Quality-consciousness in Large-scale Content Distribution in the Internet

A Thesis
Presented to
The Academic Faculty

by

Minaxi Gupta

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

College of Computing
Georgia Institute of Technology
July 2004

Quality-consciousness in Large-scale Content Distribution in the Internet

Approved by:

Professor Mostafa H. Ammar
(College of Computing), Advisor

Dr. Ellen W. Zegura
(College of Computing)

Dr. Constantinos Dovrolis
(College of Computing)

Dr. George F. Riley
(School of Electrical and Computer Engineering)

Dr. Jun (Jim) Xu
(College of Computing)

Date Approved: 23 July 2004

To the memory of Motu, whose 10 *short years with us enriched our lives and made us realize the importance of co-existing with other life forms on the planet.*

ACKNOWLEDGEMENTS

I want to begin by expressing my sincere gratitude for my advisor, Prof. Mostafa Ammar, for his guidance, support, and patience throughout my PhD. Mostafa provided me with the flexibility to choose projects and steered me in the right direction through valuable feedback and constructive criticism. He encouraged me in my career goals and devoted time while I went through the academic job hunt in a rather trying environment. I especially cherish the invaluable advice and support I got from Mostafa in the last years of my PhD. I always felt taken care of under his guidance and have learnt much from him.

My dissertation committee members, Constantinos Dovrolis, George Riley, Jim Xu, and Ellen Zegura have been very accommodating and have provided me with very insightful feedback. I also want to acknowledge Prof. Mustaque Ahamad and Prof. Mani Subramanian for their support throughout my years at Georgia Tech. I could not have made it this far without their faith in me. Prof. Ling Liu gave me very useful advice during my academic job hunt for which I am very thankful.

The Georgia Tech **N**etworking and **T**elecommunications **G**roup is the most congenial research environment I could have hoped for. I admire the skill-set available within the group and have leaned on the group's support structure on many occasions. I especially want to thank Amogh Dhamdhere, Sanjeev Dwivedi, Ruomei Gao, Christos Gkantsidis, Qi He, Pradnya Karbhari, Abhishek Kumar, Richard Liston, Shashidhar Merugu, and Sridhar Srinivasan for their friendship and for providing me with very useful feedback on my research on many occasions. All the members of the NTG group have played an important role in making my years at the GCATT building nostalgic.

My husband Arun is a big reason why I am writing this section of my dissertation today. He motivated and supported me throughout my PhD and when I was a fledgling graduate student in Physics. His faith in my abilities is what helped me discover my passion for computer science and networking. My parents have always provided me with unconditional

love and their dedication toward my career is unparalleled. I can only hope to be like them one day. My brother Gaurav and my in-laws have also been very encouraging and their pride in me is very important to me. My family is my biggest strength and a constant source of inspiration.

My friends have always been an important part of my life. I want to take this opportunity to thank Sameer Adhikari, Quynh Dinh, Bhavna Kumar, Bharathi Mani, Arnab Paul, Nilakshi Raut, Valeria Saponara, Reena and Rajul Soni for sharing their lives with me and giving me and opportunity to share mine with them. Last but not the least, I want to thank the College of Computing's support staff and the aerobics instructors at the Georgia Tech athletic complex for making my years at Georgia Tech fun and smooth.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Content distribution is the primary function of the Internet today. Technologies like multicast and peer-to-peer networks hold the potential to serve content to large populations in a scalable manner. While multicast provides an efficient transport mechanism for one-to-many and many-to-many delivery of data in an Internet environment, the peer-to-peer networks allow scalable content location and retrieval among large groups of users in the Internet.

Incorporating quality-consciousness in these technologies is necessary to enhance the overall experience of clients. This dissertation focuses on the architectures and mechanisms to enhance multicast and peer-to-peer content distribution through quality-consciousness. In particular, the following aspects of quality-consciousness are addressed: 1) client latency, 2) service differentiation, and 3) content quality.

Data analysis shows that the existing multicast scheduling algorithms behave unfairly when the access conditions for the popular files changes. They favor the popular files while penalizing the files whose access conditions have not changed. To maintain the **client latency** for all files under dynamic access conditions we develop a novel multicast scheduling algorithm that requires no change in server provisioning.

Service differentiation is a desirable functionality for both multicast and peer-to-peer networks. For multicast, we design a scalable and low overhead service differentiation architecture. For peer-to-peer networks, we focus on a protocol to provide different levels of service to peers based on their contributions in the system.

The ability to associate reliable reputations with peers in a peer-to-peer network is a useful feature of these networks. Reliable reputations can help establish trust in these networks and hence improve **content quality**. They can also be used as a substrate for a service differentiation scheme for these networks. This dissertation develops two methods of tracking peer reputations with varying degrees of reliability and overheads.

CHAPTER I

INTRODUCTION

The Internet of today primarily focuses on content distribution. The growth in the variety of content and the numbers of users accessing that content has led to the invention of many technologies for large-scale content distribution in the Internet. The focus of this dissertation are two such large-scale content distribution technologies: multicast and peer-to-peer (P2P) networks.

Multicast is capable of performing one-to-many and many-to-many delivery of data scalably and efficiently in an Internet environment. Even when as few as 20-40 receivers are simultaneously served by multicast, the bandwidth savings are estimated to be 60-70% compared to the unicast delivery of the same data to the same set of receivers [9].

P2P networks have revolutionized the concept of content distribution. In February 2003, a study of digital music behaviors by market research firm Ipsos [1] estimated the music file-sharers just within the U.S to be over 40 million. With each peer capable of being a server in addition to being a client, these networks have opened up new possibilities for large-scale content distribution without the need for dedicated centralized server infrastructure.

Incorporating quality-consciousness in both multicast and P2P networks is necessary to enhance the overall experience of the users of these technologies. This dissertation focuses on the architectures and mechanisms to enhance multicast and P2P content distribution through quality-consciousness. In particular, the following aspects of quality-consciousness are addressed: 1) client latency, 2) service differentiation, and 3) content quality.

- **Client Latency:** Analysis of server access logs revealed that the small percentage of files that account for the most load on the server exhibit a very dynamic popularity behavior. The existing multicast scheduling algorithms behave unfairly under variable

[1] http://www.ipsos-pa.com/dsp_d_us.cfm?id_to_view=1743

access conditions and favor the popular files while penalizing the files whose access conditions have not changed. To maintain the client latency for all files under dynamic access conditions we develop a novel multicast scheduling algorithm *Minimum Waiting Time (MWT)* that requires no change in server provisioning.

- **Service Differentiation:** Service differentiation is a desirable functionality for both multicast and P2P networks. To provide scalable service differentiation for multicast, we design an architecture comprised of two low overhead inter-operable limited branching techniques. The proposed techniques set up and tear down multicast state in the routers as new receivers join and leave while keeping the core of each domain simple. All the complexity of receiver join and leave is pushed to the edge routers.

  Due to the goodwill nature of P2P networks, incentives are necessary to motivate peers to contribute to the common good of the system. The promise of a better service to peers (through service differentiation) that contribute more could be a useful strategy. We design a protocol to provide different levels of service to peers based on their contributions in the system.

- **Content Quality:** The ability to associate reliable reputations with peers in a P2P network is a useful feature of these networks. Reliable reputations can help establish trust in these networks and hence improve content quality. They can also be used as a substrate for a service differentiation scheme for these networks.

  This dissertation develops two methods of tracking peer reputations with varying degrees of reliability and overheads. We first focus on a formally verifiable reputation tracking scheme. By compromising some reliability in reputation tracking, a more flexible and lower overhead scheme becomes feasible. We then develop a lower overhead reputation tracking scheme and discuss the reliability and overhead trade-offs in each of the schemes.

The remainder of this dissertation is organized as follows. In chapter 2 we review the related work. Chapter 3 discusses the performance results of the existing multicast scheduling algorithms and presents the *MWT* algorithm. Chapters 4 and 5 focus on service

differentiation for multicast and P2P networks respectively. In chapters 6 and 7, we present the two mechanisms for tracking reputations of peers in a peer-to-peer network that vary in overheads and reliability. Finally, chapter 8 gives a summary of the contributions of this dissertation and discusses the future research directions arising out of this work.

CHAPTER II

RELATED WORK

A significant amount of research is available on the technologies for large-scale content distribution. In describing the related work in the following sections, we focus only on the work closely related to the focus areas of this dissertation, namely, quality-consciousness in multicast and per-to-peer (P2P) networks. Specifically, we review the work related to client latency for multicast clients in section 2.1, work related to service differentiation for P2P networks and multicast in section 2.2, and that related to content quality in P2P networks section 2.3.

2.1 Client Latency

One of the factors affecting *client latency* for multimedia multicast clients is the manner in which the multicast server schedules files. The multicast server scheduling algorithm in chapter 3 is motivated by the observations about the popularity dynamics of multimedia files from server access log analysis. As a result, we split the related work for this section in two categories: 1) analysis of multimedia server access logs and 2) multicast server scheduling.

2.1.1 Analysis of Multimedia Access Logs

Acharya et al [2] characterized non-streaming multimedia content stored at the web servers. Authors in [3] present an analysis of a six-month trace data from the mMOD (multicast Media on Demand) system that had a mix of educational and entertainment videos. Both of these studies observed a high temporal locality of accesses, and that the rankings of the video titles by popularity did not fit a Zipf distribution.

The studies of client accesses to the audio content from the MANIC system [39] and low-bit rate videos from the Classroom2000 system [28] analyze the accesses to the educational media servers in terms of the daily variation in server loads, distribution of media session durations, and client interactivity.

With the goal of identifying important parameters for generating synthetic workloads, study [8] analyzes educational media server workloads for two media servers: eTeach and BIBS. The client arrivals in BIBS can be characterized as Poisson, and arrivals in eTeach workload are closer to a heavy-tailed Pareto distribution. The authors also observe that the media delivered per session depends on the file length.

In [16] Chesire et al analyze media *proxy* workload at a large university. The authors conclude that most of the multimedia sessions are less than 10 minutes long and that 78% of the media objects are accessed only once.

Workload analysis of enterprise media server workloads done in [15] found that a small percentage of newly introduced files constituted most of the accesses in any given month and that 45%-50% of the accesses to the most popular files occured during the first week of their introduction. This observation suggests that the small percentage of files that account for the most load on the server exhibit a very dynamic popularity behavior. We investigate this issue further in chapter 3 with access logs from Georgia Tech's web and FTP server logs.

2.1.2 Multicast Server Scheduling

The client latency depends in part on the scheduling of multimedia files at the multicast server. Many multicast scheduling algorithms are available in research literature. Dan et. al. [19] have proposed first come first served (FCFS), maximum queue length (MQL), and FCFS-n. In FCFS, when a channel becomes available, the server multicasts the stream to the client that has been waiting the longest. All the clients that have requests already queued for the same video also get served. In MQL, the video that has the most number of requests queued for it is selected when a channel becomes available. FCFS-n is similar to FCFS, except that n channels are pre-allocated for the most powerful videos. FCFS is fair since it always serves the client that has been waiting for the longest duration before others. MQL attempts to maximize the number of clients served by being biased for the more popular videos, but it does so at the expense of not being fair to the clients that request the unpopular videos. FCFS-n was not observed to improve the performance of

FCFS significantly and requires choosing an appropriate n.

MQL tends to be too aggressive in scheduling popular videos considering only the queue length, while FCFS completely ignores the queue length and focuses only on the arrival time to reduce defections. Aggarwal et. al. [6] proposed maximum factored queue length (MFQL), a scheduling policy with a notion of factored queue length. The factored queue length is obtained by weighting each video queue length with the square root of its popularity, a factor which is biased against the popular videos. The authors show that MFQL yields excellent empirical results in terms of standard performance measures such as average latency, reneging rates, and fairness.

With on-demand data broadcast in mind, Aksoy at. al. proposed RxW [7], a parameterized broadcast scheduling algorithm that makes scheduling decisions based on the current request queue and adapts well with client population and access pattern changes. At each scheduling decision, the RxW algorithm chooses to broadcast the page with the maximal $R * W$ value where R is the number of outstanding requests for a page and W is the time the the oldest outstanding request for that page has been waiting.

2.2 Service Differentiation

2.2.1 Service Differentiation for P2P networks

Service differentiation in P2P networks is a relatively untouched area of research. Just like ours, both the existing approaches in this area utilize the reputations of peers to provide service differentiation. Work in [33] proposes an admission control scheme for differentiating among the requests for content based on peer reputations. The decision criterion in this scheme is binary in that the requests for content are either admitted or denied. Detailed parameters impacting content search and download functionalities are not considered.

Widely deployed decentralized P2P software, Kazaa, uses the notion of *participation level* [32] in order to track peer reputations in the form of their contributions to the system. Kazaa defines a participation level for each peer based on the MBytes it transfers and the integrity of the files it serves. Each user rates the integrity of the files it downloads as *excellent*, *average*, *poor*, or *delete file*. Based on the ratio of Mbytes uploaded and

downloaded and the integrity rating of the files, the peers are assigned to three categories: *low*, *medium*, and *high*. The participation level score varies between 0 and 1000. A new user starts at a *medium* participation level of 100. The participation level score is used in prioritizing among peers during periods of high demand. Though this solution has very little overhead, it offers no security against selfish peers that know how to alter the part of their software that computes participation level. This approach is closest in spirit to the service differentiation protocol proposed in chapter 5 and the reputation systems proposed in chapters 6 and 7.

2.2.2 Service Differentiation for Multicast

IETF's differentiated services (DS) framework [10] proposes a service differentiation architecture to provide quality of service (QoS) for unicast communication. A DS domain is comprised of *boundary nodes* and *core nodes*. Boundary nodes interconnect the DS domain to other DS or non-DS capable domains while core nodes only connect to other core or boundary nodes within the same DS domain. Traffic enters a DS domain at an ingress node and leaves at an egress node.

The DS framework uses a six bit *DS field* from the IP header to define DS codepoints. All the packets with the same codepoint that cross a link in a particular direction form a behavior aggregate. The DS boundary nodes at the customer egress set the appropriate codepoint in each packet in accordance with the customers' service level agreement (SLA) and the packet joins the correct behavior aggregate. From this point on, subsequent boundary or core nodes in various DS domains have no information about a particular customer's flow, they only deal with behavior aggregates. This contributes significantly to the scalability of the architecture.

The DS framework is specified with unicast in mind. There has been some work in the direction of utilizing the DS framework for providing service differentiation for multicast. Bless and Wehrle [11] pointed out the challenges in using the existing DS architecture for multicast communication. They proposed to extend the multicast routing tables to include codepoints to provide QoS for multicast in the DS framework. This involves changing IP

multicast protocols.

Striegel and Manimaran [52] proposed an encapsulation-based approach called *DSMCast* for providing multicast support in a DS domain. Their approach consists of adding a DSMCast header to each packet at the edge of the DS domain by the ingress router. Upon receiving such a packet, a core router will inspect the packet to determine which interfaces the packet should be replicated on based on the information contained in the DSMCast header. This solution keeps the core routers simple but incurs bandwidth overhead for every multicast data packet. This approach is scalable in terms of the number of multicast groups, but not in terms of bandwidth overhead because the DSMCast header size is dependent on the number of receivers in each DS domain.

Our approach in chapter 4 is scalable both in terms of the number of multicast groups, as well in terms of the number of receivers. There are two kinds of overheads in both of our techniques: 1) signaling overhead to set up and tear down state in appropriate routers, a small *one-time* overhead incurred for each receiver join and leave and is independent of the duration of data flow and 2) bandwidth overhead in terms of extra packet hops incurred because of possibly moving the branching point, a topology dependent overhead. There is no bandwidth overhead in terms of extra headers in individual data packets.

2.3 Content Quality in P2P Networks

An approach to improving content quality in P2P networks is to track peer behavior and to map it to the reputation of the peer. This can help increase trustworthiness of entities and content in these networks, and hence improve content quality. Aberer and Despotovic [1] have proposed a *binary* trust model for P2P networks, i.e. a peer is either trustworthy or it is not. Assuming that maliciousness is an exception, the peers in this model only store information about their view of the malicious behavior of the peers they interact with. The overall trust is computed on the fly by querying appropriate peers. This system does not have any preventive measure against inserting arbitrary complaints about peers.

The proposal for tracking reputations in P2P networks by Demiani et. al. [18] involves keeping separate local repositories for resources and peers. They assume *binary* values

for each of the repositories. Peers update their local repositories for the resources and their offerers upon finishing transactions. The criteria for such updates are subjective. To compute trust values for resources and the peers on the fly using votes, they enhance the 2 phase Gnutella search and download protocol into a 5 phase protocol called *XRep*. The first phase of this protocol enhances the resource searching to include sending a *digest* of the resource. Upon selecting a resource, in the second phase the querying peer broadcasts other peers to find out the reputations of the offerers and their view of the resource. The third phase involves the evaluation of votes to judge the reputation of the resource and their offerers. In the fourth phase, the querying peer explicitly checks the selected peer to counter any spoofing attacks. The final phase is similar to the download in Gnutella. While this work addresses many security considerations for both P2P networks and the reputation system, it offers no incentive to the peers to participate in XRep. Moreover, reputations in this work are subjective and the inference involves on-demand computations that have high overheads and latency and introduce unknown amount of inaccuracy due to the high churn rates observed in these networks.

The Free Haven project [20] is a system of anonymous storage with goals of resisting powerful adversaries to find or destroy any stored data. For accountability purposes, they develop a centralized reputation system that attempts to limit the damage done by misbehaving servers. Each server locally keeps track of reputation and credibility values for all the other servers it trusts, along with the confidence ratings. Servers broadcast referrals in circumstances like when they log an honest completion of a trade, or when they suspect some data is lost, or when the reputation and credibility values for any server change considerably.

EigenRep [31] is a reputation management system for P2P networks. Each peer locally stores its own view of the reputation of the peers it does transactions with. The global reputation of each peer is computed by using the local reputation values assigned to it by other peers, but weighted by the global reputation of the assigning peers. This method of reputation inference rules out the possibility of malicious peers maligning the reputation of other peers.

NICE [35] is a platform for implementing cooperative distributed applications. Peers in NICE gain access to the remote resources by bartering local resources. The reputation in NICE is stored in the form of a *cookie* which can take real values in the $[0, 1]$ interval and is based on a peer's subjective satisfaction from the transaction. As against all the above reputation systems where the locally stored reputations were an indication of that peer's view of the credibility of the peers it had had transactions with, the locally stored reputations in NICE are an indication of the satisfaction of others peers that it served. The system does not assume peers will store cookies that have low values. Since the peers store their own reputations, no cooperation is required from other peers for storage purposes. However, to compute the reputations when needed, cooperation from the other peers in the system is a must, just like in the case of other reputation systems. Work in [21] explores similar ideas in order to form a rating system for P2P networks.

PeerTrust [56] is also a feedback based trust management system where peers quantify and compare the reputation of other peers. The trust for a peer in this system is also a non-decreasing scalar and is subjective but differs from the other reputation systems in that it is computed based on the three factors: 1) the amount of satisfaction received by the other peers in the system, 2) the total number of interactions, and 3) a balancing factor to offset the impact of malicious peers that misreport other peers' service. Each peer is mapped to maintain a small database that stores a portion of the global trust data. Though this reputation storage scheme differs from that in other reputation systems, it still requires cooperation from the peers for storing the reputations. Maliciousness is countered by having multiple peers responsible for storing the same portion of the database. Voting can be used if these databases differ. Trust is computed on the fly through querying potentially multiple databases over the network.

TrustMe [48] introduces the notion of anonymity in computing and storing peer reputations. Several other aspects of this system are similar to those of PeerTrust. For example, it requires cooperation from the peers for storing reputations of other peers.

Work by Vishnumurthy et. al. [55] comes closest in spirit to the reputation system proposed in this paper in that it uses a scalar value called $KARMA$ that tracks each peer's

resource consumption and contribution. KARMA's however in this system are not stored locally with the peer's but with a dedicated group of nodes.

CHAPTER III

A NOVEL MULTICAST SCHEDULING SCHEME FOR MULTICAST SERVERS WITH VARIABLE ACCESS PATTERNS

3.1 Introduction

Multicast communication accomplishes one-to-many and many-to-many delivery of data in an Internet environment. It is scalable and efficient because it outperforms unicast even for a small number of receivers. It has been observed in [9] that even for 20-40 receivers, multicast can be 60-70% more efficient than unicast in the Internet.

A multicast server accomplishes multicast of a video in essentially two phases: the first phase, *batch scheduling*, involves selecting a batch of requests for a particular video; the second phase, *channel allocation*, involves deciding how the channel should be allocated for the selected video.

There are two major approaches for channel allocation for unicast and multicast: *persistent* channel allocation, and channel *merging*. In persistent channel connection, once a channel is allocated to a video, it is used for multicasting the entire video to that batch of clients. Channel merging algorithms like the ones proposed in [29, 30, 5, 22] classify channels into *regular* channels and *patching* channels. The basic idea is to transmit part of the video on the patching channel and possibly merge the clients on to the persistent channel that is in the process of transmitting the same video. This can be done only if the clients are able to receive data at a higher bit rates. We assume persistent channel allocation for simplicity and focus on the performance of the batch scheduling schemes.

Various batch scheduling algorithms have been proposed in the literature. In first come first served (FCFS), when a channel becomes available, the server multicasts the stream to the client that has been waiting the longest. All the clients that have requests already

queued for the same video also get served. In maximum queue length (MQL [19]), the video that has the most number of requests queued for it is selected when a channel becomes available. Maximum factored queue length (MFQL [6]) is a batch scheduling policy with a notion of factored queue length. RxW [7] is a parameterized broadcast scheduling algorithm that makes scheduling decisions based on the current request queue and adapts well with client population and access pattern changes. It is important to note that all these batch scheduling schemes are designed for static server access patterns.

Studies of media server workloads [8, 14] show that the accesses to the server vary highly with time and the reason for this is the dynamic access patterns of a small number of popular videos. To test our conjecture that other kinds of servers may also experience variable number of accesses due to dynamic popularity of a small percentage of their popular files, we collected logs from a FTP and a web server. Analysis of these logs confirmed that the accesses to these servers are also highly variable.

The latency experienced by the multicast clients depends not only on the batch scheduling and channel allocation schemes but also on the server access pattern, network conditions, and the location of the receivers. Focusing on batch scheduling, our first goal is to evaluate the performance of the existing batch scheduling schemes under variable server access patterns. To that end, and motivated by the observations from the server logs, we generate three synthetic logs with different file popularity dynamics. For evaluating the performance of the existing batch scheduling algorithms on these synthetic logs we consider client latency and reneging of requests. While even during constant number of accesses to the server some batch scheduling schemes perform better than others, we observe that all of them degrade in performance when the accesses to the multicast server fluctuate over time. During the periods when a small percentage of popular files exhibit dynamic profiles all the schemes favor the dynamic files, giving them much lower client latency compared to the times when they have constant accesses. They do so at the cost of penalizing the less popular files that have not experienced a change in access patterns. Also, the reneging during periods of higher accesses is very high.

To correct this situation, our second goal is to develop a novel multicast scheduling

scheme called *M*inimum *W*aiting *T*ime (MWT) that provides lower client latencies to files that do not have dynamic profiles, while maintaining the response time for dynamic files. It is more fair compared to existing multicast scheduling schemes because it does not provide better performance to the popular files that have dynamic profiles. By trading the lower than average client latencies for the dynamic files for providing lower latencies for the files with static profiles MWT also reduces the reneging of requests, leading to better server resource utilization.

3.2 File Popularity Dynamics

3.2.1 Server Log Analysis

The study of educational media server workloads [8] found that there were very few periods of stationary relative file access frequency. This indicates that the accesses to the media servers fluctuate with time and that the media file popularity patterns are dynamic.

Analysis of enterprise media logs [14] observed that the file popularity for the media server workloads can be approximated by a Zipf-like distribution at varying time scales and that in any given month, total accesses to the media servers are dominated by the new files introduced in that month. Furthermore, approximately 50% of the accesses to any file occur in the first week of their introduction. These findings corroborate the observations of [8] in terms of fluctuation in the number of server accesses and the dynamic nature of the file popularity. The duration of server logs used in these studies varied from 1 month to 29 months, implying that these observations are not related to the duration of the logs. While the media servers used in both of these studies were not very busy servers (educational media servers in study [8] had 538 and 606 requests per day and enterprise media servers used in study [14] had 23 and 753 requests per day), they raise an important question about the file popularity dynamics. We ask the question if similar file popularity dynamics exist in the workloads of other kinds of more busy servers, i.e., FTP and web servers; implying that variable server accesses may indeed be a widely prevalent phenomenon.

To investigate this issue, we collected access logs from two unicast servers: *GT-FTP* (FTP logs collected from Georgia Tech's Linux Mirror site); and *GT-CoC* (HTTP logs

collected from Georgia Tech's College of Computing web server). Table 1 lists some of the characteristics of each of the logs. It shows that these logs vary from the server logs used in the multimedia studies [8, 14] in terms of duration, number of requests per day, and the number of unique files.

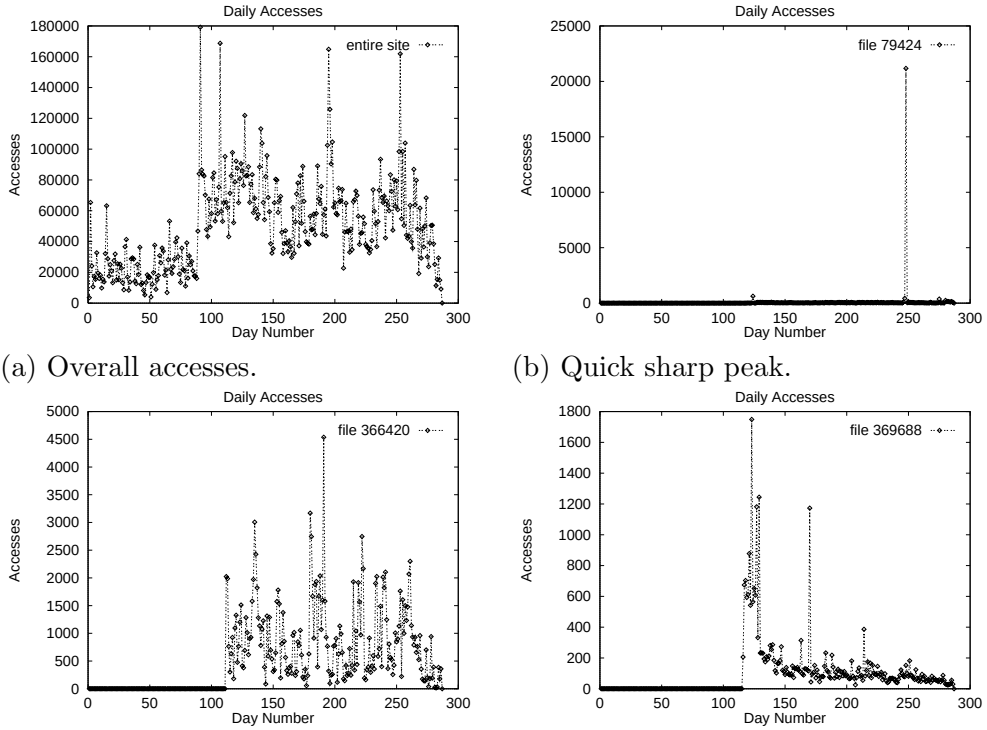
Table 1: Profile of log collection sites.

	GT-FTP	GT-CoC
Duration	12/22/00-10/23/01	4/30/01-10/23/01
Total Sessions	14,344,037	37,372,732
# Unique Clients	84,144	1,103,692
# Unique Files	676,315	492,084
Average File Size (in MBytes)	.737	.046
Median File Size (in MBytes)	.017	.0015

Table 1 shows that these logs vary from the server logs used in the multimedia studies [8, 14] in several ways. The duration of these logs, 10 months and 6 months falls in between that of the study in [8] (1 month and 3.5 months) and the study in [14] (21 months and 29 months). The number of requests that the FTP and web servers experience per day, $46,876$ and $211,145$ respectively, are much higher. The number of unique files accessed in FTP/web logs, $676,315/492,084$ respectively is also much higher than the 73, 1506 for study [8], and 2999, 412 for the study [14].

We experimented with various daily and overall (for the entire duration of the logs) access thresholds for individual files to investigate the access patterns of the most popular files in the above logs. Choosing the daily access thresholds of $4,000$ for GT-FTP, and $15,000$ for GT-CoC; and overall access thresholds of $15,000$ for GT-FTP and $150,000$ for GT-CoC narrowed down the number of popular files to 13 (.002% of total) for GT-FTP and 23 (.005% of total) for GT-CoC. The graphs in figure 1 show the overall access pattern at GT-FTP and the three types of access patterns that the most popular files exhibited. 6 out of the 13 most popular files peaked up in popularity for a very short time (see figure 1(b) for an example) and then were not accessed throughout the logs, 4 files had a modest profile (see figure 1(c) for an example), and 3 files peaked up quickly in popularity and then there

was a slow decay (see figure 1(d) for an example). As the graph of the overall accesses indicates, the FTP server gained additional popularity into the $100th$ day into the logs, hence the most popular files belonged to the latter part of the logs.



