

Design and Performance Evaluation of a System for Epidemic Peer-to-Peer Content Distribution

by

Ali Alagöz

**A Thesis Submitted to the
Graduate School of Sciences and Engineering
in Partial Fulfillment of the Requirements for
the Degree of**

**Master of Science
in
Computational Sciences and Engineering**

Koc University

August 2006

Koc University
Graduate School of Sciences and Engineering

This is to certify that I have examined this copy of a master's thesis by

Ali Alagöz

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Committee Members:

Assistant Prof. Öznur Özkasap, (Advisor)

Associate Prof. Mine Çağlar, (Co-Advisor)

Associate Prof. Attila Gürsoy

Assistant Prof. Alkan Kabakçioğlu

Assistant Prof. Sibel Salman

Date: 09/08/2006

To My Family

ABSTRACT

Peer-to-peer (P2P) cooperative systems are becoming extremely popular as they find diverse applications. One major application area is the content distribution over large-scale networks. As the usage of the Internet grows up, the number of large contents such as software packages and popular movie files, and also the user population retrieving these contents increase exponentially.

In this thesis, we propose and design a peer-to-peer system, SeCond, addressing the distribution of large sized content to a large number of end systems in an efficient and effective manner. In contrast to prior work, it employs an epidemic dissemination scheme for state propagation of available blocks and initiation of block transmissions. In order to support heterogeneity of peers, ease of deployment, scalability, and adaptivity to dynamic peer arrivals/departures, and also to increase the utilization of the system resources, we propose mechanisms such as adjusting protocol parameters according to the bandwidth usages dynamically. We describe our protocol SeCond and its discrete event simulation model. A comprehensive performance evaluation has been accomplished for a wide range of scenarios. A well known and widely used P2P content distribution system is BitTorrent which we also model and compare as a benchmark. Performance results include scalability analysis for different arrival/departure patterns, flash-crowd scenario, overhead analysis, and fairness ratio. The major metrics we study include the average file download time, load on the primary seed, uplink/downlink utilization, communication overhead, and the fairness ratio. SeCond peers download the file faster compared to BitTorrent peers for most of the scenarios and the protocol is as fair as BitTorrent although it has no explicit strategy addressing free-riding. We show that SeCond is a scalable and adaptive protocol which takes the heterogeneity of the peers into account. We also illustrate the applicability of an analytical fluid model to the behavior of SeCond.

ÖZETÇE

Uygulama alanlarının artması, eşler arası yardımlaşmalı sistemler üzerine olan ilginin artmasına neden olmaktadır. Eşler arası yardımlaşmalı sistemlerin ana uygulama alanlarından biri de içerik dosyalarının geniş boyutlu ağlar üzerinde dağıtılmasıdır. İnternet kullanımının artması, yazılım yamaları veya filmler gibi büyük boyutlu verilerin ve bu verilere olan ilginin hızlı bir şekilde artmasına yol açmaktadır.

Bu tez çalışmasında, büyük boyutlu içeriğin çok sayıda son kullanıcıya verimli ve etkin bir biçimde dağıtılmasını hedefleyen bir eşler arası sistem önerilip, SeCond: Eşler arası epidemik içerik dağıtım protokolü, tasarımı gerçekleştirilmiştir. Önceki çalışmalardan farklı olarak, SeCond diğer eşlerin erişilebilir bloklardan haberdar edilmesi ve blok aktarımlarının başlatılması için epidemik dağıtım mekanizmasını kullanmaktadır. Eşlerin farklılığı, uygulanma kolaylığı, ölçeklenebilirlik ve dinamik eş varış/ayrılışları destekleyebilmek gibi özelliklerin yanı sıra, sistem kaynaklarından yararlanmayı artırmak için de bant genişliğinin kullanımına göre protokol parametrelerini ayarlayan mekanizmalar geliştirilmiştir. SeCond protokolü ve onun ayırık durum simülasyon modeli tanımlanmıştır. Birçok senaryo için, protokolün kapsamlı bir performans değerlendirilmesi yapılmıştır. Modellediğimiz ve SeCond ile karşılaştırdığımız BitTorrent protokolü, tanınan ve geniş bir kullanıcı kitlesine sahip olan bir eşler arası içerik dağıtım sistemidir. Performans sonuçları farklı geliş/ayrılış kalıpları için ölçeklenebilirlik analizini, ani yoğunluk senaryosunu, ek yük analizini ve eşitlik oranını içermektedir. Kullandığımız ana başarımlı ölçütleri ortalama içerik indirim zamanı, orijinal içerik yayıncısı üzerindeki yük, indirme/yükleme bant genişliği yararlanma oranı, eşitlik oranı ve iletişim ek yükünden oluşmaktadır. SeCond eşlerinin gerçekleştirilen senaryoların birçoğu için dosyayı BitTorrent eşlerine göre daha hızlı indirdiği ve protokolün özel bir mekanizma önermediği halde BitTorrent kadar adil dağıtım yaptığı görülmüştür. SeCond'un eşlerin farklılığını göze alan, ölçeklenebilir ve uyarlanabilen bir protokol olduğu ortaya konulmuştur. Ayrıca analitik bir akış modelinin, SeCond protokolünün davranışına uygulanabilirliği gösterilmiştir.

ACKNOWLEDGEMENTS

I wish to express my sincere gratitude to my thesis supervisors Assistant Prof. Öznur Özkasap, and Associate Prof. Mine Çağlar for their support, valuable criticism, and endless patience throughout my study. I would like to thank Associate Prof. Attila Gürsoy, Assistant Prof. Alkan Kabakçioğlu and Assistant Prof. Sibel Salman for their valuable comments and time in reviewing this thesis as my committee members. I would like to thank to my close friend Emrah Ahi for his encouragement and support. I was fortunate to have such great friends close to me during my graduate education; Aydın Varol, Uğur Kaplan, Mert Sedef my roommates, Emre Dikmen, and Murat Ötkür my officemates. Finally, I would like to express my special thanks to my family; my parents Nursen and Kaya, my brother Erdinç, my wife Aylin, and my wife's parents Meserret and Erol for their endless love and patience throughout my study.

TABLE OF CONTENTS

LIST OF TABLES	ix
LIST OF FIGURES	x
NOMENCLATURE.....	xii
Chapter 1: Introduction	1
Chapter 2: Related Work.....	5
2.1 Infrastructure-based Content Distribution Solutions	6
2.2 Cooperative Content Distribution Solutions	6
2.2.1 Multicast-Based Schemes	7
2.2.2 Mesh Cooperative Architectures.....	8
2.2.3 Erasure Codes	9
2.3 BitTorrent.....	10
2.4 Slurpie	13
2.5 Epidemic Algorithms	14
Chapter 3: SeCond: A System for Epidemic P2P Content Distribution	16
3.1 Model of the Index Server Agent.....	17
3.2 Model of the Peer Agent	19
3.3 Structure of the file	23
3.4 State propagation.....	24
3.5 View Construction and Update	25
3.6 Download/Upload Decisions	27
3.7 Protocol Parameters, Algorithms, Events and Message Structures	28
3.7.1 Properties and Algorithms of Index Server Agent.....	29
3.7.2 Properties and Algorithms of Peer Agent	31

Chapter 4: Simulation Model.....	37
4.1 Development Environment and Simulation Model	38
4.2 Design and Implementation Details	39
4.2.1 SeCond	39
4.2.2 BitTorrent.....	40
4.3 Shared File	42
4.4 Peer Arrival/Departure Patterns and System Size.....	42
4.5 Network and Link Properties	43
4.6 Terminology of Simulation Evaluation.....	45
4.7 Default Simulation Settings	46
Chapter 5: Simulation Results	47
5.1 Parameter Analysis	47
5.1.1 Gossip Interval	48
5.1.2 Parallel Upload Limit.....	51
5.1.3 Connection Size	54
5.2 Performance Evaluation	56
5.2.1 Arrival Rate Analysis.....	57
5.2.2 Departure Rate Analysis	59
5.2.3 Scalability Analysis for Different Arrival Patterns.....	61
5.2.4 Fairness Ratio.....	68
5.2.5 Overhead Analysis	71
Chapter 6: Analytical Approaches on P2P Cooperative Systems	75
6.1 Markovian Models	75
6.2 Fluid Models	77
6.3 Studies on Heterogeneous Systems.....	79
6.4 Analytical Comparison	80
Chapter 7: Conclusion and Future Work.....	82
BIBLIOGRAPHY	85
VITA.....	89

LIST OF TABLES

Table 3.1. Events handled by the index server agent.....	29
Table 3.2. Special variables of index server agent.....	29
Table 3.3. Protocol parameters of index server agent.....	29
Table 3.4. Data structures of index server agent.....	29
Table 3.5. Functions called by index server agent.....	29
Table 3.6. Message formats	30
Table 3.7. Pseudo-code for events and functions.....	30
Table 3.8. Events handled by the peer agent.....	31
Table 3.9. Functions called by peer agent.....	31
Table 3.10. Data structures of peer agent.....	32
Table 3.11. Special variables of peer agent	32
Table 3.12. Protocol parameters of peer agent.....	32
Table 3.13. Pseudo-code for function and events	33
Table 3.14. Message formats	36
Table 4.1. Bandwidth distribution for heterogeneous system.....	45
Table 4.2. Common simulation settings.....	46
Table 4.3. SeCond parameter settings.....	46
Table 4.4. BitTorrent parameter settings	46
Table 5.1. Bandwidth distribution according to peer ids	69

LIST OF FIGURES

Figure 2.1. Categorization of content distribution solutions.....	05
Figure 2.2. Simple epidemic message dissemination	15
Figure 3.1. State diagram for the index server	18
Figure 3.2. Functional model of an index server	19
Figure 3.3. State diagram for peer.....	21
Figure 3.4. Stored data for the peers in the gossip receivers list and gossip senders list...	22
Figure 3.5. Functional model of a peer agent	23
Figure 3.6. Gossip Message Construction	25
Figure 3.7. Illustration of view update decision process.....	26
Figure 3.8. Illustration of upload decision process	28
Figure 4.1. Overview of the simulation process	38
Figure 4.2. Illustration of the block dissemination in SeCond.....	44
Figure 5.1. System with limited bandwidth; 512/128 Kbs Downlink/Uplink bandwidths	49
Figure 5.2. System with high bandwidth; 1500/512 Kbs Downlink/Uplink bandwidths	50
Figure 5.3. Effect of smart peer policy on uplink/downlink utilizations	50
Figure 5.4. Effect of smart peer policy on average file download time.....	51
Figure 5.5. Parallel upload limit vs. Downlink and uplink utilizations	53
Figure 5.6. Parallel upload limit vs. Average file download time	53
Figure 5.7. Parallel upload limit vs. Load imposed on the primary seed.....	54
Figure 5.8. Connection size vs. Average file download time	55
Figure 5.9. Connection size vs. Data size uploaded by primary seed.....	56
Figure 5.10. System size	58
Figure 5.11. Arrival rate vs. Load imposed on the primary seed.....	58

Figure 5.12. Arrival rate vs. Average file download time	59
Figure 5.13. Arrival rate vs. Time.....	60
Figure 5.14. Mean departure time vs. Average file download time.....	60
Figure 5.15. Mean of exponential departure time vs. Load imposed on the primary seed.....	61
Figure 5.16. Arrival Patterns	62
Figure 5.17. Maximum number of active peers vs. System Size.....	63
Figure 5.18. Average file download time for increasing arrival pattern vs. system size	64
Figure 5.19. Average file download time for decreasing arrival pattern vs. system size	64
Figure 5.20. Average file download time for constant arrival pattern vs. System size	65
Figure 5.21. Average file download time for triangular arrival pattern vs. System size ...	65
Figure 5.22. Load imposed on the primary seed for increasing arrival pattern vs. system size	66
Figure 5.23. Load imposed on the primary seed for decreasing arrival pattern vs. system size	67
Figure 5.24. Load imposed on the primary seed for constant arrival pattern vs. system size	67
Figure 5.25. Load imposed on the primary seed for triangular arrival pattern vs. system size	68
Figure 5.26. Average file download time vs. System Size	69
Figure 5.27. Load imposed on the primary seed vs. System Size.....	70
Figure 5.28. Fairness Ratio	70
Figure 5.29. Histogram for BitTorrent state update messages.....	72
Figure 5.30. Histogram for SeCond gossip messages.....	73
Figure 5.31. Number of state update messages sent by BitTorrent peers.....	73
Figure 5.32. Number of gossip messages sent by SeCond peers.....	74
Figure 6.1. Normalized number of leechers and seeds as a function of time	81

NOMENCLATURE

ARPA	Advanced Research Project Agency
WWW	Word Wide Web
CERN	European Organization for Nuclear Research
HTTP	HyperText Transfer Protocol
HTML	HyperText Markup Language
P2P	Peer-to-Peer
MP3	MPEG-1 Audio Layer 3
CPU	Central Processing Unit
RMTP	Reliable Message Transport Protocol
SRM	Scalable Reliable Multicast Protocol
FTP	File Transfer Protocol
URL	Uniform Resource Locator
TCP	Transmission Control Protocol
FIFO	First in First out Queue
TFT	Tit-for-tat policy
LRF	Local Rarest First policy

Chapter 1

INTRODUCTION

Roots of the Internet appeared in 1969 with the ARPAnet [1] system. When World Wide Web (WWW) was invented at European Organization for Nuclear Research (CERN) between 1989 and 1991, main goal of the developers was creating a platform for researchers on which they would exchange reports, photos, drawings and other documents easily. In early 1990s, the HTTP, HTML, Web Server, and the first browser, the four main components of the Web, were introduced in CERN. The explosion of the Internet happened after the emergence of the World Wide Web, which brought the Internet into the usage of millions of people worldwide. From early 90's up to now, Web has found lots of different application areas such as e-mail, electronic commerce, instant messaging, and P2P file sharing of Mp3 audio files or movies. With the second millennium, most of the daily routines can be handled by the Web such as financial transactions, music and video downloads, and video conferencing with a person from the other side of the world. However, the main idea behind the Web has not changed: letting people reach information and contents rapidly.

Most of the applications designed to be deployed on the Internet is based on the client-server architecture. Basically, client-server architecture separates computers or processes into two categories: clients and servers. Each client can send requests to a server and waits for reply. A server is the passive component and waits for requests. Upon receiving a request, it serves and sends a reply. Client-server approach leads to the centralization of many services when a powerful server can handle many clients at the same time. As an example, many users can visit the same web page. Such centralization leads to a lot of problems caused by the limitation of resources such as

the bandwidth of communication links, the storage space, channels and the computer power.

Emergence of new digital technologies made it possible to store movies, music, images in digital environments with high quality. The Internet is the most convenient and cheapest way to sell, to buy, and to share them. On the other hand, structure of the WWW was not convenient enough to share files. The scale achieved by the Web and its distributed structure make it difficult for users to find the files they need. In 1999, first P2P file sharing system, Napster [2], was introduced. Napster was a localized index for MP3 files shared by the users. However, because of its centralized structure, Napster encountered legal troubles and was shut down in 2001. To get around this problem, decentralized solutions came into market. Gnutella [3] was the first example of this kind of solutions. Kazaa [4] was another P2P file sharing platform introduced in 2001. Despite being only a few years old, P2P file sharing systems have gained tremendous popularity and now include millions of users.

Beginning from the second half of the 1990s, Internet and its content have evolved continuously. With respect to recent statistics, 15.7 % of all people take advantage of the Internet for some reason. Moreover, it is reported that the usage growth of the Internet equals to 183.4 % between the years 2000-2005 [5].

As the usage of the Internet grows up, the number of contents and the number of people retrieving these contents increase exponentially. That content may be a HTML page, a software packet or a popular movie. As the number of end systems retrieving that content simultaneously increases, the accessibility of the content is affected adversely. Especially, distribution of popular large files may be very obstructive due to bottleneck in the server of the file. As soon as a popular file is released in the file server, a flash-crowd scenario is expected in which lots of users strive to achieve a copy of the file suddenly. This popularity may result in a failure of distribution for all of the users because of the collapse of the file server. Even if we assume that the file server can handle large number of requests simultaneously, the time period required to obtain a full copy of the file increases linearly with respect to system size. On the other hand,

P2P systems create a platform where people find many files to transfer, but generally they do not intend to disseminate a popular file or alleviate the load of the original source such as [2], [3], and [4]. The main goal of these P2P applications is locating sources for the desired files, not increasing number of the copies of the file as fast as possible.

BitTorrent [6] and Slurpie [7] are two important examples of cooperative P2P protocols which utilize system resources during the dissemination of the file. The main idea of cooperative P2P systems is exploiting resources of the clients and alleviating the load imposed on the original file source. Namely, instead of only requesting blocks of the file from the file source, peers are also involved in the dissemination process. Cooperative P2P protocols let the end systems decide the source for the data they strive to achieve locally. This locality increases the utilization of the system resources.

In this thesis, we propose a cooperative content distribution protocol titled as SeCond, a System for Epidemic Peer-to-Peer Content Distribution, which targets the distribution of a large file to many users simultaneously in an efficient and effective way. The major metrics we study include the average file download time, load on the primary seed, uplink/downlink utilization, communication overhead, and the fairness ratio.

The contributions of this thesis can be summarized as follows:

- A new cooperative content distribution protocol, SeCond, has been defined for peer to peer distributed systems.
- An efficient state propagation mechanism inspired by the epidemic methods for the new protocol has been developed.
- An adaptive mechanism has been developed for the protocol to utilize the system resources for heterogeneous systems.
- The protocol has been modeled and a discrete event simulator has been implemented in Java programming language.
- The comprehensive performance evaluation of the protocol has been performed by a large set of simulations for a wide range of scenarios. It has been shown that

SeCond is a scalable, adaptive protocol which takes the differences among the peers into account.

- The effect of the protocol parameters on the performance of the protocol has been analyzed by the simulation studies.
- To compare performance of our protocol with a real life application, a discrete event simulator modeling the BitTorrent protocol has been implemented in Java programming language.
- It is shown that SeCond peers download the file faster for most of the scenarios compared to BitTorrent peers and the protocol is as fair as BitTorrent although it has no explicit strategy addressing free-riding.
- A survey of analytical approaches on cooperative content distribution is made and a comparison of an analytical model with the simulation results of SeCond is accomplished.

Chapter 2

RELATED WORK

The general problem of getting popular content from heavily loaded servers is well studied. There are many studies addressing content distribution. We can categorize these solutions into two main categories of infrastructure-based solutions and cooperative content distribution solutions as given Figure 2.1. In this chapter, we will study benefits and drawbacks of these solutions. In section 2.1 and 2.2 we will review the infrastructure-based solutions and cooperative solutions respectively. Multicast-based schemes, mesh cooperative architectures and erasure codes will be given as subsections of section 2.2. Although two cooperative content distribution solutions BitTorrent and Slurpie can be stated under mesh cooperative architecture section 2.2.2, we prefer to study them separately in sections 2.3 and 2.4 respectively since they are highly related to our work. Finally, we will introduce epidemic algorithms in section 2.5.

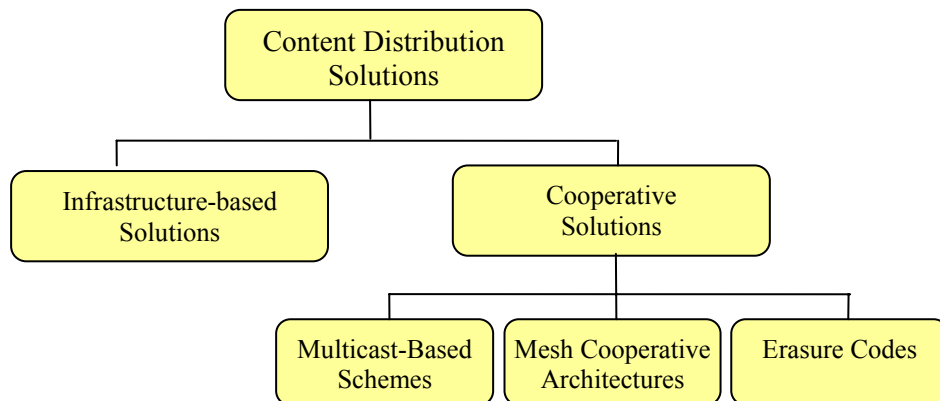


Figure 2.1. Categorization of content distribution solutions

2.1 Infrastructure-based Content Distribution Solutions

If infrastructure can be organized, the load of the server caused by content distribution can be alleviated, the efficiency of the process can be increased in turn. One example to existing solutions addressing content distribution is mirroring or replication of server. Akamai [8] is the best example deployed in the Internet for years which runs tens of thousands of servers. Although mirroring seems to be the best solution to increase server's resources, in order to reply large number of requests in flash-crowd scenarios, serving capacity of the file server should be increased linearly with respect to the number of requests which is impossible in most cases.

