November 1984.

[88] Saroiu, S., Gummadi, P., and Gribble, S., "A measurement study of peer-to-peer file sharing systems," in *Proceedings of Multimedia Computing and Networking*, January 2002.

[89] Savage, S., Anderson, T., Aggarwal, A., Becker, D., Cardwell, N., Collins, A., Hoffman, E., Snell, J., Vahdat, A., Voelker, G., and Zahorjan, J., "Detour: Informed Internet Routing and Transport," *IEEE Micro*, February 1999.

[90] Sen, S. and Wong, J., "Analyzing peer-to-peer traffic across large networks," in *Second Annual ACM Internet Measurement Workshop*, November 2002.

[91] Sivasubramanian, S., Szymaniak, M., Pierre, G., and van Steen, M., "Replication for web hosting systems," *ACM Computing Surveys*, vol. 36, no. 3, pp. 291–334, 2004.

[92] "Speedera." http://www.speedera.com/, December 2002.

[93] Spero, S., "Session Control Protocol." http://www.w3.org/Protocols/HTTP-NG/http-ng-scp.html, 1998.

[94] Sripanidkulchai, K., "The popularity of gnutella queries and its implications on scalability." http://www-2.cs.cmu.edu/ kunwadee/research/p2p/gnutella.html, 2001.

[95] Stephanos Androutsellis-Theotokis and Diomidis Spinellis, "A survey of peer-to-peer content distribution technologies," *ACM Computing Surveys*, vol. 36, no. 4, pp. 335–371, 2004.

[96] Stoica, I., M., R., Karger, D., Kaashoek, M. F., and Balakrishnan, H., "Chord: A scalable peer-to-peer lookup service for internet applications," in *Proceedings of ACM Sigcomm*, August 2001.

[97] Subramanian, L., Stoica, I., Balakrishnan, H., and Katz, R., "OverQos: An Overlay based Architecture for Enhancing Internet QoS," in *Symposium on Networked Systems Design and Implementation (NSDI)*, March 2004.

[98] Sundararaj, A. and Duchamp, D., "Analytical characterization of the throughput of a split tcp connection," in *Tech Report*, December 2003.

[99] Swany, M. and Wolski, R., "Improving throughput with cascaded tcp connections: the logistical session layer," in *UCSB Tech Report*, June 2002.

[100] Touch, J., "TCP Control Block Interdependence," Request for Comments 2140, Internet Engineering Task Force, April 1997.

[101] Touch, J., "Dynamic Internet Overlay Deployment and Management Using the X-Bone," *Computer Networks*, pp. 117–135, July 2001.

[102] "Ultrapeer Specifications." http://www.limewire.com/developer/Ultrapeers.html, December 2003.

[103] Urvoy-Keller, G. and Biersack, E., "A congestion control model for multicast overlay networks and its performance," in *Proceedings of the Fourth International Workshop on Networked Group Communication*, October 2002.

[104] "Windump." http://windump.polito.it/, December 2003.

[105] Zhao, B. Y., Huang, L., Stribling, J., Rhea, S. C., Joseph, A. D., and Kubiatowicz, J. D., "Tapestry: A Resilient Global-Scale Overlay for Service Deployment," *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 1, 2004.

[106] Zou, L. and Ammar, M., "A File-Centric Model for Peer-to-Peer File-Sharing Systems," in *IEEE International Conference on Network Protocols (ICNP)*, November 2003.

VITA

Pradnya Karbhari received her B.E. in Computer Engineering at the Veermata Jijabai Technological Institue (VJTI) in the University of Mumbai in 1998. She was in Texas A&M University for a year and moved to Georgia Institute of Technology in 1999. She completed her doctorate under the able guidance of Dr. Mostafa Ammar and Dr. Ellen Zegura on the topic of "Throughput and Fairness Considerations in Overlay Networks for Content Distribution". She has worked as a Summer Manager at AT&T Labs- Research with Dr. Misha Rabinovich in the summers of 2000 and 2001. Her research interests include overlay networks, peer-to-peer networks and content distribution networks.