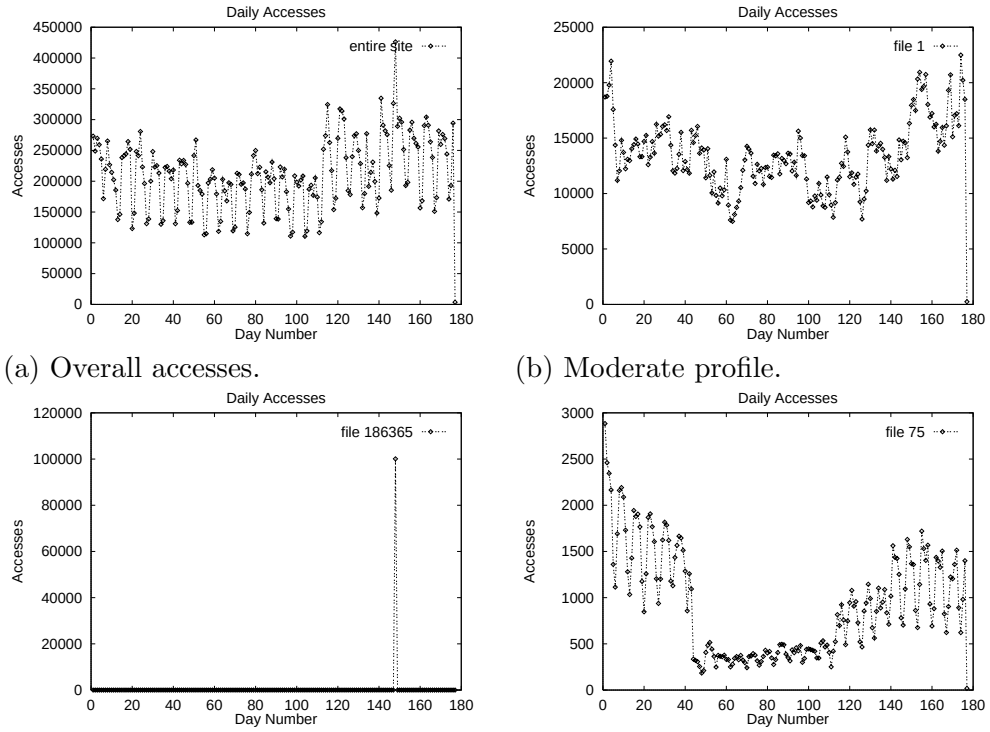
(a) Overall accesses. (b) Quick sharp peak.

(c) Moderate profile. (d) Sharp rise, slow decay.

Figure 1: FTP server access patterns.

Analysis of file access dynamics for GT-CoC led to different file dynamics. The graphs in figure 2 show the overall access pattern at GT-WWW and the three types of access patterns that the most popular files exhibited. 20 out of the 23 most popular files exhibited a very modest profile (see figure 2(b) for an example), 2 files sharply peaked in popularity for a short time and then were not accessed again for the duration of the logs (see figure 2(c) for an example), and only 1 file showed a popularity pattern consisting of multiple popularity plateaus (see figure 2(d)).

While the mix of popular files with modest and other kinds of profiles may be site dependent, the above analysis leads to the conclusion that dynamic file popularity is indeed a very pervasive phenomenon.



(a) Overall accesses.

(b) Moderate profile.

(c) Quick sharp peak.

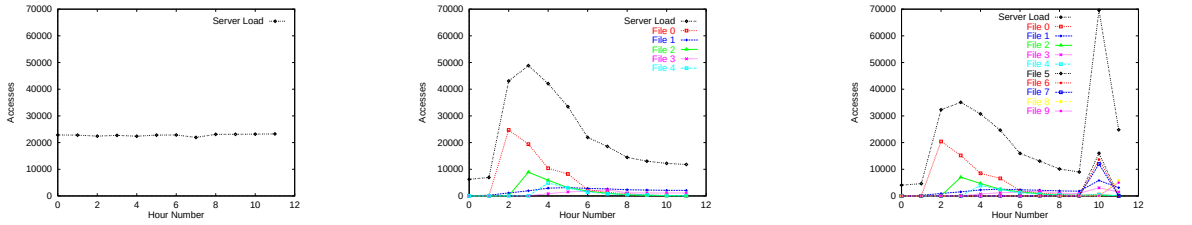
(d) Multiple plateaus.

Figure 2: WWW server access patterns.

3.2.2 Synthetic Logs

To test the impact of file popularity dynamics, we generated three synthetic logs, 12 hour each, for a multicast server that serves 100 files whose sizes range from 0 to 5MBytes. The ideas was to have shorter duration logs with the same characteristics in terms of file access patterns as that of longer duration logs, only compressed in time. These logs are Zipf distributed with a parameter of 1.0. The exponentially distributed request arrival rate for these logs is 350 requests per minute. The requests for files in the Ist log stay constant on an hourly basis. 95% of the files in the IInd log have a constant access pattern throughout the logs but the 5% most popular files in the Zipf distribution (numbered 0 through 4) exhibit a dynamic profile, fluctuating the server accesses. In the IIIrd log, the top 10% of the popular files (numbered 0 through 9) exhibit a dynamic profile, forming two peaks in activity. In each peak of activity, 5% of the files participate. Also, the manner in which the 10% popular files exhibit dynamic profiles is different. The files in the first peak of

activity rise gradually in popularity and either continue to be popular or gradually decrease in popularity. This peak in server accesses occurs from the 2nd hour until the 6th. On the other hand, the files in the second peak quickly rise in popularity at the 10th hour and the popularity subsides just as quickly. The graphs in figure 3 show the hourly server accesses and the hourly accesses for the individual popular files exhibiting a dynamic profile. An important point to note about these logs is that all the three logs have the same total load on the multicast server in the 12 hour duration. This is important to study the impact of dynamic file profiles on the multicast batch scheduling.



Ist Log (0% Popular Files) IInd Log (5% Popular Files) IIIrd Log (10% Popular Files)

Figure 3: Synthetic workloads for server and popular files.

3.3 Evaluation of Multicast Scheduling Schemes

We begin by reviewing various batch scheduling algorithms proposed in the literature. Dan et. al. [19] proposed first come first served (FCFS), maximum queue length (MQL), and FCFS-n as the batching policies. In FCFS, when a channel becomes available, the server multicasts the stream to the client that has been waiting the longest. All the clients that have requests already queued for the same video also get served. In MQL, the video that has the most number of requests queued for it is selected when a channel becomes available. FCFS-n is similar to FCFS, except that n channels are pre-allocated for the most powerful videos. FCFS is fair since it serves the client that has been waiting for the longest duration. MQL attempts to maximize the number of clients served by being biased for the more popular videos, but it does so at the expense of fairness to the clients that request the unpopular videos. FCFS-n was not observed to improve the performance of FCFS significantly and requires choosing an appropriate n.

MQL tends to be too aggressive in scheduling popular videos considering only the queue length, while FCFS completely ignores the queue length and focuses only on arrival time to reduce defections. Aggarwal et. al. [6] proposed maximum factored queue length (MFQL), a batch scheduling policy with a notion of factored queue length. The factored queue length is obtained by weighting the queue length of each video with the square root of its popularity, a factor which is biased against the popular videos. The authors show that MFQL yields excellent empirical results in terms of standard performance measures such as average latency, reneging rates, and fairness. With on-demand data broadcast in mind, Aksoy at. al. proposed RxW [7], a parameterized broadcast scheduling algorithm that makes scheduling decisions based on the current request queue and adapts well with client population and access pattern changes. At each scheduling decision, the RxW algorithm chooses to broadcast the video with the maximal $R * W$ value where R is the number of outstanding requests for a video and W is the time the the oldest outstanding request for that video has been waiting.

Next, we evaluate the performance of FCFS, MQL, MFQL, and RxW batch scheduling schemes under dynamic file popularity using the synthetic logs we generated in section 3.2.2. The metrics we consider are client latency and client reneging. Client latency can be defined in a variety of ways. One can define it to be the time it takes for the client to start receiving the file from the time of its request. Varying network conditions would make it hard to judge the impact of file popularity dynamics on each of the scheduling policies. Moreover, it is hard to simulate real world network conditions. Hence, we decided not to take into account the network conditions while finding the client latency. We define *client latency* to be the time a client request spends waiting for the server. It is the time from the point the server receives a request to the point it starts serving this request. After the request has been waiting in the queue for some amount of time, the client may decide to go away without waiting for the delivery of the file. The amount of time after which the client does so is called the *reneging time*.

For our event driven simulations, we considered a multicast server with 100 transmission channels and a transmission rate of $100kbps$. For each of the logs, we studied the average

hourly client latency for all the files, just the popular files that are causing the increase in server accesses (dynamic files), and the rest of the files.

Recall that the MFQL batch scheduling algorithm requires the square root of file popularity in addition to the queue length. Since the file popularity of some of the files changes dynamically, we decided to experiment with various values of smoothing parameter α in the following standard moving weighted average equation:

$$f_{new} = \alpha * f_{old} + (1 - \alpha) * f_{sample}$$

f_{old} in the above equation represents the previously estimated popularity for a particular file, f_{sample} represents the number of requests for this file in this estimation interval, and f_{new} is the new popularity value for this file. We experimented with $\alpha = 0.1, 0.5, 0.8$ for two different estimation intervals. The first interval was 50 minutes, meaning that we estimated the popularity of each file every 50 minutes. We chose this interval to ensure the misalignment of the popularity estimation interval with increase in file popularity in the synthetic logs. The second interval we experimented with was 8 minutes. It turns out that $\alpha = 0.5$ and popularity estimation interval of 50 minutes produced the best results and ways of popularity estimation do not differ significantly from each other. We also experimented with *static* MFQL (static-MFQL) where the Zipf probabilities were used as static popularity values for the simulations (we refer to this case as the *static-MFQL*). In general such information is not likely to be available ahead of time.

The graphs in figure 4 show the average hourly client latency and reneging for MQL, MFQL (50 minute popularity estimation interval and $\alpha = 0.5$), *static-MFQL*, and RxW for the case when the load on the server is Zipf distributed but constant (basically, using Ist synthetic log). We eliminate presenting FCFS results because FCFS performs poorly on both counts. We tested the schemes for three reneging times: 1 minute, 2 minutes, and 5 minutes.

As expected, MFQL (both static-MFQL and when $\alpha = 0.5$ and popularity estimation interval 50 minutes) outperforms all other scheduling schemes in terms of providing the

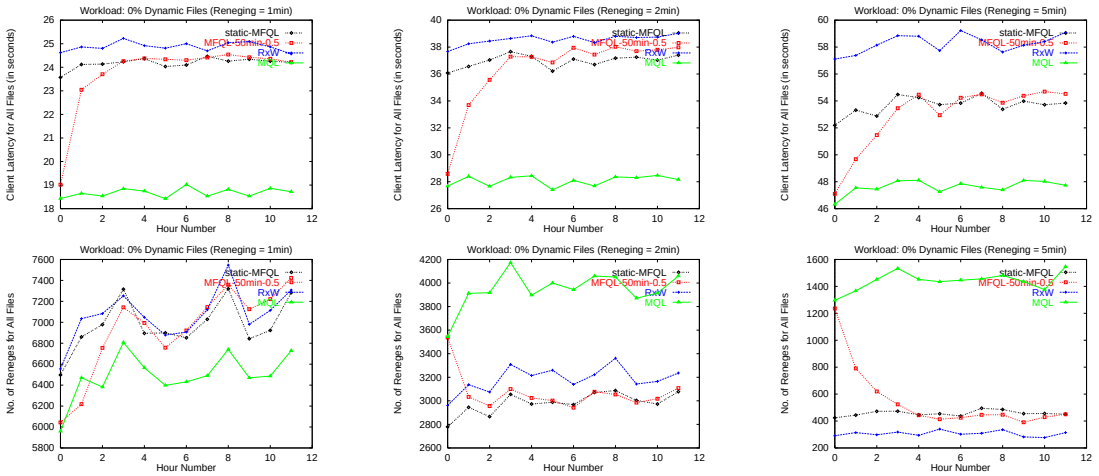


Figure 4: Average client latency and reneging for all files combined for synthetic log I.

clients with lower access latency and in terms of the number of clients reneging every hour. We then tested the performance of all the above schemes for IInd and IIIrd logs, where the most popular 5% and 10% of files have dynamic access profiles. The graphs in figure 5 show the average hourly client latency and reneging for all the files, just the top 5% dynamic files, and the rest of the 95% of the files for the IInd synthetic log when the reneging happens after 2 minutes of waiting. The results for log III and other reneging times were similar.

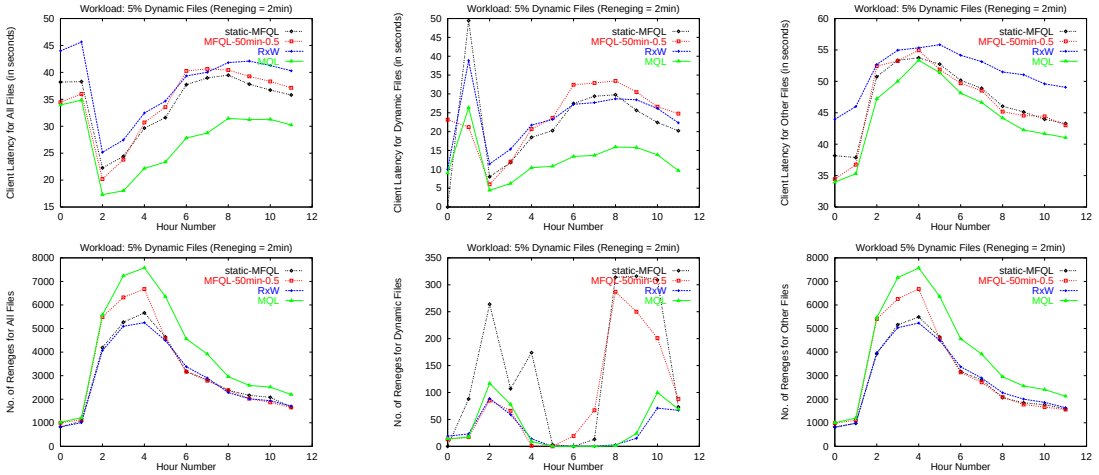


Figure 5: Average client latency and reneging for synthetic log II.

As figures 5 show, MQL performs the worst among all the schemes in terms of graceful degradation when the accesses to the server is increased because of temporary increase in

the number of accesses to the most popular 5% of the files. Even though the average client latency for all the files combined from hours 2 through 6 is lower than the case of Ist log (when the access patterns for all files were constant), it comes at the cost of penalizing the rest of the 95% of the files, whose access patterns are constant throughout the logs. The top 5% of the dynamic files get much better performance for all the schemes. Reneging during the peak access period also goes up considerably and even the best performing schemes (RxW and static-MFQL) experience about 50% more reneging for above graphs compared to the constant access case. Once again, most of the reneging happens for the 95% of the files and the top 5% popular files fair much better.

3.4 Minimum Waiting Time Scheduling Scheme

Next, we propose a new multicast batch scheduling scheme called MWT (*M*inimum *W*aiting *T*ime) for the multicast servers that experience variable number of accesses. MWT is designed with the following objective.

Objective: To provide similar average client latency to all files, irrespective of their popularity under all access conditions, while keeping the reneging to a minimum.

To illustrate the objective, we observe that during the period the server is experiencing higher number of accesses:

- a file f_i experiencing a surge in demand should get similar client latency to a file f_j whose accesses have not changed over time.
- the client latency of any file f_i should be the same as it would have been when the server accesses were constant.
- the system reneging should be as close as possible to that experienced when the access pattern is constant.

MWT attempts to be as fair in terms of which files experience reneging, as is important to reduce the number of requests that experience reneging. It tries to reduce the reneging of requests for files whose demands have not changed, at the same time not increasing the reneging of files contributing to the heavy access conditions much. It accomplishes this goal

by leveraging the fact that the files contributing to the heavy access conditions experience lower client latencies during peak access conditions than when their demand patterns are constant.

Since MFQL exhibits the best performance under constant access conditions and is known to be fair, during the periods when a multicast server is experiencing constant accesses for all its files according to their Zipf popularity, MWT behaves just like MFQL. When the accesses for the popular files increase, causing the accesses to the multicast server to increase as well, MWT gives priority to the files not experiencing a variation in their access pattern. If the first request for any of the dynamic files has been waiting for less time than a factor of the reneging time MWT schedules the files whose access pattern has not changed instead. The factor that decides the minimum waiting time for the dynamic files is called the *min wait factor*. Figures 6 show the pseudo-code for how MWT chooses one of the two files f_i and f_j to schedule an available channel under heavy access conditions:

```
if (f_i == dynamic_file and f_j != dynamic_file) {
   if (f_i->first_request_wait_time <
       reneging_time/min_wait_factor) {
      schedule f_j
   } else {
      do as MFQL would
   } } else {
   do as MFQL would
}
```

Figure 6: MWT algorithm for heavy access conditions.

For reneging times of 1 minute, 2 minutes, and 5 minutes, we experimented with various *min wait factors* (4, 3, 2, 1.5, 1) for synthetic logs II and III. The *min wait factor* that provides the best results depends on the reneging time of the file. We assume that all the files have the same reneging time in one run. Assuming that the multicast server has the knowledge about reneging times it can use an appropriate wait factor to increase the fairness among the files in terms of client latency and increase system utilization by reducing reneging. The graphs in figures 7 and 8 compare the performance of MWT to static-MFQL and show the results for the case when the reneging time is 2 minutes for synthetic logs II and III.

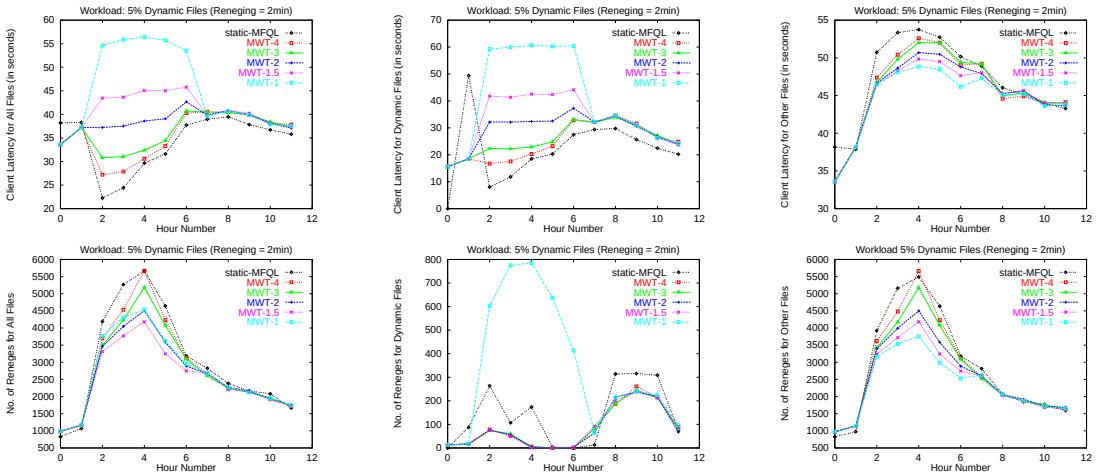


Figure 7: MWT for Synthetic Log II (Reneging=2min)

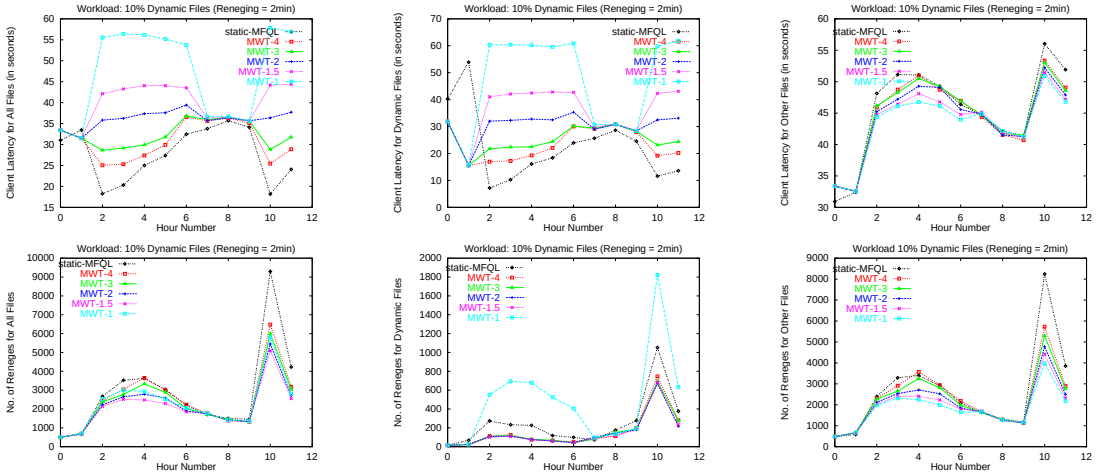


Figure 8: MWT for synthetic log III (reneging=2min).

As the graphs in figures 7 show, the average client latency for all files for a *min wait factor* of 2 comes close to the average latency for the constant access case of log I, as shown in figure 4. Although the average client latency for the 95% of the files is better for *min wait factor* of 1.5 and 1, it is not desirable, because the benefit comes at the cost of significantly degrading the average client latency for the 5% dynamic files, impacting the overall latency as well. The overall reneging as well as the reneging for the 5% dynamic and the rest of the 95% files are significantly better than static-MFQL for both *min wait factors* of 2 and 1.5. However, when the *min wait factor* decreases to 1 (causing the first request of 5% files to

wait at least the same amount as the reneging time) the overall reneging increases because of the reneging in the requests for popular files.

The results for log III are similar. This log has two peaks, one between the hours of 2 and 6, and another sharp one at the 10th hour. The first peak is contributed by the top 5% most popular files and the second one by the next 5% most popular files. The best *min wait factor* for log III is also 2. As the graphs in figures 8 show, the benefits of using MWT are more pronounced for log III in terms of decrease in the reneging of requests for the second peak.

The results of experimenting with various *min wait factors* for other reneging times (1 minute and 5 minutes) for both synthetic logs II and III were similar. In general, the bigger the reneging time, the smaller the *min wait factor* required (implying larger the average delay in scheduling the dynamic files) by MWT.

3.5 Conclusion

This work makes two main contributions. First, using the logs from FTP and web logs it recognizes the fact that the accesses to multicast servers vary significantly over time. This is primarily due a small percentage of popular files that exhibit dynamic profiles. Second, using synthetic workloads it concludes that the existing multicast schemes do not perform well under variable access patterns. To alleviate that situation, this work develops a novel multicast scheduling scheme, MWT. MWT keeps the average client latency for all files during periods of heavy accesses similar to that during constant server access conditions. It trades the lower than average latency for the dynamic files during peak access conditions for better response time to the files whose request patterns remain unchanged over time. It also reduces the reneging of requests for all files compared to MFQL.

CHAPTER IV

LIMITED BRANCHING TECHNIQUES FOR PROVIDING MULTICAST COMMUNICATION IN A DIFFERENTIATED SERVICES NETWORK

4.1 Introduction

The differentiated Services (DS) architecture [10] scalably implements service differentiation for unicast communication in the Internet. Due to the dynamic join/leave of multiple potentially heterogeneous receivers unique challenges stand in the way of providing service differentiation for multicast utilizing the scalable DS paradigm. This is because when new receivers join the multicast group, branches may get added to the existing multicast tree without prior resource allocation and this can adversely affect the unicast and multicast traffic for which resources have previously been reserved.

This chapter proposes a scalable architecture called M-DS, to provide QoS for multicast communication utilizing the DS framework. The architecture allows each Internet domain to choose between the two proposed limited branching techniques, namely, *edge-router branching* and *limited-core branching*. For each of the techniques, we define two signaling protocols; one for resource allocation on membership discovery and another for resource deallocation on membership termination on subnetworks. The signaling for resource allocation configures state in appropriate routers to be used during multicast data transmission. After this phase, data can begin to flow for multicast with QoS, as it would be in the case of DS for unicast. The signaling for resource deallocation resets the configuration changes made by the signaling for resource allocation. For both the techniques, flows are aggregated for scalability, just as in the DS framework for unicast. Also, both the techniques use the multicast state already set up in individual domains for routing packets and inter-operate with each other. Not changing multicast tables also facilitates

the co-existence of IP multicast without the QoS requirements. These techniques have low message overhead during signaling and no per-packet overhead during data transmission in terms of extra headers in individual data packets. However, they require changes in the routers to accommodate these techniques in a DS framework.

4.2 Differentiated Services (DS) Architecture

A DS domain is comprised of *boundary nodes* and *core nodes*. Boundary nodes interconnect the DS domain to other DS or non-DS capable domains while core nodes only connect to other core or boundary nodes within the same DS domain. Traffic enters a DS domain at an ingress node and leaves at an egress node. Figure 9 shows a DS domain $DS1$, boundary nodes $B1$ and $B2$, and core nodes $C1$, $C2$, and $C3$. $B2$ connects $DS1$ to a second domain $DS2$.

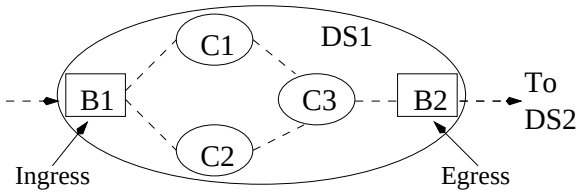


Figure 9: Various entities in a DS domain.

The DS framework uses a six bit *DS field* from the IP header to define DS codepoints. All the packets with the same codepoint that cross a link in a particular direction form a *behavior aggregate*. The DS boundary nodes at the customer egress set the appropriate codepoint in each packet in accordance with the customers' service level agreement (SLA) and the packet joins the correct behavior aggregate. From this point on, subsequent boundary or core nodes in various DS domains have no information about a particular customer's flow, they only deal with behavior aggregates. This contributes significantly to the scalability of the architecture. The DS architecture specifies various components needed to treat packets belonging to various behavior aggregates at the boundary nodes of the DS domains. Traffic classifiers separate submitted traffic into different classes. The packets from different classes matching some specified rule are subjected to metering, shaping, policing and/or remarking

to ensure that traffic entering a DS domain conforms to the SLA. Traffic forwarding in the core DS nodes is very simple; the core nodes apply the appropriate per hop behavior to each behavior aggregate.

4.3 Challenges

Providing quality of service (QoS) for multicast communication using the DS framework poses unique challenges. Multicast sources generally do not know the identity of receivers. The source sends out one copy of the data and the IP layer multicast routers make duplicate copies where needed in the network to reach all receivers of the group. Also, group membership in multicast is dynamic and the receivers are heterogeneous. Out of these issues, heterogeneity is not a stumbling block because at the application layer, receivers with different resource requirements can join different multicast groups [36], so within a particular multicast group, the resource requirements are homogeneous. However, dynamic group membership makes it challenging to provide QoS for multicast communication.

The DS architecture for unicast can not be used as is to accomplish QoS for multicast without affecting other traffic adversely. The main reason for this is because scalability in the DS architecture for unicast is achieved by distinguishing between the functionality of core and boundary routers in each domain and by traffic aggregation. Except for the routers near the source, all routers along the way deal only with behavior aggregates in the DS architecture. In multicast, however, new members may join a multicast group dynamically and as a result, several routers in the core of the network may duplicate packets to reach the new receivers. Keeping the core routers simple to preserve the DS scalability would imply core routers assign the same codepoint to the duplicated packets as to the original packets. As a result, new branches can get added to the existing multicast tree without prior resource reservation. This problem is termed as *non-reservation subtree (NRS) problem* in [11]. NRS can potentially lead to violation of SLAs between the DS peers and hence compromises QoS for one or more classes of traffic.

4.4 Architecture Components and Assumptions

An example of the various entities of the M-DS architecture is shown in figure 10. It shows two DS domains, with the multicast source attached to domain $DS1$ and receivers $R1$ and $R2$ on the same subnetwork attached to domain $DS2$ through the designated router (DR). Each domain has boundary and core routers. The figure also shows the bandwidth brokers $BB1$, and $BB2$, for domains $DS1$ and $DS2$ respectively. The DRs initiate the signaling with the BB of their domain upon each multicast group membership discovery and termination on their respective subnetworks. The bandwidth brokers (BB) handle resource allocation and deallocation requests in their domain by contacting their peers.

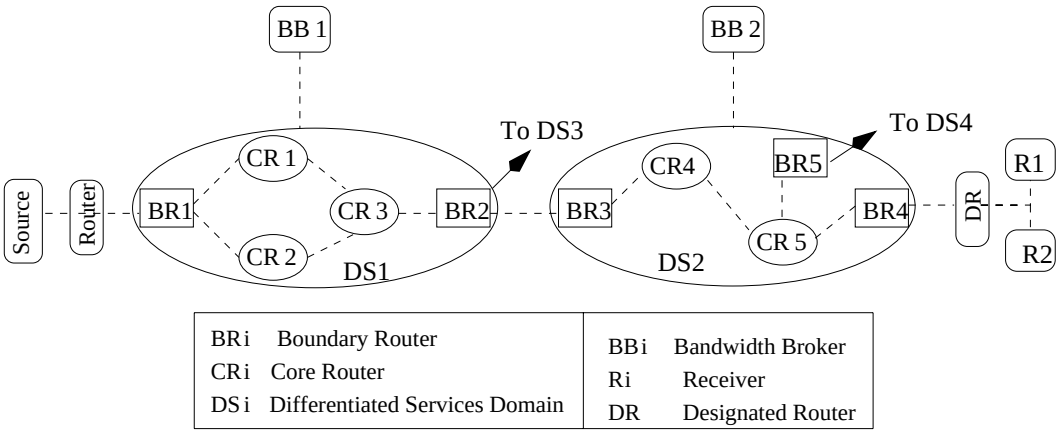


Figure 10: Components of the M-DS architecture.