Another approach that can be stated under this title is caching hierarchies. One example to this kind of solutions is Squid [9]. It stores the requested Internet objects on a system closer to the requesting site than to the source in order to reduce the load on the server and the client return time.

However, all the solutions under this title require infrastructure support which is usually an expensive supply. Moreover, demand on the content should be predictable so that the content providers can make provision against the bottlenecks in data distribution. But, of course, it is not possible to anticipate the demand accurately all the time. Misconstructions may lead to the waste of resources.

2.2 Cooperative Content Distribution Solutions

Instead of only requesting blocks of the file from the server, nodes may also be involved in the distribution of the specified popular file. This approach increases the quality of the service that the peers get. Actually, as the number of nodes requesting service in the system increases, in most of the cases, the capacity of the resources of the server (e.g. CPU, bandwidth) can not be raised in parallel to the growth rate of the system. If this rate exceeds the upper bound that the server can handle, it will lead to the collapse of the server which is the same thing as failure of content distribution. Hence, collaboration is not only for the benefit of nodes, but also, it seems to be an

obligation especially for a flash-crowd scenario. The main question that should be asked is how this collaboration is organized. The performance of the system mostly depends on the effectiveness of this collaboration. Although the idea lying behind the cooperative solution seems very simple, implementing such a protocol organizing nodes effectively in a dynamic system where arrival and departure of nodes occurs very frequently is a challenging topic. There are many studies addressing content distribution problem in the literature. We may classify them into 3 main categories as will be discussed next.

2.2.1 Multicast-Based Schemes

As multimedia files and programs such as video conferencing and movies find more places in the Internet, bandwidth requirements increase in parallel to this deployment. Usually, it is impractical or too costly to improve bandwidth capacities of the network. Hence, instead of sending copies of the data separately to requesters, multicasting sends a single copy of the data to those clients who request it. Multiple copies of data are not sent across the network. Multicast solutions strive to deliver the information to a group of destinations using the most efficient strategy to deliver the messages over each link of the network only once and only create copies when the links to the destinations split.

Multicast schemes can be categorized according to layer they serve on. Although there are many working at IP layer like [10], since they require infrastructure support, those running on application layer such as [11], [12], [13], [14] are more easy to deploy. Another criterion to classify multicast schemes is the degree of reliability they offer. The first group is multicast protocols offering reliable multicast. The reliability may include atomicity or message delivery ordering. Atomicity property requires that the data is delivered to the entire group or no one obtains it. Protocols offering strong reliable multicast schemes suffer from scalability. Moreover, their performance can become unstable under stressed conditions [15], [16], [17] which is something that we try to solve in distribution of popular contents. Other group offers a better scalability, but offers best-effort reliability. Internet MUSE protocol [18], Reliable Message

Transport Protocol (RMTP) [19], Scalable Reliable Multicast protocol (SRM) [20] can be given as examples of this group. Bimodal Multicast is another multicast protocol offering scalability and probabilistic reliability [21]. Although Bimodal Multicast provides a high level of reliability and scalability, it is not adaptable to dynamic systems. Actually, most of the multicast solutions do not intend to operate well under conditions where node arrivals and departures occur frequently. Moreover, nodes do not decide on the sources of the data dynamically by taking network conditions into consideration. The path constructed for data flow is optimized for data stream for most of the multicast schemes.

Hence, although multicast solutions seem to operate well for video conferencing, the transmission of live or recorded events using audio and video, reliable and scalable multicast has still many difficulties that should be overcome.

2.2.2 Mesh Cooperative Architectures

Unlike the traditional client-server based file sharing solutions like FTP, P2P file sharing solutions have the advantage that all the clients in the system also act as a file server. That cooperation between clients makes the P2P file sharing system more scalable. However, in a system where millions of peers request and share files, P2P protocol should have the ability that it matches the requesters and servers for the files easily. Moreover, this matching should not be done without taking utilization of peer sources into account since performance of the file distribution is directly affected by utilization of the sources.

Classic P2P systems create a platform where people find many files to transfer, but generally they do not intend to disseminate a popular file or alleviate the load of the original source. However, the main goal of these kinds of P2P applications is locating sources for the desired files, not increasing the number of copies of a file as fast as possible. Two important examples of P2P protocols whose main goal is organizing the peers sharing and requesting the same file into an overlay network are BitTorrent [6] and Slurpie [7]. This kind of P2P protocols creates a mesh and tries to replicate the

content rapidly. Fast replication of the file leads to an increase in number of seeds which is the solution for popular content distribution. The details of these protocols are given in sections 2.3 and 2.4.

Julia content distribution protocol [22] is another study addressing distribution of large files over P2P networks. Julia is a network aware protocol. Namely, it tries to reduce the overall network overhead and incur a balanced load on the network during dissemination. At the beginning, the nodes do not have information about delays and bandwidths of other participants in the system. As the download progresses, the nodes gather statistics about the network. This knowledge is used to initiate transmissions from the closer nodes. Distance is measured with a combination of bandwidth and latency. As the download progresses, nodes from the closer group are chosen for transmissions. Moreover, the amount of data exchanged between the end points of the transmission is increased with respect to utilization of the connection. Through simulations, it is shown that Julia achieves slightly slower average finishing times relative to BitTorrent, and reduces the network overhead by 33%.

An epidemic based distribution is studied in [23]. A gossip protocol that disseminates k blocks to the whole group in $\theta(k + \ln n)$ rounds is proposed, where n is the system size. It is assumed that k blocks of the file are distributed among k nodes of the system and at each gossip round a node may initiate one outgoing transmission and one incoming transmission. In order to have a uniform distribution of blocks in the group so that peers do not encounter ‘rare’ block problems, a coloring mechanism is deployed.

2.2.3 Erasure Codes

In this set of approaches, a message of m blocks is transformed into a message with n blocks where $n > m$ such that the original message can be recovered from a subset of those blocks. Instead of distributing m different blocks of the data over different paths, linear combinations of the initial blocks are forwarded to nodes. Upon receiving these

re-encoded blocks, new linear combinations of the blocks received before are forwarded to other nodes. The main advantage of this approach is that a node can reconstruct the original message with any linearly independent combinations of m blocks. This prevents bottlenecks due to rare blocks since no specific combination is more important than any other combination.

Erasur codes have been used actively to transfer bulk data [24], [25], [26]. The system becomes resilient to message loss and retransmissions are refrained by erasure codes. In Digital Fountain approach, to recover a message of size n , nodes require obtaining a subset of encoded symbols of size n . Since these sets acquired by the nodes may intersect, collaboration and reconciliation are required between the nodes which are more challenging compared to unencoded data distribution. Encoding is done only by the file server. Without knowing the data sent is useful for the receiver nodes, the nodes only forward messages. If the constructed topology includes cycles, it leads to waste of resources. In a large scale dynamic system, keeping the topology in a state that cycles do not appear is another hard issue to overcome. As the number of different erasure codes increases in the system, the probability of finding useful codes increases also. But, since erasure codes are generated by only a server, it is directly correlated to the capacity of server. The protocol proposed in [27] is based on network coding. It can be thought as a generalization of Digital Fountain Approach where encoding is done by nodes also, not only by the file server. With Network coding, the probability of forwarded messages being gainless is reduced.

2.3 BitTorrent

BitTorrent is a popular file distribution protocol for publishing large files. Actually, BitTorrent is a P2P application deployed in the Internet for several years. A BitTorrent system consists of ordinary web server, meta info file with extension .torrent, tracker, and the seed [28]. A user who wants to download the file should know the URL of the web server holding the torrent file. This torrent file contains length, name, hashing information and URL of the tracker. After obtaining the necessary torrent file,

downloader goes to the tracker to get a list of active downloaders and at least one seed. Trackers are not involved in the actual file distribution. They only hold information about active peers in the system and help them to find each other. To hold peers list update, active peers send a message that reports their existence. These messages are sent every 30 minutes periodically. Similarly, a leaving message is also sent. A peer with a complete copy of the file is called seed. Other peers are said to be leechers.

Upon joining the torrent, standard tracker algorithm returns a random list of active peers of size 50. However, newly joined peer tries to maintain 20-40 connections among given list. If the number of active connections falls below 20, additional peers are requested from the server.

To enable parallel downloading [29], BitTorrent divides the file into blocks of size 256 KB each. Whenever a block is downloaded from a seed or a peer, it is announced to other peers in the view. However, in order to assume a block is downloaded, an integrity check must be applied to the block with the help of hash information supported by the torrent file.

The idea behind BitTorrent is organizing the peers in such a way that the load of the seed is distributed to the whole system. To achieve this aim, peers should help each other. More precisely, they should upload the blocks they have downloaded. However, it is reported in [30] that 66 percentages of Gnutella users were free-riders in 2000. This value reaches %85 percentage in 2005 [31]. Similarly it is noted that as high as 30% of the Napster peers that report their bandwidth as 64 Kbps or less actually have a significantly greater bandwidth. In systems similar to Napster, a peer reports a smaller bandwidth than the real value, in order to discourage others from initiating downloads and consuming the peer's available bandwidth [32]. Unfortunately, these results show the peers are not willing to cooperate. Hence, since the performance of BitTorrent like systems are directly influenced by the level of cooperation among peers, *tit-for-tat* strategy is used which encourages peers to upload. This means as long as you serve, you are served. By default, a peer serves at most 4 peers simultaneously. These 4 peers are

updated with respect to upload rates peers provide at every 10 seconds. On the contrary, a seed looks for the best 4 downloaders with respect to download rates they offer. This selection process is called as choking. However, choking a peer does not mean that this peer will also choke the counterpart since choking is not reciprocal. To give a chance to peers that may offer better upload rates, an optimistic unchoke is performed. At every 30 seconds, a peer is unchoked regardless of the upload rate it offers. On the other hand, a simulation based study [33], shows that tit-for-tat strategy does not prevent peers to upload more than they download in heterogeneous systems.

Another selection procedure in BitTorrent is block selection. It is more likely to download rare blocks from the seed. Hence, BitTorrent strives to increase the number of rare blocks in the system. *Rarest first* algorithm is used to decide which block will be downloaded next. However, upon joining the torrent, the aim of a peer is to have at least one block to upload. Because of this, *random selection* algorithm is used at the beginning.

Since BitTorrent has been deployed in the Internet for several years, there are some measurement studies. The analysis in [34] shows that both downloaders and seeds upload data to other clients. Moreover, the upload and download rates are positively correlated which indicates the success of tit-for-tat. The portion of seeds is found to be higher than 20% with a peak at 40% during the measurements and the amount of the data uploaded by seeds is approximately twice the data uploaded by downloaders. They also evaluate BitTorrent as a client which downloaded the file from 20 peers, 8 of which are seeds. But, none of these peers were in the initial peer list given by the tracker.

A similar measurement was done in [35]. One of the issues they investigated is the behavior of the peers after completion of a download. The authors report that only 17% of the peers that finish downloading remains in the system. This percentage falls below 4% after 10 hours.

2.4 Slurpie

Slurpie is another P2P paradigm based cooperative data transfer protocol [7]. Slurpie is basically a P2P protocol designed for flash crowd scenarios. One of the two main goals intended in this P2P protocol is reducing the download times of large popular files for clients. The second goal is reducing the load on the servers. The underlying idea of Slurpie is exploiting source of clients during dissemination of the file as it is intended in BitTorrent.

There are two kinds of servers in Slurpie, one of which is called a source server. This server is the place where complete file is physically stored. The other one is called a topology server. Topology server is responsible for maintaining the list of the peers downloading the same file. Before starting to download the file, a peer should register itself to topology server. Topology server returns the address of the last registered peer as a response to registration query message. Actually, it returns last ψ peers, a protocol parameter taking small values. Contrary to BitTorrent, a peer constructs its own mesh using this given peer id. It can receive update messages from that node and add new edges to its mesh using these update messages. Mesh formation process of Slurpie is similar to that given in [11]. Slurpie also deploys swarming technique which divides the file into small size of blocks to enable parallel downloading. The chosen block size is 256 KB in Slurpie. This value is the smallest size that TCP overhead is effectively amortized. Propagation of available blocks is handled by the help of the update messages forwarded through the meshes. Using these update messages, a peer can either download a given block from a peer or from the source server.

Each node has η neighbors. The number of neighbors n is continually updated depending on the available bandwidth and as the new nodes are discovered. There is a bandwidth estimation algorithm running once a second in Slurpie. If it returns underutilized, a new randomly selected neighbor is added to mesh of the peer. Actually, each node tries to maintain $\eta \geq \theta(\log n)$ where “ n ” is an estimation of the nodes in the system.

The block list is presented as a bit vector. Information about U other nodes is stored in each node where U is calculated locally. The bit vectors are stored in a data structure known as update tree. In this protocol, before downloading any block from the server, blocks stored in the nodes are downloaded. If there is more than one node containing the same block in the system, one peer is selected randomly. Once a TCP connection is established, the node continues downloading any blocks that it does not have from that peer. The number of opened parallel connections is calculated by the help of bandwidth estimation algorithm. A block which is not stored in any peer is requested from the source server with a probability k/n to control the load at the server.

In comparison to BitTorrent, Slurpie alleviates the load on the topology server. However, performance of the Slurpie in the real world is not examined. Moreover, its algorithm is more complex than BitTorrent and requires a successful estimation of the actual group size.

2.5 Epidemic Algorithms

In recent years, epidemic algorithms have gained popularity as an effective solution for information dissemination in large scale settings, particularly for P2P systems deployed on Internet or ad-hoc networks. Since they are robust, resilient to failures, and easy to deploy, they have found many application areas such as database replication [36], failure detection [37], data aggregation and monitoring [38]. In addition to their inherent scalability, they can easily adapt themselves to dynamic conditions. It is possible to adjust the parameters of an epidemic algorithm to achieve high reliability despite process crashes and disconnections, packet losses, and a dynamic network topology.

Epidemic algorithms mimic the spread of a virus in the community. Infected individuals pass on a virus to those with whom they come into contact. Although, spread of a disease in the community may have dramatic consequences, utilizing the idea behind the epidemics may be very effective for information dissemination in large scale settings. Each process in a distributed system relays the received messages to

randomly chosen peers rather than to a server or cluster of servers in charge of forwarding it. In turn, each of these processes forwards the information to other randomly selected processes, and so on.

Epidemic algorithms alleviate the load imposed on the original message source since they utilize the system resources. Each node in the system is potentially involved in the distribution of the message. Every node buffers received messages up to a limited buffer capacity b , and at each gossip round, selects f distinct nodes from its view as gossip targets to forward the message. The parameter f is called as *fan-out*. A message is removed from the buffer after being forwarded t times. Reliability of the algorithm may be tuned by adjusting these parameters. Through epidemic dissemination, eventually the message will be received by all members with high probability in the order of $\log n$ rounds. Figure 2.2 illustrates simple message dissemination

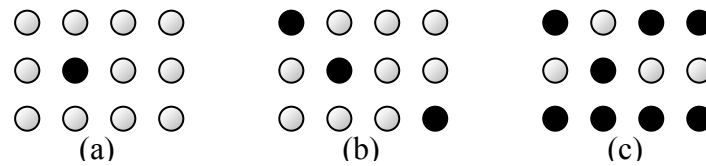


Figure 2.2. a) Source (Black circle) initiates information dissemination. b) In the first round, two randomly selected peers are infected. c) Later, these peers are also involved in the dissemination. Eventually, all the peers in the system will be infected.

In the simplest form of epidemic dissemination, if a member receives a message it directly forwards the message to a randomly chosen member. This scheme is fast but has a problem. A message can be sent to a node several times unnecessarily. The more sophisticated approach used in recent algorithms as well as this thesis is *anti-entropy* where the unnecessary transmission problem is handled by introducing a request-transmit procedure. Each member periodically chooses f random members from its neighbors and sends its message history. Upon receiving of these gossip messages, the nodes may request some messages they have missed from gossip sender (*pull-based* approach) or they may send the messages missed by gossip sender to it (*push-based* approach). The combination of both methods is also possible.

Chapter 3

SeCond: A SYSTEM FOR EPIDEMIC P2P CONTENT DISTRIBUTION

It is well known that client-server based solutions are not appropriate for distribution of large files such as CD images, and software patches. They are ineffective since they can not handle flash-crowd scenarios. Likewise, they do not utilize system resources efficiently. During distribution of the content, resources of the participating peers such as upload bandwidths, and processing capacities stay idle. In order to alleviate the load imposed on the content source, these idle resources can be utilized. Hence, a well designed protocol should have the following properties:

- **Scalability:** As the popularity of the released content increases, the number of users trying to achieve the file simultaneously also increases. Hence, a well designed content distribution protocol should be able to handle large set of users at the same time.
- **Adaptive to dynamic arrivals and departures:** During distribution of the content, for most of the cases users' arrival rate and arrival times may not be anticipated before. Similarly, a user may leave the system without notice. An efficient protocol should be able to operate under dynamic conditions.
- **Easy to deploy:** Although some of the protocols seem to operate well in theory, it is hard to deploy them in real life. That might be due to the requirement of router support or difficulties in the implementation of protocols.
- **Heterogeneity:** Among millions of geographically distributed users, download and upload bandwidths, hardware properties (such as CPU speed) differ from one user to another. Similarly, different network conditions may be observed at different

locations. In order to operate efficiently, the platform for content distribution has to take these differences into account.

SeCond is a P2P paradigm based protocol addressing the distribution of large sized files to large number of end systems in an efficient and effective way. During distribution of the file, participant peers cooperate with each other by uploading downloaded blocks of the file. SeCond deploys epidemic information dissemination scheme for propagation of the available blocks and initiation of the block transmissions. Moreover, in order to achieve the quality factors stated above and increase the utilization of the system resources, we propose some methods such as adjusting protocol parameters according to the bandwidth usages dynamically. SeCond's P2P network consists of peers and the index server. All peers striving to achieve a copy of the file should visit the index server which is the entrance point of the system. Upon obtaining a subset of the active peers in the system, peers can start to cooperate. Namely, while they are downloading some chunks of the file, they serve the chunks downloaded before. In this chapter, details of the SeCond and basic agents working on the index server and the participating peers are described.

3.1 Model of the Index Server Agent

Corresponding to each file available for download, there is a central component called the index server that keeps track of the peers currently in the system. Mesh of a peer consists of the participants with which the peer cooperates. Construction of meshes in order to cooperate with other participants is done via index server. Index server is the entrance point to the system. It helps the peers to find each other. To initiate a file download, a peer should register itself to the index server. After registration, the index server returns a random subset of peers currently downloading the file which is called the *view* of the peer. The list returned by the index server contains peers *randomly* selected among the registered ones. Since some peers may leave the system without notification or may not be willing to add a new peer to its mesh, normally, the length of

the returned list is greater than the view size of the peers. The state diagram for the index server during registration period of a newly joined peer is given in Figure 3.1. To update global view of the index server, active peers report their state to the server periodically. Moreover, if a peer realizes that one of the peers in its partial view is not active, it reports that dead peer also. The index server periodically runs dead peer detection routine given in Table 3.6. If the number of active peers of a peer falls below a predefined lower bound, called as *minimum connection size*, it asks the index server for new neighbors.