The architecture makes the following assumptions about the infrastructure. First, each DS domain has static unicast SLAs for aggregates of flows with its peer domains for all its ingress and egress router pairs. Admission requests for both unicast and multicast are handled dynamically by the BB of each domain by contacting the BBs in peer domains. These assumptions are consistent with those of the *QBone* BB work group [53]. Second, each BB is configured as a BGP-4 router and hence has TCP connections for communicating with its peer BBs and the DRs of its domains for communication during signaling. It also has access to unicast and multicast routing information. Third, each BB has a data repository containing router configurations and policy information. It needs this to be able to make decisions to allocate and deallocate resources. Fourth, sources register with the BB of their

domain before sending data. The BBs propagate this information to peer BBs to inform about the active multicast sources.[1] Fifth, we assume that all receivers know what QoS to ask for. The resource request can be carried in a manner similar to the one prescribed by the RSVP specification [12].

Next, we describe the signaling protocols for the join and leave of multicast receivers in edge-router branching technique.

4.5 Edge-router Branching

Edge-router branching uses existing unicast SLAs and exploits multicast scalability at the domain granularity because it allows branching to occur only at the ingress and egress routers. If there are any branching points in the core of the domains, they are moved to the ingress of the domain. This keeps all the complexity confined to the edge of the network and hence the core nodes are kept scalable as in the case of DS for unicast.

4.5.1 Signaling Protocol for Admission

Signaling is initiated by the DRs of the domains when they discover a multicast group in their subnetwork. The DR sends a message to the BB of its group, giving it its own IP address, the QoS requirements and the multicast group address G. Subsequently, this BB may contact its peer BB to convey this information and so on. Notice that for each subsequent receiver joining the same subnetwork served by this DR, no action needs to be taken as long as the QoS requirements and G are the same.

The BB extracts the QoS information and G and then finds out if the resources for this request have already been allocated in its domain. Because if they are, no new resources are to be allocated. The BB needs two pieces of information to make that decision.

1. Appropriate entries from the *current resource allocation table*: this table contains resource allocation information for each pair of ingress and egress routers in the domain along with the multicast groups they serve. It also contains the number of different

[1] This is essentially similar to how MSDP [38] requires the RPs [23] in each domain to advertise active sources.

subnetworks (identified by the IP addresses of the DRs) each ingress-egress router pair serves, directly or indirectly. Keeping the number of the subnetworks served corresponding to each ingress-egress router pairs helps the BB know when to deallocate the resources in its domain. Entries in this table are filled after making a decision that resources can be granted. To find out the appropriate entry, the BB first finds out the pair of ingress and egress routers in its domain in the path from the multicast source for group G to the DR. It does that using the routing information that it has access to as a BGP-4 router.

2. If the *branching point* for G lies in its domain: while the signaling protocol is in progress, multicast state is being set up in the domain using the IP multicast protocol in use in that domain. The core routers in every domain that determine that they are going to be the branching point for group G communicate this information to their BB by sending the corresponding multicast forwarding table entry. The BB sets a timer to get this information from the branching point router(s) in its domain. If the timer expires without the BB getting a reply, it assumes that no branching point exists in its domain.

Based on the above information, three cases arise for the ingress-egress router pair needed to satisfy this request, as outlined in figure 11.

4.5.1.1 Case 1: Allocation Exists

This case arises if the BB finds an entry in the current resource allocation table with the requested QoS corresponding to the ingress and egress routers needed to serve the new request for group G. This implies that adequate resources have already been allocated. Nothing needs to be done other than updating the number of subnetworks served by this ingress-egress router pair in the current resource allocation table and the BB replies in the path to the DR affirmatively and signaling is terminated. The new receivers are grafted to the existing multicast tree and since multicast state is already set up, they can start getting data.

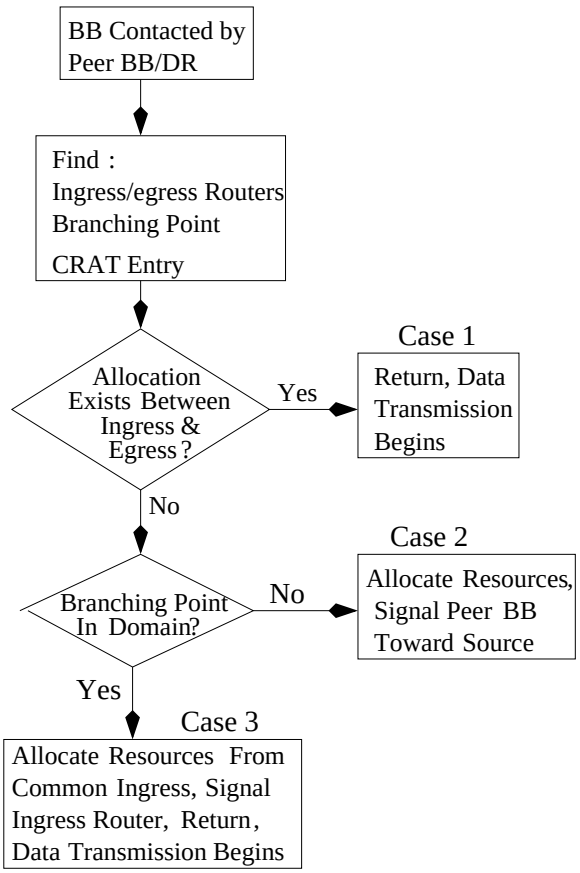


Figure 11: Signaling steps in edge-router branching technique upon membership discovery.

Figure 12 depicts the signaling for the case when $R1$ is present and $R2$ wants to join group G. The signaling for this case returns from $DS2$ and does not need to go all the way to domain $DS1$.

4.5.1.2 Case 2: No Allocation, No Branching Point

The second case arises if no resources have been allocated for G in this domain with the requested QoS. In this case, the BB does not find any entry with the requested QoS in the current resource allocation table corresponding to the ingress and egress routers needed to serve the new request. Also, no branching point router replies to the BB. The BB makes a decision to grant resources in the form of an SLA between the ingress-egress router pair based on policy information and SLA availability.

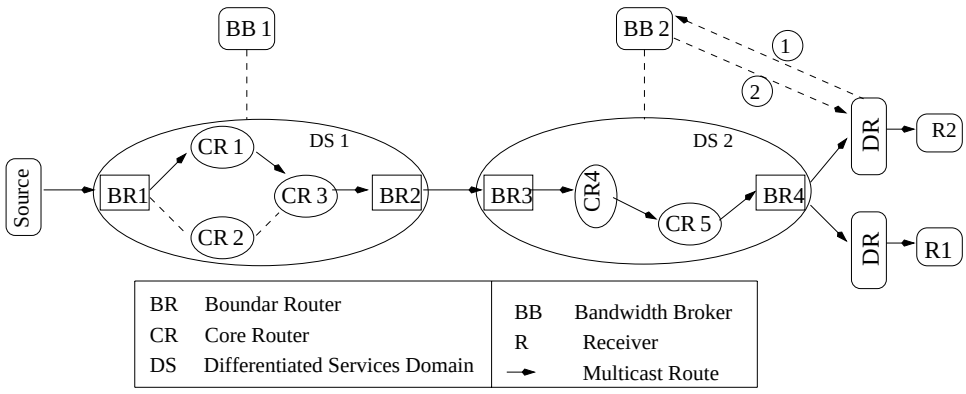


Figure 12: Signaling for case 1.

Note that to be able to use the existing multicast routing state during actual data transmission, the SLAs have to be *multicast routed unicast SLAs*, i.e., unicast SLAs that follow the same path from ingress to egress routers as the multicast route would take. If not, there are two options. The first involves communicating with all routers in the unicast and multicast routes between ingress and egress routers and *re-installing* the multicast routing state to reflect the new route according to the available unicast SLA. The second option is to use IP tunneling [40] to use normal unicast SLAs.

If the resources are granted, the BB updates the current resource allocation table for the latest bandwidth allocation corresponding to the ingress-egress router pair involved and sets the subnetwork count for the corresponding entry to one. The signaling does not terminate here for this case and the BB signals the upstream peer BB. Upon receipt of the message, the peer BB runs the same protocol as this BB.

Figure 13 depicts the signaling for the case when $R1$ joins group G in a two domain network. No resource allocations exist for G in either domain $DS1$ or $DS2$, so new resources are allocated in both domains when $R1$ joins. The signaling goes all the way to the multicast source's domain before returning back to the DR in this case.

4.5.1.3 Case 3: Branching Needed

The third case arises if one or more core routers send the corresponding multicast forwarding table entry to the BB of their domain, informing that they would be the branching point for

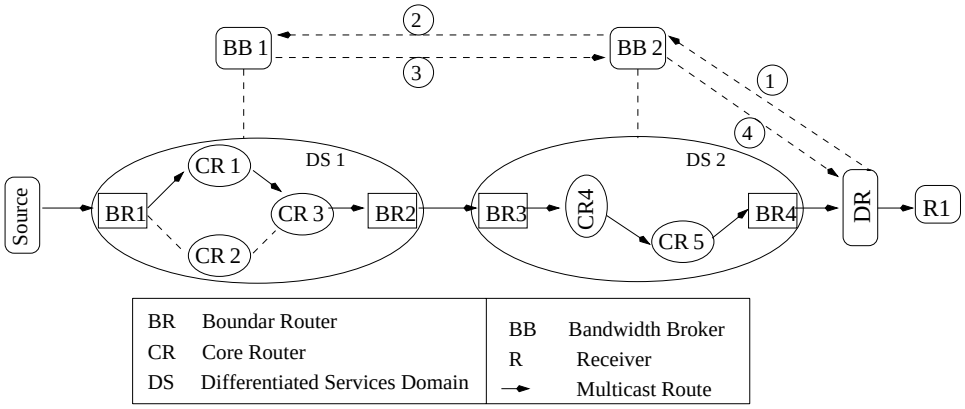


Figure 13: Signaling for case 2.

group G. This implies that the multicast tree passes through this domain but branching is required to graft the new receivers to the existing multicast tree. If the allocated resources are adequate for the new QoS request, new receivers can be grafted to the existing tree. The edge-router branching technique does not allow any branching point in the middle of any domain, hence to be able to graft new receivers to group G, an additional SLA from the ingress router to the new egress is used if one is available.

Figure 14 shows the use of two separate SLAs in a domain when the branching point exists in a domain.

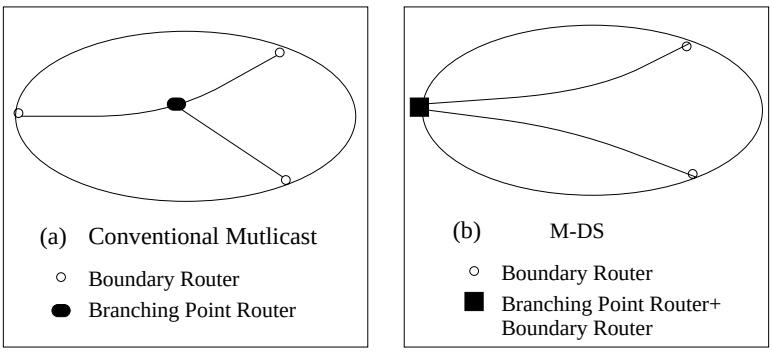


Figure 14: Two different SLAs when branching point exists in the domain.

Since all SLAs are unicast SLAs, to be able to use the new SLA, the BB moves all branching points from its core to the ingress of the domain and enhances the role of the ingress router to act as a branching point in addition to being an ingress router. This

technique makes the core router functionality very scalable, just as in the DS for unicast communication but doing so amounts to introducing some changes in multicast routing. The BB extracts the new routing information using the multicast forwarding table entry sent to it by the core branching point router(s) and conveys the appropriate routing information to the ingress router. The details on how the ingress router performs routing during data transmission and how core branching point router(s) use their existing multicast routing state are explained in section 4.7. There is no per-packet bandwidth overhead during data transmission and some changes are required to be made to router functionality.

After moving the branching point to the ingress, the BB creates a new entry for this ingress-egress router pair in the current resource allocation table and sets the subnetwork count for the corresponding entry to one. To complete the signaling, the BB signals affirmatively in the path toward the DR and data transmission begins for the new receivers since multicast state is already set up.

Figure 15 depicts the signaling for the case when $R1$ and $R2$ are present and branching is needed in $DS2$ for data to reach $R3$. New resources are allocated in $DS2$ and signaling returns to the DR without going all the way to domain $DS1$.

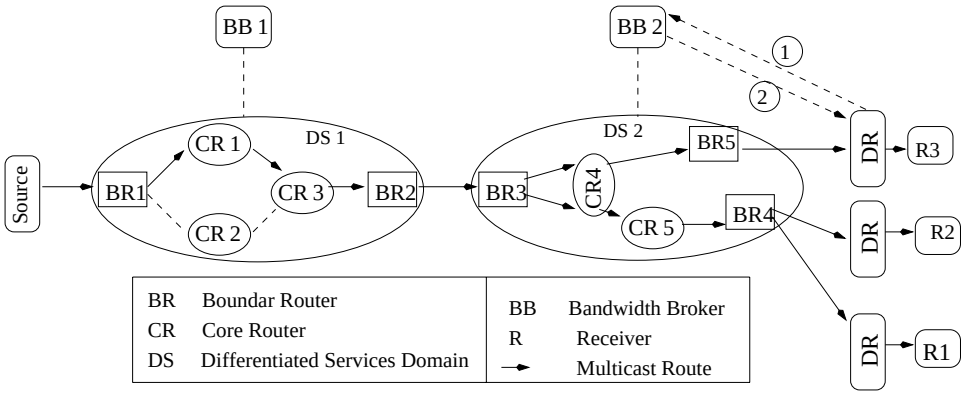


Figure 15: Signaling for case 3.

At any point in the path to the source, if resources are denied, signaling returns from that BB all the way back to the DR, freeing all the resources and canceling all configuration changes and table updates. Most IP multicast routing protocols are based on soft state protocols and if data transmission does not begin, routing state vanishes automatically due

to lack of a refresh. In our case, since the source will not be sending data to the new receivers unless resources are allocated all the way, the IP multicast routing state would vanish automatically.

4.5.1.4 Additional Considerations

In addition to the signaling overhead described above that is incurred when DRs make resource allocation/deallocation requests, depending on the protocol in use in a domain, there may be some state maintenance overhead also for our protocols. Many multicast protocols like PIM-SM [23] are based on the notion of soft state. Hence, routers periodically send state join/prune messages to reflect the latest receiver population for various multicast groups. If the underlying routing changes are reflected in these periodic messages, the action of moving the branching points from the core of a domain to the ingress (as in the third) case described above will have to be carried out in that domain. In addition, depending on the amount of traffic from particular sources, PIM-SM switches from shared tree to source specific tree, which can lead to a change in the branching points in one or more domains. In this case also, the action of moving the branching point from the core to the ingress router will have to be carried out.

4.5.2 Signaling Protocol for Departure

The protocol for the case when resources need to be deallocated for multicast group G is very similar to that of the case when they need to be allocated; only allocation of resources become deallocations. As in the case of allocation of resources, the DRs first contact the BB responsible for their domain. The BB uses the routing information it has to look up the ingress-egress router pair serving this receiver in its domain. There are two possible scenarios. First is that the subnetwork count corresponding to the ingress-egress router pair is one. This means this was the last receiver group served from this branch of multicast tree. Second is that the subnetwork count is greater than one, implying that there are more receivers on subnetworks in this domain or other domains that are being served by this ingress-egress router pair. In the first case, the resources can be deallocated and the corresponding entry from the current resource allocation table can be removed. Also, the

changes made to the functionality of the ingress router of this domain are undone and peer BB in the path toward the source is signaled to carry out the leave protocol as well. In the second case, the subnetwork count is decremented by one because resources can not be deallocated yet and the adequate changes are made to the functionality of the ingress router. For the second case, the signaling can return from this point back to the DR confirming that the last receiver's leave from group G is complete.

4.5.3 Formal Description of Edge-router Branching

In this section, we describe the basic state transitions of all the entities of the M-DS architecture (with respect to a particular multicast group G and QoS requirements) that undergo changes with respect to their functionality in the case of the DS for unicast. Such entities in each domain are: the receivers, DRs, core routers, ingress routers, and the BB.

Receivers A receiver R is in one of the three states at any point of time: READY (ready to receive data), WAITING (has contacted the DR with group G and QoS information and is waiting to receive data), or RECEIVING (*success*fully receiving data). If t denotes the time, T the event of a time-out that prompts the receiver to contact its DR again, and N the maximum number of time-outs, the transitions for R are as shown in figure 16. We use *done* to denote the transition onto the starting state for all the entities.

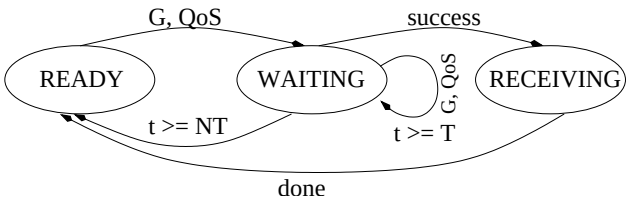


Figure 16: State transition diagram for multicast receivers.

Designated Routers The state transitions for a DR look almost identical to those of a receiver. Upon DISCOVERY (multicast discovery), the DR transitions to state WAITING (by providing its IP address, multicast group G, and R's QoS requirements to the BB of its domain). It transitions to state RECEIVING when it starts getting data for R (denoted by

success). Assuming that the time-out period is T, the transitions are as shown in figure 17.

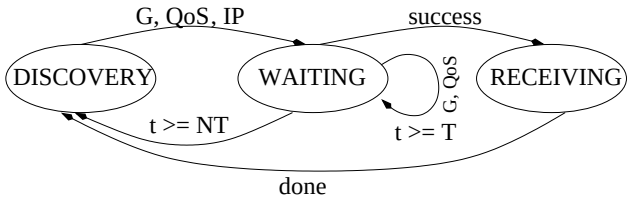


Figure 17: State transition diagram for DR.

Core Routers While the IP multicast routing state is being set-up in a particular core router, it is in state SETUP. If we denote the event of core router discovering it is a branching point router by BP, depending on whether or not it is a branching point router, it transitions into state CONTACTING, whereby it contacts the BB of domain to give the BB its appropriate multicast forwarding table entry. The state when it is receiving data and processing it according to the M-DS routing rules described in section 4.7 is called RECEIVING. These states yield the state transition diagram shown in figure 18.

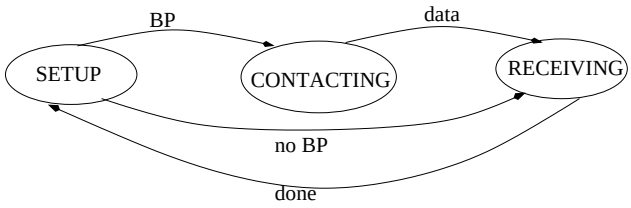


Figure 18: State transition diagram for the core router.

Ingress Routers An ingress router starts out in state CONTACTED when it is contacted by the BB for duplicating the packets for multicast group G instead of the original core branching point router. While in this state the ingress router records information sent by the BB for use during data transmission for G. Upon receiving multicast data, the ingress router processes and forwards it according to the rules described in section 4.7 and transitions into state RECEIVING. The state transition diagram is as shown in figure 19.

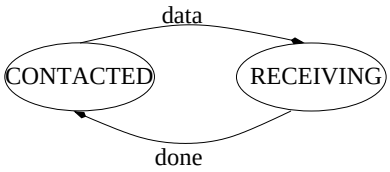


Figure 19: State transition diagram for the ingress router.

Bandwidth Broker We begin by describing the ten states of a BB that involve either contacting a peer BB or a DR of its domain, or the states BB is in after being contacted by a peer BB or the DR of its domain. We use *downstream* to indicate the direction of data flow from the source to the receiver and *upstream* for the direction from the receiver to the source. The BB is in state CONTACTED/RA when it is contacted by its downstream peer BB or DR for resource allocation. It is in state CONTACTED/RD when it is contacted for resource deallocation by the downstream peer BB or DR. The states where it is contacted for successful allocation/successful deallocation/failed signaling by upstream BB are CONTACTED/SA, CONTACTED/SD, and CONTACTED/FS respectively. The corresponding five states for the case when the BB has to *c*ontact its peer BBs or DRs are given by CONTACT/RA, CONTACT/RD, CONTACT/SA, CONTACT/SD, and CONTACT/FS, with the upstream becoming downstream and vice versa. The transitions between these states occur because of the following events:

- looking up the routing information (RI) to find out the ingress and egress routers in its domain that the SLA for group G requires, denoted by L-RI.
- looking up the current resource allocation table (CRAT) to find out if appropriate resource allocations exist between ingress and egress routers for group G, denoted by L-CRAT.
- expiration of timer that was started to get a response from all the core routers that act as branching point routers for group G, denoted by T.
- getting a response from the core branching point router (CR) before the expiration of the timer, denoted by R-CR.

- availability of an SLA between the ingress and egress routers for G, denoted by A-SLA.
- using the forwarding table entry (entries) sent by the core branching point router(s) to send M-DS routing information to the new branching point router (the ingress router), denoted by S-RI.
- updating the count of networks that a particular ingress/egress router pair serves (for deallocation purposes), denoted by C.
- BB discovering that it does not have any peer BB on the upstream (implying that the source is in the BB's domain), denoted by S.
- undoing the routing changes sent to the branching point router (ingress router) and deallocating resources, denoted by U.

Figure 20 shows the transitions between the ten BB states. The BB can start out in any of the five *CONTACTED* states and end up in any of the five *CONTACT* states. The figure shows dashed lines for the case when the BB is contacted by the *upstream* router and solid lines for when the BB is contacted by the *downstream* router.

4.6 Limited-core Branching

The edge-router branching technique uses the available unicast SLAs in each domain and concatenates them to make up end to end multicast SLAs. It exploits multicast scalability only at the domain level, and hence incurs extra packet hops. This simplifies the core of the domain but introduces extra packet hops during data transmission. A good trade-off between making all core routers that are branching points complex versus incurring extra packet hops is to limit the number of core routers that would be allowed to be branching point routers. This is the motivation behind *limited core branching*.

For the edge-router branching technique, we assumed that the SLAs in a domain existed only between ingress and egress router pairs, for the purposes of the limited-core branching technique, we assume that additional SLAs have been defined from certain special *core routers* to some egress points in addition to the usual unicast SLAs. These special core

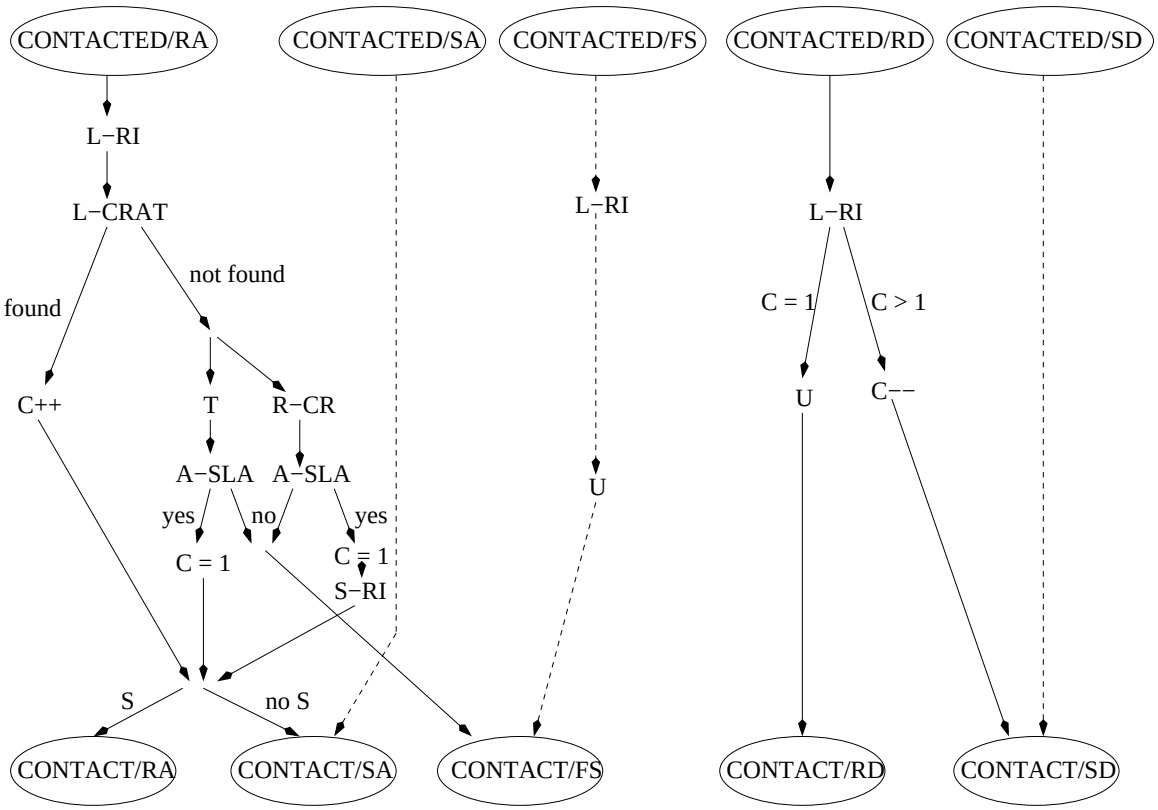


Figure 20: State transition diagram for the BB.

routers are the only ones that can be used as branching points in the domain. We do not address the issue of optimal placement strategy of special core routers, optimal number of such routers, and whether special core routers can be dynamically changed when receiver population changes. That remains an issue of future investigation.

The rest is a simple extension of the three cases we described for edge-router branching. The manner in which the first two cases, namely when no resource allocations exist and when exact resource allocations exist and no branching is needed, are dealt with as before. The only difference is in the way the third case is handled. For every new membership discovery if the BB finds out that a branching point exists in its domain it first finds out if there is a special core router configured in its domain that is also a branching point. If there is, and appropriate resources are available, then for the path subsequent to that core router in the domain the core's functionality is enhanced to be similar to that of the ingress router in edge-router branching. The ingress router can still be configured as a branching

point router as a fall back mechanism.

Figure 21 shows an example of a domain that runs limited-core branching with one special core router. In general, the special core routers do not have to be the branching point routers but in order to keep the routing described in section 4.7 simple we assume that a special core router acts as a branching point only if it was the original branching point router. This simplification is necessary because otherwise the routing state for the other core routers may also to be changed. Figure 21 shows a situation when the receivers have joined a multicast group G and require exiting at 5 egress routers in a domain. The role of branching point $B4$ has been upgraded to that of special core router SC, while the behavior of other branching points $B1$, $B2$, and $B3$ remains the same as in edge-router branching. As a result, the ingress router will be instructed to make 4 duplicate copies of packets, one to reach receivers $R1$ and $R2$, and one each to reach $R3$, $R4$, and $R5$. The special core router will be instructed to make two copies to reach $R1$ and $R2$. The special core router has become more complex than $B1$, $B2$, and $B3$ because it performs a functionality similar to the ingress router (but for fewer traffic flows) but the number of packets traversing the path between the ingress I and special core router has been reduced compared to the edge-router branching.

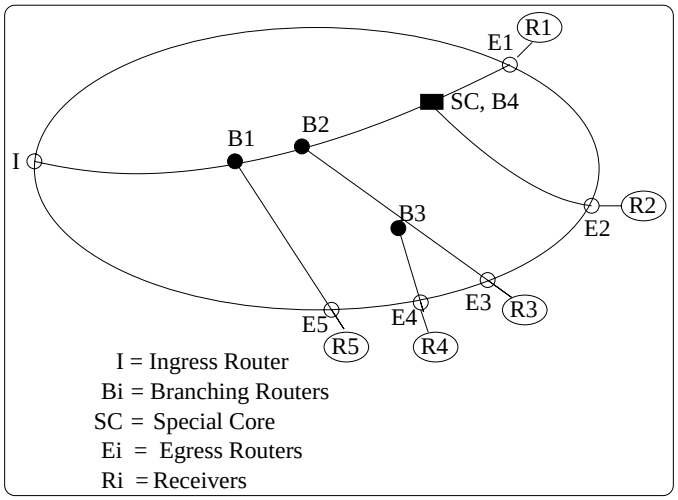


Figure 21: Different roles of routers in limited-core branching.

Limited-core branching reaps the benefits of minimized packet hops during data transmission while keeping the technique scalable by limiting the number of special core routers allowed at the cost of a slight increase in complexity, which is controllable by individual domains. Any domain can opt to using limited-core branching independently of other domains and can configure as many special core routers as it chooses. This is because both the edge-router branching and limited-core branching techniques inter-operate.

4.7 Routing Under M-DS Architecture

Both the edge-router branching and limited-core branching techniques require modifications to the functionality of the routers. Assuming that the special core routers are only allowed to act as branching point routers for limited-branching technique if they were the original branching point routers (to avoid changing the multicast state for other core routers in a domain), the routing concerns for both the techniques are similar. We now describe the routing details for the edge-router branching technique for the case when branching is required to graft the new receivers to an existing multicast tree.