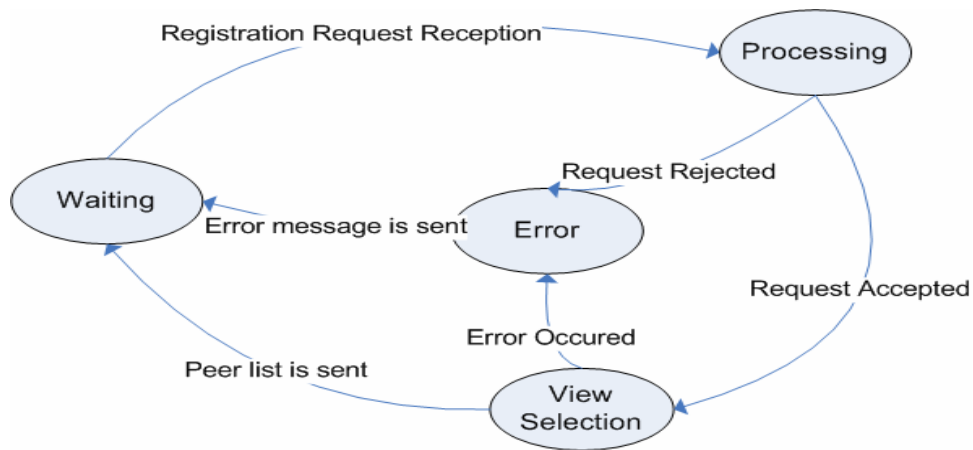


Figure 3.1 State diagram for the index server

To meet the requirements of the protocol, the agent working on the index server should be able to perform the following basic functionalities:

- It should be able to establish and maintain connections with the peers. It must be able to accept incoming/send outgoing messages from/to the peers.
- It must be able to maintain view of the entire system. State of the system is changeable. New peers may arrive and some peers may leave the system.
- It must be able to process the incoming messages from the peers.

The index server has the following components:

- **Queuing Module:** Includes a message buffer to store incoming messages.

- **Processing Module:** Includes functions and procedures for operations such as communication with other peers, helping peers to construct their views, i.e. returning a randomly selected subset of the registered peers, managing the view of the system.
- **Peer-Base:** It stores the data about the registered peers. This data includes unique address of the peer, type of the peer e.g. a seed or a leecher, registration time, and last state update time.
- **Content-Base:** Contains basic data about the shared content: size of the file, number of blocks the file is divided into, publisher of the file.

Figure 3.2 shows these components and their functional relations.

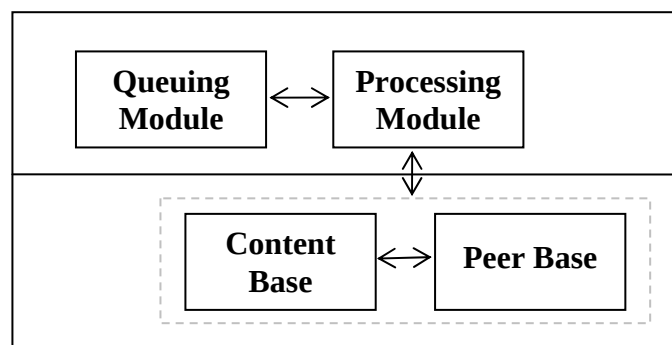


Figure 3.2. Functional model of an index server

3.2 Model of the Peer Agent

A peer takes an important part in the distribution of the file. It communicates, cooperates, keeps track of the peers in its view, and acts as a file server for the part of the data it downloaded before. A peer may join the P2P network anytime and leave the system without notification. To satisfy the protocol requirements, a peer should meet the following functionalities:

- Communication with the other peers and the index server. It must be able to accept incoming and to send outgoing messages from /to them.
- Maintaining the *partial* view of the whole network. Partial view of the peer is changeable. New peers may arrive and some peers may leave.
- Processing the messages coming from the peers.
- Management of the peer states. A peer's *state* represents the ids of the available blocks. This state information is continuously updated via messages sent by the peers.
- Management of the data about the shared content, i.e., size of data, densities of the blocks in the local view of the peer, downloaded blocks up to that time, reconstruction of the file after completion of the download for all the blocks.

In Second, a peer communicates with two sets of peers. The first one, *Gossip Receiver List*, includes the peers to which the peer uploads the available blocks. The second one, *Gossip Sender List*, consists of the peers from where the peer downloads the blocks of the file. Although one peer may exist in both of the lists, these two lists may not be identical. Figure 3.4 shows the data stored for the peers. Agent id states the order of the peer agent in the list. Second columns are Internet addresses defined by *Internet protocol (IP)*. For the gossip senders, states of the peer are also held. *Peer state* stands for the blocks downloaded by the peer. Density decisions of the blocks are given locally with respect to state information. Time entry is used to determine the time of the last incoming block transmission from the given peer. For the gossip receiver list, time entry states times of the last outgoing block transmissions. Each peer stores an array where downloaded block ids are ordered according to block download times. Indexing the blocks in this way is used to inform the other participants in the gossip receivers list about the state information of the peer. Third column entry of the gossip receiver's data in the figure below, *Gossip Order*, states the index of the block (download order of the block) the peer is informed most recently. The details for usage of the gossip order entry is given in the section 3.4.

Maximum size of the gossip receiver and gossip sender lists, *maximum connection size*, is a protocol parameter. Initially, peers in the gossip receivers list are determined with the help of the list returned by the index server after registration process. Details of the management of these lists will be given in section 3.5.

Peers may be categorized as *leechers* and *seeds* in accordance with the role they play in dissemination. A peer with a complete copy of the file is said to be a seed. Peers striving to obtain a full copy of the file are leechers. Although we call the original publisher of the file as the primary seed, there is no functional difference between the primary seed and any other seed. On the other hand, different from an ordinary seed, primary seed is expected to participate in dissemination continuously to ensure to completeness of downloads. In fact, the presence of seeds is a key feature, since it greatly enhances the ability to scale to large client populations and reduces the load imposed on the primary seed. However, a peer may leave the system at any time without completing the download or after completion of it. Figure 3.3 shows these states.

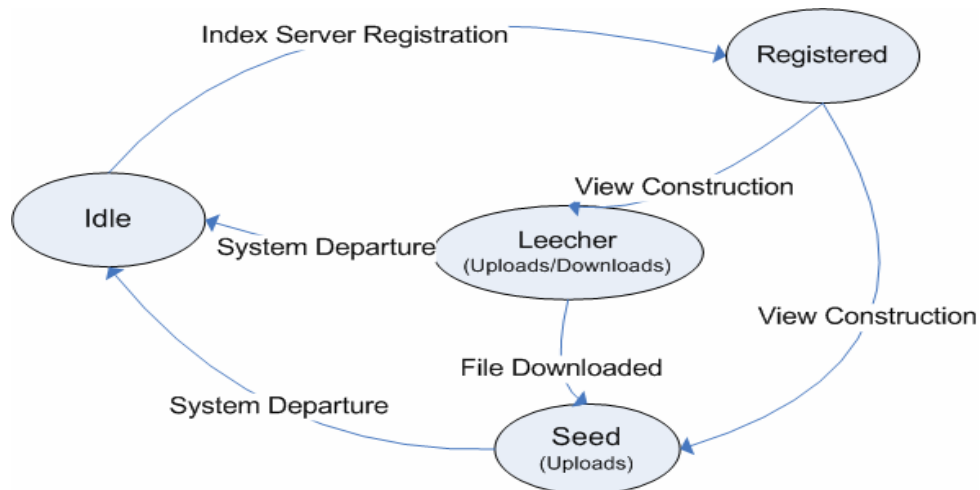


Figure 3.3 State diagram for peer

	Address	Peer State	Time
Id 1	Address 1	{b ₁ , b ₃ , b ₅ }	Time 1
Id 2	Address 2	{b ₂ , b ₃ , b ₆ }	Time 2
...	...	...	...
...	...	...	...
Id m	Address m	{b ₁ , b ₃ }	Time m

a) Data stored for gossip senders

Agent Id	Address	Gos. Order	Time
Id 1	Address 1	3	
Id 2	Address 2	1	Time 2
...	...	...	...
...	...	...	
Id n		2	Time m

b) Data stored for gossip receivers

Figure 3.4 Stored data for the peers in the gossip receivers list and gossip senders list

Peers are the building blocks of the P2P file sharing network. A peer manages data about the downloaded content, connections with other participants, and deal with different operations during dissemination. Accordingly, a peer contains the following components:

- **Queuing Module:** Includes a message buffer to store incoming messages.
- **Processing Module:** Includes functions and procedures for operations such as processing the incoming messages and organizing the communication among other modules and bases.
- **Connection Module:** Constructs views, i.e. gossip sender and gossip receiver lists, and manages view of the peer.
- **Transmission Module:** Gives decisions about the blocks that will be uploaded or downloaded, accepts or refuses the download requests of the peers.
- **Connection-Base:** Manages the data of the active connections with other peers as given in Figure 3.4.

- **Content-Base:** Contains basic data about the downloaded content: size of the file, number of the blocks the file divided into, the blocks that has been downloaded and so on.

Figure 3.5 shows the components of peer agent and their functional relations.

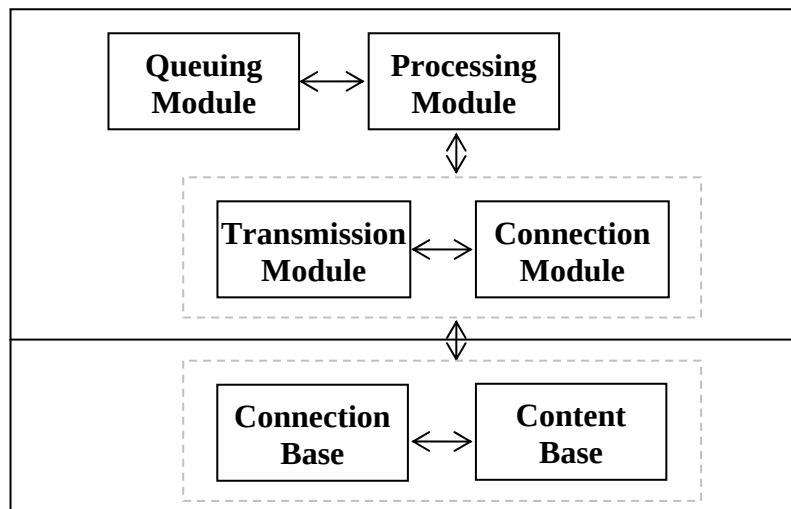


Figure 3.5. Functional model of a peer agent

3.3 Structure of the file

SeCond is a P2P protocol whose goal is to enable fast and efficient distribution of large files. The basic idea is to divide the file into equal-sized blocks and have nodes download the blocks from multiple peers concurrently. Small block size increases TCP connection overheads. On the other hand, larger blocks reduce parallelism and utilization of the link capacities since peers start uploading only if they obtain a complete copy of a block. The SeCond protocol breaks down files into a number of small blocks, typically a quarter of a megabyte (256 KB) in size.

3.4 State propagation

State of a peer stands for the blocks that have been downloaded by the peer up to that time. A peer propagates its state to the peers in the gossip receiver list. Propagation of the state information is performed by use of gossip messages. At each gossip round, a peer selects f distinct peers randomly from the gossip receivers list. The number of peers f selected as gossip targets in a gossip round is the *fan-out* parameter of the protocol. The rate of gossip messages are determined by the protocol parameter, *gossip interval*. A gossip message carries the ids of the downloaded blocks that the receiver is not aware of. Upon receiving a gossip message, a receiver peer looks for the blocks that it does not have but the sender has. If there are such blocks, the receiver peer requests missing ones from the sender. It should be noted that sometimes requesting blocks may not initiate block downloads. Details of the initiation of actual block transmissions and block selection to download are given in section 3.6. Another important point is that gossiping is not reciprocal. Namely, holding some peer in the partial view and sending gossip messages to that peer does not necessitate that destination peer also sends gossip messages to counterpart.

SeCond employs the smart peer policy which enables adjustment of the fan-out parameter f dynamically based on the current uplink utilization. The peers increase their fan-out parameter by 1 with a probability $P(1-Uplink\ Utilization)$ at each gossip round. Otherwise, the fan-out is decreased by 1.

After selection of the gossip destinations uniformly from the gossip receivers list, a gossip message is constructed per target. Downloaded blocks are indexed according to their download times. During construction process of the gossip message, the block about which destination peer is informed most recently is determined by looking at the gossip order entry held for each peer in the gossip receiver list. Following this, ids of the blocks downloaded after the determined block are included in the gossip message. Finally, the gossip order entry for the destination peer is updated with the most recent

downloaded block index. The example in Figure 3.6 illustrates the gossip message construction process for a destination peer.

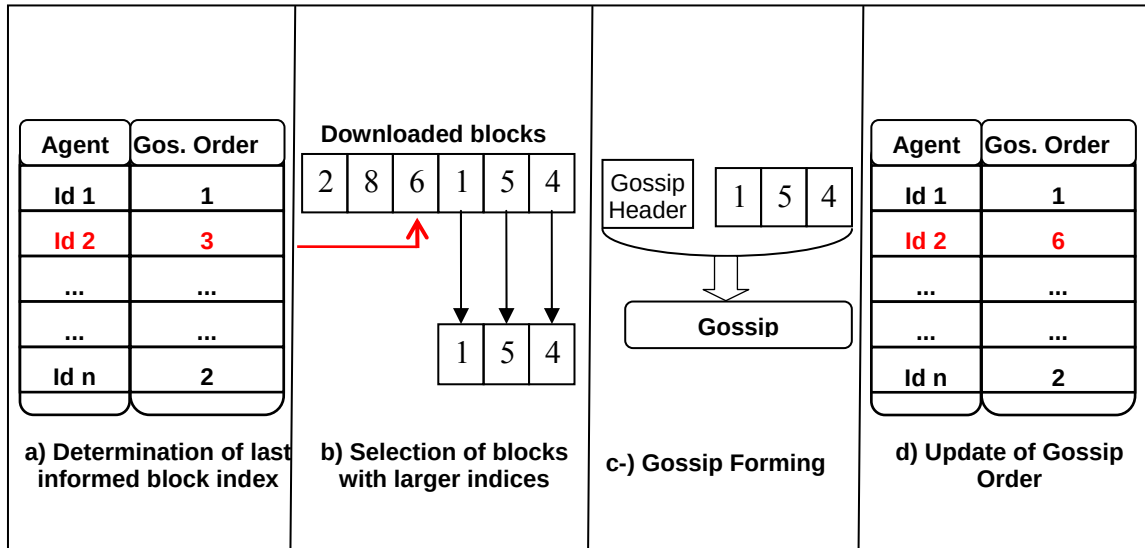


Figure 3.6. Gossip Message Construction for the peer with id 2: a) Index of the block that the peer with id 2 has been informed most recently is determined b) Blocks downloaded after block 6 are selected to be added to gossip message c) Gossip message is formed by adding header to gossip data d) Gossip order, index of the block the destination peer is informed most recently, is updated.

3.5 View Construction and Update

In order to increase the performance of content distribution, cooperation among peers should be maximized. Hence, peers should be able to upload and download blocks. Gossip receiver and sender lists are two views managed by a peer. However, the peers in these lists may have left the system, or almost all of the block transmissions among the peers might have been done. This leads to waste of link capacities of the peers. In such a case, a peer is forced to download blocks from the primary seed or stays idle. To avoid this situation, peers update their views during dissemination.

Initially, the gossip receiver list is constructed with the help of the list returned by the index server. Following the registration process, index server returns a list which

consists of active peers (both seeds and leechers) randomly selected among the registered ones. Typically, the size of the returned list is greater than *default connection size* that the peer tries to maintain. After registration, newly joined peer sends a request message to peers in the returned list indicating that it wants to add the counter part to the gossip receiver list. If there is free space in the gossip sender list of the receiver, the sender is directly added to that list. However, if the capacity of the list is full, then the receiver looks for a passive peer. A peer in the gossip sender list from which no block transmission has been initiated in a given time interval, called *passive peer interval*, is said to be a *passive peer*. If there is a passive peer, it is replaced with the new one, and the gossip sender list is updated. Otherwise, a rejection message is sent. On the other hand, regardless of the response returned to the requester, the receiver checks whether the requester is in its gossip receiver list. If not, it looks for free space or a passive peer in the gossip receiver list to place the requester. In this case, passive peer stands for a peer that has not initiated any block download for the passive peer interval. The decision processes for updates of gossip sender and gossip receiver lists are illustrated in Figure 3.7 a and b respectively.

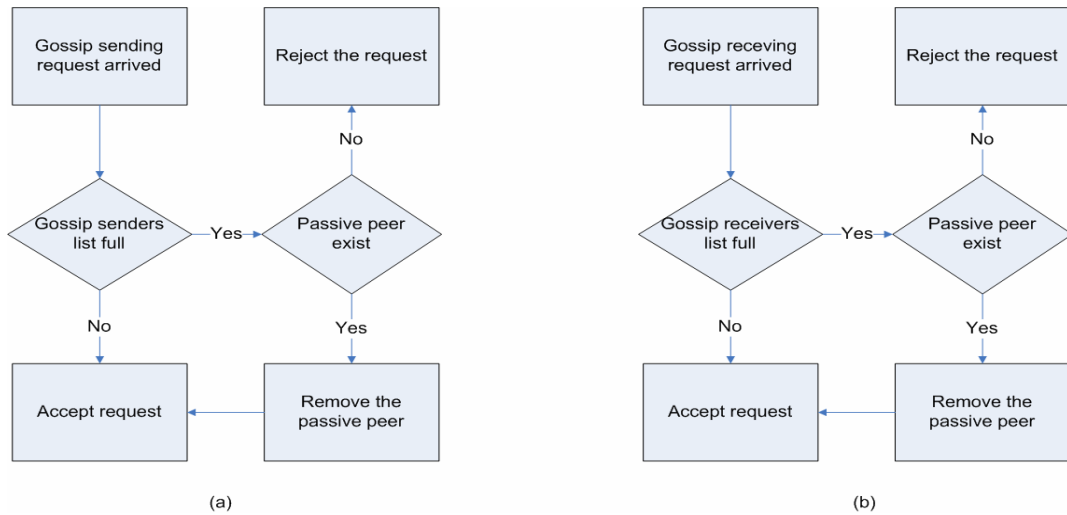


Figure 3.7 Illustration of view update decision process

If the number of peers in gossip sender or receiver lists falls below a lower bound, *minimum connection size*, during the dissemination, index server is asked to return

addresses of new peers. As it is stated in section 3.2, the timer entry indicates the last incoming/outgoing block transmission times of the peers in the gossip receiver/sender lists. These entries are used for determining the passive peers and are updated with the initiation of new block transmissions. Both the passive peer interval and minimum connection size are protocol parameters.

3.6 Download/Upload Decisions

In the beginning, primary seed is the only one containing all blocks of the file. As the dissemination continues, peers owning blocks of the file increase in the system. When a peer receives a gossip message from another peer, in addition to updating the state information of the message sender, the receiver checks whether there is any block that it does not have but the sender has. If there are such missing blocks, it sends a request message containing n missing blocks ids. These are least duplicated blocks in the system. Recall that peers hold the state information of the peers in the gossip sender list. Hence, the decision for the density of the blocks is given locally by the help of this data. The source prefers to forward the least uploaded block among the requested ones. The objective is to consistently equalize the density of each block in the system since the main goal of our protocol is to make copies of the blocks available in the system as fast as possible. This makes it unlikely that the system will get bogged down because of “rare” blocks that are difficult to find.

Requesting some blocks from a source does not guarantee the initiation of block download. Since the concurrent upload capacity of the peers is limited, the request may be refused. It is bounded by a protocol parameter, *parallel upload limit*. Otherwise, download rates of transmissions for the blocks obtained from the sources may decrease below an acceptable lower bound. As a result, replication of the blocks in the system may take longer. On the other hand, letting small number of peers to download simultaneously may lead to waste of uplink capacities. To avoid this problem, although an initial value is assigned to maximum parallel upload limit for each peer, this value can be adjustable by each peer locally according to the utilization of uplink capacities.

If the maximum parallel downloading limit is reached and another upload request is received, the source checks the utilization of its uplink capacity. If this utilization is greater than a bound between 0 and 1, called *utilization threshold*, new incoming request is rejected. Otherwise, it lets the requester download without taking the maximum parallel upload capacity into consideration. Moreover, we deploy a queue not to refuse requests and to increase utilization of the uplink capacities. Peers put the received requests into their upload queues if they cannot accept another parallel upload. Though, originally we operate the first in first out queue (FIFO) scheme on request queues, different schemes may be implemented, that is, priorities may be given to peers that upload more to the owner of the queue. This decision process is illustrated in Figure 3.8. *Maximum queue size* is also limited and it is a protocol parameter.

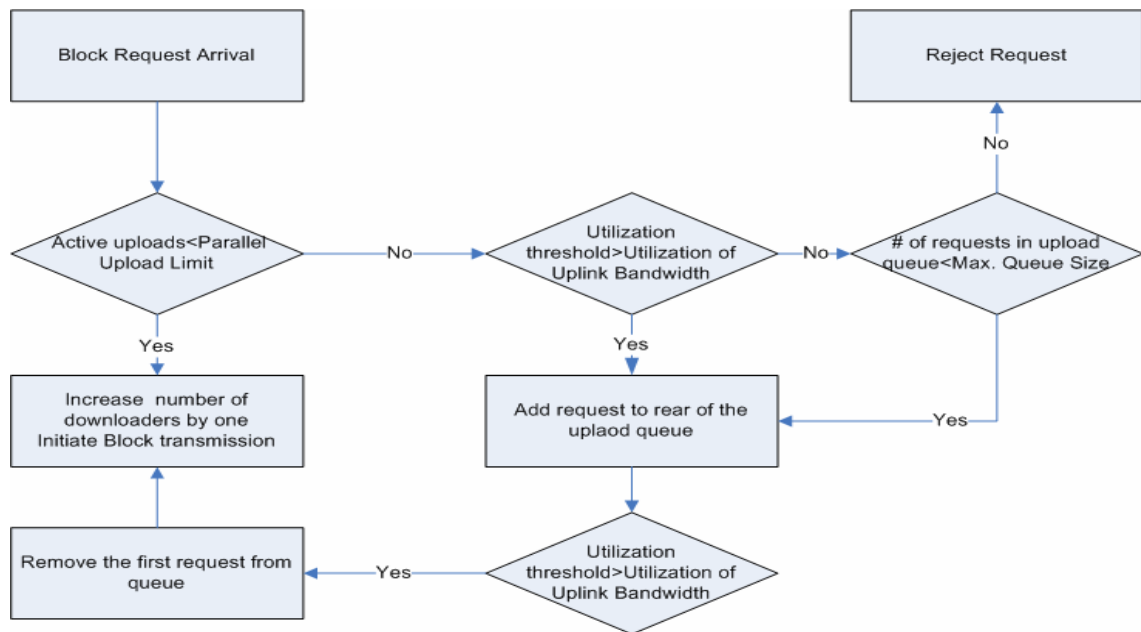


Figure 3.8 Illustration of upload decision process

3.7 Protocol Parameters, Algorithms, Events and Message Structures

This section summarizes the main data structures and special variables, and protocol parameters of Second.

3.7.1 Properties and Algorithms of Index Server Agent

Table 3.1, 3.2, and 3.3 shows the events, special variables, and protocol parameters handled by the index server respectively. Data structures, functions, and message formats are given in Table 3.4, 3.5, and 3.6. Finally algorithms of the functions and events are illustrated in Table 3.7.

Table 3.1 Events handled by the index server agent

Event	Description
<i>Registration Request Reception</i>	Reception of a registration request
<i>Peer Request Reception</i>	Reception of a request for addresses of new peers
<i>State Report Reception</i>	Reception of a message stating the sender is alive
<i>Dead Peer Detection Round</i>	Detection of dead peers registered to system

Table 3.2 Special variables of index server agent

Special Variable	Description
<i>File Descriptor</i>	Unique descriptor of the file
<i>System Size</i>	Number of the peers registered to system at any moment
<i>Requested Peer Size</i>	Number of the requested peers
<i>Report Time</i>	Time of the last state report message received from the peer

Table 3.3 Protocol parameters of index server agent

Protocol Parameter	Description
<i>List Size</i>	Size of the list returned to the peers after registration
<i>State Report Interval</i>	Time interval at which each peer reports its existence

Table 3.4 Data structures of index server agent

Data Structure	Description
<i>Registration Message</i>	Received registration message
<i>Peer Request Message</i>	Received peer request message
<i>State Report Message</i>	Received state report message
<i>Peer Base</i>	The base where the tracks of the registered peers are kept
<i>Content Base</i>	The base where the data about the shared file is kept

Table 3.5 Functions called by index server agent

Function	Description
<i>View Selection</i>	Selection of the peers for returning to requester

Table 3.6 Message formats

Registration Request Message		
Message Type 1 octet	Size of the message 2 octets	Request Data (i.e. descriptor of the shared file) Max 2048 octets
Peer Request Message		
Message Type 1 octet	Size of the message 2 octets	Request Data (i.e. number of the requested peers) Max 2048 octets
State Report Message		
Message Type 1 octet	Size of the message 2 octets	Statistical data (i.e. percentage of the completed download) Max 2048 octets

Table 3.7 Pseudo-code for events and functions

Registration Request Reception
If (The requester peer is not registered currently and requested file is same as the shared file) then; - View Selection (List Size) - Send selected list to requester peer - Add requester to Peer-Base endif
Peer Request Reception
If (The requester peer is registered) then; - Get number of the requested peers - View Selection (Number of requested peers) - Send selected list to requester peer endif
State Report Reception
If (The requester peer is registered) then; - Update the Report Time of the peer endif
Dead Peer Detection Round
for all registered peers in the peer base If (Current time – Report Time of the peer > State Report Interval) then; - Remove the peer from the peer base endif endfor
View Selection (Requested Peer Size)
If (Requested Peer Size > List Size) then; - Send an error message endif If (Requested Peer Size < System Size) then; - Select requested number of peers uniformly from the peer base - Return selected peers endif else - Return all the peers in the peer base endif

3.7.2 Properties and Algorithms of Peer Agent

Events, functions, and data structures handled by a peer agent are given in Table 3.8, 3.9, and 3.10 respectively. Table 3.11, 3.12, 3.13 shows the special variables, protocol parameters, and algorithms of the events and functions.

Message formats are illustrated in Table 3.14. Gossip message, block request message, upload acceptance message, queue acceptance message, gossip sending request message, and gossip receiving message are message types defined for communication among the peers. Indicator value given in gossip sending request message is used to determine whether this message is forwarded upon reception of a gossip receiving request message or not. Similarly, same entity is in gossip receiving request message to determine whether the message is sent upon reception of a gossip sending request message.

Table 3.8 Events handled by the peer agent

Event	Description
<i>Gossip Round</i>	Gossip dissemination procedure operate periodically
<i>Gossip Message Reception</i>	Reception of a gossip message
<i>Block Request Reception</i>	Reception of a block upload request
<i>Gossip Sending Request</i>	Reception of request to be added to gossip senders list
<i>Gossip Receiving Request</i>	Reception of request to be added to gossip receivers list
<i>Completion of Block Download</i>	Completion of a block transmission

Table 3.9 Functions called by peer agent

Function	Description
<i>Gossip Message Construction</i>	Forming process of the gossip message
<i>Block Request Construction</i>	Forming of a block request
<i>Passive Sender Detection</i>	Detection of a passive peer among the gossip senders
<i>Passive Receiver Detection</i>	Detection of a passive peer among the gossip receivers
<i>Block Selection to Upload</i>	Selection of the block among the requested ones to upload

Table 3.10 Data structures of peer agent

Data Structure	Description
<i>Gossip Message</i>	Gossip message to be sent
<i>Block Request Message</i>	Request message including ids of requested blocks
<i>Gossip Sender List</i>	The list containing the peers sending gossip messages
<i>Gossip Receiver List</i>	The list containing the peers to which gossip messages are sent
<i>Last Block Download Times</i>	Times of the last block downloads from peers in gossip senders
<i>Last Block Upload Times</i>	Times of the last block uploads to the peers in gossip receivers

Table 3.11 Special variables of peer agent

Special Variable	Description
<i>Upload Count</i>	Number of the active uploads
<i>Uplink Utilization</i>	Rate of the aggregate uplink traffic to uplink capacity
<i>Queue Size</i>	Number of the requests in the upload queue
<i>Sender List Size</i>	Number of the peers in the gossip sender list
<i>Receiver List Size</i>	Number of the peers in the gossip receiver list
<i>Interested Block Count</i>	Number of the blocks peer doesn't have but counter part has
<i>Last Block Download Time</i>	Time of the last block download from a peer in gossip senders
<i>Last Block Upload Time</i>	Time of the last block upload to a peer in gossip receivers list
<i>Block Density</i>	Number of the peers in the gossip sender list owning the block

Table 3.12 Protocol parameters of peer agent

Protocol Parameter	Description
<i>Fan-Out</i>	Numbers of peers gossip messages are sent to at each gossip
<i>Gossip Interval</i>	Time interval at which each peer gossips
<i>Block Index</i>	Download order of the block
<i>Gossip Order</i>	Index of the block the peer is informed about most recently
<i>Parallel Upload Limit</i>	Maximum number of parallel uploads
<i>Utilization Threshold</i>	Threshold value for deciding utilization of uplinks
<i>Gossip Order</i>	Index of the block the peer is informed about most recently
<i>Max. Connection Size</i>	Maximum size of the gossip receiver sender lists
<i>Passive Peer Interval</i>	Time period used to determine passive peers
<i>Max. Queue Size</i>	Maximum size of the queue where incoming requests are stored
<i>Request Size</i>	Number of the block ids attached to a block request message
<i>Min-Fan-Out</i>	Minimum value that the fan-out parameter can be set

Table 3.13 Pseudo-code for functions and events

<i>Gossip Round</i> -Determine uplink utilization - Select a random real number between 0 and 1 - If(selected number > uplink utilization) -Increase fan-out by one - else -Set fan-out parameter to max {min-fan-out, fan-out-1} - endif -f = min {fan-out, receiver list size} -Select f different peers uniformly from the gossip receivers list. - for each peer selected as gossip destination -Gossip Message Construction (Destination Peer) -Send gossip message to destination - endfor
<i>Gossip Message Reception</i> - If (Gossip Sender is not in Gossip Receiver List) then; -Throw an error message - endif -Read the newly downloaded block ids from the gossip message and update state information of the gossip sender according to new information. -Update the density information of the blocks -If (there is any block in which the peer is interested) -Block Request Construction (Gossip Sender) - Send block request message to gossip sender - If (Request is accepted) -Initiate the block transmission from the source - else - If (Request is added to upload queue of the gossip sender) -Mark the block as currently being downloaded not to request the same block from any other source later. - endif - endif - endif
<i>Block Request Reception (Request Message)</i> -If (Request sender is not in gossip receiver list) -Reject request - endif - If (Upload count < Max. parallel upload limit) -Block Selection to Upload (Request Message) -Send an upload acceptance message containing the id of the selected block to be uploaded - else - If (Queue size < Max. Queue Size) - Block Selection to Upload (Request Message) -Send a queue acceptance message containing the id of the selected block - else - Send an upload rejection message - endif - endif

Gossip Sending Request

```

If(Sender List Size < Max. Connection Size)
  -Send an acceptance message
  -Add the sender to gossip message senders
else
  -Passive Gossip Sender Selection
  -If (passive peer is found)
    -Replace it with the new peer
    -Send an acceptance message
  else
    -Send a rejection message
  endif
endif
If(Request is not forwarded after reception of a gossip sending request sent by this peer)
  If (Sender is not in gossip receiver list)
    -If(Receiver List Size < Max. Connection Size or there is a passive peer)
      - Send a gossip sending request
      - If request is accepted, add peer to the gossip receiver list
    endif
  endif
endif
endif

```

Gossip Receiving Request

```

If(Receiver List Size < Max. Connection Size)
  -Send an acceptance message
  -Add the sender to gossip message receivers list
else
  -Passive Gossip Receiver Selection
  -If (passive peer is found)
    -Replace it with the new peer
    -Send an acceptance message
  else
    -Send a rejection message
  endif
endif
If(Request is not forwarded after reception of a gossip receiving request sent by this peer)
  If (Sender is not in gossip sender list)
    -If (Sender List Size < Max. Connection Size or there is a passive peer)
      - Send a gossip receiving request
      - If request is accepted, add peer to the gossip sender list
    endif
  endif
endif
endif

```

Passive Gossip Sender Selection

```

for each peer in the gossip sender list
  if(Current time – last download time entry of the peer > passive peer interval)
    Return index of the peer
  endif
Return -1 to indicate that passive peer could not be found.

```

Passive Gossip Receiver Selection

```

for each peer in the gossip receiver list
  if(Current time – last upload time entry of the peer > passive peer interval)
    Return index of the peer
  endif
Return -1 to indicate that passive peer could not be found.

```

Gossip Message Construction (Destination Peer)

```

If (Destination Peer is not in Gossip Receiver List) then;
  -Throw an error message
endif
-Get index of the block the destination is informed most recently by looking gossip order variable for the peer.
- Add ids of the blocks having larger indices, i.e. blocks downloaded after the last gossip message sent to destination peer.
-Set gossip order variable of the destination peer to largest block index, i.e. index of the lastly downloaded block

```

Block Request Construction (Source Peer)

```

If (Source Peer is not in Gossip Sender List) then;
  -Throw an error message
endif
- n = min {number of interested blocks, request size}
- Select n blocks starting from the densest one among the interested blocks, i.e. among the blocks the source has got but the peer hasn't got
- Piggyback the selected block ids to block request message

```

Block Selection to Upload (Request Message)

```

-Read requested block ids and their density information
for each requested block
  -Calculate the sum of the given density data of the block obtained from the request message and local density data of the block
endfor
-Select the block with the smallest sum of the local and given density data

```

Completion of Block Download
-Update state information If(Sender has any other block in which the peer is interested) - Block Request Construction (Gossip Sender) - Send block request message to gossip sender If (Request is accepted) -Initiate the block transmission from the source else If (Request is added to upload queue of the gossip sender) -Mark the block as currently being downloaded not to request the same block from any other source later. endif endif endif endif

Table 3.14 Message formats

Gossip Message			
Message Type 1 octet	Size of the message 2 octets	Gossip Data (i.e. ids of the newly downloaded blocks) Max 2048 octets	
Block Request Message			
Message Type 1 octet	Size of the message 2 octets	Request Data (i.e. ids of n requested blocks and their densities)	
Block Upload Acceptance Message			
Message Type 1 octet	Size of the message 2 octets	Acceptation Data (i.e. id of the block to be uploaded) Max 2048 octets	
Queue Acceptation Message			
Message Type 1 octet	Size of the message 2 octets	Acceptation Data (i.e. id of the block added to upload queue) Max 2048 octets	
Gossip Sending Request Message			
Message Type 1 octet	Size of the message 2 octets	Indicator Value 1 octet	Optional Data (i.e. percentage of download) Max 2048 octets
Gossip Receiving Request Message			
Message Type 1 octet	Size of the message 2 octets	Indicator Value 1 octet	Optional Data (i.e. percentage of download) Max 2048 octets

Chapter 4

SIMULATION MODEL

In order to present key characteristics and behavior of our protocol SeCond, a simulator has been designed and implemented in this thesis study. We have tried to observe behavior of the system for different settings of the protocol parameters. Similarly, effect of different arrival patterns on the content distribution has been studied. In addition to these, a simulator modeling the BitTorrent protocol has been developed. To reflect BitTorrent's behavior accurately in the simulation environment, we have integrated all the protocol properties specified in BitTorrent official web site into simulation code.

Figure 4.1 shows the fundamental steps followed during the development process of the simulators. Initially, we determined appropriate design decisions about the simulators, i.e. we modeled them as single thread discrete event simulators. In addition, since the official BitTorrent's protocol specification [28] outlines the protocol in somewhat general terms, and lacks behavioral detail in some areas, an additional research has been made to clarify missing real implementation details. After implementation, we utilize unit testing procedure to verify that the implemented code operates as it is intended. During the experimental phase, the models have been executed in order to generate results. Finally, we analyze the output of these runs. Details of the simulation models are described next.

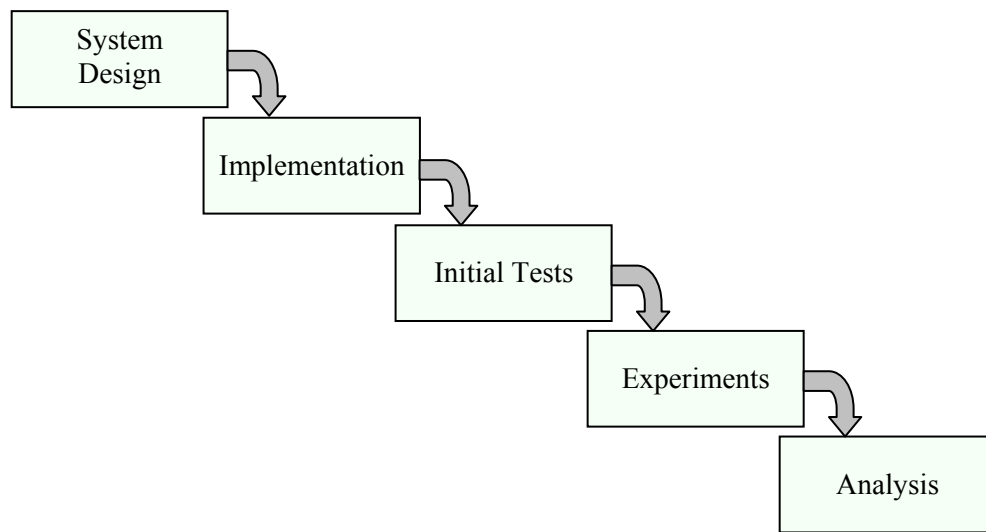


Figure 4.1. Overview of the simulation process

4.1 Development Environment and Simulation Model

Both the SeCond and BitTorrent simulators are implemented in Java programming language. JDK 1.5.0.6 (Java platform, Standard Edition 1.5.0.6) is used as development platform. To verify the correctness of the implemented code, unit tests of the implemented codes has been done. These tests were handled by the help of the JUnit [39] testing framework. We run our simulations on a P4 2.4 GHz, 1 GB RAM machine.

We employ discrete event simulation scheme in our simulators. Discrete event simulation is one way of building up models to observe the time based behavior of a system. Inside the any simulation software there is a number of important components, namely entities and logical relationships.

Entities are the tangible elements found in the real world. *Logical relationships* link different entities together. The logical relationships are the key part of the simulation model; they define the overall behavior of the model.

Another key part of any simulation system is the *simulation executive*. The executive is responsible for controlling the progress of the time. A central clock is responsible for keeping track of the time. The executive will control the logical

relationships between the entities and advance the clock to the new time. The simulation executive is central to providing the dynamic, time based behavior of the model. The executive manages the passage of time and ‘steps’ the model into the future, executing the relevant logical relationships along the way. The nature of the jumping between significant points in time allows models to be evaluated more quickly.

Each event handled by the simulation executive has two key data items one of which is the occurrence time. Second one is the reference to the operations defined to be handled during the execution of the event. The events are stored in order according to their occurrence time. The executive removes the first event from the list and executes the procedures for the event. Any new events occurring as a result are inserted on the list at the appropriate point. Similarly, execution of an event may lead to changes in occurrence times of the other events in the list.

We utilize object-oriented techniques for modeling real entities of the protocols in the simulation. Random generator class of Java util package is used to provide randomness.

4.2 Design and Implementation Details

In this section, implementation and design details of both Second and BitTorrent protocols are given.

4.2.1 SeCond

The P2P network of SeCond consists of two main components one of which is peers and the other one is index server. Peers are also categorized as *leechers* and seed *peers*.

For every participant of the SeCond’s P2P network except the index server, two main agents have been implemented. The first one, gossiping agent is responsible for processing the incoming gossip messages, constructing outgoing gossip messages, and managing gossip sender and gossip receiver lists. Actual block transmissions are handled by the second agent, transmission agent. All the decisions given for block

downloads and uploads, i.e. the block ids requested from a gossip sender upon reception of a gossip message, are given by this agent.

On the other hand, there is only one agent working on the index server. It is responsible for registering newly joined peers, helping peers to find each other by returning randomly selected list of active peers, and managing the list of active peers in the system.

All the protocol properties defined for peers and index server are implemented in the simulator. Details of the SeCond protocol are introduced in chapter 3.

4.2.2 BitTorrent

Details of the BitTorrent protocol definitions and their implementations are given below.

a) Tracker Algorithm

Tracker is the server which is responsible for keeping track of the registered peers. Our tracker algorithm, similar to official deployment, returns a list of randomly selected peers among the currently registered ones. The size of the returned list is set to 50. But, peers attempt to establish connections to about 40 of them. Peers whose active neighbor number falls below the lower bound 20, specified by the BitTorrent protocol, revisit the tracker to obtain additional peers. We set a revisiting time interval for missing neighbors. Although the maximum number of neighbors is not directly stated in the official BitTorrent specification, we limit this number by 55 which is used in some of the real protocol implementations.

b) Choking/Unchoking Policy

BitTorrent employs a *tit-for-tat* (TFT) policy to avoid free riding. A peer uploads to the peers providing the best download rates. Selection process of the neighbors to upload is called as choking/unchoking. This policy is employed once every 10 seconds.