The basic idea behind the new routing is to keep the core of the domain simple. As a result, any branching point is moved from the core of the domain to the ingress. The ingress router would now need to duplicate packets (and hence act as the new branching point router) to reach all the receivers instead of the original core branching point router. The complication arises when these identical duplicate packets reach the original branching point router in the core of the domain. The technique described in this section helps the ingress router in putting information in the packets that guides the original core branching point router to use the multicast routing information it already has and forward the data packets on the correct outgoing interfaces without any duplication.

Routing Entry Sent by the Core Branching Point Router to the BB During the signaling for resource allocation when a core router of a domain determines that it is going to be the branching point router to graft the new receiver of group G to the existing multicast tree it sends its corresponding multicast forwarding table route entry to the BB of

its domain. A forwarding table route entry in an IP multicast routing table includes fields such as source address, multicast group address, incoming interface from which the packets are accepted, list of outgoing interfaces to which packets are sent, and the corresponding TTLs (Time To Live).

Routing Information Sent by the BB to the New Branching Point The BB uses the multicast forwarding entry sent by the core branching point router of its domain to deduce and communicate the routing information to the new branching point router (ingress router). It can aggregate multiple forwarding entries for the same multicast group from different core branching points in doing so. The routing information sent by the BB contains the multicast address for which the packets have to be duplicated, outgoing interface number for the ingress router, and the number of entries; one for each original core branching point router. Each entry contains the IP address of the core branching point router for identity purposes, the number of duplicate packets that the core branching point router would have made, TTL, and the individual *bit vectors* corresponding to each outgoing interface of the original core branching point router, one each corresponding to the number of duplicate packets (see figure 22). The BB determines the new branching point router's outgoing interface corresponding to the original core branching point router's incoming interface using the router configurations it has.

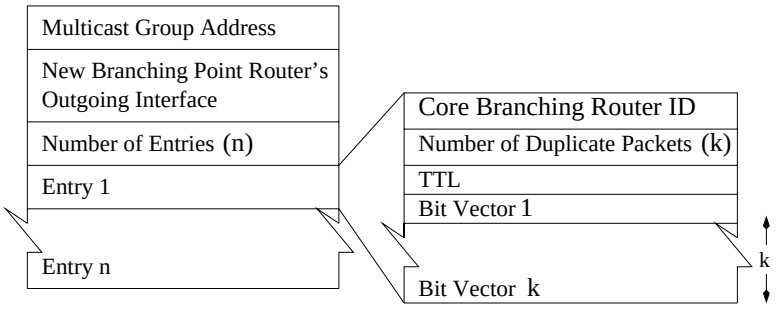


Figure 22: Routing information for the new branching point router.

The BB fills out the TTL by adding hops between the new branching point router and original core branching point router to original core branching point router's TTL. The number of duplicate packets depends on the number of receivers served out of an outgoing

interfaces in the original core branching point router's multicast forwarding table entry.

During data transmission, the ingress router uses the bit vector in the new routing information to put routing information in IP packets. This routing information is used by the original core branching point router in deciding which of its outgoing interfaces should the packet be forwarded on. The values that this bit vector can take will become clear once we define how the packets will be processed at the core branching point router.

Since all routers have a fast path, for efficiency purposes, we describe two methods of processing the multicast packets. There is a 2 bit unused field adjacent to the codepoint field in IP. If the maximum number of outgoing interfaces corresponding to any single incoming interface in a actual multicast routing table entry is 4, it would be faster to use those 2 bits to forward packets compared to the other option we will define shortly. In this case, the bit vector that the BB would send in the new routing information during signaling phase would be $xx111111$. Each xx bits correspond to one outgoing interface corresponding a routing table entry. For example, 00 would be put for packets that are to be forwarded on interface 1 for the branching point router. Similarly, values 01, 10, and 11 corresponding to outgoing interfaces 2, 3, and 4. Notice that the actual number of outgoing interfaces for the branching point router may be more than the number of outgoing interfaces in the forwarding table route entry.

New Branching Point Router Functionality The ingress router extracts the entries for each core branching point router from the information sent by the BB. For each entry, it makes as many packets as indicated by the *number of duplicate packets* field. To help the original core branching point router process each of these duplicate packets, we propose a solution that utilizes the IP options field as shown in figure 23. In each of the duplicate packets, the new branching point router copies the *core branching router ID*, and the *bit vector*.

Figure 23: New IP option.

The first bit in the first octet of figure 23 is to ensure copying of this new field in all fragments. The next two bits define a new option class for DS for multicast, and the last 5 bits define a new option type with $number = 1$. The second octet contains the length of this IP option. Following that are the core branching point router identity and the bit vector.

Core Branching Point Router Functionality Upon seeing a multicast packet, the original core branching point router extracts the ID field from the IP option field along with the bit vector. If the ID field matches its own IP address, it processes the packets, otherwise, it just forwards it without doing anything. The processing of the packets is simple. The bit vector from the packet (assume of size one octet) can take 256 values, one corresponding to each outgoing interface of core branching point router. The core branching point router uses this bit vector and the relevant multicast forwarding table entry for group G, and maps the outgoing interface in the multicast entry to the bit vector. After the mapping, instead of making duplicate packets using the multicast entry, it forwards the packet on that interfaces.

4.8 Performance Evaluation

The proposed techniques move the multicast branching points to the edge of the domains for scalability, incurring extra packet hops in the multicast tree. Also, they require signaling protocols to be run for membership discovery and termination on subnetworks. For evaluating the performance we designed simulations to estimate the extra packet hops and signaling overheads in terms of the number of messages.

The main results of evaluating these overheads can be summarized as follows. First, the total signaling overhead per membership discovery and termination on subnetworks under both techniques is similar for all the topologies tested. It varies between 3.75 and 7 messages per receiver join and is small compared to the average per second routing overhead of 23 messages/second per BGP-4 router [54]. Second, the simplicity of the edge-router branching technique comes at the cost of extra bandwidth consumption in terms of packet hops during

data transmission. The effect is more pronounced when receivers are clustered together. Third, the minimal controllable additional complexity of limited-core branching technique compared to edge-router branching saves the extra bandwidth consumption compared to edge-router branching by about 40%.

4.8.1 Simulation Setup

We used GT-ITM [13] to generate various topologies comprising of 744, 2646, and 6384 nodes each. Compared to the size of the Internet, these topologies seem small, but we believe that our simulations on these topologies produce results that prove the feasibility of deployment of both the techniques. The reason for this is the following. Signaling overhead in the proposed techniques depends on the number of domains in the topologies, not the number of nodes. In choosing the number of domains in the topologies, we used the results of a simple traceroute experiment we performed using random destinations across the globe. Our results indicated that most packets cross between 3 and 5 domains between source and destination. All the paths in our topologies have these properties. Simulations were conducted using a custom simulator written in *C*. It implements Dijkstra's shortest path algorithm for multicast routing. The details about the number of transit, stub, and total domains in these topologies are shown in table 2.

Table 2: Simulation topologies.

#Nodes	#Transit domains	#Stub domains per transit domain	#Total domains
744	4	3	12
2646	6	4	24
6384	7	5	35

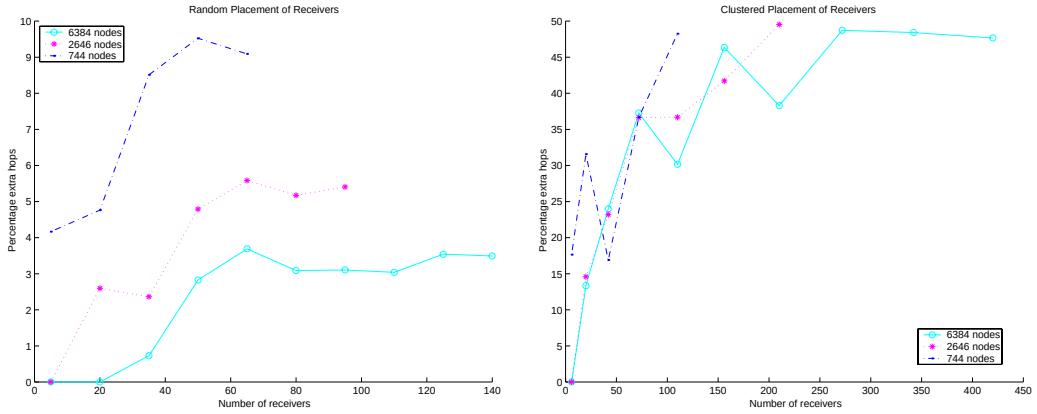
To estimate the extra packet hops and the signaling overhead for both the techniques, we experimented with two types of placement of receivers, random and clustered. For random placement of receivers, we specified the total number of receivers to be placed in an entire topology. For clustered placement, we specified the number of stub domains the receivers should be placed in and the number of receivers per stub domain. Our simulator

implements Dijkstra's shortest path algorithm for multicast routing. To estimate the bandwidth overhead in terms of extra packet hops compared to the multicast routing under both techniques, we placed the receivers statically. To estimate the signaling overhead for both the techniques, we experimented with dynamic join and leave of receivers. We varied the mean time in the group for the receivers and experimented with uniform and exponential distributions of receivers' time in the multicast group. All simulations assume a maximum of one receiver per subnetwork, making the results presented in this section conservative. In real scenarios there may be more than one receiver on each subnetwork and signaling will take place only once for the membership discovery and termination in this case. Further, the simulations assume that all the receivers join the same multicast group, and that a receiver join request is never denied for lack of resources.

4.8.2 Bandwidth Overhead

Figures 24 (a) and (b) show the percentage extra packet hops (compared to the total hops using shortest path multicast routing) for various topologies for edge-router branching. The receivers are statically placed randomly and in clusters. The graph for clustered placement of receivers is not smooth because the extra hops are closely tied to the actual placement of receivers. We ran the tests on various topologies for each number of nodes, with similar results. The overhead is less for random placement of receivers, about 10% extra packet hops compared to total hops for 140 receivers; compared to about 40% when the same number of receivers are clustered. This is expected because random placement of multicast receivers does not lend itself to as much savings in packet hops because of lack of commonality in the path to the receivers. The overhead does not increase substantially beyond a certain percentage when more receivers are added to the clustered placement. This is evident from the fact that in going from 140 to 450 clustered receivers, the overhead only goes from 40% to 50%. This seems logical because adding more receivers to the same domain after a point would not increase the overhead further.

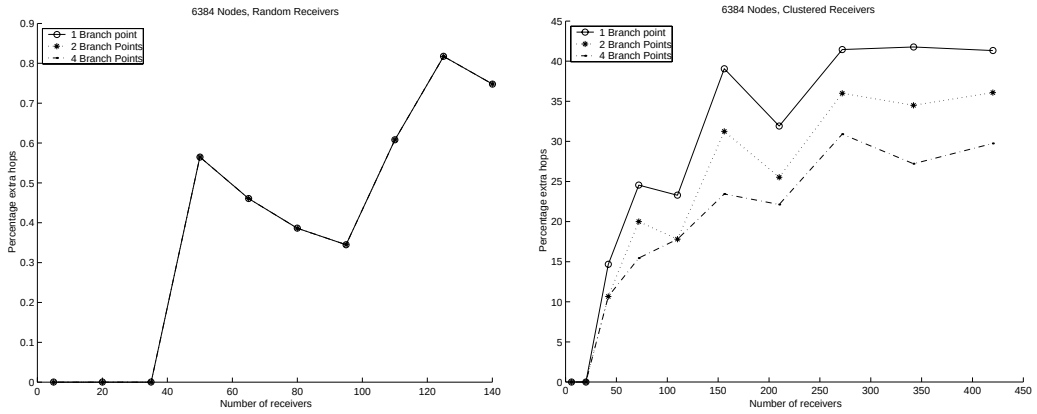
Figures 25 (a) and (b) show the percentage extra packet hops for random and clustered placement of receivers for limited-core branching. The plots show three cases, when 1,



(a) Random placement. (b) Clustered placement.

Figure 24: Percentage extra hops for edge-router branching.

2, and 4 actual branching points are configured as special routers in each domain for the topology with 6384 nodes. For clustered placement of receivers, configuring 4 branching points as special routers in each domain brings down the percentage of extra hops to about 30% of actual packet hops under multicast routing, which is about 40% saving compared to edge-router branching technique. Also, note that configuring more branching points as special routers for random placement of receivers does not make a difference in the savings. This implies that the decision about configuring branching point depends on the placement of receivers.

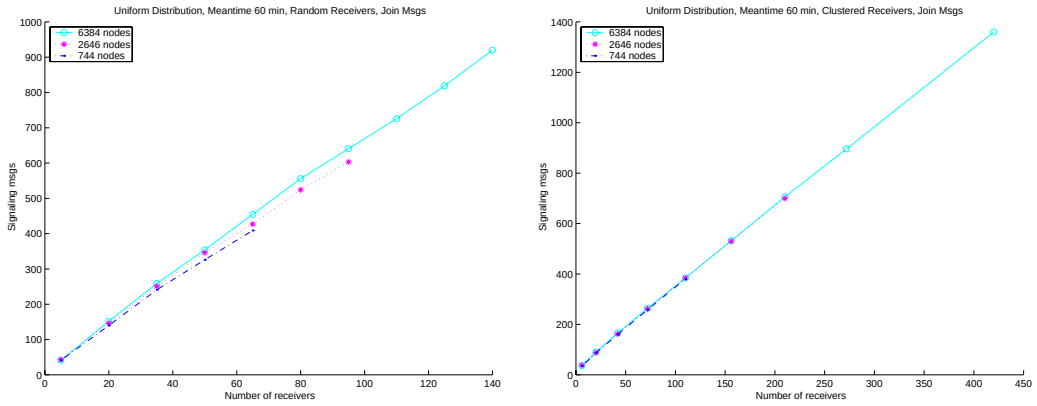


(a) Random placement. (b) Clustered placement.

Figure 25: Percentage extra hops for limited-core branching.

4.8.3 Signaling Overhead

To estimate the signaling overhead, figures 26 (a) and (b) show the number of signaling messages for when the receivers join dynamically. They show it for random and clustered placement of receivers for all three topologies for edge-router branching. The mean time between the arrival of receivers in the group for these plots is 60 minutes and the distribution of receivers' time in group is uniform. Since signaling is at the domain level and all the simulation topologies have a similar number of domains in the paths to the receivers the signaling message overhead is almost independent of the number of nodes in the topology. As expected, the signaling overhead is more for random placement of receivers than for the clustered placement. The message overhead for 140 clustered placement receivers is about 50% less compared to similar number of randomly placed receivers. This is so because for clustered receivers, the signaling does not have to go all the way to the sender because of the presence of other receivers.



(a) Random placement. (b) Clustered placement.

Figure 26: Signaling messages for receiver join in edge-router branching.

We experimented with exponential distribution as well with the mean time in group ranging from 2 minutes to 2 hours, the results were almost the same. Since the protocols involve a similar amount of signaling overhead for both join and leave of each receiver, the corresponding plots for the overhead when receivers leave were identical. The signaling in the case of limited-core branching involves only as many additional messages as the number of branching points in each domain. Hence, the signaling overhead for it is expected to be

similar to that in the case of edge-router branching.

Assume 100 receivers join the same multicast group, one every 2 minutes. From figure 26, for random placement of receivers, the approximate number of join messages per receiver is about 7 (700 messages/100 receivers). The corresponding figure for clustered placement is 375/100 = 3.75. The number of receivers per second is about approximately 1 for both cases. Using these two numbers for each of random and clustered case, we find that the number of messages per second is about 7 for the random case and 3.75 for the clustered case. Each BGP-4 router processes about 23 messages/second on an average [54]. For edge-router branching, all domains combined that are involved in getting data to the receiver process between 3.75 to 7 messages per second for our topologies, a small overhead.

The above results on the signaling are conservative because we assume only a maximum of one receiver per subnetwork. In reality, because of the manner in which IGMP works, signaling would be carried out once for every membership initiation and termination on a subnetwork. hence, for subsequent receivers on the same subnetwork, there is no signaling overhead.

4.9 Conclusion

The DS framework is a scalable way to provide QoS for unicast communication. Dynamic join/leave of receivers in multicast poses unique challenges in utilizing the DS framework to provide support for multicast communication. This chapter proposes M-DS, a scalable architecture that consists of two inter-operable techniques for providing service differentiation for multicast utilizing the DS paradigm. Both the techniques, *edge-router branching* and *limited-core branching* define two signaling protocols each to be run upon membership discovery and termination on subnetworks. Signaling allocates and deallocates resources and sets up state in the relevant routers to be used during data transmission. Upon the completion of signaling, data flows in as in the case of DS for unicast. The edge-router branching technique exploits the multicast scalability at a domain granularity and moves all the branching points required to graft new receivers to the existing multicast tree to the ingress of the domain. The limited-core branching technique allows a limited number of

branching points to exist in a domain. This introduces a small amount of extra complexity but exploits multicast scalability in a more effective manner.

Even though both techniques use the IP multicast routing state already set up in individual domains for forwarding packets during actual data transmission, incorporating them in the DS framework requires that the functionality of all the routers be enhanced. But for actual data transmission, only a few routers would need to use this enhanced functionality.

Simulations show the feasibility of the proposed techniques since they incur modest signaling and bandwidth overheads. Because these techniques do not introduce any extra headers in the data packets, they do not involve any per packet bandwidth overhead during data transmission.

CHAPTER V

REPUTATION-BASED SERVICE DIFFERENTIATION IN PEER-TO-PEER NETWORKS

5.1 Introduction

Peer-to-peer (P2P) networks have introduced a new paradigm in content distribution. Each peer is both a client and a server in these networks. Users are drawn to these networks due to the ability to locate a wide variety of multimedia content. The popularity of these networks can be judged by the fact that the query-response traffic in $2000 - 2001$ due to Gnutella comprised about 1.7% of the total traffic in the Internet backbones in December 2000 [44].

P2P networks essentially come in three flavors: 1) centralized P2P networks like Napster, 2) decentralized unstructured networks like Gnutella and Kazaa, and 3) decentralized structured networks like CAN [42] and CHORD [51]. All are founded on the fundamental principle of cooperation among the peers. Getting content from a P2P network involves two phases: 1) *content search* and 2) *content download*. Content search in centralized P2P networks is facilitated by a central server while it is carried out by the peers themselves for both the decentralized flavors. Content download in all the flavors occurs directly between the downloading peer and the serving peer.

Each peer in a P2P network is capable of being a client and a server. Perhaps this inherent notion of equality of peers is one of the reasons why service differentiation issues have not received much attention from the research community. However, as the population of these networks increases, issues like free-loading behavior[1] make it necessary to differentiate among peers. The presence of free-loaders is evidenced by several measurement studies of the deployed instances of P2P systems like Napster and Gnutella. It has been reported [4]

[1]Free-loaders are peers who only download content but do not serve it to the other peers.

that at the time of the study, nearly 70% of Gnutella users shared no files, and nearly 50% of all responses were returned by the top 1% of sharing hosts. Also, the study in [46] quotes the free-loaders to be about 25% in Gnutella but much less in Napster.

There are two main directions one can pursue in order to provide service differentiation in P2P networks. First, peers can be explicitly charged money depending on the class of service they belong to. However, due to the absence of the traditional client-server model of delivery in P2P networks, pricing issues in these networks are yet to be resolved. Second, the fact that the peers differ from each other in the services and resources they provide (or are able to provide) can be exploited. A measure of the latter is generally referred to as the *reputation* of the peer and has been explored extensively in various contexts to motivate participants to cooperate for the common good.

Price and credit-based approaches have been explored in ad-hoc networks for stimulating cooperation among the nodes [57, 41]. The concept of reputations has been explored toward the same goal in P2P networks [1, 18, 31, 35, 56, 26]. The basic idea behind these reputation systems is to assign a reputation to each peer based on the satisfaction experienced by the other peers from the services it provides to them. While being able to find reputable peers is beneficial from the perspective of a peer wishing to retrieve content, it may serve as a disincentive for peers acting as servers to have a good reputation for the fear of being overwhelmed. An effective service differentiation scheme could serve as a motivation for the peers to possess good reputations.

Focusing on Gnutella-like P2P networks, this work takes the approach of using peer reputations to address service differentiation issues in P2P networks. Essentially, Kazaa uses a similar concept. Its *participation level*[2] can be viewed as a simplified form of a reputation. The participation level is used to differentiate among the peers in that peers with a higher participation level are served before others during heavy access conditions.

Assuming that a service differentiation scheme consisting of three levels of service (LoS) is used, figure 27 shows an example of how the peer reputation scores (RSs) can be mapped

[2]The participation level is locally updated for each peer based on the number and size of *integrity rated* files served by it.

to various LoSs using parameters a and b. In this scheme, peers for whom $RS < a$ are eligible for *basic LoS*. $a \leq RS < b$ provides *enhanced LoS* to the peers. Peers with $RS > b$ receive *premium LoS*. The parameter a and b are expected to be known to each peer in the P2P network.

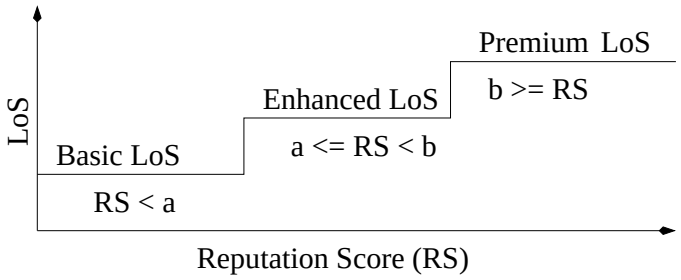


Figure 27: An example of three levels of service in a P2P network.

The goal of service differentiation is not to provide hard guarantees but to create a distinction among the peers based on their contributions to the system. The basic idea being, the more the contribution, the better the relative service. While service differentiation parameters are well understood and studied in the context of the Internet (e.g. delay, jitter, bandwidth), they are still to be defined for the P2P networks. Focusing on Gnutella like P2P networks, this work makes three main contributions: 1) it defines a set of parameters that are suitable for service differentiation in P2P networks, 2) it proposes SDP, a protocol to accomplish service differentiation using the proposed parameters, and 3) it identifies a set of features that are necessary in the reputation system to be used as a basis for service differentiation.

5.2 Service Differentiation Parameters

Peers carry out three main functions in a P2P network: 1) bootstrapping, 2) content search, and 3) content download. The overall experience of a peer in a P2P network depends on the network conditions and the services and resources provided by the other peers during each of these functions. The network conditions depend on many factors that may not be controllable within a P2P overlay topology and as a result are not considered in this paper.

Bootstrapping is required to allow peers to join the network. In Gnutella like P2P

networks, to search for content, a querying peer generates a query with appropriate *keywords* and sends it to all the peers that it is directly connected to in the overlay topology. The peers who process this query reply back if they have the content in their shared directory and forward the request to the peers they are directly connected to depending on the *hop-count* (or the TTL) of the query. This forwarding continues until the TTL specified by the querying peer is exhausted. Upon receiving the replies, the querying peer selects a peer to download the content from. At that point, the content download typically uses a HTTP or a TCP connection with the selected peer.

Next, we describe the set of parameters that can be mapped to each LoS. They are guided by the factors that create service differentiation during the bootstrapping, content search, and content download functions in a P2P network; and hence the peer's perception of service quality. The parameters described here are based on the salient features of the widely deployed Gnutella-like P2P networks and the results of the current research on similar unstructured decentralized P2P networks.

5.2.1 Factors Affecting Bootstrapping

During the bootstrapping process, the type of peers a peer directly connects to in the overlay topology play an important role in its overall satisfaction from content search and download functions later on. For example, apart from how cooperative the connecting peers are, their actual network distance, processing power, memory, bandwidth, and storage capacity are important factors.

5.2.2 Factors Affecting Content Search

The success of the search phase requires that the other peers be online, agree to search for the content from their shared directory, and forward the query further depending on the hop count of the query. The overall experience during this process is impacted by the type, quality, and quantity of the content other peers place in the shared directory. The following factors can be used as a basis for service differentiation because they impact the perception of service for peers during content search:

Number of hops: To search for content, the querying peer sets the maximum number

of hops in the overlay topology its query would take, by denoting the *hop-count*. While the success of the content search phase depends on many other factors as well, the number of hops plays an important role. Hence, setting a *hop-limit* could act as a component of the service differentiation scheme.

Premium content: Peers can choose to classify some of the content they share as premium content, which they can make available only to peers eligible for certain minimum LoS. This classification can be done through some system wide guidelines.

Hard to find content: A special utility of the P2P networks for many peers comes from being able to access *hard-to-find* content. Although classifying content as hard-to-find may be based on subjective criteria, peers can potentially reserve the hard-to-find content only for peers with a certain minimum LoS.

Query caching: Sripanidkulchai [49] found that the popularity of search strings in Gnutella follows a Zipf-like distribution and that caching a small number of queries results in a significant decrease in the traffic in the Internet. In order to distinguish among the peers with various LoSs, the outcome of caching queries may be made available only to peers with a certain minimum LoS eligibility.

Cached content: Kazaa distinguishes between the functionality of *supernodes* and the rest of the peers in its P2P network. Peers with higher bandwidths can choose to become supernodes in a Kazaa P2P network. During idle periods, the supernodes actively query other peers in the network and cache the content so retrieved. This gives the supernodes access to additional content and when queries for the cached content arrive at the supernodes, they can get served faster. For faster retrieval of content in unstructured P2P networks, Cohen et. al. [17] have also proposed caching strategies. Due to its potential to improve peer experience, caching could be used to distinguish among peers with different LoS eligibility.

Interest-based locality: By exploiting interest-based locality, Sripanidkulchai et. al. [50] have proposed an efficient content search solution for unstructured P2P networks. The basic idea is for peers that share similar interests to create *shortcuts* to each other. These shortcuts can then be used to locate content faster. The basic Gnutella content search paradigm remains as a backup mechanism. In creating such shortcuts, peer LoS eligibility

may be used as an additional deciding factor.

Load balancing: An enhancement to maintaining a shortcut to peers that share common interests (as described above) would be to maintain the most recent load for those peers as well. Such an information can help in avoiding already overwhelmed peers and potentially get the content faster. The availability of such information however would require a periodic protocol to assess the load on the peers with similar interests. But if this information is available, it can be provided to peers eligible for certain minimum LoS during the content search phase.

5.2.3 Factors Affecting Content Download

Successful content download requires that the chosen peer be online and serve the content when requested. The quality of the downloaded content is an important consideration in the overall experience from the downloaded content. The processing power of the serving peer, its storage capacity, and the bandwidth at which the actual download occurs also contribute to the satisfaction of the peers interested in the content. A high bandwidth querier peer is likely to have a better experience with the system if it downloads content from another high bandwidth peer. During the content download phase, the following factors can be used as a basis for service differentiation because they impact a peer's overall experience:

Rate of transfer: During content download from the chosen peer, the rate of transfer may be dependent on the LoS the downloading peer is eligible for. The basic idea is to restrict the portion of capacity used to serve the peers with less than a certain LoS. These restrictions may either be in effect all the time or may be used only during periods of heavy loads.

Scheduling policy: During periods of heavy load or even otherwise, peers may use various scheduling policies in order to give priority in serving content to the peers with with premium LoS over the peers with enhanced LoS. Similarly, they may prioritize enhanced LoS peers over those with basic LoS eligibility.

and to enhanced LoS over basic LoS.

5.3 Service Differentiation Protocol (SDP)

SDP enhances the basic functionality of Gnutella-like P2P protocols to include the service differentiation functionality. It assumes that the peers have immediate access to their own reputation scores. Assuming that the P2P network stores reputations in a decentralized manner, one way to accomplish this is through local storage of reputations. However, security issues in such a storage need to be carefully addressed. Another alternative is to compute reputations *just-in-time* from a distributed storage. These issues are described in detail in section 5.4.

SDP assumes that the mapping of the reputation scores to the LoSs is known to all the peers in the P2P network. Such a mapping could be statically configured into the software or could be downloaded from the bootstrapping infrastructure[3] when a peer joins the network. SDP is flexible about the structure of peer reputations. It only requires that the structure of the reputation scores be known to all peers.

The anonymity issues in SDP are dealt with in a manner similar to those in the popular unstructured P2P protocols. The details of this and other security issues are discussed in section 5.6.

5.3.1 SDP Details

During the bootstrapping process, most popular unstructured P2P protocols provide an option to connect only to *high-capacity* (in terms of bandwidth, processing power, memory, and storage) peers. Such high-capacity peers are referred to as *supernodes* in Kazaa and *ultrapeers* in Limewire (www.limewire.com). Additionally, *binning* scheme of the type proposed in [43] can be used to allow peers to connect to peers close-by in the Internet topology. These factors impact the quality of service (QoS) perceived by the peers and can be incorporated in a service differentiation scheme. In this paper, we focus only on the service differentiation during the search and download process of the content retrieval.

For the subsequent SDP description, we build on the Gnutella specification [24]. Consequently, all enhancements proposed by SDP are on top of such a protocol and use similar

[3]The bootstrapping infrastructure used by Gnutella is called GWebCache [27].

terminology.

5.3.1.1 Content Search:

This section describes how the content search part of unstructured P2P protocols can be modified to incorporate the service differentiation functionality using the parameters described in section 5.2.2.