At every choking/unchoking round, a peer determines 4 interested peers which it has the best download rates from. These are called as unchoked neighbors. Other neighbors are temporally refused to download from the peer, namely they are choked. If a peer which has a better upload rate becomes interested, the unchoked peer having the worst upload rate gets choked. A peer with a complete copy of the file, a seed, uses its upload rate rather than its download rate to decide whom to unchoke. Since choking is not reciprocal, in general the set of neighbors that a node is uploading to may not exactly coincide with the set of neighbors it is downloading from.

In order to give a chance to peers that may offer better download rates and prevent the peers from getting bootstrapped which have got nothing to upload, an *optimistic unchoking* is deployed once every 30 seconds. For any time there is only one *optimistic peer*, a peer which is unchoked regardless of its upload rate. If the optimistic peer is interested in the available blocks, it is counted as one of the four allowed downloaders. However, optimistic peer is not selected uniformly among the neighbors. Peers that have not obtained a block of the file are three times as likely to start as the current optimistic unchoke as anywhere else in the rotation.

c) **Block Selection Policy**

In general, a peer has a choice of several blocks that it could download. In order to make the number of replicas of each block as evenly distributed as possible, BitTorrent employs a *local rarest first* (LRF) policy in selecting the block to download. Each peer tries to download the block that is least replicated among its neighbors. *Random selection policy* is made as an exception to the local rarest first policy in the case of a new peer that has not downloaded any blocks yet. Since it is important to get a complete block as quickly as possible for the newly joined peer, blocks to download are selected randomly until the first complete piece is obtained.

d) Endgame Model

To policy which enables peers to download different sub-blocks of the last chunk from different neighbors simultaneously is called *end-game-mode*. The intuition for this policy is preventing delays due to last block transmission.

To be able to reflect BitTorrent's behavior accurately in a simulation environment, we have integrated all the protocol properties specified in BitTorrent official web site into simulation code. All the BitTorrent policies stated above except the endgame policy are implemented. We believe that endgame policy does not have a significant effect on simulation results to appreciate BitTorrent's performance as it refers only to the last chunk.

4.3 Shared File

In our simulation model, the file is broken into blocks of size 256KB and the size of the shared file is 200MB which may be size of a game patch. Whenever a peer completes downloading of a block successfully, it starts sharing that block. At the beginning of the simulation, there is only one seed which is the original file publisher. Unless otherwise is stated, peers leave the system as soon as they complete the file download.

4.4 Peer Arrival/Departure Patterns and System Size

Dynamic arrivals and departures of the peers have also been taken into account. Except for the scalability analysis, the number of peers requesting the file is set to 1000 in the simulations. For scalability analysis, we performed simulations for various population sizes up to 8000.

Five different arrival patterns flash-crowd, constant, increasing, decreasing, and triangular arrival are defined. Flash crowd arrival scheme refers to scenario in which all the peers request the file simultaneously. Constant arrival corresponds to the Poisson process with constant rate. Increasing and decreasing arrivals stand for non-stationary

Poisson processes with increasing and decreasing rates respectively. Triangular arrival pattern assumes a Poisson process in which the arrival rate increases up to some point and then starts to decrease. Unless otherwise is stated, we run simulations for flash-crowd arrival process.

We define two different departure patterns. *Immediate departure* corresponds to departure of peers immediately after completion of the download. The second one, *random departure*, assumes that the time period in which the peers remain connected in the system after completion of download is exponentially distributed with mean μ . By default, immediate departure pattern is employed.

4.5 Network and Link Properties

The main issue that we have made some important simplifications is modeling of underlying network topology. One of the details we ignore in the simulation is the propagation delay of packets. The incentive to this simplification is that the file download duration is mainly determined by the transmission times of the blocks. We believe that this simplification does not have a significant impact on our results. Another assumption is that the bottleneck during packet transmission arises either on uploader or on downloader side of the packet. Namely, download time of the block transmission is determined by the upload capacity of the block sender and download capacity of the block receiver. Lastly, we do not model the dynamics of TCP connections. Instead, we use a model which assumes that the flows traversing a link share the link bandwidth equally. Similar simplifications are introduced in [33].

An example describing the restrictions and calculations is illustrated in Figure 4.2. The list indicated as $B[\text{block id list}]$ represents blocks downloaded by the corresponding peer up to that time, P represents the number of peers that can be served additionally, and $[\text{upload capacity}, \text{download capacity}]$ represents the download and upload bandwidth of the peer. $T\#:[a,c]$ represents the transmission of block a with transmission rate c . Unit of bandwidth capacities is Kbps. In Figure 4.2, shared file consists of 5 blocks. At round 1, peer 2 initiates transmission of block 1 from peer 1. Bandwidth

initially reserved for this transmission is equal to the minimum of the upload capacity of peer 1 and, the download capacity of peer 2, namely 128 Kbps. Assuming that no block transmission is completed, peer 1 can serve at most 1 other peer in the remaining rounds. Similarly, a transmission of rate 128 Kbps from the server is started in round 1. At round 2, peer 1 starts to download block 2 from peer 3 with rate 128 Kbps. However, peer 4 also initiates a transmission from peer 1 at round 2. Since upload capacity of a peer is shared out equally among downloaders, both of the rates reserved for transmissions 1 and 3 are set to 64 Kbps. At round 3, peer 2 initiates a block download from peer 4. Reserved bandwidth for this transmission is equal to the minimum of the download capacity of peer 2 and the upload capacity of peer 4, which is 192 Kbps. Remember that peer 2 uses 64 Kbps of its download capacity for transmission 1. Without completion of any block transmission, no one can initiate a transmission after round 3.

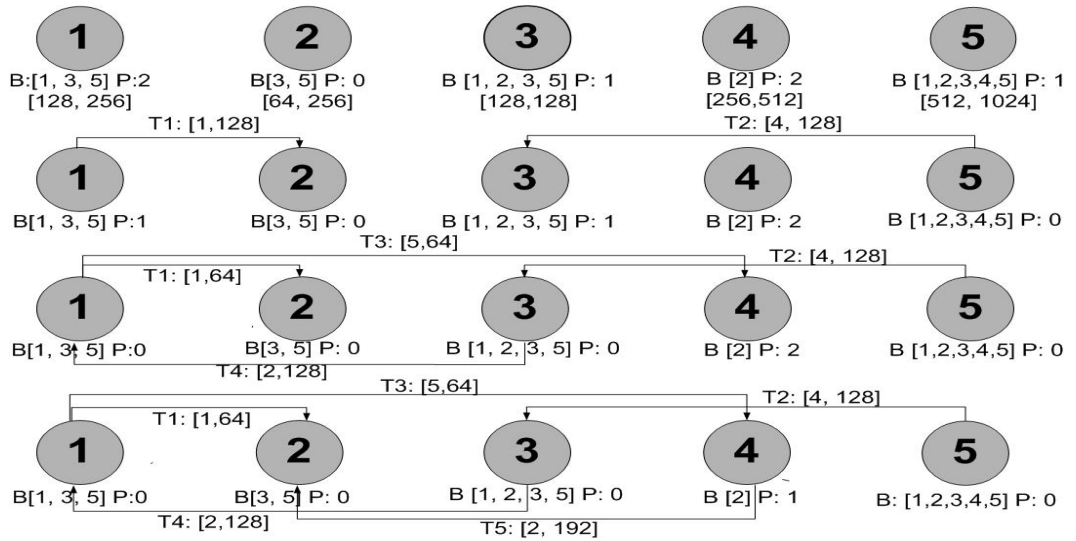


Figure 4.2. Overview of the simulation process

In the simulations we have tried to make a performance evaluation on both homogenous and heterogeneous systems. Homogeneous system stands for the case in which all the peers have equal uplink and downlink capacities, except the primary seed.

Uplink capacity of the primary seed is set to 6000 Kbps. All the other peers have 1500 Kbps download, 500 Kbps upload capacities. Since the size of the shared file is 200MB, the lower bound for the file download time of a peer is 1093 seconds for the homogeneous system. On the other hand, peers in the heterogeneous system may have different uplink and downlink capacities. Distribution of the bandwidths is given in Table 4.1 which is obtained from a measurement study of the Gnutella network [32]. For the heterogeneous system, the lower bound for the file download time would be 163 seconds.

Table 4.1 Bandwidth distribution for heterogeneous system

Downlink (kbps)	Uplink (kbps)	Fraction
784	128	0.2
1500	384	0.4
3000	1000	0.25
10000	5000	0.15

4.6 Terminology of Simulation Evaluation

Average File Download Time: The file download time of a peer is the time between the initiation and the completion of the download. Average of all file download times of the peers in the system is evaluated using this metric.

Load on the Primary Seed: A peer obtains blocks of the file either from the primary seed or from another peer. This metric denotes the size of the data uploaded by the primary seed.

Uplink/Downlink Utilization: The utilization is the ratio of the aggregate traffic flow on all uplinks/downlinks to the aggregate capacity of all uplinks/downlinks in the system.

Fairness Ratio: Ratio of the uploaded data size to downloaded data size by a peer.

4.7 Default Simulation Settings

Default protocol parameters used through simulations are given in Table 4.2, 4.3 and 4.4 below.

Table 4.2 Common Simulation Settings

Setting	Description	Value
<i>File Size</i>	Size of the shared file	200MB
<i>System Size</i>	Number of peers requesting the file	1000
<i>Constant Seed Count</i>	Number of seeds that continuously serve during dissemination	1

Table 4.3 SeCond Parameter Settings

Parameter	Description	Value
<i>Fan-Out</i>	Number of peers gossip messages are sent to at each gossip	3
<i>Gossip Interval</i>	Time interval at which each peer gossips periodically	20Sec.
<i>Parallel Upload Limit</i>	Maximum number of parallel uploads	5
<i>Utilization Threshold</i>	Threshold value for deciding utilization of uplinks	0.9
<i>Max. Connection Size</i>	Maximum size of the gossip receiver and sender lists	50
<i>Passive Peer Interval</i>	Time period used to determine passive peers	300Sec.
<i>Max. Queue Size</i>	Maximum size of the queue where incoming requests are stored	3
<i>Request Size</i>	Number of the block ids attached to a block request message	3
<i>Min-Fan-Out</i>	Minimum value that the fan-out parameter can be set	3
<i>List Size</i>	List size returned by index server after registration	50

Table 4.4 BitTorrent Parameter Settings

Parameter	Description	Value
<i>Choking Interval</i>	Time interval at which choking policy is employed	10 Sec.
<i>Optimistic Choking Inter.</i>	Time interval at which optimistic peer is selected	30Sec.
<i>Parallel Upload Limit</i>	Maximum number of parallel uploads	5
<i>Max. Connection Size</i>	Maximum size of the gossip receiver and sender lists	55
<i>List Size</i>	List size returned by index server after registration	50

Chapter 5

SIMULATION RESULTS

In this chapter, we present our simulation results and analysis for the SeCond and BitTorrent system models. A comprehensive performance evaluation of SeCond and its comparison with the BitTorrent system model have been accomplished for a wide range of scenarios. Performance analysis results include the scalability analysis for different arrival/departure patterns, flash crowd scenario, overhead analysis, and fairness ratio. The major metrics we study include the average file download time, load on the primary seed, uplink/downlink utilization, and the fairness ratio. In the first section, the performance metrics are studied in terms of protocol parameters. Performance evaluation of SeCond and its comparison with BitTorrent are given in the second part.

5.1 Parameter Analysis

Parameter settings may affect the performance of the protocol significantly. Analysis of the protocol parameters is a crucial requirement for finding the appropriate settings providing the optimum performance results. It is also very useful for observing the behavior of the protocol for different scenarios and hence making possible improvements on the protocol.

In this section, analysis and discussion of the main protocol parameters such as gossip interval, parallel upload limit, view size, are studied through the simulator developed for the SeCond. During the simulations for parameter analysis, we assume a *flash-crowd* arrival, that is, all of the requesters join the system simultaneously. Similarly, *immediate-departure* is employed. Namely, peers leave the system as soon as the download is completed.

5.1.1 Gossip Interval

Gossip messages are used for both state propagation and initiating actual block transmissions. At each gossip round, each peer forwards gossip messages to f randomly selected targets. The number of the peers the gossip message is forwarded is the *fan-out* parameter of the protocol. In addition to fan-out, frequency of the gossip messages is determined by the *gossip interval*. In this section, we investigate the effect of gossip interval on the simulation results.

Firstly, we deactivated the smart peer policy to observe the effect of gossip interval more clearly. Recall that smart peer policy enables adjustment of the *fan-out* parameter dynamically based on the current uplink utilization. Peers increase their fan-out parameter by one with a probability $P(1 - \text{Uplink Utilization})$ at each gossip round. Otherwise, the fan-out is decreased by one. Uplink/Downlink utilization is the ratio of the aggregate traffic flow on all uplinks/downlinks to the aggregate capacity of all uplinks/downlinks in the system.

Figure 5.1 and 5.2 show the average uplink and downlink utilizations versus gossip interval. The homogeneous system stated in Figure 5.1 has relatively limited bandwidth resources in comparison to the system in Figure 5.2. We observe that increasing the gossip interval leads more significant degradation on the uplink and downlink utilizations for the system consisting of peers having more uplink/downlink capacities. The difference is due to the block transmission times. For the system with more bandwidth, the time required to complete a block transmission is shorter compared to the other system. Hence, since the rate of gossip messages which inform other peers about available blocks is reduced, it is more likely for a peer to be idle between two successive incoming gossip messages. This reduces the downlink utilization. Similarly, the uplink utilization decreases faster. Hence, the gossip interval affects the performance of the protocol.

We run the simulator employing smart peer policy for the system settings with more bandwidth. When the smart peer policy is activated, peers reconfigure their fan-out

parameter. This decreases the probability of not finding a peer for cooperation. In general, although increasing gossip interval parameter reduces the overhead, it decreases uplink/downlink utilization. However, Figure 5.3 shows that smart peer policy alleviates this negative effect of the increase in gossip interval on the utilization. Likewise, the smart peer policy reduces the average download of the file as shown in Figure 5.4.

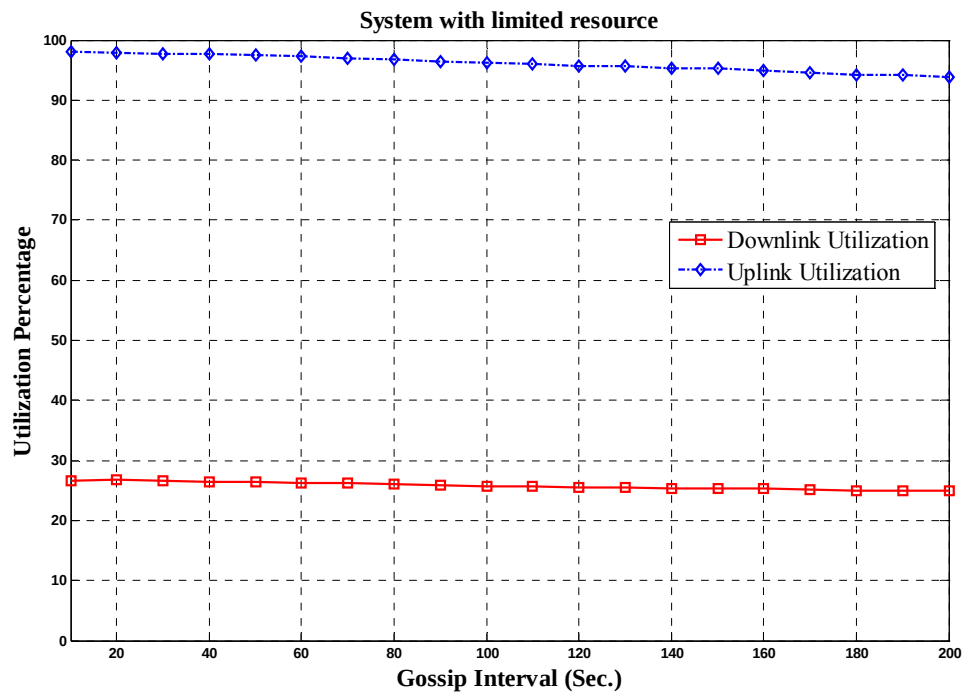


Figure 5.1 System with limited bandwidth; 512/128 Kbs Downlink/Uplink bandwidths

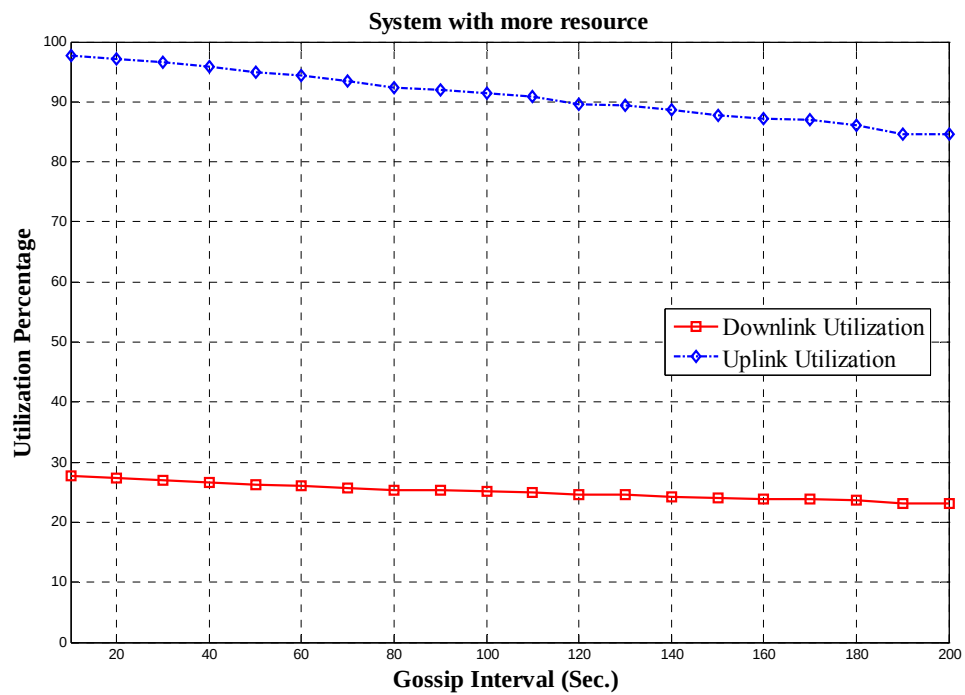


Figure 5.2 System with high bandwidth; 1500/512 Kbs Downlink/Uplink bandwidths

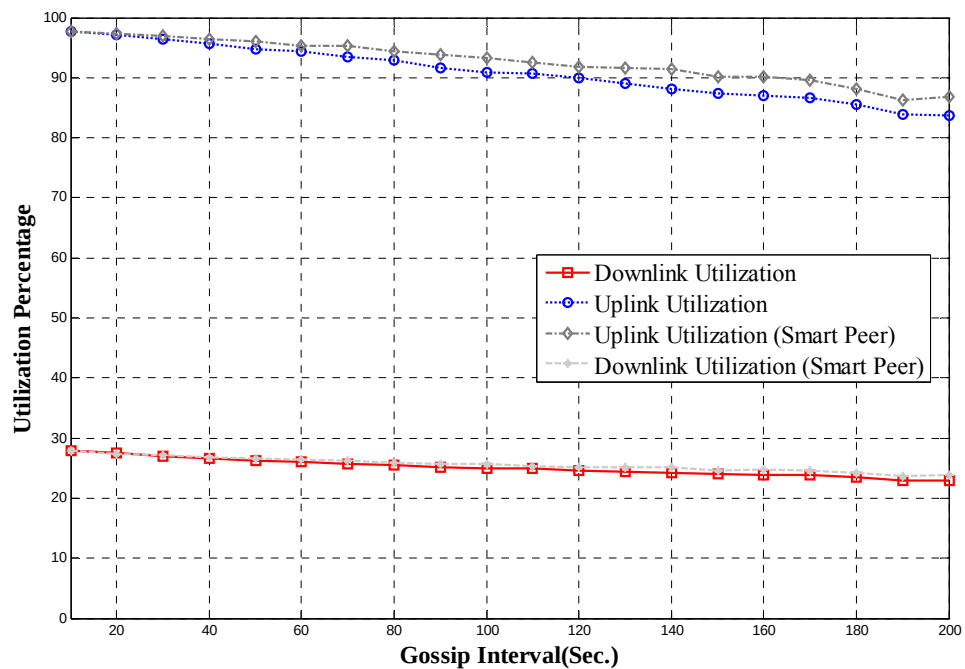


Figure 5.3 Effect of smart peer policy on uplink/downlink utilizations

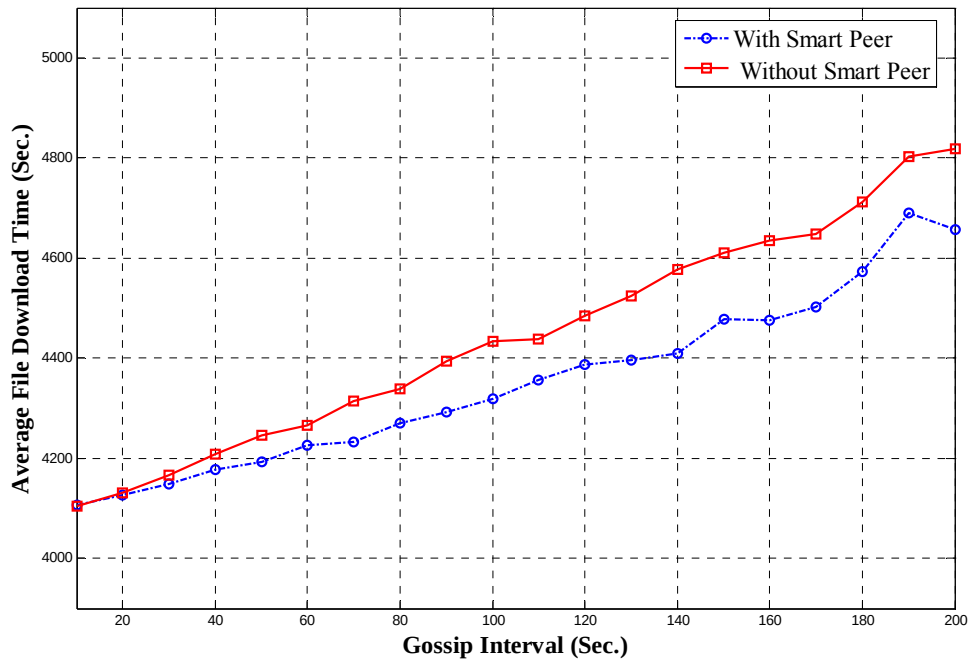


Figure 5.4 Effect of smart peer policy on average file download time

Finally, note that although decreasing the gossip interval has a positive effect on the state propagation, it brings additional protocol communication load on the system.

5.1.2 Parallel Upload Limit

The number of peers a source can serve simultaneously is limited by the parallel upload limit. However, this parameter is not the only decision factor for accepting or rejecting an incoming upload request. If the maximum parallel uploading limit is reached and another upload request is received, the source checks the utilization of its uplink capacity. If this utilization greater than a value set between 0 and 1, called utilization threshold, new coming request is rejected or added to upload queue of the source. Otherwise, it lets the requester to download without taking the maximum parallel upload capacity into consideration.

Setting the parallel upload limit to a specific value for each participant may lead to waste of uplink capacities. As an illustrative example, assume that the source peer has

6000 Kbps uplink capacity and it has already been serving 4 peers whose downlink capacities are 750 kbps. If the source does not accept any other upload request, source uplink utilization can reach at most 0.5. On the other hand, assuming that there are 10 peers served for the time being whose downlink capacities are 10000 Kbps, accepting new upload request may lead to latency in replication of the blocks in the system. Taking the uplink utilization into consideration in upload decisions handles these kinds of scenarios.

We run our simulator for different parallel upload limit values. During the simulations, the system consists of peers whose uplink and downlink distributions are as given in Table 4.1. Figure 5.5 shows that increasing parallel upload limit has a positive effect on the utilization of downlink capacities up to a certain value. After that point, downlink utilization stays almost constant. However, the uplink utilization does not change significantly with the increase of the parallel upload limit. When parallel upload limit is set to a small number, peers handle a small number of transmissions with higher upload rates instead of serving more peers providing lower transmission rates. Since the number of served peers is not a factor in calculation of the uplink utilizations, we cannot observe any effect on the uplink utilization. However, if we continue increasing parallel upload limit, this may lead to latency in completion of the block transmissions. As a result, since the peers cannot serve other participants without owning a complete block, the uplink utilization starts to decrease. As given in Figure 5.6, average file download time, that is, average time period required to obtain a full copy of the file, decreases as the parallel upload limit increases up to a certain value. Increasing this parameter after this threshold does not lead to further decrease in the download time. Another important result is the data size uploaded by the primary seed. As shown in Figure 5.7, amount of the data served by the primary seed decreases continuously as the parallel upload limit increases. Departure pattern of the peers may lead to such a decrease. In the simulation, the peers leave the system as soon as they complete the download. Hence, when the number of peers served simultaneously is set to small numbers, the peers that have direct connection or are close to the primary seed complete

the download in shorter periods and leave the system without uploading other peers. This forces the remaining peers to download the blocks from the primary seed.

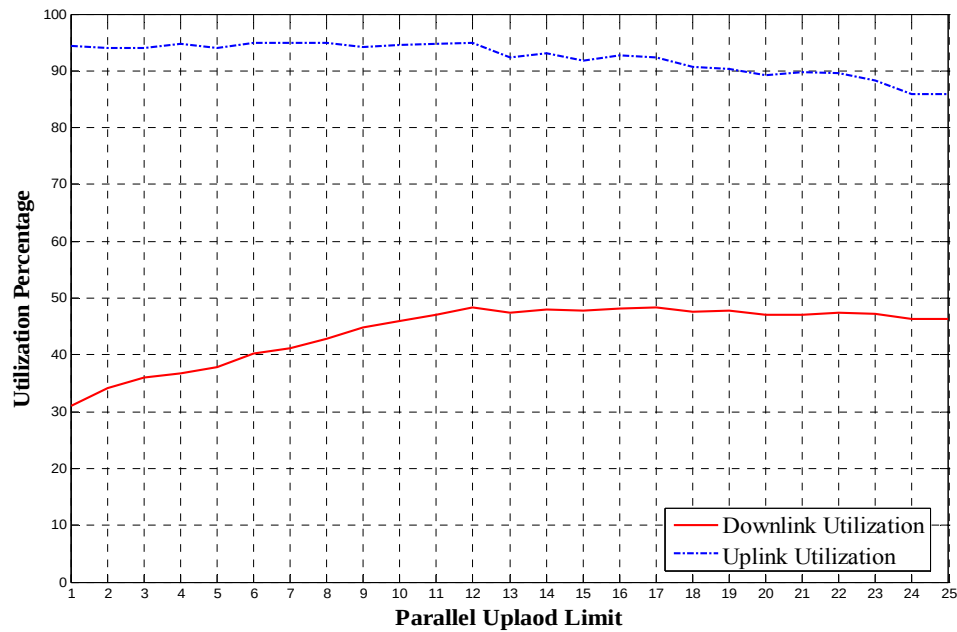


Figure 5.5 Parallel upload limit vs. Downlink and uplink utilizations

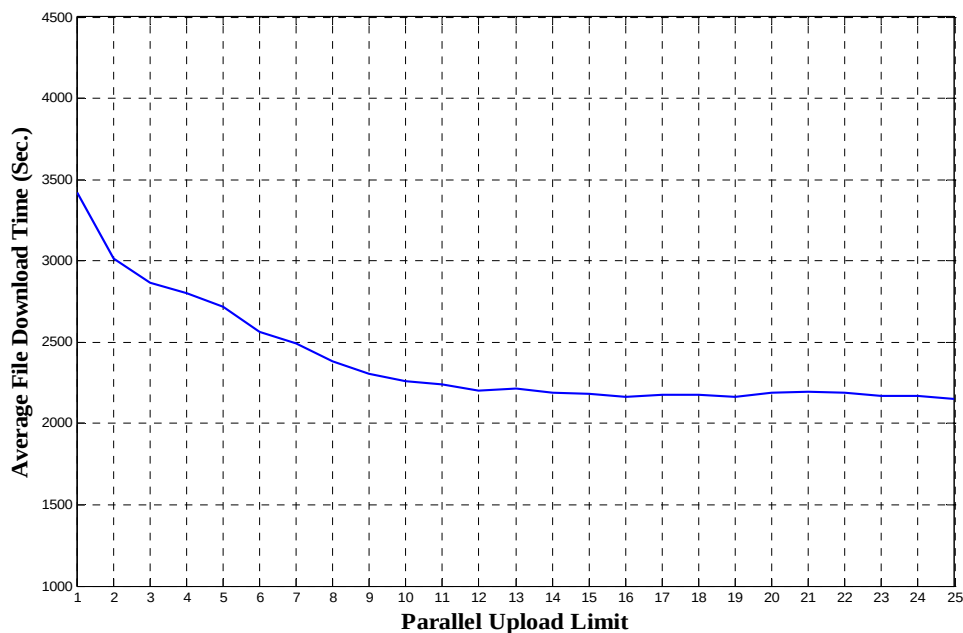


Figure 5.6 Parallel upload limit vs. Average file download time

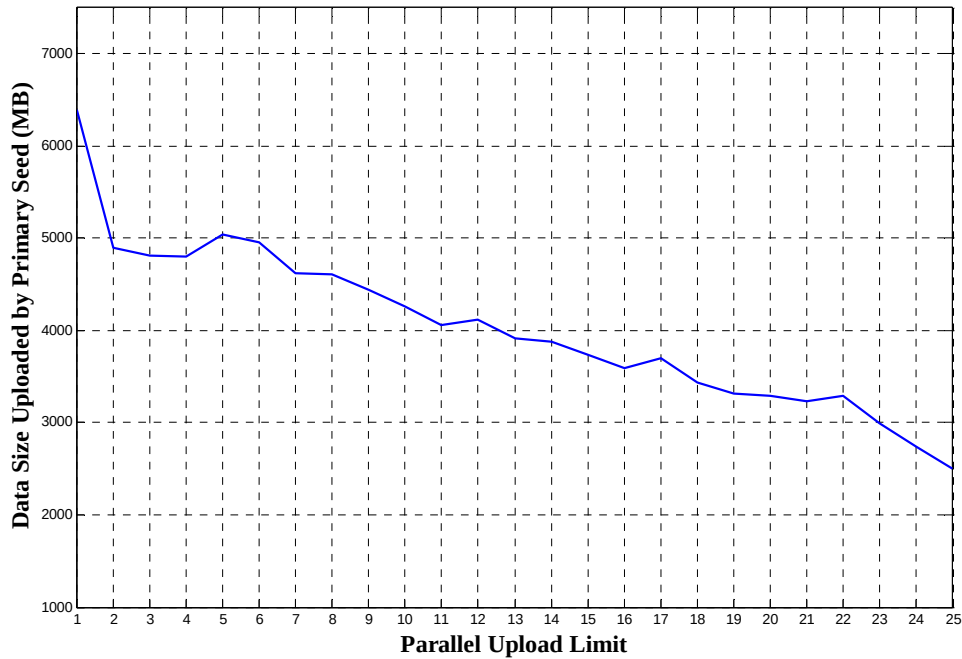


Figure 5.7 Parallel upload limit vs. Load imposed on the primary seed

5.1.3 Connection Size

The scalability requirement imposes the use of a scheme providing each peer with only partial view of the system. In SeCond, peers manage two subsets of participating peers in the system for block uploads and downloads. The first list is the gossip sender list and the other one is gossip receiver list. Recall that gossip receiver list includes the peers to which the peer uploads the available blocks and gossip sender list consists of the peers from where the peer downloads the blocks of the file. The number of neighbors that peers hold in their gossip receivers and senders lists is specified by the *connection size* parameter of the protocol. In a dynamic system, peers may leave the system at any time. However, if the number of active connections of a peer falls below a lower bound, it asks the index server for new neighbors. This prevents peers to be isolated from the system. On the other hand, if the connection size is set to small values,

peers may visit the index server for new neighbors more frequently. This may increase the workload of the index server.

In this section, we analyze whether holding more peers in these lists, that is having information about larger part of the system has a positive effect on the distribution of the file. Figure 5.8 shows that although the number of connections managed by the peers is increased, no significant change is observed in average file download time. Similarly, the amount of the data uploaded by the primary seed stays almost constant as it can be seen from Figure 5.9.

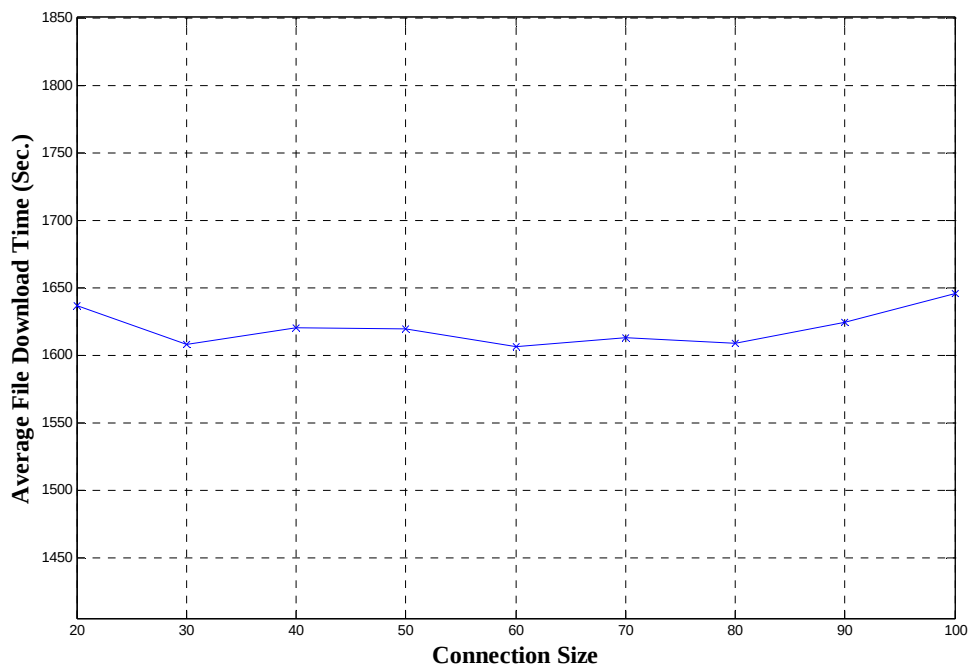


Figure 5.8 Connection size vs. Average file download time

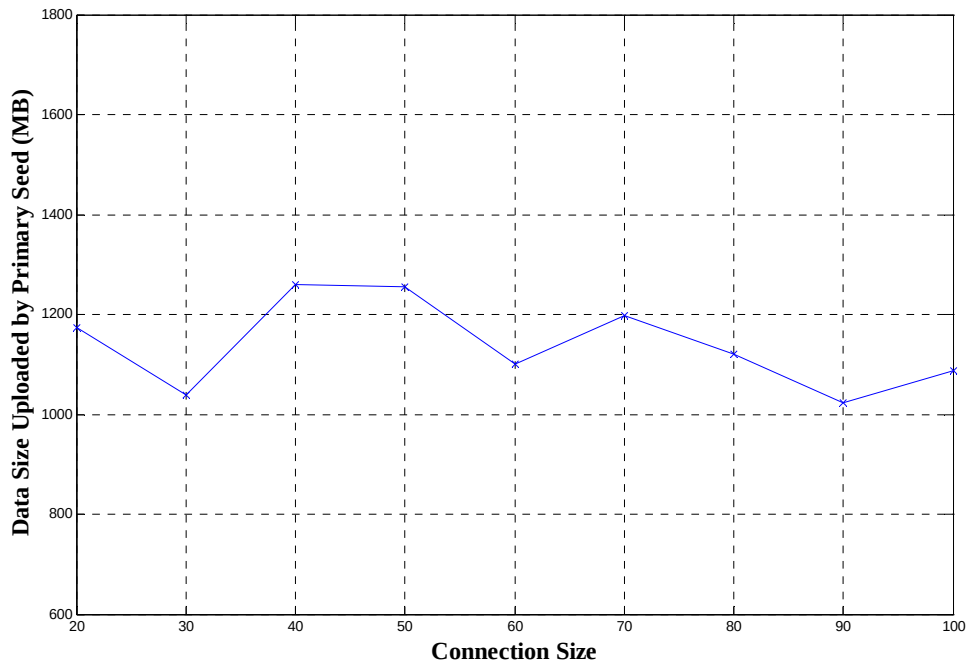


Figure 5.9 Connection size vs. Data size uploaded by primary seed

5.2 Performance Evaluation

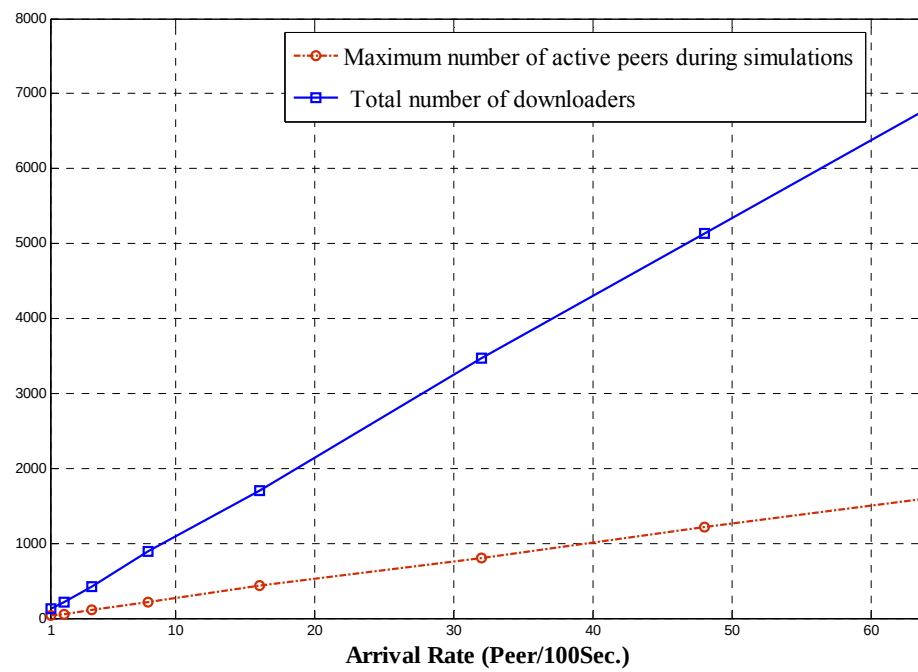
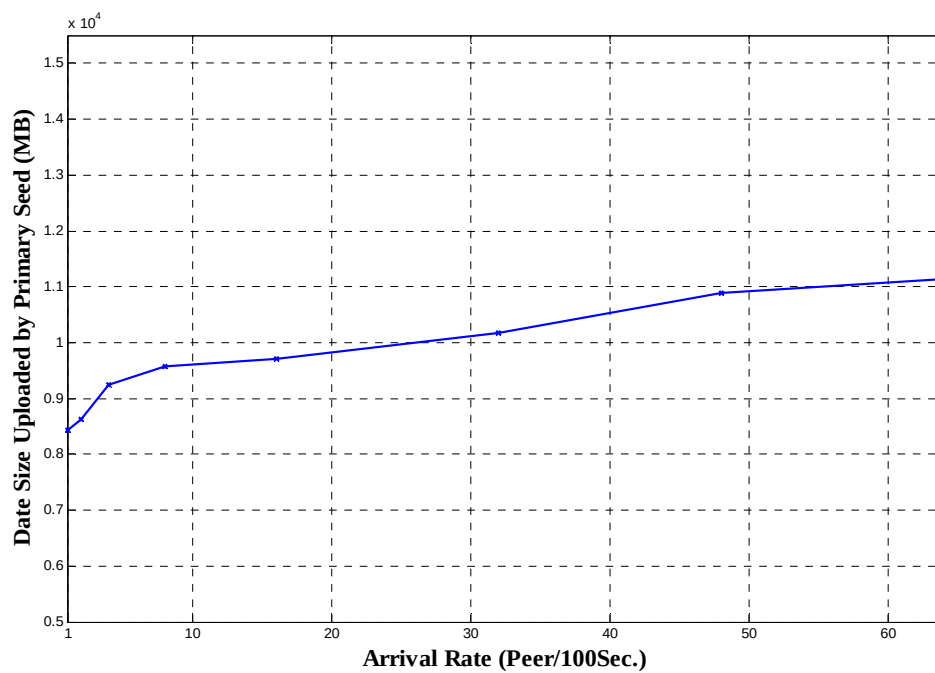
In this section, performance of the protocol is studied for different scenarios. In addition to the flash-crowd arrival rate, we employ a Poisson process for generation of the file requests. Since the rate of the file requests may vary according to popularity of the content, we assume time-dependent arrival rate $\lambda(t)$.