Search phase 1: The peer who initiates the search request sends its reputation score along with the *Query* message. We refer to this enhanced query as *Query_SDP*. It includes the peer's reputation score in addition to the standard fields like TTL, hops, search criteria etc.

Search phase 2: Each peer who receives *Query_SDP* extracts the reputation score. This score is used to map the peer to the LoS it is eligible for and for processing the query accordingly. This mapping can be done by using parameters a and b of the type described in section 5.2. The LoS specific processing is referred to as the *searchProcess_SDP*. Since the processing is dependent only on the reputation score, SDP does not require any identification for the querying peer. Also, the peers who process the query do not have to cache the reputation scores for any other peer in this scheme. This is because the reputation scores may change over time.

Examples of functions that would be a part of *searchProcess_SDP* are shown in figure 28. These functions would provide appropriate LoS using the parameters described in section 5.2.2. Assuming three LoSs implies that there are three separate functions, one for each LoS. The functions in figure 28 assume that the number of allowable hops for basic, enhanced, and premium LoS are given by *hops_basic*, *hops_enhanced*, and *hops_premium* respectively. The basic idea behind these functions is the following. If a peer's query has already traversed more hops than it was eligible for, it is dropped immediately. However, if it has not traversed extra hops but would go farther than should (based on the TTL+Hops), then appropriate value for the TTL needs to be set. Furthermore, for enhanced and premium LoS peers, additional lookups are needed for service differentiation.

After the LoS specific processing, the query continues to be processed per Gnutella guidelines. This is denoted by *process Query* at the end of each of the functions in figure 28. Since *interest-based locality* and *load balancing* parameters from section 5.2.2 require additional protocols to be run, we eliminate them from these functions. However, if such information is available, it can be incorporated easily. The functions currently use query caching results, cached content, premium content and hard-to-find content for differentiating among the peers.

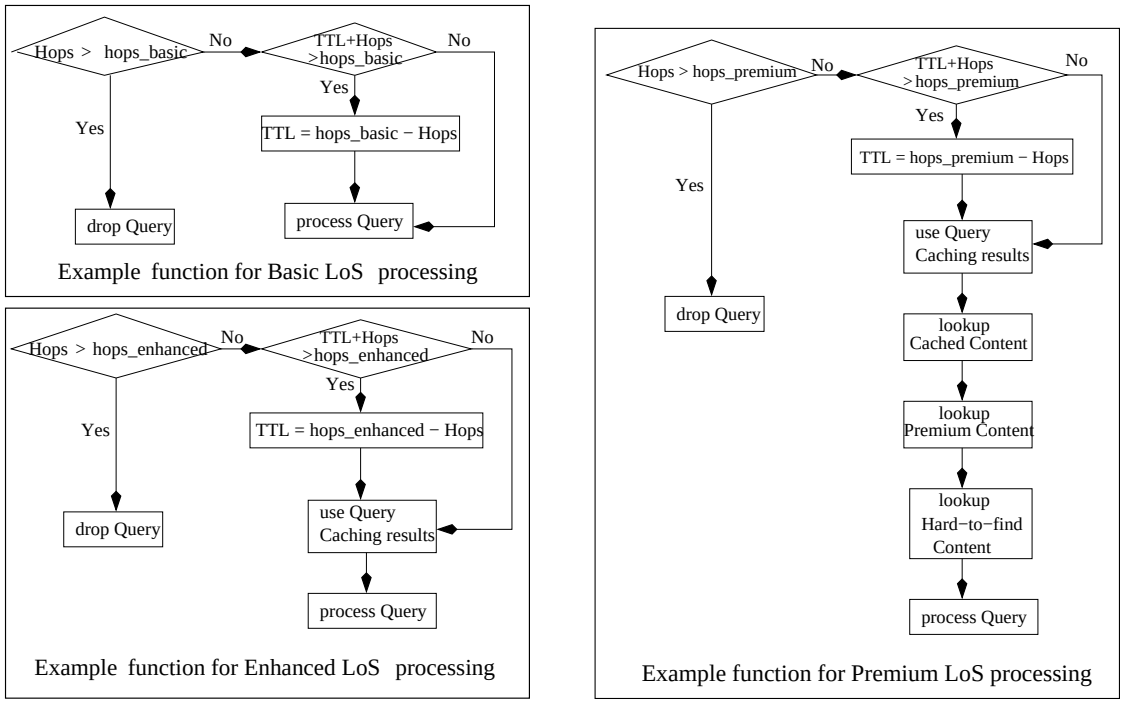


Figure 28: Functions for service differentiation during content search.

Search phase 3: Notice from the functions in figure 28 that the LoS specific query processing may amount to dropping the query. But if the query is not dropped during search phase 2, peers forward it on their outgoing interfaces according to the Gnutella specifications. In the case a response is to be sent back to the querying peer after processing the query, *QueryHit_SDP* is sent. *QueryHit_SDP* is an enhancement to the Gnutella *QueryHit* message. It allows peers who reply to optionally put their reputation scores in the response. This is to help the querying peer make a decision about who to download the content from based on the reputation of the responders.

5.3.1.2 Content Download:

This section describes how SDP enhances the content download process in unstructured P2P protocols to incorporate support for service differentiation. SDP uses the parameters described in section 5.2.3 for this phase.

Download phase 1: In Gnutella, after selecting a peer to download the content from, the querying peer connects to the selected peer using a TCP or HTTP connection. Just as in search phase 1, this phase also requires the querying peer to send its reputation score while establishing the connection for downloading.

Download phase 2: Before serving the content, the sender peer maps the reputation score to the LoS the requester peer is eligible for. Once the LoS is decided, the sender peer picks the appropriate rate of transfer and scheduling policy for the LoS.

The topic of what transfer rates to use and the particular scheduling policies employed needs more research and is beyond the current scope of this dissertation.

5.4 Desirable Reputation System Features

This section describes how the requirements of SDP translate into guidelines for a reputation system for P2P networks. It also compares the existing reputation systems for their suitability in accomplishing service differentiation using SDP. The reputation systems can be evaluated along the following dimensions:

- **Centralized or decentralized:** Storing reputations centrally is a reasonable choice for Ebay (www.ebay.com) and Slashdot (www.slashdot.org) style systems. However, having to retrieve reputations from a central location is not a scalable choice for SDP because even one search/download can lead to many peers attempting to retrieve the reputation of the requester peer.

- **Reputation inference:** The reputation systems proposed in [1, 18, 31, 35, 56] store reputations in a distributed fashion and require reputations to be computed *on-demand*. These schemes have the advantage of complete decentralization but suffer

from the following drawbacks: 1) they require cooperation from the peers in performing computations and storage, 2) latency is introduced in acquiring reputation scores, 3) as each search in SDP requires many peers to have access to the reputation scores, not having the reputation scores readily available is highly unscalable and 4) the high peer churn rates observed in deployed Gnutella-like P2P networks may introduce unpredictable inaccuracy in reputations. As a result, SDP requires that the peers store their own reputations locally, making it possible to *piggyback* the reputation values while conducting the P2P functions. Since each LoS in SDP corresponds to a range of reputation values, SDP does not require the reputations to be highly accurate, but they need to be reliable.

- **Subjective or objective criteria:** In order to keep the reputation computations distributed, all the existing decentralized reputation systems [1, 18, 31, 35, 56] are based on a subjective reputations. The recipient peers in these reputation systems assign a reputation score to the transaction based on their satisfaction from the content received. Subjectiveness is part of the reason for incurring on-demand reputation inference overheads. A mechanism that uses *objective* criteria for computing reputations could not only remove the need for on-demand reputation inference but also enable incorporating a detailed set of rules to judge a peer's cooperation in the P2P network.

- **Other features:** Reputation scores could be scalar or vector. SDP is indifferent to the structure of reputation scores as long as all the peers in the system know the interpretation of the reputation scores they receive. The range of values allowable by a reputation system is another consideration. Although several of the existing reputation systems have a limited range of values that peer reputations can assume, reputation values in most of the proposed reputation systems are *non-decreasing*. SDP can adapt to the change in the range of reputation scores in the system by changing the maping of the LoSs and hence does not prescribe a fixed range of reputation values.

5.5 Performance Evaluation

SDP can be evaluated along the following dimensions:

- **Effectiveness:** the actual differentiation in services received by peers belonging to different LoSs during the content search and download phases
- **Sensitivity to participation:** expecting that all peers in the system run SDP for it to be effective is not possible because peers could be running different versions of the the underlying Gnutella protocol and also could be malicious. As a result, gauging the sensitivity of SDP to the extent of participation required from peers is important
- **Overheads:** an estimation of overheads of SDP due to the enhancements to the Gnutella *Query* and *QueryHit* messages and extra processing on the part of peers who process the content search and download requests
- **Impact of parameter values:** the exact values of service differentiation parameters used while mapping them to various LoSs.

The *overheads* in SDP involve extra processing and sending of reputation scores along with the content search and download requests. The bandwidth consumed in sending reputations depends on their structure and the evaluation for these overheads would be specific to the reputation system used. To understand the *impact of parameter values* further research is required. As a result, the simulations presented here focus on the first two aspects from the above list.

To estimate the *effectiveness* of SDP in providing service differentiation we assume that parameters like premium content, hard-to-find content, cached queries, and cached files are constant across all the LoSs. The only parameter that differentiates among peers with different LoSs is the number of hops their queries are allowed to go. We focus only on the search aspect and assume that all the peers are running SDP. To test the *sensitivity to participation* by the peers during content search we only consider the lack of participation to mean that the peers are not SDP enhanced. As a result, they process the queries as they would in the case of Gnutella.

All simulations assume a three LoS (basic, enhanced, premium) scheme. They also assume that the peers exhibit greedy behavior in that the number of hops they specify in their searches exceed the eligibility of their reputation scores. Peers are assumed to stay in the system for the entire duration of the logs and their reputations do not change during that time period.

5.5.1 Simulation Setup

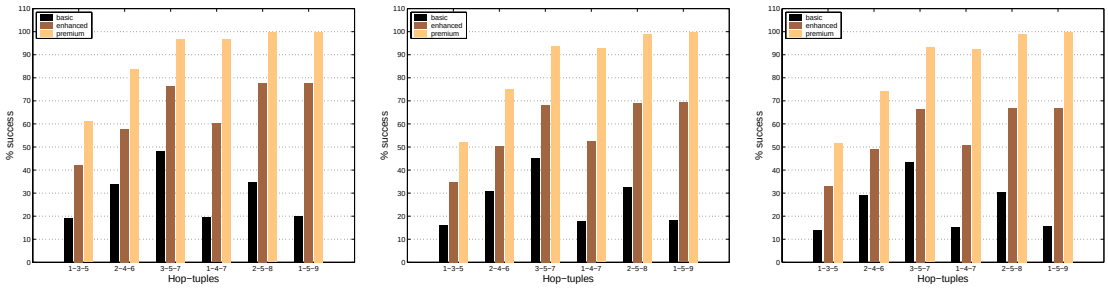
We generated connected topologies with peer populations ranging from 5000 to $25,000$. Each peer in the simulation topologies is connected on an average to about 4 other peers in the system. Compared to the actual populations of widely deployed P2P networks, these topologies are small. However, our main goal in the preliminary evaluation presented in this work is to observe the general trends in the design of SDP-like protocol.

We assume that the total number of files in each topology is the same as the peer population. The file popularities are Zipf distributed with a parameter of 1.0 and the file sizes are uniformly distributed between 0-8MBytes. At the beginning of the simulations, each peer possesses one unique file. As the peers access more files, they cache them and serve them to other peers. This leads to the file *propagation*. The *cache size* at each peer is limited, implying that peers can not cache any more than a certain number of files. The requests for files in this system are uniformly distributed among all the peers with an exponential inter-arrival time of 50 requests/second. The simulation logs are 1 hour long for each topology.

For the evaluations, we experimented with population sizes of 5000, $10,000$, and $25,000$ peers in the system. We assumed three combinations of the percentage of peers eligible for basic, enhanced, and premium LoS respectively. These combinations were $(20, 20, 60)$ $(40, 40, 20)$, and $(60, 20, 20)$. Subsequently, we refer to each of them as the *percentage-tuple* for simplicity. The first, second, and third entries of each percentage-tuple correspond to the percentage of peers eligible for basic, enhanced, and premium LoS respectively. For each population size and each percentage-tuple, we experimented with various cache sizes (50-1000 files) and various sets of *hop-tuples*. Each hop-tuple (i, j, k) corresponds to the

maximum number of hops the queries for basic, enhanced, and premium LoS peers are allowed to go respectively. With all other parameters staying constant, the effect of varying population sizes was that the smaller the peer population in the topology, the more the successes during content search (and hence lesser failures). This is expected because the effects of file propagation are visible much sooner for smaller topologies. Also, for each topology, the results of varying the cache sizes were identical. We now present the results of varying the percentage-tuple and hop-tuple for the topology with a population size of $10,000$ and a cache size of 1000 files.

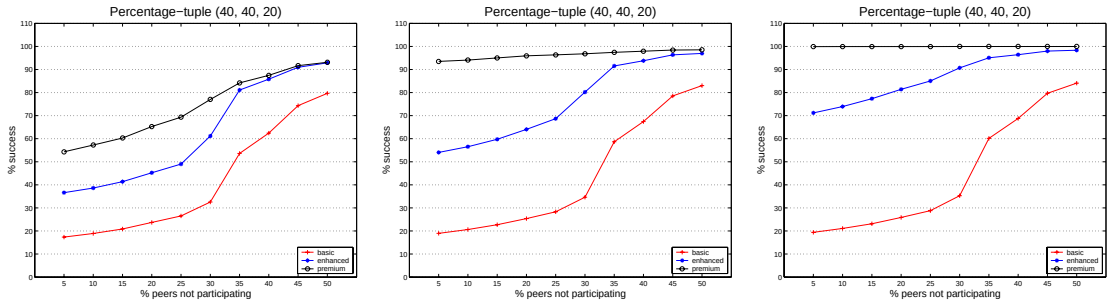
Figures 29(a), 29(b), and 29(c) show the effect of hop-tuple and percentage-tuple on the success of queries for the basic, enhanced, and premium LoS peers respectively. Irrespective of the particular hop-tuple, the success rates for premium LoS peers are the highest. The success rates for enhanced LoS peers are better than those of basic LoS but worse than the premium LoS peers. This shows that for the topologies tested, SDP is able to provide service differentiation for content search using just the number of hops. Comparing across graphs for peers belonging to the same LoS respectively, we note that for basic and enhanced LoS peers, as the percentage of peers eligible for the same service increases, the success rate deteriorates. This is expected because the presence of more basic and enhanced peers implies fewer premium peers and less successes during content search. The total effect is that of lesser propagation of files. Also, as expected, as the number of hops for an individual LoS increase, the corresponding success rates improve as well.



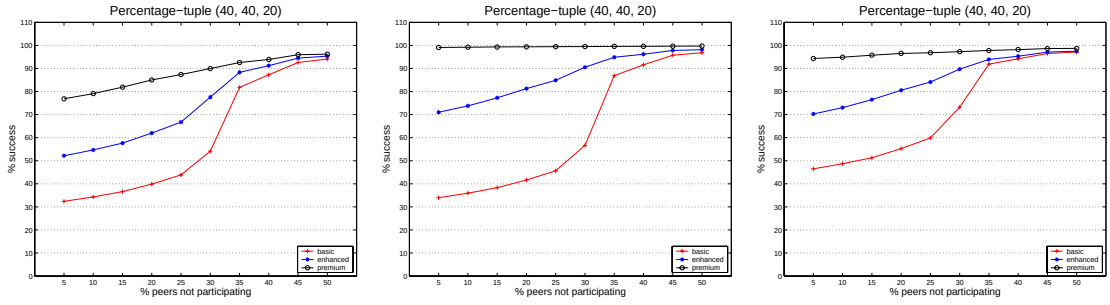
(a) %-tuple (20,20,60). (b) %-tuple (40,40,20). (c) %-tuple (60,20,20).

Figure 29: Effectiveness of SDP during content search.

Figures 30(a), 30(b), 30(c), 30(d), 30(e), and 30(f) show the effect of (lack of) participation on the effectiveness of SDP during content search for percentage-tuple $(40, 40, 20)$ and hop-tuples $(1, 3, 5)$, $(1, 4, 7)$, $(1, 5, 9)$, $(2, 4, 6)$, $(2, 5, 8)$, and $(3, 5, 7)$ respectively. The non-participating peers were preferentially chosen from peers with lower LoS eligibility since those peers are likely to lack the motivation to participate. The graphs for percentage-tuples $(20, 20, 60)$ and $(60, 20, 20)$ were similar.



(a) Hop-tuple: (1, 3, 5). (b) Hop-tuple: (1, 4, 7). (c) Hop-tuple: (1, 5, 9).



(d) Hop-tuple: (2, 4, 6). (e) Hop-tuple: (2, 5, 8). (f) Hop-tuple: (3, 5, 7).

Figure 30: Effect of participation on SDP during search.

The first observation from these graphs is that as the difference in hops between basic, enhanced, and premium LoS increases, so does the difference in the % of successes for queries. This is expected because a query that traverses more hops has a higher chance of being successful. Also, as a smaller percentage of peers participates in SDP, the overall trend in all the graphs is that of decreased service differentiation among the basic, enhanced, and premium LoS peers. Under the assumption that the peers are greedy, the presence of more

non-SDP peers means that more peers with lower LoS will be able to get content search replies from farther away than their LoS justifies. For all the figures 30(a), 30(b), 30(c), 30(d), 30(e), and 30(f), when the percentage of peers not participating increases beyond 30-35%, the difference in service received by peers belonging to various LoSs diminishes very quickly.

5.6 Discussion of Security and Participation Issues

Decentralization is one of the core strengths of the P2P networks because it makes them robust to failures. However, the same decentralization makes it hard to enforce rules in any light-weight manner. As a result, enforcing the service differentiation proposed in this paper is not possible without incurring high overheads. We now discuss the security and participation issues for SDP.

Security Issues: Malicious peers can thwart SDP in two primary ways. First, they may not give better LoS to peers who are eligible for enhanced or premium LoS. Second, they may collude with other peers and give each other a better LoS than their reputation justifies. The latter is not a serious issue because peers often interact with a large number of other unknown peers. However, unless behavior of all peers is tracked, the occurrence of the first problem can not be eliminated. Current P2P networks do not have any provisions for isolating misbehaving peers and work on the goodwill of majority of the peers. SDP functions on the belief that if a peer trusts the reputation of the other peer, it will be willing to provide them with appropriate LoS. Moreover, the rewards from malicious behavior while providing service differentiation are limited unlike the one's that are possible by being able to assume a good reputation score without earning it.

Another important issues is that of anonymity. Gnutella like protocols do not address issues related to anonymity. Since SDP does not change the anonymity characteristics of the underlying protocol, it does not provide any anonymity to the peers. However, since SDP makes the service differentiation decisions based only on the reputation scores and not on who sent it, it is independent of whether or not the underlying P2P network provides anonymity.

Participation Issues: The lack of participation in SDP may not necessarily arise out of malicious behavior. Some peers may fail to participate because they may be running a non-SDP enhanced version of the P2P software. As the evaluation results show, SDP is relatively insensitive to a small percentage of peers not providing the appropriate LoS. During the content search phase, the redundancy of results may offset the effect of such peers because peers who do not send their reputation scores with their search replies may be avoided for content download.

5.7 Conclusion

In this chapter we presented SDP, a protocol for service differentiation in Gnutella like P2P networks. Preliminary simulation results show that SDP is able to create service differentiation during content search and that the service differentiation is sensitive to the percentage of non-participating peers in the system. When the percentage of non-participating peers increases beyond a certain point, the distinction across peers belonging to various LoSs vanishes rapidly.

In this work we proposed a set of parameters that can be used by SDP to provide different LoSs. These parameters can affect the outcome of content search and download in Gnutella-like networks. More research is required in order to assess the impact of individual parameters on service differentiation. This work is also preliminary in terms of the mapping of parameters to the various LoSs.

The eligibility of peers for various LoSs in SDP is decided based on their reputations. To function efficiently, SDP requires the peer reputations to be universally comparable and locally stored. Further, the reputations need to be reliable, but not highly accurate. In chapter 6, we discuss the design of a reputation system that fits these criteria.

CHAPTER VI

RELIABLE REPUTATIONS FOR PEER-TO-PEER NETWORKS

6.1 Introduction

The peer-to-peer (P2P) networks like Gnutella and Kazaa can perform large-scale content distribution without the need for dedicated high-capacity servers. However, among other issues, these networks suffer from *lack of trust* and *free-loading*[1]. The content quality in P2P networks suffers due to the lack of trust because a downloading peer cannot be certain about the quality of content it receives.

The ability to associate reliable reputations with peers that are based on the contributions of peers in the system can greatly improve trust in P2P networks, and hence content quality. Reliable reputations are also a basis for the service differentiation protocol (SDP) described in chapter 5. SDP can serve to motivate free-loaders to be cooperative members of the P2P community. Further, reputations can be used in making decisions about who to serve content to and who to request content from. When a peer comes online, it can potentially use reputations to decide who to directly connect to in the overlay topology.

In this work we investigate the design of a reputation system in which a peer's *reputation* is based on its past behavior. Tracking peer reputations in a centralized P2P network like Napster is not difficult because the search for content is facilitated by a central server. The lack of any central authority in the functioning of decentralized P2P networks makes the problem of accurate reputation tracking a challenging one. In fact, we find that a certain amount of centralization is necessary to build a viable reputation system. Since unstructured P2P networks are the most prevalent today, the reputation system proposed in this work mainly focuses on unstructured decentralized P2P networks like Gnutella, but

[1] *Free-loaders are peers who only download content but do not serve it to the other peers.*

it can easily be adapted for structured decentralized P2P systems also.

Reputations can be tracked either *objectively* or *subjectively*. In subjective reputation systems ([1], [18], [31], [35], [56]) peers use their own opinion about a transaction to associate a score with the transaction. Subjective reputations have the advantage that the reputations can be tracked in a completely decentralized manner but they require cooperation from peers in storing the reputation scores. Also, the reputations so computed are not universally comparable, a requirement of SDP (chapter 5). Inferring the reputation of any peer in a system that tracks reputations subjectively requires on-demand computations which could have high overheads and unknown amount of inaccuracy due to unpredictable peer churn rates. Table 3 highlights the main differences between objective and subjective reputations.

Table 3: Distinguishing features of subjective and objective reputations.

	Subjective reputations	Objective reputations
Decentralized reputation computations possible	**Yes**	No
Universally comparable reputations	No	**Yes**
On the fly computation required for inferring reputations	Yes	**No**
Cooperation required for storage and computation of reputations	Yes	**No**

Our work takes the approach of defining a set of objective criteria for tracking universally comparable reputations. The proposed mechanism, *debit-credit reputation computation* (DCRC), essentially tracks the resources contributed to and used by the peers during the normal P2P functions of content search and download by means of *non-negative* points that represent a peer's reputation score. It credits peer reputation scores for serving content and debits for downloading. It also offers additional credits for query processing and forwarding, and staying online.

For the reputation scores to be reliable, they have to be updated and stored securely to prevent malicious peers from thwarting the reputation system. An ideal solution will be light-weight, completely distributed, and compute trustworthy reputation scores that can be locally stored. Since a fully distributed solution does not seem possible, we introduce a

partially distributed solution that uses an infrastructure of reputation computation agents (RCAs) to offer reliable reputation tracking by defeating attacks from *selfish* peers[2]. This design not only renders reliability to the reputations but also allows reputations to be stored *locally* (another requirement of SDP) at the peers for fast retrieval. We *formally specify* and *verify* the reliability properties of the DCRC scheme.

6.2 Details of the Reputation System

In decentralized unstructured P2P networks like Gnutella, content retrieval involves a *content search* phase and a *content download* phase. To search for the desired content, a peer generates a query with appropriate *keywords* and sends it to all the peers that it is directly connected to in the Gnutella overlay topology. The peers who process this query reply back if they have the content in their shared directory or forward the request to the peers they are directly connected to depending on the *hop count* of the query. This forwarding continues until the hop count specified by the querying peer is exhausted. Once the querying peer receives all the replies, it selects a peer to download the content from. At that point, the content download typically uses a HTTP or a TCP connection.

The success of the search phase in Gnutella-like P2P networks requires that the other peers be online, agree to search for the content from their shared directory, and forward the query further depending on the hop count of the query. The success of the download phase requires that the chosen peer be online and serve the content when requested. For successful content retrieval, the type, quality, and quantity of the content each peer places in the shared directory plays an important role. The reputation system proposed here essentially measures each peer's participation in the system based on these objective factors.

Next, we describe the DCRC scheme which credits peer reputation scores for serving content and debits them for downloading. Additionally, it offers credits for query processing and forwarding, and staying online. In a perfect world, each peer's local software can update and store its reputation score. However, this simple mechanism could be thwarted by the

[2]The security threats to reputation tracking arise from peers that are *malicious* and/or *selfish*. The goal of malicious peers is only to attack the system, not to benefit from it. Our focus is on selfish peers whose goal is to maximize their reputations.

peers by altering the score computations to their benefit or by tampering with the value of the stored counter. Our solution to prevent such occurrences is discussed in section 6.3. For now, for describing how reputations are computed in the DCRC scheme we assume that the updates to reputation scores take place locally.

6.2.1 Debit-Credit Reputation Computation (DCRC) Scheme

DCRC uses two tunable system parameters: 1) *file size factor* f, $f \in integer$ and 2) *time factor (in hours)* t, $t \in integer$. The file size factor, f, determines how many MBytes of data transfer results in a unit increment to the reputation score and the time factor, t, is used to determine the granularity at which peer cooperation in the system for sharing and staying online is rewarded. These parameters ensure that the reputation scores stay within a certain range of non-negative values. They do so by tuning the system for larger file sizes, and longer stay in the system. Utilizing these parameters, a peer's total reputation score is computed using the following four components:

Query-Response Credit (QRC): The DCRC scheme uses average query-response message size to credit peer reputations for processing the query-response messages. If the average query response message size is denoted by QR, the number of points earned for each query processed are given by:

$$QR$$

It has been reported [44] that the average query-response traffic in a Gnutella network is about .75 KB per second per connection. Also, most connections generated about 15 messages per second. This gives a rough estimate of the average query-response message size to be 0.00006 MBytes. This value can be used to specify QR.

Upload Credit (UC) and Download Debit (DD): Peers get credit for serving content and debit for downloading it. For a file of size s MBytes the debit and credit to the reputation score is given by:

$$\frac{s}{f}$$

Sharing Credit (SC): During the content search phase, in addition to peer availability, another important factor is the amount of content shared. Some peers may be sharing *hard-to-find* content. QRC, UC, and DC may not give any credit to such peers because by definition, such content may not be heavily accessed. SC is intended to capture this effect. In practice, it is very hard to implement this correctly in a light-weight fashion[3]. But assuming it can be implemented, if a peer shares n files where the size of jth file is given by s_j, at the elapse of each time factor the number of points it will earn are given by:

$$\sum_{j=1}^{n} \frac{s_j}{f}$$

The total reputation score for a peer k who processes a query-response messages, facilitates b uploads, performs c downloads in d time factors is given by:

$$\text{Reputation Score}_k = (a \times QRC + \sum_{l=1}^{b} UC_l - \sum_{m=1}^{c} DD_m + d \times SC)$$

where UC_l and DD_m are the upload credit and download debit for files l and m respectively.

6.3 Reliable Reputation Computations

As the local computation of reputations using the objective criteria described in 6.2.1 are not reliable, DCRC utilizes an infrastructure of RCAs. We first discuss the assumptions about the functionality of the RCA infrastructure (section 6.3.1) and the terminology used (section 6.3.2) before the details (6.3.3).

6.3.1 Infrastructure

Gnutella uses an infrastructure of bootstrapping servers [27] to bootstrap new peers into the network. The functionality of the existing bootstrapping servers can be enhanced to incorporate the functions required of the RCA infrastructure. For simplicity of subsequent description, we assume that the RCA infrastructure is a single entity. Issues in the division of functionality among various RCAs are discussed in section 6.5.

[3]For this reason for the rest of this paper we assume SC is not provided.

A peer who is interested in getting its reputation tracked first enrolls itself with the RCA to get a (public, private) key pair. This distribution of (public, private) key pair can be thought of as similar to that of public key infrastructure (PKI) in the sense that only one *enrolled* identity is permissible per peer[4]. Alternately, a scalable public key distribution infrastructure such as the one proposed in [37] can also be used, if one is available.

The digest of a peer's public key is used to identify it. We assume that the (public, private) key pair of the RCA is denoted by $\{PK_{RCA}, SK_{RCA}\}$ and that each peer has access to the RCA's public key and that the peers can obtain public keys of other peers in the system when needed.

The RCA is expected to have a copy of all content served by the enrolled peers for the purposes of ensuring content reliability. The creators of the content can provide the RCA with the content when they initialy create it. Further, we assume that the RCA is not malicious but peers can collude with other peers in self-interest.