The presence of seeds is a key feature for the performance of the file distribution. It enhances the ability to scale to large set of requesters since the load of the primary seed is alleviated. We assume two different departure scenarios for the peers completing the download. *Immediate-departure* stands for a scenario in which the peers leave the system as soon as they complete the file download. For the second departure pattern, *random departure*, the time period in which the peer continues participating in the system as a seed is determined by exponential distribution.

As it is stated in chapter 4, a discrete event simulator of BitTorrent is implemented. We study different arrival patterns through BitTorrent simulator as well as SeCond to make a comparison between them.

5.2.1 Arrival Rate Analysis

In the first set of simulations, we assume that peers join the system in a three hour period. By increasing the arrival rate λ , our aim is analyze how the protocol operates under heavy traffic. Figure 5.10 shows the maximum number of active peers downloading the file simultaneously and the total number of peers requesting the file. During distribution of the file, the arrival rate is assumed to be stationary and peers departure as soon as they obtain a full copy of the file. We increase the arrival rate λ from 1 to 64 having the *unit number of peers per 100 seconds*. The number of active peers at any time during our simulations reaches 1600 when λ is set to 64. As shown in Figure 5.11, data size uploaded by the primary seed increases logarithmically with respect to arrival rate. Similarly, Figure 5.12 shows that there is not any significant change in average download time despite the increasing arrival rate. These two figures indicate that SeCond scales well even if the peers depart immediately after completion of the download.

**Figure 5.10** System size**Figure 5.11** Arrival rate vs. Load imposed on the primary seed

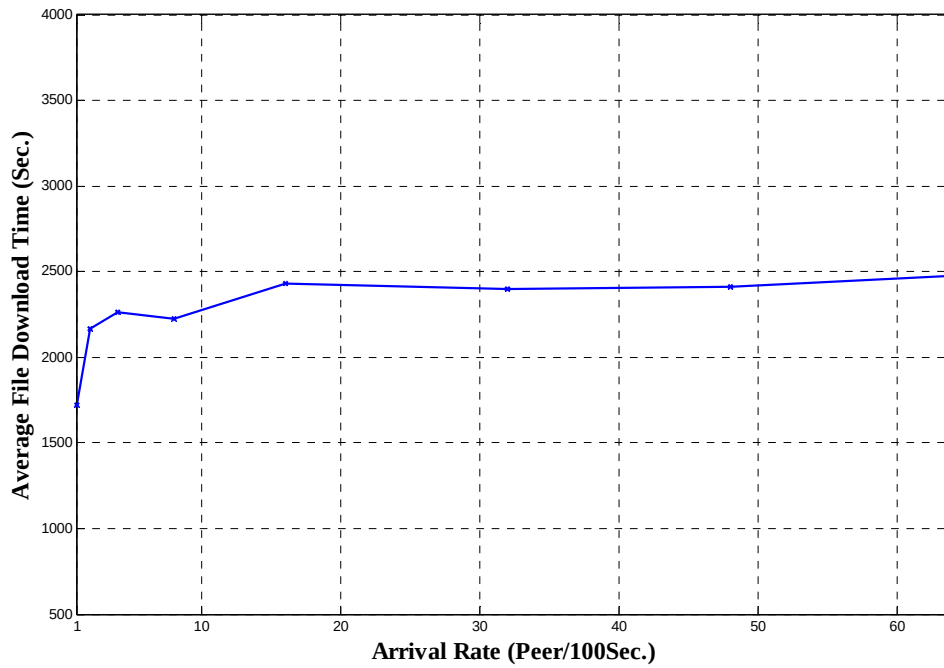
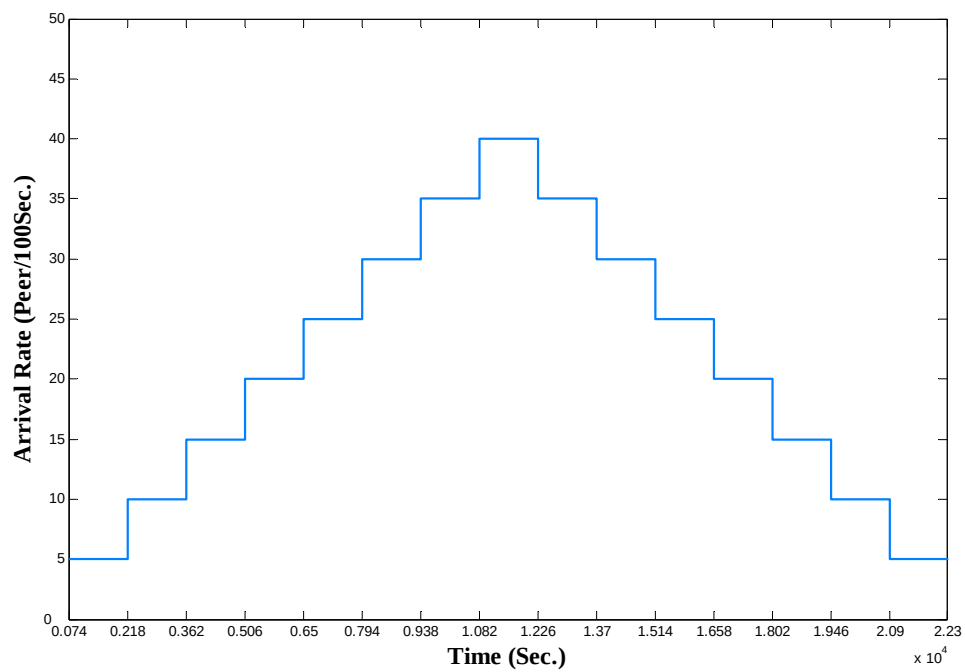
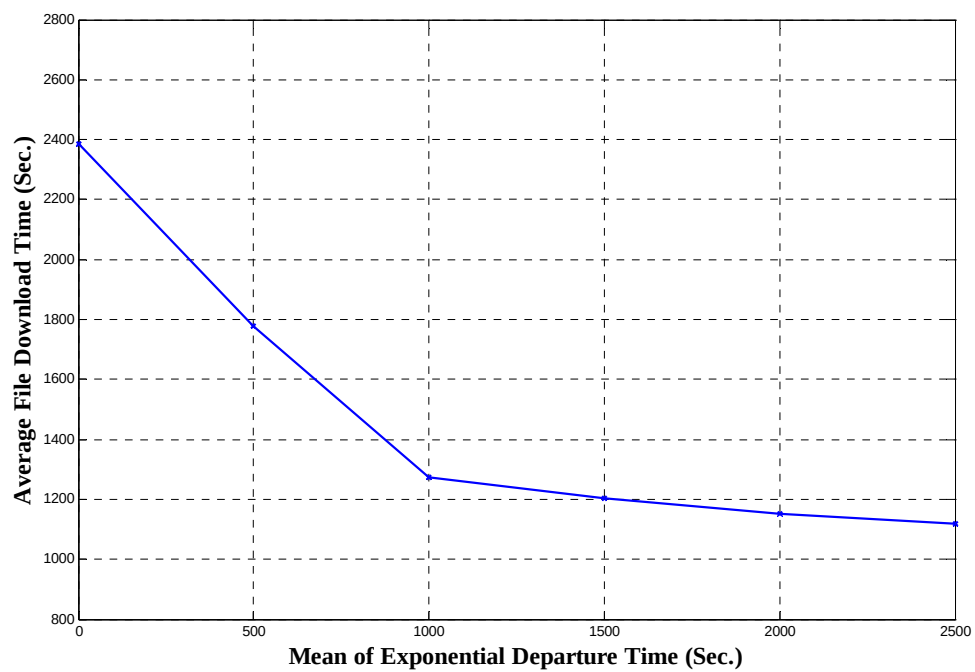


Figure 5.12 Arrival rate vs. Average file download time

5.2.2 Departure Rate Analysis

Although peers do not have to remain connected to the system after completion of their download, if they continue participating in dissemination as seeds and serve other peers, the performance of the system enhances significantly. We assume that the time period in which a peer remains connected in the system after completion of its download is exponentially distributed with mean μ . We run simulations for different μ values. Zero mean stands for *immediate-departure* which corresponds to the departure of peers immediately after the completion of download. In these sets of simulations, we deploy a non-stationary arrival rate $\lambda(t)$ which is given in Figure 5.13. Figure 5.14 illustrates that the average file download time decreases dramatically if the mean *departure time*, the time period peers remain connected after download of the file, increases. Similarly, the load of the primary seed is alleviated when the departure time of peers in the system is increased as it shown in Figure 5.15.

**Figure 5.13** Arrival rate vs. Time**Figure 5.14** Mean departure time vs. Average file download time

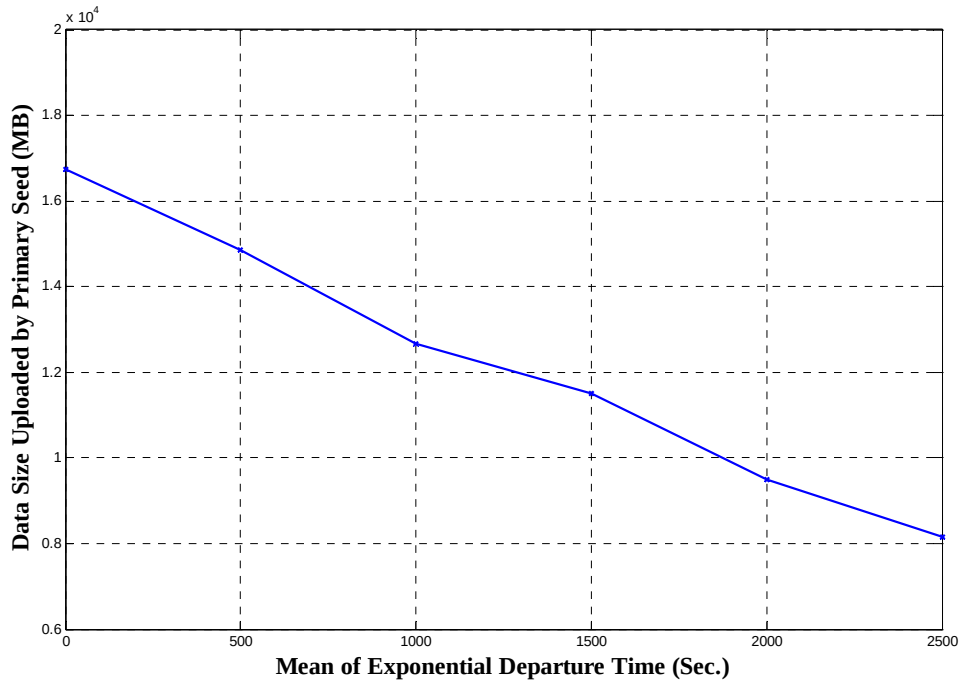


Figure 5.15 Mean of exponential departure time vs. Load imposed on the primary seed

5.2.3 Scalability Analysis for Different Arrival Patterns

According to the popularity of the published content, the arrival rate of file requests can follow different patterns. For this purpose, we analyze five different arrival patterns, namely *flash-crowd*, *constant*, *increasing*, *decreasing*, and first increasing then decreasing arrival rate which we simply call *triangular*. Flash-crowd stands for joining the system at the same time which can be thought as a storm. While the constant arrival corresponds to a Poisson process with constant rate for the generation of file requests, the increasing and decreasing arrival rates stands for non-stationary Poisson processes with increasing and decreasing rates, respectively. In general, the published content gets more popular in time after it is released. However, this popularity starts to decline as there are more peers owning a full copy of the file. Triangular arrival rate models such a case. Figure 5.16 illustrates our implementation of these arrival scenarios.

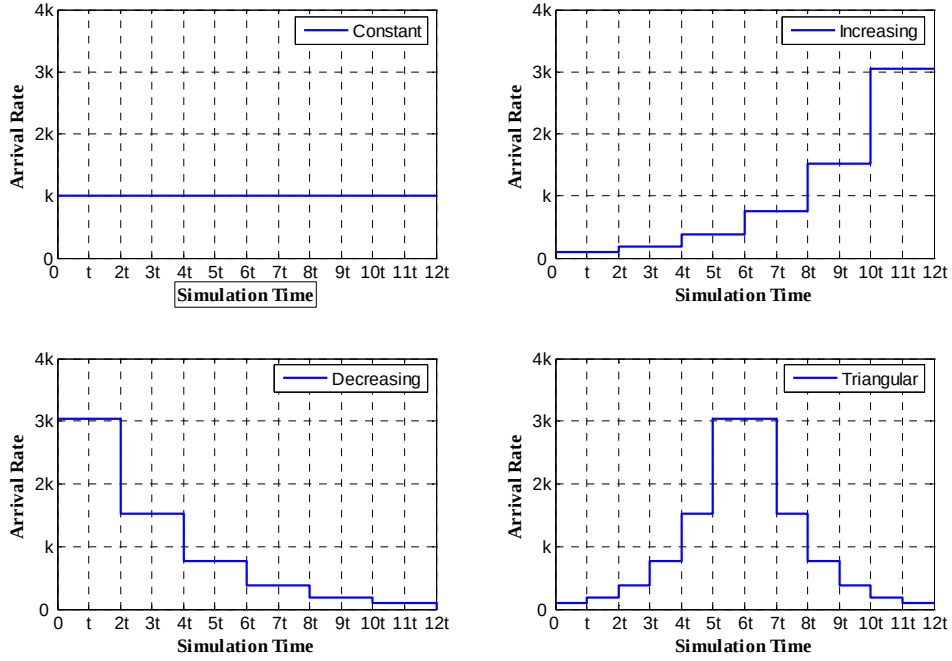


Figure 5.16 Arrival Patterns

The time period in which new peers join the system is divided into 12 equal time intervals. The initial arrival rate is duplicated/halved per two subintervals for increasing/decreasing arrival patterns. For example, if the initial arrival rate is λ , it reaches 32λ in the last two subintervals during the increasing arrival period. In the Bell-Shaped pattern, the initial arrival rate is duplicated per subinterval up to the half of the total arrival period then for the rest of the period it is halved per subinterval.

We run both the BitTorrent and SeCond simulators for the arrival patterns described. In the following results for different arrival patterns, it is assumed that peers join the system within a three hour period and follow an immediate departure pattern. The system consists of heterogeneous peers whose bandwidth distribution was described in Section 4.5. For the scalability analysis, we vary the peer population joining the system from 1000 to 8000.

Figure 5.17 shows the maximum number of active peers as a function of system size for different arrival patterns. It is observed that the system is the most crowded for the increasing arrival pattern and the least for the constant arrival rate. The effect of

system size on the average file download times are given in figures 5.18-5.21 for four arrival patterns. These four figures show that the average file download time is almost constant even if the scale of the system is eight folded for both the Second and BitTorrent models. There is no significant difference in the average download times corresponding to arrival patterns in SeCond simulation results. On the other hand, for the BitTorrent's results, the increasing arrival pattern causes the largest average file download time. For these arrival scenarios, SeCond peers download the file in shorter time periods comparing to BitTorrent peers. In order to exploit the system resources, SeCond proposes strategies such as smart peer and view update policies. These results endorse that these policies increases the utilization of the system resources and leads to smaller download periods.

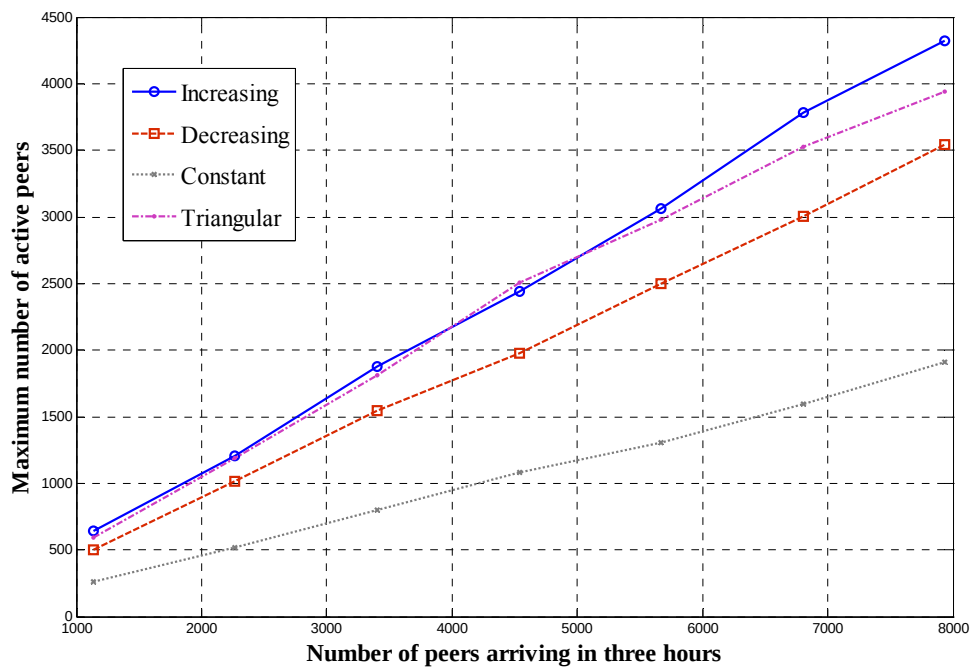


Figure 5.17 Maximum number of active peers vs. System Size

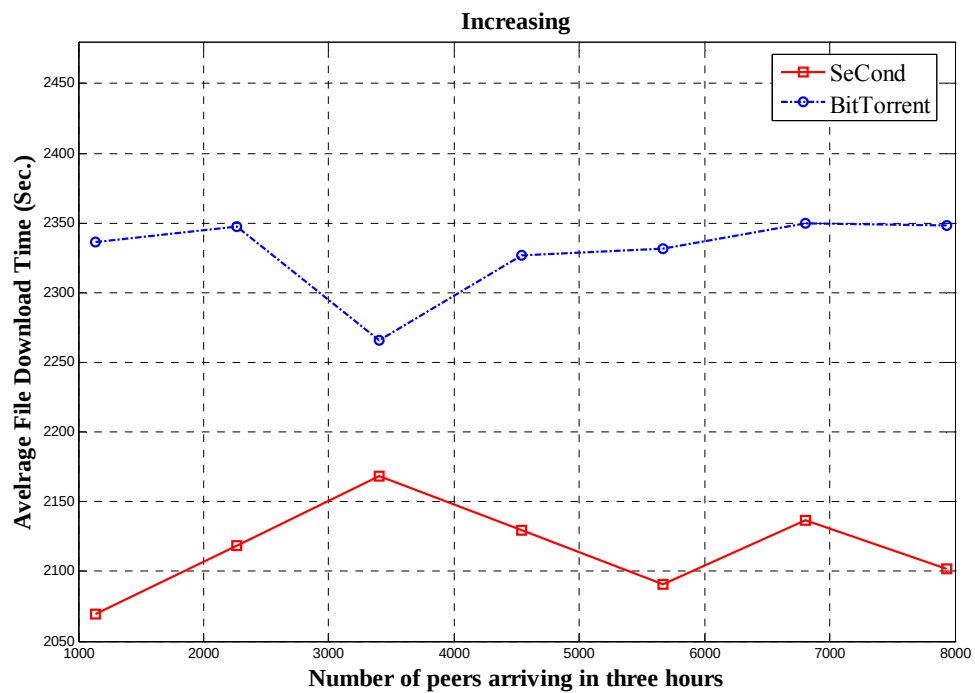


Figure 5.18 Average file download time for increasing arrival pattern vs. System size