The DCRC scheme involves additional overheads to keep the most up-to-date view of each peer's reputation which some peers may not want to incur. Also, some peers may not want to get their reputation tracked for privacy reasons. Existing designs of these networks do not provide peer anonymity and our goal in this work is not to propose alternate designs for Gnutella style P2P networks. For these reasons, enrollment in the reputation computations is *voluntary*. Peers who choose not to enroll always maintain a default reputation score of 0, the minimum allowable by the system. Peers who enroll can enhance their scores by being good citizens of the P2P network. They can also save their reputation scores across sessions. Thus, a cooperative peer can maintain benefits of its participation in the system in spite of being off-line for a while.

6.3.2 Terminology

A P2P system comprises of only one type of entities - the peers. All the peers in a P2P system are treated homogeneously in spite of the heterogeneity because each peer is capable of performing the task of both a client as well as a server. The introduction of voluntary

[4]A peer can however generate any number of *unenrolled* identities.

enrollment in reputation computations distinguishes between peers that are enrolled and peers that are unenrolled. With the introduction of the RCA to perform reputation computations reliably, a P2P system now can be thought of as three types of entities: 1) enrolled peers, 2) unenrolled peers, and 3) RCAs.

Depending on the stage of the content retrieval and the role of a peer (enrolled or unenrolled), its designation changes. We refer to the peer that generates the query as the *querying peer/querier* in this paper. All the peers that receive and process the query are called *searching peers/searchers*. Once the querying peer receives all the replies, it chooses a peer to download the content object from. At that point, it becomes a *downloading peer/downloader*. The peer that serves the content is referred to as the *serving peer/server*. The (public, private keys of querier, searcher, downloader, and server are denoted by $\{PK_{query}, SK_{query}\}$, $\{PK_{search}, SK_{search}\}$, $\{PK_{download}, SK_{download}\}$, and $\{PK_{serve}, SK_{serve}\}$ respectively.

6.3.3 Details

The reputation tracking in the DCRC scheme is essentially done in two steps: 1) searchers save proofs of their contributions during content search to collect query-response credit (QRC) and the RCA saves *transaction state* about content download that can be converted into upload credit (UC) for the servers and 2) searchers periodically send QRCs to the RCA. The RCA processes the QRC for the searchers, UC for the servers, and the corresponding download debit (DD) for the downloaders. It then sends the encrypted QRCs and UCs as reputations to each for keeping locally but stores the DDs with itself to ensure they are not dropped by the recipient peers.

6.3.3.1 Proofs of Peer Contributions

To save the proofs of their contributions, for every query-response message processed during content search the enrolled searchers save $\{$*searcher_identity*, *querier_identity*, *query_keywords*, *time_stamp*$\}_{SK_{query}}$ as the *proof of searching (pSearch)*.

The RCA facilitates content download for reliability of content delivery. The following protocol takes place between an enrolled downloader and enrolled server at the time of the

file download[5]:

Step 1: The downloader sends a request for content to the server, denoted by *DownloadReq*. This message contains $\{$*downloader_identity, content_identity, time_stamp*$\}_{\mathrm{SK_{download}}}$.

Step 2: The server generates a session specific symmetric key called *SymKey* using the information in *DownloadReq* and encrypts the content with it. This encrypted content is then sent to the downloader as *ContentDelivery*. Encrypting the content guarantees that the downloader is not able to access the content unless it has access to the SymKey.

Step 3: The server sends *KeyTransfer* message which contains $\{\{$*server_identity, downloader_identity, content_identity, time_stamp*$\}_{\mathrm{SK_{serve}}}$, *SymKey*$\}_{\mathrm{PK_{RCA}}}$ to the RCA. Encryption by PK_{RCA} ensures no one but the RCA is able to decrypt the information and signing by SK_{serve} ensures that the identity of the server is verified. This message is necessary because neither the downloader nor the server are trusted entities of the system. As a result, a trusted third party is necessary in order to ensure a fair transaction between them.

Step 4: After receiving the encrypted content, the downloader sends *ContentRcvd* message containing $\{$*server_identity, downloader_identity, content_identity, encrypted_content_digest, time_stamp*$\}_{SK_{\mathrm{download}}}$. This message confirms that the downloader received the encrypted content. The *time_stamp* in this message indicates the time at which the downloader received the encrypted content. The digest is useful in verifying the integrity of the encrypted content and can be produced by using an algorithm like MD5 [45].

Step 5: After decrypting the *ContentRcvd* message from the downloader, the RCA first encrypts the content locally stored under *content_identity* (using SymKey from the *KeyTransfer* message) and calculates its digest. The matching of *content_identity* ensures that the downloader is not tricked into receiving any other content than the one it selected to get. We assume that the RCA uses the same algorithm to compute the digest as the downloader. If the digests match, the integrity of the content is verified.

To complete the transaction between the serving and requesting peers, the RCA can now pass the SymKey key securely to the downloader. It sends a *KeyDelivery* message

[5]It is important to note that if *any* of the peers are not enrolled, the following exchange does not take place. This means that by creating additional *unenrolled* identities peers cannot earn any credit to their reputations.

with $\{content\_identity,\ RCA\_ID,\ SymKey\}_{\mathrm{PK}_{\mathrm{download}}}$ to the downloader. Encrypting this message with downloader's public key ensures that the message can not be snooped upon and that only the downloader is able to decrypt it. At this point, the downloader has received both the content and the SymKey and can decrypt the content.

The RCA does not store the SymKey with itself after delivery. However, it does store *transaction state* of the form (*downloader_identity*, *server_identity*, *file_name*, *file_size*, *time_stamp*, *credit_processed_list*) for reputation credit inference purposes. The credit_processed_list is the list of peers who have already received the credit. It could have multiple entries for each peer depending on what type of credit it has already received. For example, the list could have one entry each for QRC, UC, and DD for each peer. The RCA uses this list to ensure that peers receive each type of credit only once.

Figure 31 shows the communication between the RCA, searcher, and the downloader in the above protocol.

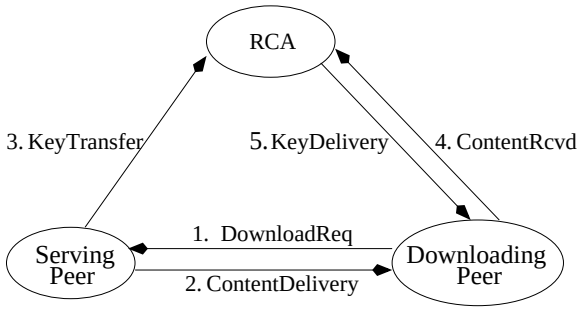


Figure 31: Protocol for secure content download.

6.3.3.2 Processing at the RCA

Periodically, the enrolled queriers send the pSearchs collected thus far to the RCA. Using the pSearchs and the transaction state, the RCA processes the QRCs, UCs, and the DDs.

Processing upload credit (UC) and download debit (DD): Using the criteria prescribed in section 6.2.1, the RCA infers the total UC for the servers and the corresponding DD for the downloaders. It then updates the credit_processed_list for the relevant entries of the transaction state with the identities of the peers that received UC and DD and sends an encrypted reputation score of the form {*RCA_identity*, *time_stamp*, *reputation_score*,

server_identity$\}_{SK_{RCA}}$ to the serving peers. The encryption allows the peers to store their own reputations locally without being able to tamper with them. To avoid having the downloaders drop negative reputation scores, the RCA retains the DDs in the form of *debit state* with itself until those peers send some credits for processing.

Processing query-response credit (QRC): Since peers can forge QRCs for the files that were never downloaded, the QRCs from the pSearchs are not processed until the corresponding UCs and DDs are processed, irrespective of when they arrive. Also, the RCA uses the topology snapshots it maintains while processing the pSearchs. After inferring the QRCs using pSearchs and the transaction state, the RCA updates the credit_processed_list and sends an encrypted reputation score of the form $\{$*RCA_identity*, *time_stamp*, *reputation_score*, *searcher_identity*$\}_{SK_{RCA}}$ to the peers that sent the QRCs.

Processing sharing credit (SC): Without active monitoring, SC cannot be processed for the searchers in a manner that gives each peer credit for staying online to serve the hard-to-find content and the overheads of monitoring could be prohibitive. An alternative is to use the transaction state to infer how long a peer was online and process the SC. As in the case of UC and QRC, the RCA updates the credit_processed_list and sends an encrypted reputation score to the appropriate peers.

6.3.4 Formal Specification

The formal specification of the DCRC scheme using a version of *Abstract Protocol Notation* [25] is presented in the APPENDIX. It contains a process each for the 5 entities: *queriers*, We assume that there are n peers and m distinct files in the system and that the encryption and decryption are denoted by NCR and DCR respectively.

6.4 Attack Analysis

For peers that behave in self-interest, we prove that the mechanisms to ensure the reliability of reputations for enrolled peers in the DCRC scheme satisfy the following properties:

1. *only peers that contribute to the search and download should be able to collect credit.*

2. *no downloading peer should be debited unless it receives the content completely.*

3. *no serving peer should be able to collect credit without serving the content completely.*

4. *the credit and debit settlement should occur at most once for each content download.*

We divide collusion scenarios into two types: 1) those where peers acting out of self-interest collude to increase each other's reputation scores and 2) those where the increase in one peer's reputation score results in a penalty to the reputation score of the peer facilitating it. The above properties are only valid for the first scenario. The peers contributing to the second type of collusion do not act out of self-interest but do not fall under the category of malicious peers either. Section 6.5 discusses the implications of the second type of collusion.

6.4.1 Attacks and Actions

To defeat the above security goals an adversary can mount the following attacks to compromise the reliability of reputation computations:

- **a0:** earn credit without doing any work in the system.
- **a1:** earn credit for useless work.
- **a2:** earn credit for partial file transfer.
- **a3:** earn credit instead of someone else (with or without their consent).
- **a4:** earn credit more than once for the same work[6].
- **a5:** get content without earning a debit.

To launch each of the above attacks, a selfish peer can use one or more *actions*. These actions can broadly be divided into 3 categories: 1) *identity related*, including *impersonation* and *repudiation*, 2) *IP address related*, including *spoofing* and *TCP hijacking*, and 3) *message* and *content related*. Out of these three categories of actions, only the last one can be used to mount attacks on on the DCRC scheme. The identity related actions are ruled out because each enrolled peer is uniquely identifiable by the digest of its public key and the process

[6]Since every credit has a debit, earning credit more than once implies some other peer receives undue debit.

of (public, private) distribution is assumed to be safe. The IP address related actions are not of concern because a peer is identified only through the digest of its public key and is allowed to legitimately change IP addresses while maintaining the same identity. We next formalize the definitions of possible message and content related *actions* that can be used by the selfish peers to launch attacks on DCRC:

Forgery: Generation of fake messages or content by an enrolled peer in collusion with, or without another enrolled peer.

Modification: Alteration of messages or content in transit through interception.

Replay: Retransmission of a valid message either by the originator or by another peer who snooped on the message in transit.

6.4.2 Formal Verification

Figure 32 shows the flow of messages in the DCRC scheme between the *querier (q)*, *searcher (sc)*, *downloader (d)*, *server (sr)*, and the *RCA (r)*. The content search and download related messages are shown by solid lines and the debit-credit settlement messages are shown by dashed lines.

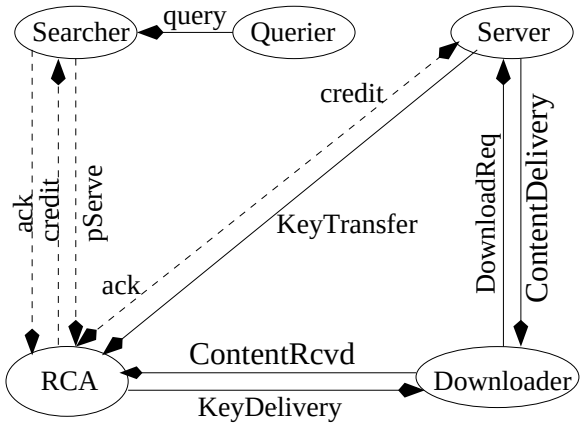


Figure 32: Interactions among the five entities.

The interactions among the five entities in figure 32 can be divided in three main categories: 1) between the querier q, searcher sc, and the RCA during content search and the corresponding credit settlement (denoted by $q-sc-r$), 2) between the downloader d, server sr, and the RCA r during content download (denoted by $d-sr-r$), and 3) between the

RCA r, downloader d, and the server sr during debit-credit settlement (denoted $r-d-sr$). Table 4 shows a listing of possible attacks during each of the three interactions.

Table 4: Attacks possible during each interaction.

	$a0$	$a1$	$a2$	$a3$	$a4$	$a5$
$q-sc-r$	✓			✓	✓	
$d-sr-r$		✓	✓	✓	✓	✓
$r-sr$	✓			✓		

We use *convergence theory* to verify the reliability of the DCRC scheme. In convergence theory, a protocol is called *secure* if it satisfies the following three conditions [34]:

Closure: In each protocol computation whose first state is *safe*[7], every state is safe.

Convergence: In each protocol computation whose first state is *unsafe*[8], there is a safe state.

Protection: In each protocol transition whose first state is unsafe, the critical variables of the protocol do not change their values.

As argued in [34] each protocol satisfies the closure condition. This is because the protocol *states* are defined from a valid domain of values and the *transitions* occur only when actions whose guards are true are executed. Hence, to formally prove a protocol is secure, it is sufficient to show that the protocol satisfies the convergence and protection conditions.

For the three categories of interactions, we now describe how the attacks described in table 4 are launched using forgery (F), modification (M), and replay (R) and how the security built in DCRC counters them. In the subsequent discussion and in figure 33, the safe states are denoted by Si, $i \in \mathcal{I}$, with S_0 being the initial state for each interaction. The unsafe states are denoted by Ui, $i \in \mathcal{I}$ and the actions of forgery, modification, and replay are labeled as F, M, and R respectively.

[7] A computation of a protocol is an infinite sequence $(p0, p1, ...)$ of protocol states such that each pair $(pi, p(i+1))$ of successive states in the sequence is a protocol transition. A state is *safe* if it occurs in any protocol computation $(p0, p1, ...)$ where $p0$ is an initial state of the protocol.

[8] A state is *unsafe* if it can be reached by some adversary action starting from a safe state, or if it occurs in some protocol computation $(p0, p1, ...)$ where p0 is an error state of the protocol.

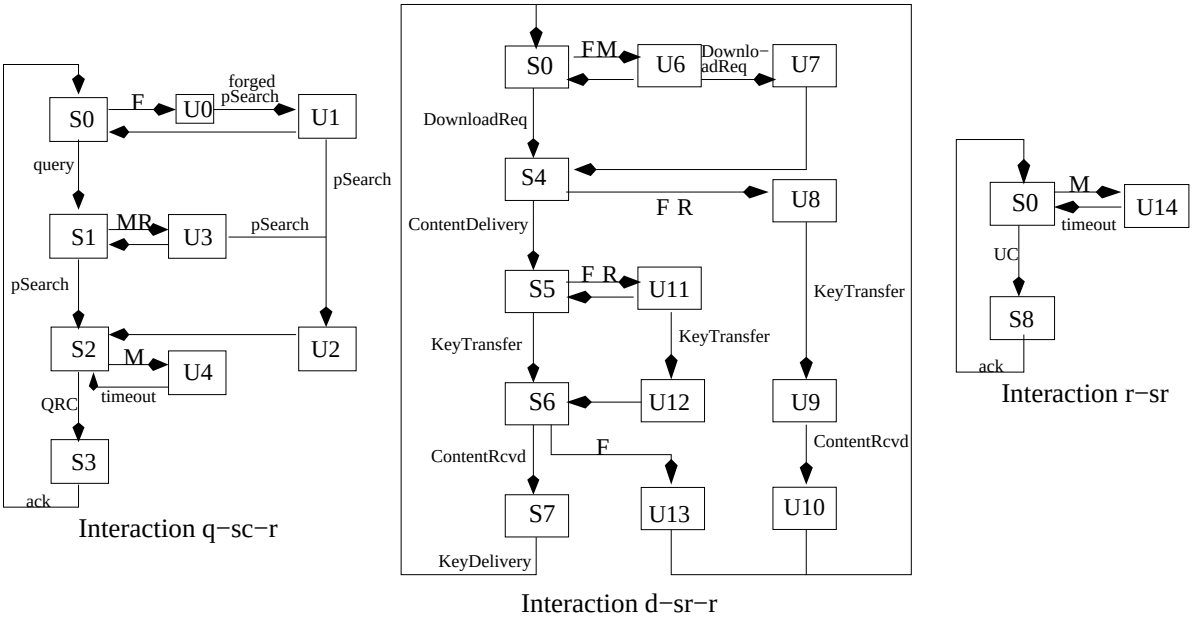


Figure 33: State transition diagrams

6.4.2.1 Interaction q-sc-r:

Attacks $a0$, $a3$, and $a4$ can be launched by selfish peers during credit settlement for content search contribution.

The protocol starts in state $S0$ when a searcher sends out a query in search for content. During normal operation of the protocol, the enrolled searchers save pSearchs as a proof of their work, leading the protocol to move to $S1$. When these pSearchs are sent to the RCA, the protocol moves to state $S2$. Upon receiving the pSearchs, the RCA infers the query-response credit (QRC) and sends them to the searchers, leading the protocol to state $S3$. Finally, when the searchers send acks for receiving QRC, the protocol returns to state $S0$.

From $S0$, the protocol moves to unsafe state $U0$ instead if selfish enrolled peers save forged pSearchs and launch attack $a0$. There are several possibilities for forging pSearchs: 1) peers can guess the key words, time stamps, and the entities involved in valid queries in the system (given that the number of files and peers in a P2P network are likely to be large, forging legitimate transactions and their time stamps is not possible), 2) they can snoop on queries not encountered in their section of the network, or 3) they can obtain information

about valid queries from other colluding peers. When the forged pSearchs are sent to the RCA, unsafe state $U1$ results. However, since the RCA maintains snapshots of connectivity information, the forged pSearchs fail to earn any credit and the protocol returns to the initial state $S0$. In state $U1$ however, if a searcher sends pSearchs, unsafe state $U2$ results. The RCA can identify a valid pSearch from a forged one using the topology snapshots. As a result, effectively the protocol is in safe state $S2$ and the normal operation of sending the corresponding QRC can be carried out, defeating attack $a0$.

In state $S1$, selfish peers can launch attacks $a3$ and $a4$ by modifying valid pSearchs on the way to the RCA and by replaying old pSearchs respectively. Either of these actions cause the protocol to move to unsafe state $U3$. The transaction state maintained by the RCA helps avoid replays and the signing of pSearchs rules out the possibility of modification. The result is that the protocol returns to safe state $S1$, defeating the attacks. If however, in $U3$, a valid pSearch arrives at the RCA, the protocol moves to unsafe state $U2$. As the modified and replayed pSearchs fail to fetch credit,the net effect is that the protocol returns to state $S2$ and the valid credits can now be processed.

Attack $a3$ can also be launched in state $S2$ when QRCs are being sent out. An enrolled selfish peer can modify a QRC destined for a searcher leading the protocol to an unsafe state $U4$. Since the QRCs are signed, modified QRCs are not of any use. Also, not receiving an ack for the QRC causes the RCA to timeout and the protocol resumes in state $S2$. It is important to notice that QRCs can also be snooped upon in transit. However, that does not cause any interruption to the protocol operation.

6.4.2.2 Interaction d-sr-r:

The interaction between the downloader, server, and the RCA is susceptible to attacks $a1$ through $a4$.

The interaction during content download content starts in state $S0$. When a downloader sends a *DownloadReq* to the server, the protocol progresses to state $S4$. Upon *Content-Delivery*, the state changes to $S5$. The *KeyTransfer* message leads the protocol to state $S6$. When the downloader sends *ContentRcvd* message, state $S7$ results. Finally, when the

RCA sends the *KeyDelivery* message, the protocol returns to its original state.

To launch attack $a5$ a selfish enrolled peer can modify the *DownloadReq* or a selfish searcher could forge an incorrect *DownloadReq* to hide its identity to avoid debit. These actions lead the protocol to unsafe state $U6$. However, since *DownloadReq* is signed, the attack is avoided and the protocol returns to safe state $S0$. If however, a valid *DownloadReq* arrives in state $U6$, unsafe state $U7$ results. The RCA identifies such a request and safe state $S4$ results.

While in state $S4$, a selfish server can launch attacks $a1$ and $a2$ by delivering incorrect or partial content through actions F and R and cause the protocol to reach unsafe state $U8$. Though the server may not be able to save itself much effort, it may do so in the event it does not possess the actual content the downloader selected to get from it. In $U8$, the selfish server can send the *KeyTransfer* message to the RCA which results in state $U9$. The *SymKey* in this message may or may not be useful to decrypt the content previously delivered but that detail is immaterial because the content is not what the downloader expected to get. When the downloader sends the *ContentRcvd* message to the RCA with the digest of the received content, unsafe state $U10$ results. At this point, the RCA checks the message digest of the actual content with the one in *ContentRcvd* and concludes that incorrect content has been sent to the downloader. The effect is that the downloader times out waiting for the *KeyDelivery* message from the RCA and the protocol returns to the initial state $S0$, defeating the attacks.

In state $S5$ after *ContentDelivery* by a server, a selfish peer can launch attacks $a3$ and $a4$ to claim the upload credit (UC) instead of the server (perhaps because it also has the content denoted by *content_id*). It can do so by forging the *KeyTransfer* message or replaying an old one and causing the protocol to enter unsafe state $U11$. If the *KeyTransfer* message was just encrypted, the RCA could not distinguish a valid one from a forged or replayed message. However, since the *KeyTransfer* message includes a signature that identifies the server, downloader, time stamp, and the content_id, these attacks will fail and the protocol will resume in state $S5$. There is a possibility that a valid *KeyTransfer* message will arrive at the RCA in state $U11$, causing the system state to change to $U12$. Since the RCA can

distinguish a valid message from a forged or replayed one, the net effect is for the protocol to resume in safe state $S6$.

To avoid debit for the received content, a selfish downloader can launch attack $a5$ in state $S6$. It can do so by forging its identity in the *ContentRcvd* message, causing the protocol to enter unsafe state $U13$. Since in this case, the identity of the downloader in the *ContentRcvd* message will fail to match that in *KeyTransfer* message, this attack will fail and the protocol will revert back to safe initial state $S0$.

Since *KeyDelivery* is encrypted by the RCA, no attack can be launched in $S7$. It is also important to note that snooping of *ContentDelivery*, *KeyTransfer*, and *KeyDelivery* is possible by selfish peers in order to launch attack $a5$ but can cause no harm to the protocol because all three are encrypted. As a result, no state changes are required if these messages are snooped.

6.4.2.3 Interaction r-sr:

Periodically, the RCA starts in state $S0$ and processes the upload credit (UC) and the download debit (DD) for the servers and the downloaders using the maintained *transaction state*. Upon finishing the processing, it sends the UC to the respective servers, causing the protocol to enter state $S8$. When the servers acknowledge the receipt of UC, state $S0$ is restored.

Attacks $a0$ and $a3$ can be launched by selfish peers by snooping on the UCs in transit or by modification. Since the UC is signed, snooping does not cause damage to the protocol operation. However, modification causes the protocol to enter unsafe state $U14$. When the UC does not reach the server, the RCA times out waiting for the acknowledgment of the UC. The result is that the initial protocol state $S0$ is restored and the RCA can resend the UC. In any event, both the attacks are defeated.

The specification of the DCRC scheme satisfies the *convergence* and *protection* conditions. The convergence condition is satisfied because for every unsafe state Ui, $i \in \mathcal{I}$ $\exists$ Si, $i \in \mathcal{I}$. The protection condition is satisfied because the critical variables: transaction state maintained at the RCA, searcher_id, server_id, downloader_id, content_id, and time_stamp

are not modified in any unsafe state.

6.5 Deployment Considerations

We now discus the practical considerations in incorporating the DCRC scheme in the deployed instances of P2P networks like Gnutella and Kazaa.

Multiple identities: Since the enrollment in reputation tracking is voluntary, the scheme cannot guard against multiple identities. Even enrolled peers who cannot create more than one enrolled identity can create other identities that are not enrolled in the reputation tracking. However, since the RCA processes QRC, UC, and SC only for the enrolled peers, no credit can be earned in the system by the unenrolled identities.

Collusion: In analyzing the attacks possible on the proposed reputation tracking schemes, we focussed only on the selfish peers. Since the underlying Gnutella infrastructure itself is vulnerable to attacks from the malicious peers, we chose not to focus on them for reputation tracking purposes. However, there is a third category of peers that deserves attention. Certain enrolled peers may be willing to incur debits to their reputation scores in order to help other targeted peers through collusion. They can do so by agreeing on fake content downloads. Though the RCA can detect simple collusion of this nature, in general it cannot prevent a collusion where certain enrolled peers agree to penalizing themselves for the sake of others. However, given that most peers are expected to be *unknown* to each other, an occurrence of such a collusion should be insignificant in practice.

Inaccuracies in reputation computations: There are several sources of inaccuracies in reputation computations. Since the RCA does not synchronize the processing of pSearchs, by the time certain peers choose to contact the RCA to obtain the QRCs, the corresponding transaction state may have been erased due to the upper bounds on memory requirements for storing it. Similarly, the download debit state maintained by the RCA may be lost by the time those peers contact the RCA with credit requests. In fact, peers can avoid debits stored at the RCA by not contacting the RCA with credit requests for a long enough duration such that the debit state gets erased (although, they will also not be able to collect any credit for their work in the system in the meanwhile).

A P2P transaction between an enrolled peer and an unenrolled peer causes the enrolled peer to be not be able to collect the credit for its work in the system. Thus, the enrollment of peers in the reputation computations also impacts the accuracy of reputation computations. The net result of all these sources of inaccuracies is the decreased accuracy of reputation computations. It is important for the applications using peer reputations as a substrate to understand these trade-offs.

Reliance on the RCA infrastructure: The DCRC scheme relies on the RCA infrastructure for reliable reputation tracking. The deployed Gnutella-like P2P networks depend on an infrastructure of *voluntary* bootstrapping servers [27] to allow new peers to join the system. The existing bootstrapping infrastructure can be enhanced to provide the RCA functionality. However, since the RCA needs to be trustworthy one cannot depend on voluntary compliance.

Choosing an RCA: In describing the schemes we assumed that the RCA is a centralized entity. Clearly, this assumption is not robust against failures. One way to share the load among an infrastructure of RCAs is for each RCA to be made responsible for certain number or types of files. This would distribute the load among the RCAs. The RCAs dealing with a certain number or types of files can further be replicated for robustness. However, doing so would require synchronization of transaction state among them.

Consolidation of reputations: Periodically, when the peers send the pSearchs to the RCA, they collect encrypted and time-stamped reputation scores to be stored locally. For ease of presenting aggregate reputation scores that the applications using reputation scores may require, the RCA can perform consolidation of reputation scores.

Though reputations can be saved across sessions, to prevent peers from earning a good reputation once and never contributing resources again to the P2P system, upper bounds on the life of reputations need to be defined. These upper bounds can also be used during consolidation of reputations.

Reputations for new peers: All peers start with a reputation score of 0. The new peers start earning reputation by placing content in their shared directory or by serving the downloaded content. When their uploads exceed the downloads, their reputation will

become non-negative. Further, they can earn QRC for simply being a part of the system.

6.6 Performance Evaluation

For the performance evaluation, we focus on two aspects: 1) overheads of reputation computations in the DCRC scheme and 2) the trade-offs in the accuracy of reputation computations with the state maintenance at the RCA, percentage enrollment of peers in reputation tracking, and the frequency at which the RCA processes the debits and credits to reputation scores.

6.6.1 Overheads

In comparison with the Gnutella protocol, the DCRC scheme incur four types of overheads for each content download: 1) extra messages during content download, 2) messages exchanged periodically between the RCA, searchers, and servers for reputation computation purposes, 3) extra computations in order to encrypt and decrypt messages using public and shared key cryptography and in performing hash computations, and 4) crawl overhead for the RCA in order to obtain periodic snapshots of P2P network topology.

Table 5: Upper bounds on overheads in DCRC.

	Search	Download	Reputation related
Extra messages	0	3	$2 + 3\sum_{k=1}^{h} n^k$
Public key encryptions	$\sum_{k=1}^{h} n^k$	5	$1 + \sum_{k=1}^{h} n^k$
Public key decryptions	0	4	$1 + 2\sum_{k=1}^{h} n^k$
Symmetric key operations	0	3	0
Hash computations	0	2	0

Table 5 summarizes the upper bounds on these overheads for each download assuming that all peers in a P2P network are enrolled in reputation computations (the RCA crawl overheads are not considered). In table 5, the average number of neighbors each peer is connected to is denoted by n, and the *hop-count* for queries is denoted by h. We now describe the entries of the table.

Extra messages: DCRC does not require any extra messages during content search. Content download in Gnutella requires a request to download content before the actual download. In addition to this, the DCRC protocol requires three messages during content download: 1) *KeyTransfer*, 2) *ContentRcvd*, and 3) *KeyDelivery*.

Using the transaction state, the RCA infers UC and DD periodically. It then sends a message containing the UC to each server. Including the acknowledgment for this message, two such messages are required for each content download. This is an upper bound because if some servers have contributed to multiple downloads in a period their UCs can be aggregated, bringing the overall messages required in DCRC.

Additionally, up to $\sum_{k=1}^{h} n^k$ searchers can send pSearchs for collecting QRC for each download. In practice, searchers collect pSearchs and send them together periodically, so the average number of messages will be lesser. Sending the corresponding QRCs and receiving their acknowledgments by the RCA requires $2\sum_{k=1}^{h} n^k$ messages.

The first row in table 5 presents the extra messages during search, download, and reputation computations. The upper bound on the total number of extra messages required for each successful content download in DCRC is: $3 + 2 + \sum_{k=1}^{h} n^k + 2\sum_{k=1}^{h} n^k = 5 + 3\sum_{k=1}^{h} n^k$.

Security related operations: We next turn our attention to quantifying the security related operations required in the DCRC scheme for each download. In contrast, Gnutella does not require any such operations. In symmetric key cryptography, encryption and decryption are symmetric. However, the encryption operations in public key cryptography are about 100 times slower than the decryption. Digest computations are typically $10,000$ times faster than the public key encryption. Due to the difference in processing for each of these operations we list these overheads separately in table 5.

In generating the pSearchs each searcher performs one signing operation using its private key, leading to up to $\sum_{k=1}^{h} n^k$ public key encryption operations during content search. Content download in DCRC involves both public and symmetric key operations and also digest computations. Steps 1 through 5 during content download require 9 public key operations (5 encryptions and 4 decryptions), 3 symmetric key operations, and 2 hash

computations.

The RCA decrypts the pSearchs before generating QRCs and signs the QRCs before sending them. Upon receipts of the QRCs, the searchers decrypt the QRCs. Thus, the upper bound on the number of public key encryptions and decryptions during content search are $\sum_{k=1}^{h} n^k$ and $2\sum_{k=1}^{h} n^k$ respectively. Similarly, the RCA encrypts the UC before sending it to the server and the server decrypts it, leading to 1 encryption and 1 decryption per download.

6.6.2 Simulation Evaluation of Reliability Trade-offs

Section 6.5 discussed the potential causes for inaccuracies in the reputations of peers computed using the DCRC scheme. We now evaluate the tradeoffs in the accuracy of DCRC reputation computations with: 1) transaction state maintained by the RCA, 2) percentage enrollment of peers in reputation computations, and 3) the periodicity at which the RCA processes the debits and credits to the reputation scores. The simulations assume that only the UCs, DDs, and the QRCs are processed and do not consider SC.

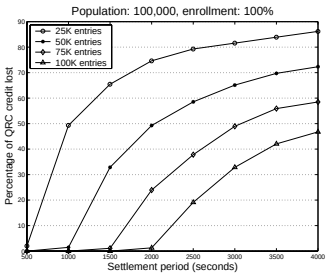
To evaluate the inaccuracies in reputation computations, we generated 6 hour long synthetic P2P request logs with exponential inter-arrival time and an average request rate of 50 requests per second. The peer population in these logs varies from $10,000$ unique peers to $500,000$. The logs assume that all the peers stay in the system for the entire duration. The enrollment of peers in getting the reputations tracked varies from 25% to 100%. The peers access 100 unique files and the accesses to these files are are uniformly distributed.

Referring to the periodicity at which the RCA processes the debits and credits as *settlement period*, figure 34(a) shows the percentage of the QRC credit lost for a peer population of $100,000$ because by the time the searchers contacted the RCA with their pSearchs the corresponding transaction state at the RCA had been erased. The peer enrollment in reputation computations is assumed to be 100% in figure 34(a) and we experimented with the transaction state at the RCA to be about $25,000$, $50,000$, $75,000$, and $100,000$ file transfers. As can be seen from the figure, there is a clear tradeoff between the amount of transaction state maintained at the RCA and the settlement period. The trends for other population

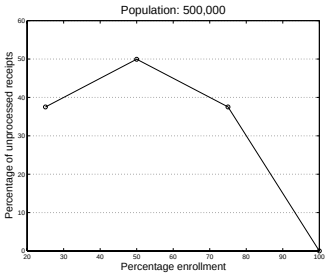
sizes were similar.

Figure 34(b) shows the impact of percentage enrollment on the percentage of UCs that could not processed because one of the peers was not enrolled to get its reputations tracked. We show the results for the logs with a peer population of $500,000$. The results for other logs were similar. As expected, the percentage of unprocessed receipts peaks when the enrollment reaches 50%.

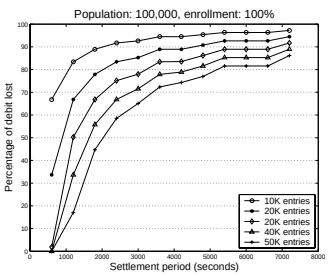
In addition to the transaction state, the DCRC scheme requires the RCA to maintain debit state for peers after processing the UCs and the corresponding DDs. This is necessary to ensure that peers do not drop the *negative* reputation scores. The amount of debit state maintained at the RCA effects the accuracy of reputation computations. Figure 34(c) shows the effect of settlement period on the percentage of debit lost as a result of the amount of debit state maintained at the RCA. These results are also for the logs with peer population $100,000$ and an enrollment of 100%. Though the number of entries in the state maintained are different, the results of figure 34(a) and 34(c) show very similar trends as expected.



(a) Accuracy of QRC vs settlement time period.



(b) Effect of enrollment on receipt processing.



(c) Accurate debit accounting vs settlement time.

Figure 34: Quantifying the accuracy of reputation computations.

6.7 Conclusion

Reliable reputations can help to establish trust and motivate peers to contribute to the common good of P2P networks. All of the previous proposals for reputation tracking in P2P networks have focussed on accomplishing the task in a *decentralized* manner. A

consequence of this choice is that the resulting reputations are based on *subjective* criteria and are not universally comparable. The latter is a requirement for protocols like SDP, proposed in chapter 5. Further, reliability issues in reputation computations have thus far not received much attention.

This chapter proposes a reputation system for decentralized unstructured P2P networks like Gnutella that uses objective criteria for computing reputations. In order to ensure the reliability of reputation computations in the presence of peers who act in self-interest and to allow the reputations to be locally stored we propose a solution that utilizes the RCA infrastructure. The requirement of trustworthiness for the RCA infrastructure suggests that a voluntary infrastructure of the kind used for bootstrapping Gnutella clients may not be sufficient.

Formally verifiable reliability in reputation computations proposed in this chapter comes at the cost of overheads to the Gnutella system. For applications that require high reliability these overheads may be acceptable. But for other kinds of applications where some compromise in reliability may be acceptable, these overheads may be an overkill. Chapter 7 is devoted to the discussion of the trade-offs between reliability and overheads in reputation computations. Using the same objective criteria discussed in section 6.2.1, in chapter 7 we propose a lower overhead reputation system.

Appendix: Formal Specification for Strong Reputations

In *Abstract Protocol Notation*, each entity of the protocol is denoted by a *process*. Each process p in turn is defined by a set of *constants*, *inputs*, *variables*, *parameters*, and *actions*. The constants of p have fixed values while the inputs can be read but not updated. The variables can be read and updated by the actions of p. A parameter declared in a process is used to write a finite set of actions as one action, with one action for each possible value of the parameter. Each *action* of a process is of the form $< guard > \rightarrow < statement >$. The *guard* of an action is one of the 3 forms: 1) a boolean expression over the constants and variables of p, 2) a receive guard of the form ***rcv***$< message >$***from*** q; where q is another process in the protocol, and 3) a *timeout* that contains a boolean expression over the constants and variables of every process and the contents of all channels in a protocol. An action is executed only when its guard is true and consists of executing the *statement* of this action. Details on the types of variables and statements can be found in [25]. Comments can be added anywhere in a process definition; each comment is placed between the two brackets { and }.

```
process querier[i: 0...n-1]
inp  K^-1_self, K_self  : integer;          {own keys}
     K_RCA              : integer;          {RCA's public key}
     K                  : array [0...n-1] of integer; {public keys of all peers}
var  new_query          : boolean;
     keywords           : string;
     reply              : integer;
par  j                  : array [0...n-1] of integer;     {for peers}
begin
     new_query = true →                     {new query generated}
          keywords := any;
          send query(keywords) to searcher[j]; {send to all direct neighbors}
          new_query := false;
     | timeout reply = 0 →                  {ready to resend query}
          new_query := true;
end
```

process searcher[i: 0...n-1]

inp $K^{-1}{}_{self}$, K_{self} : **integer**; {own keys}

K_{RCA} : **integer**; {RCA's public key}

K : **array** [0...n-1] **of integer**; {public keys of all peers}

var timer : **boolean**;

pSearch : **array** [0...p-1] **of string**; {#pSearchs< p}

keywords', time_stamp, credit', ack : **string**;

querierID : **integer**;

reputation : **array** [0...p-1] **of real**; {#reputations < p}

par j : **array** [0...n-1] **of integer**; {for peers}

begin **rcv** query(keywords') **from** querier[j] →

pSearch[k] := NCR(K^{-1},(MD5(K_{self} | querierID | keywords' | time_stamp)));

{save pSearch}

| **timeout** timer = **true** →

send pSearch[k] **to** RCA;

timer := **false**;

| **rcv** credit' **from** RCA →

reputation[k] := DCR(K_{RCA}, credit'); {save reputation}

send ack **to** RCA;

end

process downloader[i: 0...n-1]

inp $K^{-1}{}_{self}$, K_{self} : **integer**; {own keys}

K_{RCA} : **integer**; {RCA's public key}

K : **array** [0...n-1] **of integer** {public keys of all peers}

var select, timer : **boolean**;

content' : **binary**;

time_stamp, msg' : **string**;

contentID, RCA_ID, serverID, symKey : **integer**;

par j : **array** [0...n-1] **of integer**; {for peers}

begin

select = **true** → {a server selected for download}

send downloadReq(NCR($K^{-1}{}_{self}$, (MD5(K_{self}) | contentID | time_stamp))) **to** server[j];

select := **false**;

| **rcv** contentDelivery(content') **from** server[j] →

send contentRcvd(NCR($K^{-1}{}_{self}$, (serverID | MD5(K_{self}) | contentID | MD5(content') | time_stamp))) **to** RCA;

| **timeout** timer = **true** →

select := **true**; {resend request}

timer := **false**;

| **rcv** keyDelivery(msg') **from** RCA →

(contentID, RCA_ID, symKey) := DCR($K^{-1}{}_{self}$, (msg'));

DCR(symKey, content); {decrypt content}

end

process server[i: 0...n-1]

inp $K^{-1}{}_{self}$, K_{self} : **integer**; {own keys}

K_{RIA} : **integer**; {RIA's public key}

K : **array** [0...n-1] **of integer**; {public keys of all peers}

var content : **array** [0...m-1] **of binary**;

time_stamp, request', ack : **string**;

contentID, downloaderID, symKey, credit' : **integer**;

reputation : **array** [0...p-1] **of real**;{#reputations < p}

par j : **array** [0...n-1] **of integer**; {for peers}

begin

rcv downloadReq(request') **from** downloader[j] →

(downloaderID, contentID, time_stamp) := DCR(K_j, (request'));

symKey := DES(contentID | downloaderID | time_stamp);

{generate symKey using DES}

send contentDelivery(NCR(symKey, (content[contentID]))) **to** downloader[j];

send keyTransfer(NCR(K_{RIA}, NCR(K^{-1}self, (MD5(K_{self})|downloaderID

|contentID|time_stamp))|symKey)) **to** RIA;

| **rcv** credit' **from** RIA →

reputation[k] := DCR(K_{RIA}, (credit')); save reputation

send ack **to** RCA;

end

```
process RCA
inp   K^-1_RCA, K_RCA: integer;        {own keys}
      K                : array [0...n-1] of integer {public keys of all peers}
var   content                                : array [0...m-1] of binary;
      keyRcvd, digestRcvd                    : array [0...n-1] of boolean;
      time_stamp, trans_his                  : string
      symKey, contentID, serverID            : array [0...n-1] of integer;
      digest, msg'                           : integer;
      downloaderID, searcherID               : integer;
      reputation, credit, debit              : array [0...j-1] of real;
par   j, l             : array [0...n-1] of integer;          {for peers}
begin rcv keyTransfer(msg') from server[j] →
          (serverID[j], downloaderID, contentID[j], symKey[j] := DCR(K^-1_RCA, (msg'));
          if digestRcvd[j] := true →{content digest already rcvd}
              send keyDelivery(NCR(K_j, (contentID[j] | RCA_ID | symKey[j]))) to downloader[j];
              digestRcvd[j] := false;
              update(trans_his);                    {update transaction history}
          fi
      | rcv contentReceived(msg') from downloader[j] →
          (serverID[j], downloaderID, contentID[j], digest, time_stamp):= DCR(K_j,(msg'));
          if keyRcvd[j] := true & digest = MD5(NCR(symKey[j], (content[contentID[j]]))) →
              send keyDelivery(NCR(K_j, (contentID[j] | RCA_ID | symKey[j]))) to downloader[j];
              update(trans_his);
              keyRcvd[j] := false;
          fi
      | timeout timer = true →      {time to process debits/credits}
          process(trans_his);       {timer for bill acks not shown}
          reputation[j] := NCR(K^-1_RCA,(serverID[j] | contentID[j] | time_stamp | credit[j])) {one msg/peer}
          if reputation[j]-debit[j] >= 0 →
              send reputation[j]-debit[j] to to server[j];
              debit[l];             {save corresponding debit for the downloader[l]}
          fi
          timer := false;
      | rcv pSearch(msg') from searcher[j] →
          if match(trans_his, DCR(K_j, (msg'))) = false → skip {state erased}
          | match(trans_his, DCR(K_j, (msg'))) = true →
              send (NCR(K^-1_RCA,(searcherID | contentID[j] | time_stamp | credit[j]))) to searcher[j]
          fi
end
```

CHAPTER VII

TRADE-OFFS BETWEEN RELIABILITY AND OVERHEADS IN REPUTATION TRACKING IN PEER-TO-PEER NETWORKS

7.1 Introduction

The success of peer-to-peer (P2P) networks hinges on the participation of peers in the system. Reliable reputations are a desirable feature of these networks because they can serve as the basis for the incentive schemes to encourage participation and address issues of *lack of trust* and *free-loading*[1].

In chapter 5, we described a service differentiation protocol (SDP) for Gnutella-like P2P networks with the goal of discouraging free-loading. SDP utilizes reputations as a substrate to accomplish its goal. Chapter 6 proposed a reputation system with features desired by SDP. Reputations in the debit-credit reputation computation (DCRC) scheme proposed in chapter 6 are *objective* and hence universally comparable. DCRC utilizes an infrastructure of reputation computation agents (RCAs) to track peer reputations reliably. The design allows for the reputations to be stored *locally* for fast retrieval.

The reliability of reputation computations in DCRC scheme are formally verifiable. However, the extent to which the RCA infrastructure is utilized to facilitate content download to ensure reliable reputation tracking introduces an undesirable centralization during P2P content download. It also creates an unavoidable dependence on the RCA infrastructure. The work in this chapter is motivated by these concerns. This chapter explores the trade-offs between reliability and overheads (and hence the dependence on the RCA infrastructure) in reputation tracking. Specifically, referring to the reputation tracking in

[1] *Free-loaders are peers who only download content but do not serve it to the other peers.*

DCRC scheme proposed in chapter 6 as *strong reputations*, this work proposes *weak reputations*. The weak reputations retain the basic properties of the DCRC scheme, namely, using objective criteria for reputation tracking and local storage of reputations. However, they remove the dependency on the RCA infrastructure during P2P content download. The RCA infrastructure in weak reputations is no longer a bottleneck in normal P2P operations and needs only be contacted periodically for reputation related computations. Evaluations show that the change in the functionality of the RCA infrastructure lowers the overheads of reputation computations. However, the lowered overheads come at the cost of decreased reliability, which may be acceptable for certain applications.

We also investigate an alternate scheme to compute low overhead (and hence lower reliability) reputations. The credit-only reputation computation (CORC) scheme uses the same objective criteria to track reputations as the DCRC scheme and allows the reputations to be locally stored with the help of the RCA infrastructure. Conceptually, the reputation tracking in CORC scheme is similar to the weak reputations in terms of the overheads and reliability. The main difference compared to the DCRC scheme is that the CORC scheme credits peer reputation scores for serving content but offers no debits[2]. The expiration on the scores instead serves as a debit. Just like in the DCRC scheme, the CORC scheme offers additional credits for query processing and forwarding, and staying online. Though the CORC scheme has several desirable properties, analysis shows that it is prone to collusion which makes it impractical.

The basic assumptions about the RCA infrastructure and the terminology used in describing weak reputations as well as the CORC scheme are the same as in section 6.3. The only difference being that since the RCA in weak reputations is not involved in content download, it does not need to have a copy of the content served by the peers. Also, just as in the case of strong reputations, a peer in weak reputations can choose not to have its reputation tracked, in which case it will always have a reputation score of 0, the minimum reputation score allowed by the system.

[2]As described in chapter 6, the DCRC scheme credits peer reputation scores for serving content and *debits* them for downloading.

7.2 Assumptions

This section reviews the assumptions about the infrastructure and terminology used in this chapter. These are essentially the same set of assumptions that were described in section 6.3.

A peer who is interested in getting its reputation tracked first enrolls itself with the RCA to get a (public, private) key pair. The digest of a peer's public key is used to identify it. This distribution of (public, private) key pair can be thought of as similar to that of public key infrastructure (PKI) in the sense that only one *enrolled* identity is permissible per peer[3]. The (public, private) key pair of the RCA is denoted by $\{PK_{RCA}, SK_{RCA}\}$ and all the peers have access to the RCA's public key. They can obtain the public keys of other peers in the system when needed.

We assume that the RCA is not malicious but peers can collude with other peers in self-interest. For simplicity of description we assume that the RCA infrastructure is a single entity.

Due to the enrollment requirement, a P2P system now has three types of entities: 1) enrolled peers, 2) unenrolled peers, and 3) RCAs. As before, we refer to the peer that generates the query as the *querying peer/querier* in this paper. All the peers that receive and process the query are called *searching peers/searchers*. Once the querying peer receives all the replies, it chooses a peer to download the content object from. At that point, it becomes a *downloading peer/downloader*. The peer that serves the content is referred to as the *serving peer/server*. The (public, private keys of querier, searcher, downloader, and server are denoted by $\{PK_{query}, SK_{query}\}$, $\{PK_{search}, SK_{search}\}$, $\{PK_{download}, SK_{download}\}$, and $\{PK_{serve}, SK_{serve}\}$ respectively.

7.3 Weak Reputations

This section describes the weak reputations. We begin with a discussion of the design space for positioning the weak reputations in terms of reliability and the overheads incurred.

[3]A peer can however generate any number of *unenrolled* identities.

7.3.1 Design Space

Figure 35 shows the design space of solutions. The leftmost end of the spectrum corresponds to minimum overhead solution. The current implementations of Gnutella like P2P networks which do not have any support for reputation tracking fall here. The rightmost end of the spectrum corresponds to a hypothetical fool-proof reputation tracking solution which is capable of guarding against all possible security attacks[4]. As expected, this is the maximum overhead solution. Any solution that tracks reputations falls in between these two ends of the spectrum.

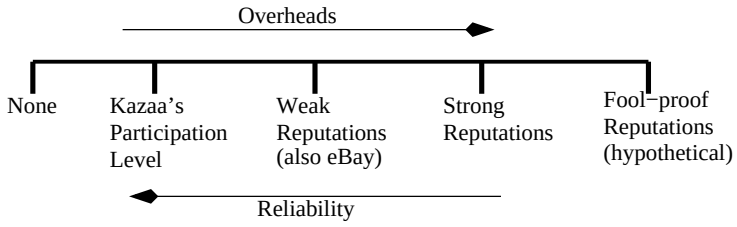


Figure 35: Spectrum of reliable reputation computation solutions.

Kazaa uses the notion of *participation level* ([32]) in order to track peer contribution. Kazaa software locally updates the net MBytes served by each peer. In times when the request rate is high, the participation level is used to prioritize among downloaders. Though this solution has very little overhead, it offers no security against selfish peers that know how to alter the part of their software that computes participation level.

The strong and weak reputations fall in between Kazaa's participation level and the hypothetical fool-proof reputation tracking solutions in terms of overheads and reliability. Both the solutions utilize the RCA infrastructure to varying degrees in offering reliability to the reputation computations. The weak reputations can conceptually be thought of as similar to the reputations maintained in eBay in terms of reliability. Buyers and sellers in eBay can leave incorrect feedback and can also collude. However, the reputations so maintained are still useful.

[4]We call it hypothetical because it is resistant against even undiscovered attacks.

7.3.2 Details

Reputation tracking in weak reputations is also done in two steps, just like in the case of strong reputations. However, the details differ from those for strong reputations (6.3.3): 1) peers enrolled in reputation tracking now save proofs of their contributions both during P2P search and download functionalities to collect query-response credit (QRC) and upload credit (UC) and 2) they periodically send these proofs to the RCA. The RCA processes UC, the corresponding download debit (DD) and the QRC and sends the encrypted reputations to relevant peers for keeping locally. Just as in the case of strong reputations, the RCA stores the DD with itself to ensure that peers do not drop them. We now describe the proof of contributions saved by the peers and the processing at the RCA.

7.3.2.1 Proof of Peer Contributions

pSearch: For every query-response message processed during the content search, a searcher peer saves $\{$*searcher_identity*, *query_keywords*, *time_stamp*, *querier_identity*$\}_{SK_{query}}$ as the *proof of searching (pSearch)*, just as in the case of strong reputations.

pServe: For saving the *proof of serving (pServe)*, the following exchange takes place between the enrolled downloader and the enrolled server peer at the time of the file download[5]:

- The downloader sends a *requester portion of the receipt (RPR)* in the form of $\{$*downloader_identity*, *server_identity*, *file_name*, *file_size*, *time_stamp*$\}_{SK_{download}}$ to the sender peer.
- The server peer verifies the information using PK_s and stores $\{\{$*downloader_identity*, *server_identity*, *file_name*, *file_size*, *time_stamp*$\}_{SK_{download}}\}_{SK_{serve}}$ as pServe. At that point, it serves the content to the downloader.

7.3.2.2 Processing at the RCA

Periodically, the enrolled peers send the pSearchs and pServes collected thus far to the RCA. The RCA uses these to maintain *transaction state* of the form (*downloader_identity*,

[5]It is important to note that if *any* of the peers are not enrolled, the following exchange does not take place. This means that by creating additional *unenrolled* identities peers cannot earn any credit to their reputations.

server_identity, *file_name*, *file_size*, *time_stamp*, *credit_processed_list*). The credit_processed_list is similar to that in strong reputations. Notice that since the RCA does not interfere with the search and download functionality, as does in the case of strong reputations, it can not maintain any state by itself. It relies on the pServes to maintain the transaction state about downloads.

Processing upload credit (UC) and download debit (DD): The RCA in weak reputations uses pServes to infer the UC for the serving peers and the corresponding DD for the downloading peers. Just as for strong reputations, it then updates the credit_processed_list and sends encrypted reputations of the form $\{$*RCA_identity*, *time_stamp*, *reputation_score*, *server_identity*$\}_{SK_{RCA}}$ to the relevant serving peers. To avoid having the downloaders drop negative reputation scores, the RCA retains the DDs in the form of *debit state* with itself until those peers send some credits for processing, just like in strong reputations.

Processing query-response credit (QRC): Since peers can forge QRCs for files that were never downloaded, the QRC from the pSearchs is not processed until the corresponding UCs and DDs are processed, even if they arrive before the pServes. After inferring QRCs using pSearchs and the transaction state, the RCA updates the credit_processed_list and sends an encrypted reputation score of the form $\{$*RCA_identity*, *time_stamp*, *reputation_score*, *searcher_identity*$\}_{SK_{RCA}}$ to the searchers.

7.4 Attack Analysis

Just like in the case of strong reputations, the security mechanisms in weak reputations are mainly designed to counter attacks launched by enrolled peers that are selfish, not malicious. The discussion in section 6.5 about enrolled peers that are not malicious or selfish but agree to suffering a penalty for others' sake applies equally well to weak reputations.

The signing of pSearchs for QRC and pServes for UC provides protection against message forgery and modification attacks. The credit_processed_list maintained by the RCA along with the time stamps in the pSearchs and pServes guard against message replay attacks. These ensure that peers can only collect credit once and that only peers that send the signed pServes and pSearchs can collect the credit. As a result, the last property out of the

4 properties listed in section 6.4 is satisfied. However, the rest of the 3 properties are not satisfied for weak reputations because of the possibility of the following attacks:

Collusion to collect QRC: Peers collect pSearchs while processing content search queries without the intervention of any overseeing authority. The RCA postpones processing of pSearchs sent by the peers until it receives the corresponding pServes. This precludes peers from generating pSearchs for downloads that did not take place in the system. However, peers can collude with each other to collect pSearchs (and hence QRC) by supplying information about the queries that any of them have processed since the RCA does not have a way of verifying the pSearchs. In practice such a collusion is not worth it for two reasons: 1) the main issue the reputations help address is that of motiving peers to contribute to the system. Peers have ample opportunity to cooperate in the system and earn QRC for processing queries they encounter and 2) since the QRC is expected to be very small as compared to the UC (as described in section 6.2.1) such a collusion would not be very beneficial.

Limited content reliability: In order to allow the server peers to collect credit for serving content, the downloaders are required to send the receiver portion of the receipt (RPR) as described in section 7.3.2.1 before they can receive the content from the server. This ensures that the downloaders cannot avoid a debit to their reputations if they consume servers' resources. However, the exchange for collecting pServe does not protect the downloaders from avoiding debits in cases where they do not get the requested content in its entirety either due to servers' selfishness or network failures. As as result, there is limited content reliability for the downloaders in weak reputations. significant cost. To circumvent the problem in a light-weight manner, the weak reputations allow the downloaders to report such servers' to the RCA. Upon receiving many such complaints, such peers may be blacklisted. Though the system offers no advantages for doing so, the misreporting is prone to peers colluding to malign the reputation of certain targeted peers and is a hard problem to solve in general.

Though the exchange to collect pServe offers only limited content reliability, including the identity of the servers in the RPR guarantees that the server peers do not collude with

other peers to share the RPR. Otherwise, multiple peers could collect pServes and each could cause a debit for the downloader. The downloader can choose to generate RPRs even for the transfers that did not take place but doing so would only cause more debits to be accumulated in its account with the RCA.

7.5 Credit-only Reputation Computation (CORC) Scheme

The CORC scheme is very similar to the weak reputations in terms of overheads and reliability except for the fact that there are no debits (DD) for content download. The time stamp on the reputations serves to expire the credits earned by searching the content (QRC), serving the content (UC), and by staying online (SC). CORC also utilizes the RCA infrastructure, just like weak reputations and allows the peers to store their reputations locally for fast retrieval.

Peers collect proof of contributions in the form of pSearchs and pServes in CORC, just like in the case of weak reputations (section 7.3.2.1). However, the exchange between the downloader and the server during pServe is slightly different. The RPR for CORC is sent after receiving the content. This is to prevent the server peers from collecting credits without actually serving the content. This exchange to collect pServe is more fool proof compared to that in the case of weak reputations. This is because although malicious receivers in this scheme can choose not to send the receipt in spite of receiving the content, doing so is not advantageous to them because CORC offers no debits to downloaders. Further, reporting such downloaders to the RCA can prevent such occurrences.

Just like in the case of weak reputations, the pSearchs and the pServes are periodically sent to the RCA. The processing at the RCA in the case of CORC differs from weak reputations only in that the RCA does not maintain any debit state for the downloaders. The RCA sends the encrypted reputation scores to the relevant peers and the time stamps in the UC and QRCs serve to expire the reputation scores.

The lack of debit component in the CORC schemes leads to some important differences with respect to the weak reputations. Table 6 summarizes the key differences. If a scheme is better on some count, it is indicated in bold.

Table 6: Key differences between the CORC scheme and weak reputations.

	CORC	Weak reputations
Possibility of collusion	More	**Less**
Consolidation of reputation scores possible	No	**Yes**
Relative safety of pServe exchange during content download	**More**	Less
RCA needs to maintain debit state	**No**	Yes

The main drawback of the CORC scheme is the possibility of *mutually beneficial collusion* among peers who act in self-interest. This is because the CORC scheme does not offer debits to the downloader. Colluding peers in CORC can achieve high reputation scores by transferring a set of files to each other continually. The debit component of DCRC prevents such an occurrence from happening. Since the RCA maintains transaction state for some amount of time about transactions, it can potentially run algorithms to detect collusion among peers to increase their reputation score. This can be done by giving fewer credits if the same set of peers have many transactions with each other, especially if the files comprise a small set. However, determined peers can easily thwart such simple algorithms. The other aspects of the vulnerabilities of the CORC scheme are similar to the attack analysis discussed in section 7.4.

The lack of debit component in CORC also gives rise to several other differences, as shown in table 6. Since the time stamps in CORC are used for expiring the reputation scores, the reputation scores collected over a period of time cannot be consolidated for ease of presenting when required. However, the exchange to collect pServe during content download is relatively more secure in CORC. Further, the RCA in the case of CORC does not have to maintain debit state.

7.6 Evaluation of Overheads

In comparison with the Gnutella protocol, the weak reputations incur three types of overheads for each content download: 1) an extra message during content download, 2) messages

exchanged periodically between the RCA, searchers, and servers for reputation computation purposes, and 3) extra computations in order to encrypt and decrypt messages using public key cryptography. Compared to the strong reputations the weak reputations have fewer overheads. They do not have RCA crawl overhead or the overhead for symmetric key operations and hash computations.

We evaluate the overheads for weak reputations in two ways. First, we evaluate the upper bounds on these overheads. These upper bounds can be compared to the upper bounds for the case of strong reputations 6.6.1. Second, we do a comparative evaluation of the overheads between weak and strong reputations through simulations.

7.6.1 Upper Bounds on Overheads

Table 7 summarizes the upper bounds on the extra messages and public key operations for each download assuming that all peers in a P2P network are enrolled in reputation computations. Just as in the case of strong reputations, the average number of neighbors each peer is connected to is denoted by n and the *hop-count* for queries is denoted by h. We now describe the entries of the table.

Table 7: Upper bounds on overheads in weak reputations.

	Search	Download	Reputation related
Extra messages	0	1	$3 + 3\sum_{k=1}^{h} n^k$
Public key encryptions	$\sum_{k=1}^{h} n^k$	2	$1 + \sum_{k=1}^{h} n^k$
Public key decryptions	0	1	$2 + 2\sum_{k=1}^{h} n^k$
Symmetric key operations	0	0	0
Hash computations	0	0	0

Extra messages: Weak reputations also do not have any extra messages during content search. However, during content download they require one extra message to be sent by the downloader.

The RCA infers UC and DD periodically using the pServes sent by the servers. It then sends a message containing the UC to each server. Including the acknowledgment for

this message, three extra messages are required for reputation inference for each content download. Additionally, up to $\sum_{k=1}^{h} n^k$ searchers can send pSearchs for collecting QRC for each download. Sending the corresponding QRCs and receiving their acknowledgments by the RCA requires $2\sum_{k=1}^{h} n^k$ additional messages.

The first row in table 7 presents the extra messages during search, download, and reputation computations. The upper bound on the total number of extra messages required for each successful content download in weak reputations is: $1 + 3 + 3\sum_{k=1}^{h} n^k = 4 + 3\sum_{k=1}^{h} n^k$. Compared to the strong reputations, there is one less message per content download.

Security related operations: In generating the pSearchs each searcher performs one signing operation using its private key, leading to up to $\sum_{k=1}^{h} n^k$ public key encryption operations during content search. Content download in weak reputations involves 2 public key encryptions and 1 public key decryption.

Further, The RCA decrypts the pSearchs before generating QRCs and signs the QRCs before sending them. Upon receipt of the QRCs, the searchers decrypts the QRCs. Thus, the upper bound on the number of public key encryptions and decryptions during content search are $\sum_{k=1}^{h} n^k$ and $2\sum_{k=1}^{h} n^k$ respectively. Further, 1 encryption and decryption each is required for sending and receiving the UC.

7.6.2 Comparison of Weak and Strong Reputations Through Simulations

To compare the overheads between strong and weak reputations through simulations, we generated connected topologies with peer populations ranging from 5000 to $50,000$. Each peer in these topologies is connected on an average to about 4 other peers in the system. The requests for files in this system are uniformly distributed among all the peers with an exponential inter-arrival time of 50 requests/second. The simulation logs are 1 hour long for each topology. The number of hops queries are allowed to go varied from 4 hops to 7 hops. The number of hops were constant for all peers for one run.

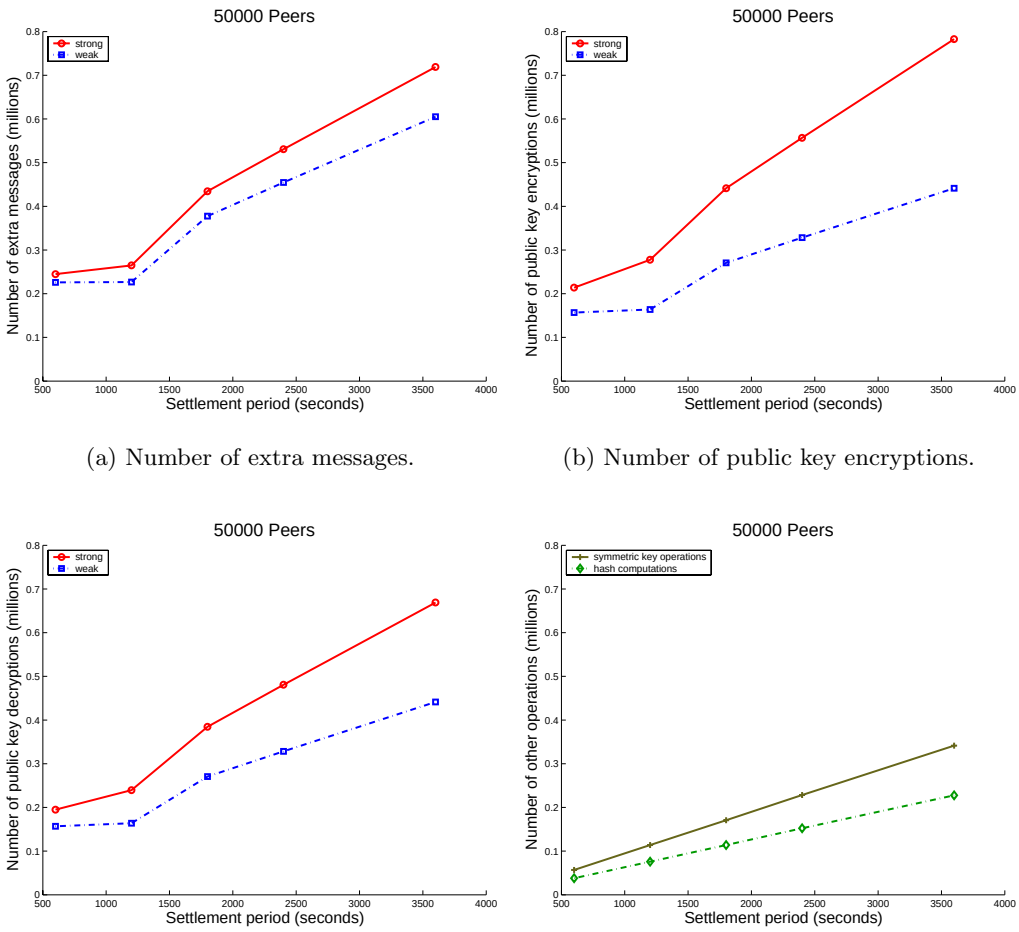
We assume that the total number of files in each topology is the same as the peer population. The file popularities are Zipf distributed with a parameter of 1.0 and the file

sizes are uniformly distributed between 0-8MBytes. To model a relatively stable state of the P2P system where some peers cache and serve more files than some others, we modeled the number of files cached by each peer in the system as a Zipf distribution with a parameter of 1.0. To create multiple logical small-worlds where some peers serve more files than the others we created 100 groups of peers for each population size. Within each group the number of cached files were Zipf distributed.

For the evaluations, we experimented with population sizes of 5000, 10,000, 25,000 and 50,000. With the number of hops staying constant, the effect of varying population sizes was that the smaller the peer population in the topology, the more the successes during content search (and hence lesser failures). This is expected because for smaller topologies, the number copies of files in the system are more. Since our goal is to compare the strong and weak reputations for their overheads, as long as they are both tested under same conditions the conclusions are not expected to change.

We now present the results of the topology with 50,000 peers where each query is allowed to go 7 hops. The *settlement period* (the periodicity at which the RCA processes the reputation related debits and credits) varies between 600 seconds to 3600 seconds for the graphs shown in figures 36(a), 36(b), 36(c), and 36(d). Figure 36(a) shows the difference in the number of total extra messages in the system between strong and weak reputations. Figure 36(b) and 36(c) show the difference in the total number of encryption and decryption operations between the two schemes respectively. Figure 36(d) shows the total number of symmetric key operations and hash computations in the system for strong reputations. Weak reputations do not have this overhead. The RCA crawl overheads required in strong reputations were not analyzed.

In all the graphs the total overheads as well as the difference in overheads between strong and weak reputations increases with the increase in settlement period. This is because the number of total downloads increase with the settlement period, leading to higher overall overheads on the system. Further, the downloads contribute most to the difference in overheads between strong and weak reputations, which leads to more divergence between strong and weak reputations.



(a) Number of extra messages.

(b) Number of public key encryptions.

(c) Number of public key decryptions.

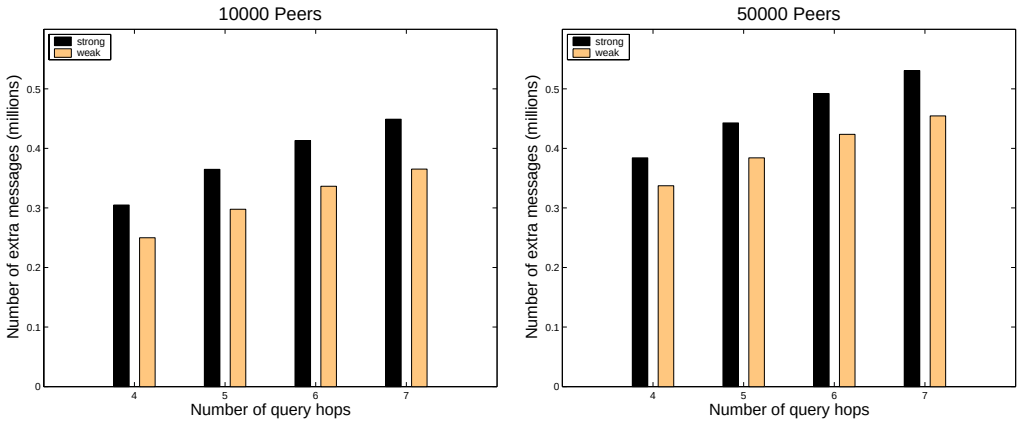
(d) Symmetric key and hash computations for strong reputations.

Figure 36: Overheads of strong and weak reputations.

Figures 37(a) and 37(b) show the affect of varying hops on the overheads for strong and weak reputations for topologies with $10,000$ and $50,000$ peers respectively. The settlement period was set to 2400 seconds for these graphs. As expected, with increase in the number of hops the queries are allowed to go, the overheads increase. Also, the difference in overheads between strong and weak reputations stayed constant.

7.7 Conclusion

The strong and weak reputations proposed in chapters 6 and 7 require the presence of a trustworthy RCA infrastructure to ensure reliability in reputation computations. Existing



(a) 10,000 peers. (b) 50,000 peers.

Figure 37: Effect of varying the number of hops.

Gnutella-like P2P networks also utilize a bootstrapping infrastructure to facilitate peer join in the network. However, the requirements on the bootstrapping infrastructure are much less stringent and voluntary compliance is sufficient. The requirement of a trustworthy RCA infrastructure in the case of strong and weak reputations amounts to changing the basic nature of a P2P network.

While many proposals to track peer reputations in P2P networks are available, to our knowledge this is the first work that explores the trade-offs in reliability and overheads in reputation tracking. Though the *strong reputations* proposed in chapter 6 track reputations with formally verifiable reliability, they introduce a centralized bottleneck during content download, require the RCA infrastructure to crawl the P2P network to maintain snapshots of the topology, and incur higher overheads on the underlying P2P network compared to the *weak reputations*. The RCA infrastructure in weak reputations on the other hand is not a bottleneck for normal P2P operations of search and downloads and only needs to be contacted periodically. Applications that utilize peer reputations need to understand these trade-offs in overheads and reliability in reputation tracking.

We considered another lower overhead (and hence lower reliability) reputation tracking scheme, the credit-only reputation computation (CORC). Though it offers certain advantages over the weak reputations, it suffers from collusion by selfish peers and is impractical

from the standpoint of further consideration.

CHAPTER VIII

SUMMARY OF CONTRIBUTIONS

Quality-consciousness in large-scale content distribution technologies can significantly enhance the overall experience of the users of these technologies. This dissertation focused on the architectures and mechanisms to enhance multicast and peer-to-peer content distribution through quality-consciousness. In particular, it addressed the following aspects of quality-consciousness: 1) client latency, 2) service differentiation, and 3) content quality. The contributions of this dissertation under each of these categories are summarized in sections 8.1, 8.2, and 8.3. Section 8.4 discusses several future work directions that arise out of this dissertation.

8.1 Client Latency

Previous research analyzing access logs for multimedia servers indicated that the server load is highly variable. This variation is caused by the dynamic nature of a small percentage of popular files. To test the conjecture that the phenomenon may not be restricted to multimedia files but may occur for a variety of content in the Internet, we collected web and FTP logs from Georgia Tech's servers. Analysis of these logs confirmed the conjecture that a small percentage of files with variable access patterns cause most of the load variability for the servers, irrespective of the content type and protocol used to retrieve it.

Guided by the file dynamics observed in the collected logs, we generated three synthetic multimedia server logs with varying number of hourly accesses to the multimedia server. The evaluation of these logs for client latency and reneging of requests led to the following contributions[1]:

- We concluded that during the periods when a small percentage of popular files exhibit dynamic access profiles, all available multicast scheduling schemes favor the dynamic

[1]This work appeared in ICC 2003.

files, giving them much lower client latency compared to the times when they have constant accesses. They do so at the cost of penalizing the less popular files that have not experienced a change in access pattern. Also, the reneging during the periods of higher accesses is very high.

- We proposed a novel multicast scheduling scheme called *M*inimum *W*aiting *T*ime (MWT) that provides lower client latencies to the files that do not have dynamic profiles, while maintaining the response time for the dynamic files during variable access conditions. It does so by ensuring a minimum waiting time for the dynamic files during heavy access conditions. MWT is fair to all the files in terms of response time. It reduces the reneging of requests due to better batching of requests, leading to a better server resource utilization.

8.2 Service Differentiation

The service differentiation aspect was addressed for multicast as well as P2P networks. The contributions for each of them are:

8.2.1 Service Differentiation for Multicast:

This dissertation proposes a scalable architecture called M-DS (multicast-DS) to provide service differentiation for multicast utilizing the differentiated services (DS) framework[2]. The architecture uses one of two interoperable limited branching techniques, namely, *edge-router branching* and *limited-core branching*. The edge-router branching technique exploits the multicast scalability at a domain granularity and moves all the branching points required to graft new receivers to the existing multicast tree to the ingress of the domain. The limited-core branching technique allows a limited number of branching points to exist in a domain. This introduces a small amount of extra complexity but exploits multicast scalability in a more effective manner.

M-DS preserves the DS scalability, does not incur any per-packet overhead due to extra headers in data packets, and routes packets using the IP multicast routing tables that are

[2]This work appeared in IC 2002.

already set up in individual domains, albeit by enhancing the router functionality. Performance evaluation through simulations in terms of signaling overhead and extra bandwidth required shows that it is practical to include the proposed techniques in the DS framework.

8.2.2 Service Differentiation for P2P Networks:

Service differentiation can be very useful in dissuading free-loading in P2P networks. We developed SDP, a protocol for service differentiation in decentralized unstructured P2P networks like Gnutella and Kazaa that utilizes peer reputations as a substrate. The goal of service differentiation is not to provide hard guarantees but to create a distinction among the peers based on their contributions to the system. The basic idea being, the more the contribution the better the relative service.

Specifically, this work makes the following contributions[3]:

- The service differentiation parameters are well understood and studied in the context of the Internet (e.g. delay, jitter, bandwidth) but they are still to be defined for the P2P networks. This work defines a set of parameters that can be used to create service differentiation in P2P networks. These parameters affect the content search and download functions of P2P networks.

- It proposes service differentiation protocol SDP. SDP uses service differentiation parameters and accomplishes the goal by enhancing the Gnutella specification. A preliminary evaluation of the service differentiation achieved during the content search phase shows the promise of the approach.

- It identifies a set of features necessary in a reputation system so that the reputation system can be used for the application of service differentiation.

8.3 Content Quality

Reliable reputations are useful in establishing trust in P2P networks and hence in improving content quality. They can also be used as a substrate for SDP proposed in chapter 5.

[3]This work appeared in NGC 2003.

Concentrating on Gnutella-like P2P networks, in chapters 6 and 7, we proposed two different schemes of tracking peer reputations that possess the properties desired by SDP. The schemes utilize *objective criteria* for updating peer reputations, making peer reputations universally comparable. Most of the other existing proposals for reputation tracking in P2P networks have focused on accomplishing the task in a *decentralized* manner. Even though the decentralized solutions are more in-tune with the spirit of P2P networks, they themselves require cooperation from the peers in the system-the very problem incentive techniques that utilize reputations try to solve.

In order to secure the updates to reputations from peers who act in their self-interest, a partially distributed approach utilizing the RCA is proposed. The use of the RCA infrastructure allows the reputations to be stored locally at the peers which is another property desired by SDP. Specifically, the contributions of chapters 6 and 7 are[4]:

- The reliability issue in reputation tracking in P2P networks has so far only been addressed in an ad-hoc manner in the existing research literature. We formally verified that the reputation tracking in *strong reputations* (referred to as the DCRC in chapter 6) can function correctly in the presence of selfish peers in the system.

- Due to the inherent decentralization in the P2P networks, a low overhead method to track accurate reputations does not seem feasible. We evaluated the trade-offs in the accuracy of reputation computations in the proposed scheme with the state maintenance at the RCA, percentage enrollment of peers in reputation tracking, and the frequency at which the RCA processes the debits and credits to reputation scores in chapter 6.

- The issue of overheads in reputation tracking had also not been addressed before this work. The overheads incurred by reputation tracking could compromise the scalability of the underlying P2P network. The reliability of strong reputations comes at the cost of introducing a centralized bottleneck during content download, and additional overheads. In chapter 7, we proposed *weak reputations* that trade some reliability in

[4]Parts of this work appeared in NOSSDAV 2003.

reputation computations for lower overheads. Analytical evaluation and simulations show that the overheads and reliability trade-offs in weak reputation are more practical for potential deployment.

- We considered another lower overhead (and hence lower reliability) reputation tracking scheme, credit-only reputation computation (CORC). Though it offers certain advantages over the weak reputations, it suffers from the possibility of collusion by selfish peers and is impractical from the standpoint of further consideration.

8.4 Future Directions

Some of the future directions arising out of the work in this dissertation are:

- On-demand P2P overlays have been explored recently [47] to handle large-scale content distribution. For flash crowds, server may be the best place to put in the necessary measures. Application layer multicast using the MWT algorithm proposed in chapter 3 is worth exploring either stand-alone or in conjunction with on-demand P2P overlays proposed in [47] to handle flash crowds.

- The M-DS architecture proposed in chapter 4 can provide scalable service differentiation for IP multicast. It has been argued that multicast functionality should be provided as an application layer enhancement. An architecture similar to M-DS but for application layer multicast is worth a consideration.

- Chapter 5 proposed a set of parameters to be used for service differentiation using SDP. Quantifying how each or a set of those parameters affect the service differentiation perceived by the users would be a useful study to undertake. It would help determine which factors should be incorporated in SDP.

REFERENCES

[1] Aberer, K. and Despotovic, Z., "Managing trust in a peer-2-peer information system," in *Ninth International Conference on Information and Knowledge Management (CIKM)*, Nov. 2001.

[2] Acharya, S. and Smith, B., "An experiment to characterize videos stored on the web," in *SPIE/ACM Conference on Multimedia Computing and Networking (MMCN)*, Jan. 1998.

[3] Acharya, S., Smith, B., and Parnes, P., "Characterizing user access to videos on the world wide web," in *SPIE/ACM Conference on Multimedia Computing and Networking (MMCN)*, Jan. 1998.

[4] Adar, E. and Huberman, B. A., "Free riding on Gnutella," tech. rep., Xerox PARC, 2000.

[5] Aggarwal, C., Wolf, J., and Yu, P., "A permutation based pyramid broadcasting scheme for video on-demand systems," in *IEEE International Conference on Multimedia Computing and Systems(ICMCS)*, June 1996.

[6] Aggarwal, C., Wolf, J., and Yu, P., "The maximum factor queue length batching scheme for video-on-demand systems," *IEEE Transactions On Computers*, vol. 50, no. 2, pp. 97–110, 2001.

[7] Aksoy, D. and Franklin, M., "Rxw: A scheduling approach for large-scale on-demand data broadcast," *IEEE Transactions Of Networking*, vol. 7, no. 6, pp. 846–860, 1999.

[8] Almeida, J., Krueger, J., Eager, D., and Vernon, M., "Analysis of educational media server workloads," in *ACM NOSSDAV*, June 2001.

[9] Almeroth, K., "Validating the multicast mystique," in *IEEE INFOCOM*, Apr. 2001.

[10] Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., and Weiss, W., "An architecture for differentiated services." RFC 2475, Dec. 1998.

[11] Bless, R. and Wehrle, K., "Ip multicast in differentiated services network." Internet Draft, Sept. 1999.

[12] Braden, R., Zhang, L., Bernon, S., Herzog, S., and Jamin, S., "Resource reservation protocol (rsvp) – version 1 functional specification." RFC 2205, Sept. 1997.

[13] Calvert, K., Doar, M., and Zegura, E., "Modeling internet topology," *IEEE Communication Magazine*, July 1997.

[14] Cherkasova, L. and Gupta, M., "Characterizing locality, evolution, and life span of accesses in enterprise media server workloads," in *ACM NOSSDAV*, May 2002.

[15] Cherkasova, L. and Gupta, M., "Analysis of enterprise media server workloads: Access patterns, locality, content evolution, and rates of change," in *IEEE Transactions Of Networking*, to appear 2004.

[16] Cheshire, M., Wolman, A., Voelker, G. M., and Levy, H. M., "Measurement and analysis of a streaming media workload," in *USENIX Symposium on Internet Technologies and Systems (USITS)*, Mar. 2001.

[17] Cohen, E. and Shenker, S., "Replication strategies in unstructured peer-to-peer networks," in *ACM SIGCOMM*, Aug. 2002.

[18] Damiani, E., di Vimercati, S. D. C., Paraboschi, S., Samarati, P., and Violante, F., "A reputation-based approach for choosing reliable resources in peer-to-peer networks," in *9th ACM Conference on Computer and Communications Security*, Nov. 2002.

[19] Dan, A., Sitaram, D., and Shahabuddin, P., "Scheduling policies for an on-demand video server with batching," in *ACM Multimedia*, Oct. 1994.

[20] Dingledine, R., Freedman, M. J., and Molnar, D., *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*, ch. 12: Free Haven. O'Reilly, Mar. 2001.

[21] Dutta, D., Goel, A., Govindan, R., and Zhang, H., "The design of a distributed rating scheme for peer-to-peer systems," in *Workshop on Economics of Peer-to-Peer Systems*, June 2003.

[22] Eager, D., Vernon, M., and Zoharjan, J., "Optimal and efficient merging schedules for video on-demand servers," in *ACM Multimedia*, Nov. 1999.

[23] Estrin, D., Farinacci, D., Helmi, A., Thaler, D., Deering, S., Handley, M., Jacobson, V., Liu, C., Sharma, P., and Wei, L., "Protocol independet multicast-sparce mode (pim-sm): Protocol specification." RFC 2362, June 1998.

[24] "Gnutella protocol specification." `http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf/`.

[25] Gouda, M. G., *Elements of network protocol design.* John Wiley & Sons, 1998.

[26] Gupta, M., Judge, P., and Ammar, M. H., "A reputation system for peer-to-peer networks," in *ACM NOSSDAV*, June 2003.

[27] "Gnutella web caching system." `http://www.gnucleus.com/gwebcache/`.

[28] Harel, N., Vellanki, V., Chervenak, A., and Abowd, G., "Workload of a media-enhanced classroom server," in *IEEE Workshop on Workload Characterization*, Oct. 1999.

[29] Hua, K. A., Cai, Y., and Sheu, S., "Patching: A multicast technique for true video on-demand services," in *ACM Multimedia*, Sept. 1998.

[30] Hua, K. A. and Sheu, S., "Skyscraper broadcasting: A new broadcasting scheme for metropolitan video on-demand systems," in *ACM SIGCOMM*, Sept. 1997.

[31] Kamvar, S. D., Schlosser, M., and Garcia-Molina, H., "EigenRep: Reputation management in P2P networks," in *World-wide Web Conference*, May 2003.

[32] "Kazaa participation level." `http://www.kazaa.com/`.

[33] Kung, H. T. and Wu, C., "Differentiated admission control for peer-to-peer systems: incentivizing peers to contribute their resources," in *Workshop on Economics of Peer-to-Peer Systems*, June 2003.

[34] Lee, D. F., *A formal presentation of electronic commerce protocols*. PhD thesis, The university of Texas at Austin, 2002.

[35] Lee, S., Sherwood, R., and Bhattacharjee, B., "Cooperative peer groups in NICE," in *IEEE INFOCOM*, Apr. 2003.

[36] Li, X., Ammar, M. H., and Paul, S., "Video multicast over the internet," *IEEE Network Magazine*, Apr. 1999.

[37] McDaniel, P. and Jamin, S., "A scalable key distribution hierarchy," Tech. Rep. CSE-TR-366-98, Electrical Engineering and Computer Science, University of Michigan, 1998.

[38] Meyer, D. and Fenner, F., "Multicast source discovery protocol (msdp)." Internet Draft, May 2001.

[39] Padhye, J. and Kurose, J., "An empirical study of client interactions with continuous-media couseware server," in *ACM NOSSDAV*, July 1998.

[40] Perkins, C., "Ip encapsulation within ip." RFC 2003, Oct. 1996.

[41] Qiu, Y. and Marbach, P., "Bandwidth allocation in ad hoc networks: A price based approach," in *IEEE INFOCOM*, 2003.

[42] Ratnasamy, S., Francis, P., Handley, M., Karp, R., and Shenker, S., "A scalable content addressable network," in *ACM SIGCOMM*, Aug. 2001.

[43] Ratnasamy, S., Handley, M., Karp, R., and Shenker, S., "Topologically-aware overlay construction and server selection," in *IEEE INFOCOM*, Apr. 2002.

[44] Ripeanu, M., Foster, I., and Iamnitchi, A., "Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design," *IEEE Internet Computing Journal*, vol. 6, no. 1, 2002.

[45] Rivest, R., "The md5 message digest algorithm." RFC 1321, Apr. 1992.

[46] Saroiu, S., Gummadi, P. K., and Gribble, S. D., "A measurement study of peer-to-peer file sharing systems," in *SPIE/ACM Conference on Multimedia Computing and Networking (MMCN)*, Jan. 2002.

[47] Sherwood, R., Braud, R., and Bhattacharjee, B., "Slurpie: A cooperative bulk data transfer protocol," in *IEEE INFOCOM*, Mar. 2004.

[48] Singh, A. and Liu, L., "Trustme: Anonymous management of trust relationships in decentralized P2P systems," in *International conference on peer-to-peer computing*, Sept. 2003.

[49] Sripanidkulchai, K., "The popularity of gnutella queries and its implications on scalability." White Paper Featured on O'Reilly's website http://www.openp2p.com/, Feb. 2001.

[50] Sripanidkulchai, K., Maggs, B., and Zhang, H., "Efficient content location using interest-based locality in peer-to-peer systems," in *IEEE INFOCOM*, Apr. 2003.

[51] Stoica, I., Morris, R., Karger, D., Kaashoek, F., and Balakrishnan, H., "Chord: A scalable peer-to-peer lookup service for internet applications," in *ACM SIGCOMM*, Aug. 2001.

[52] Streigel, A. and Manimaran, G., "A scalable approach to diffserv multicasting," in *IEEE ICC*, May 2001.

[53] Teitelbaum, B. and Chimento, P., "Qbone bandwidth broker architecture." QBone Bandwidth Broker Work Group, June 2000.

[54] "Internet performance measurement and analysis." http://www.merit.edu/ipma.trends.

[55] Vishnumurthy, V., Chandrakumar, S., and Sirer, E. G., "KARMA : A secure economic framework for peer-to-peer resource sharing," in *Workshop on Economics of Peer-to-Peer Systems*, June 2003.

[56] Xiong, L. and Liu, L., "Building trust in decentralized peer-to-peer communities," in *International Conference on Electronic Commerce Research (ICECR-5)*, Oct. 2002.

[57] Zhong, S., Chen, J., and Yang, Y. R., "Sprite: A simple, cheat-proof, credit-based system for mobile ad-hoc networks," in *IEEE INFOCOM*, 2003.

VITA

Minaxi Gupta holds a three year B.Sc. from Bombay University, India and a two year M.Sc. from Indian Institute of Technology (I.T.T.), Bombay, both in Physics. She came to Rice University (Houston, TX) to pursue a Ph.D. in High Energy Physics. While taking computer programming classes for her research in Physics, Minaxi became interested in Computer Science. She joined Georgia Tech (Atlanta, GA) in 1997 for an M.S. in Computer Science. During her M.S., Minaxi realized her real research interests lied in Networking and decided to transfer over to the Ph.D. program at Georgia Tech in 1999 where she is currently a Ph.D. candidate working with Prof. Mostafa Ammar.

Minaxi's primary research interests are in Networking and Distributed Systems. She is interested in novel architectures and technologies for large-scale content distribution for current and emergent applications from a variety of disciplines.

While a graduate student at Georgia Tech Minaxi had an opportunity to do summer internships at Motorola (Mansfield, MA), I.B.M. T.J.Waston (Hawthorne, NY), and H.P.Labs (Palo Alto, CA). Her persistent hobby is following the financial markets, which she tracks avidly. Minaxi has also done the equivalent of 1/3 of the course work required for an M.B.A. in Finance, from Dupree College of Management, Georgia Tech.