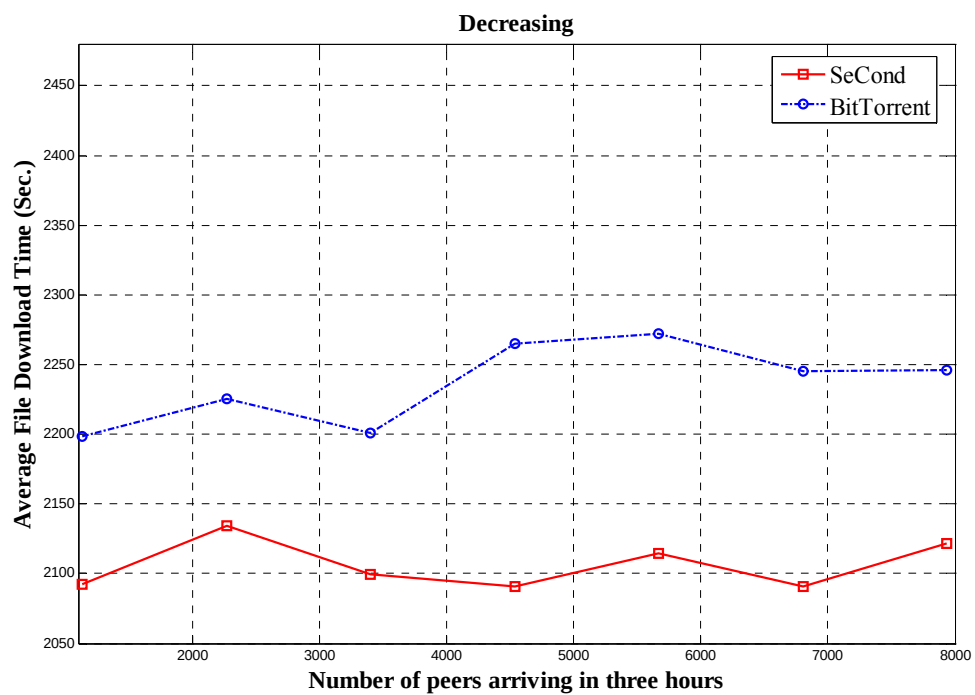


Figure 5.19 Average file download time for decreasing arrival pattern vs. System size

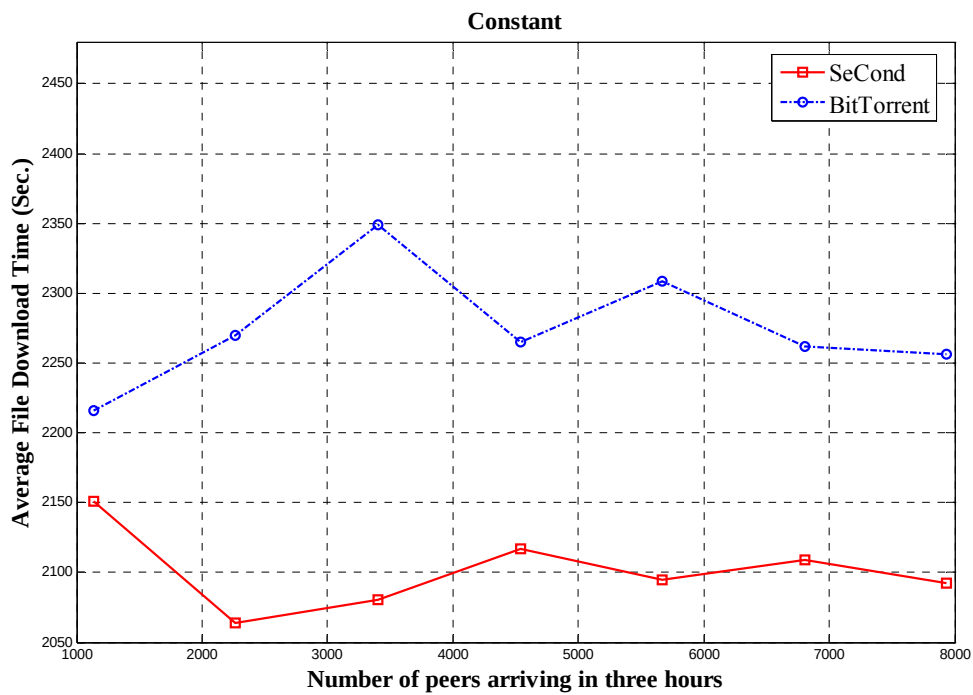


Figure 5.20 Average file download time for constant arrival pattern vs. System size

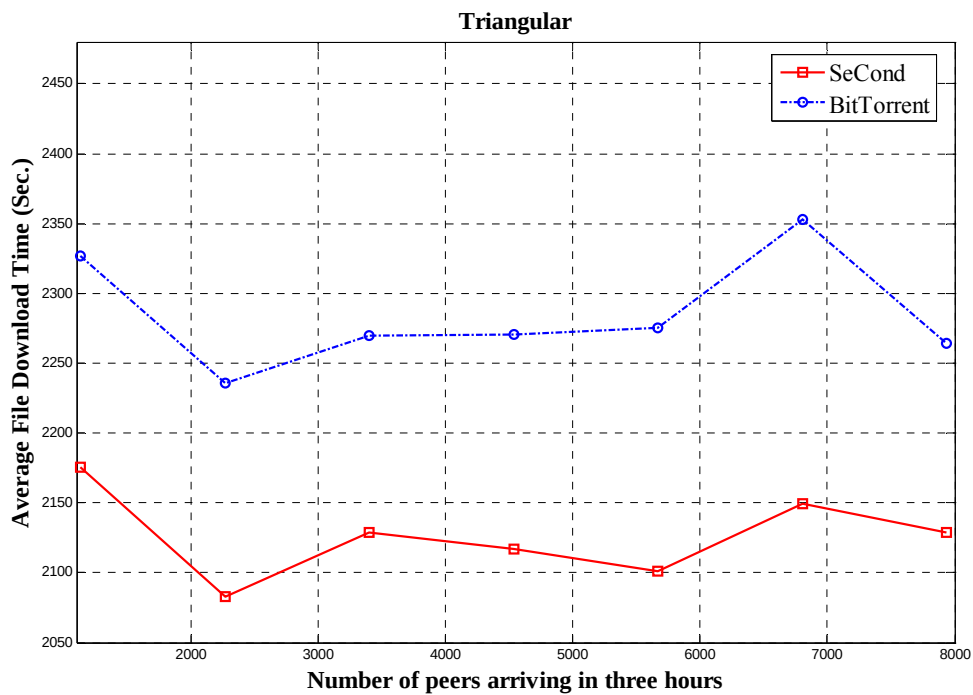


Figure 5.21 Average file download time for triangular arrival pattern vs. System size

If peers download large portion of the file from the primary seed, i.e. original publisher of the content, the time period required to obtain a full copy of the file is expected to increase since the primary seed would be overloaded and become a single point of failure. Results given above indicate that file download times stay bounded despite the increasing system size. Figures 5.22-5.25 given below show the data sizes uploaded by the primary seed. These results indicate that in both of the protocols, peers cooperate well among themselves instead of requesting the blocks from the primary seed. Although 8000 copies of the file are distributed, the ratio of the data size uploaded by the primary seed to the actual file size reaches at most 101 for the increasing rate. For the increasing arrival pattern, the load of the primary seed is relatively higher than the other arrival patterns since the peers joining the system initially cannot find so many downloaders to cooperate. Since the peers complete their download faster in SeCond, and leave the system immediately, the remaining peers download more from the primary seed in comparison to BitTorrent. This is reason for the slight difference between the loads imposed on the primary seed.

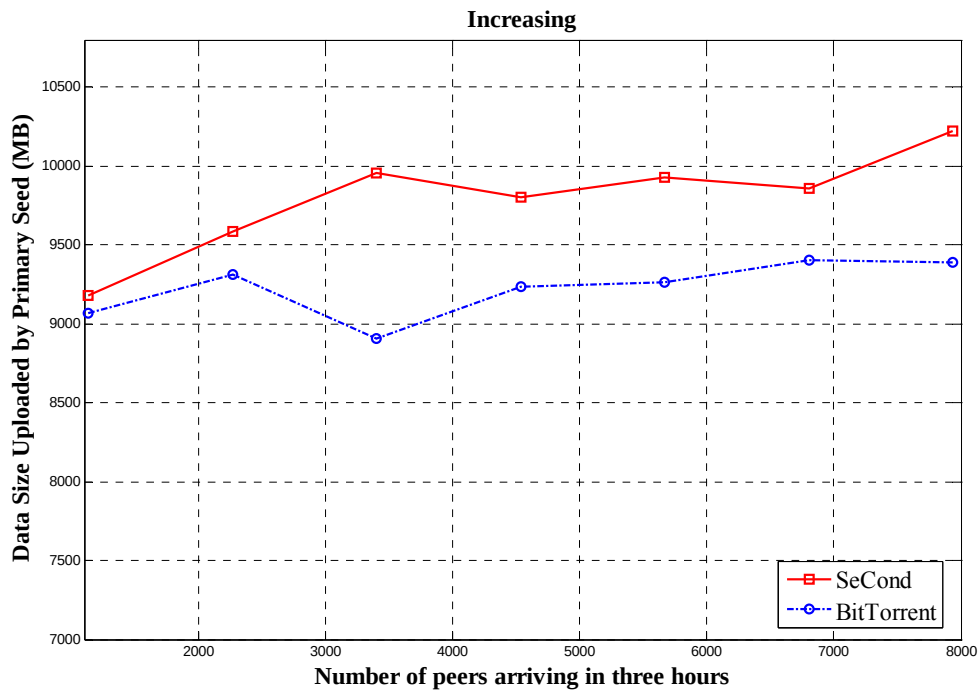


Figure 5.22 Load imposed on the primary seed for increasing arrival pattern vs. System size

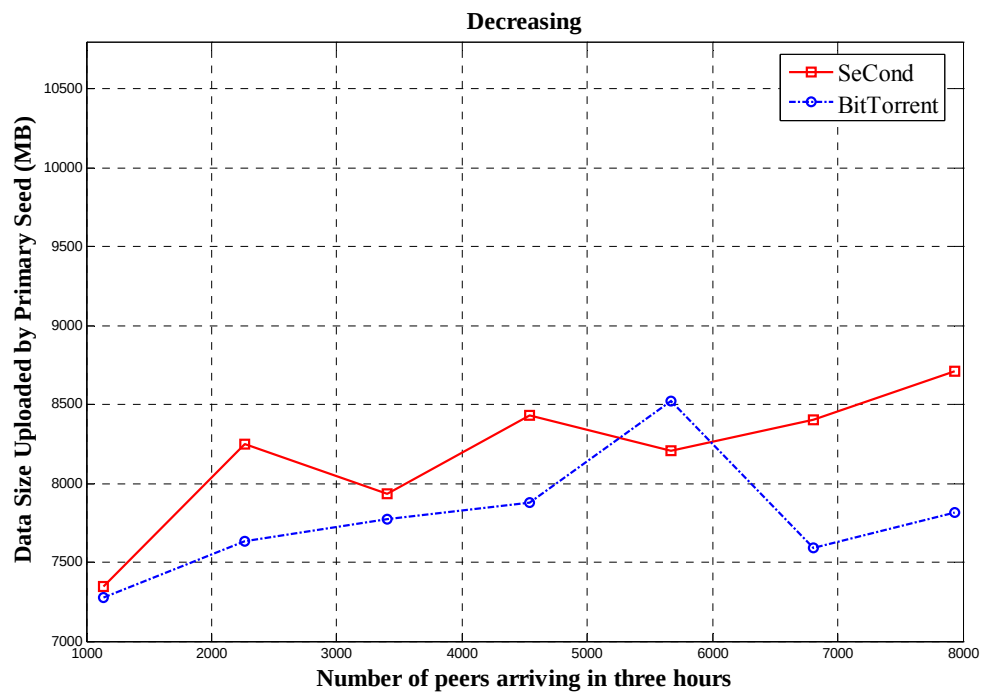


Figure 5.23 Load imposed on the primary seed for decreasing arrival pattern vs. System size

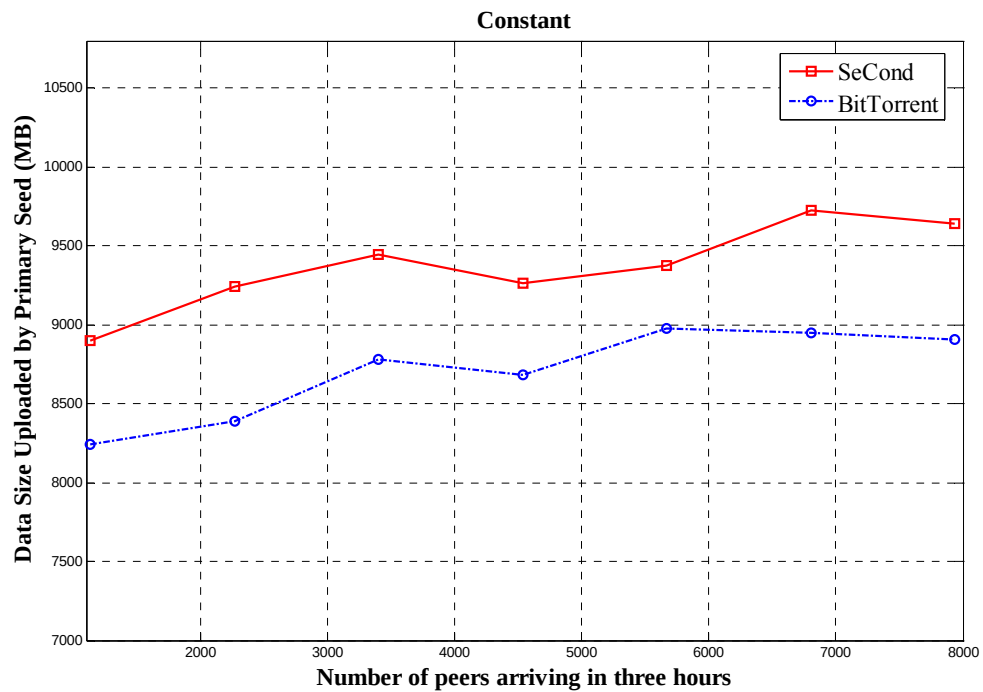


Figure 5.24 Load imposed on the primary seed for constant arrival pattern vs. System size

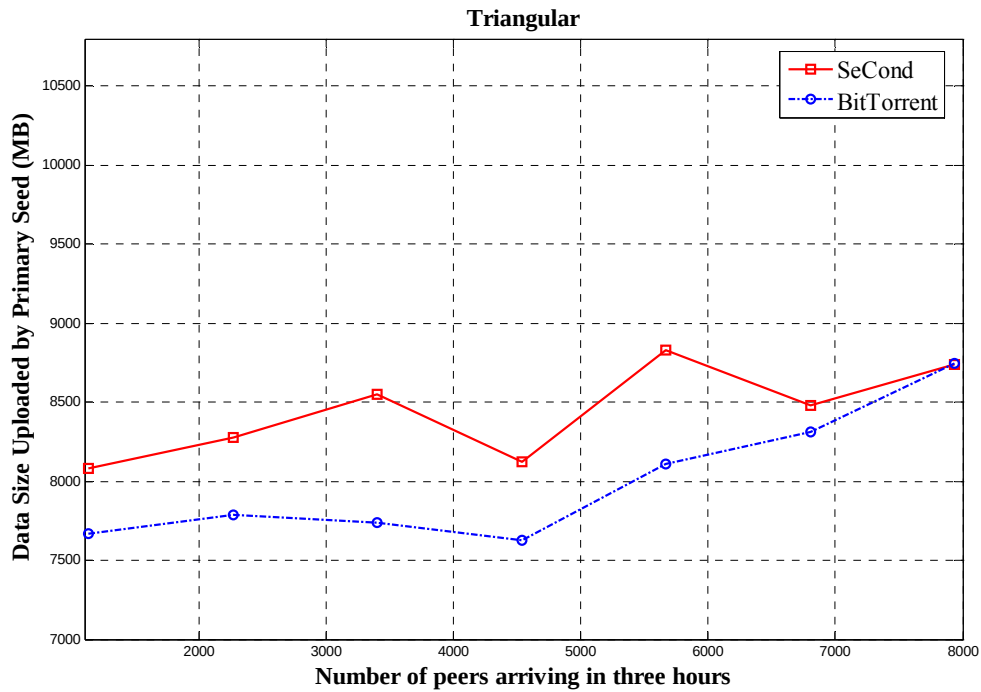


Figure 5.25 Load imposed on the primary seed for triangular arrival pattern vs. System size

In addition to analysis given above, we study scalability for the flash-crowd arrival scenario. We assume that all the peers join the system simultaneously and scale up system size up to 8000 peers. Figures 5.26 and 5.27 show that even for the worst scenario, SeCond scales well and SeCond peers complete their download in comparable time to BitTorrent peers. The primary seed uploads a little more for the SeCond system.

5.2.4 Fairness Ratio

An important issue for P2P file sharing systems is free-riding. In heterogeneous systems, some peers may overuse the resources of the other peers. In a fair world, the *fairness ratio* of a peer, uploaded data size divided by downloaded data size by that peer, is expected to be 1. BitTorrent deploys *tit-for-tat* strategy which encourages peers to upload. A peer uploads the peers providing the best download rates. On the other hand, SeCond aims to increase performance of the overall system. Each peer updates its gossip sender and receiver lists dynamically. Moreover, protocol parameters are adjusted according to the utilization of uplink bandwidths. In order to compare

BitTorrent and SeCond, we study fairness ratio for a system of size 1000. Bandwidth distribution of the peers is given in Table 5.1. Despite the fact that SeCond has no explicit strategy addressing free-riding, Figure 5.28 shows that it is as fair as BitTorrent. The average fairness ratios are recorded as 0.97 and 0.98 for the SeCond and BitTorrent respectively. Peers having larger capacities upload more than they download, since the download time of a peer is not only determined by the download capacity of the peer, but also by the upload capacities of the neighbors it has. In a heterogeneous system, peers may have neighbors having varying bandwidth capacities. For instance, in Figure 5.28, the peers having ids between 850 and 1000 have larger fairness ratio values in comparison to the peers with lower capacities.

Table 5.1 Bandwidth distribution according to peer ids

Peer Id	Uplink (kbps)	Downlink (kbps)
0-200	128	784
200-600	384	1500
600-850	1000	3000
850-1000	5000	10000

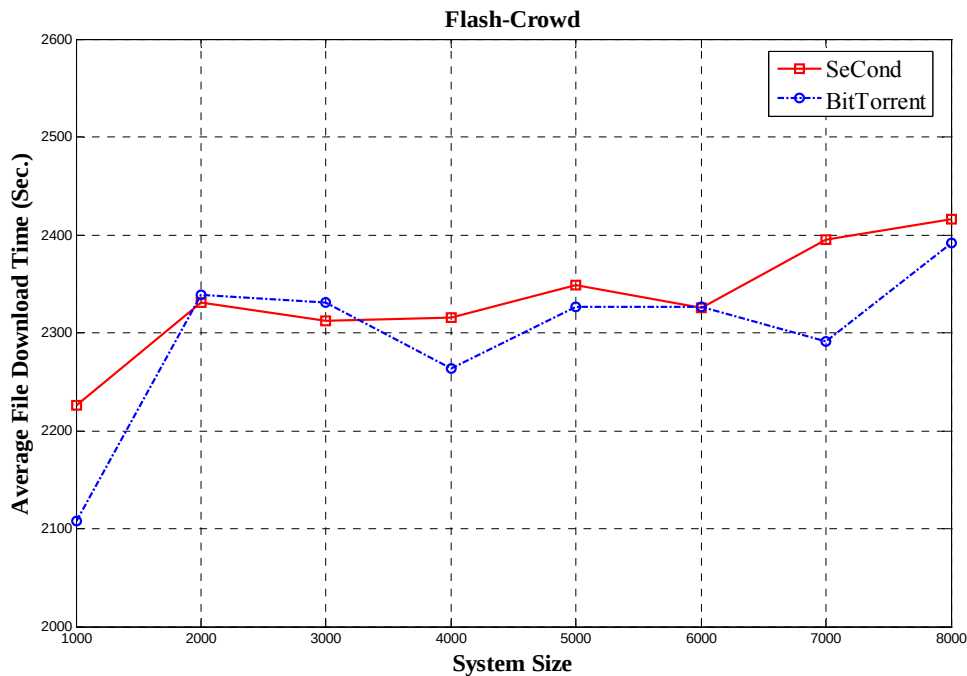


Figure 5.26 Average file download time vs. System Size

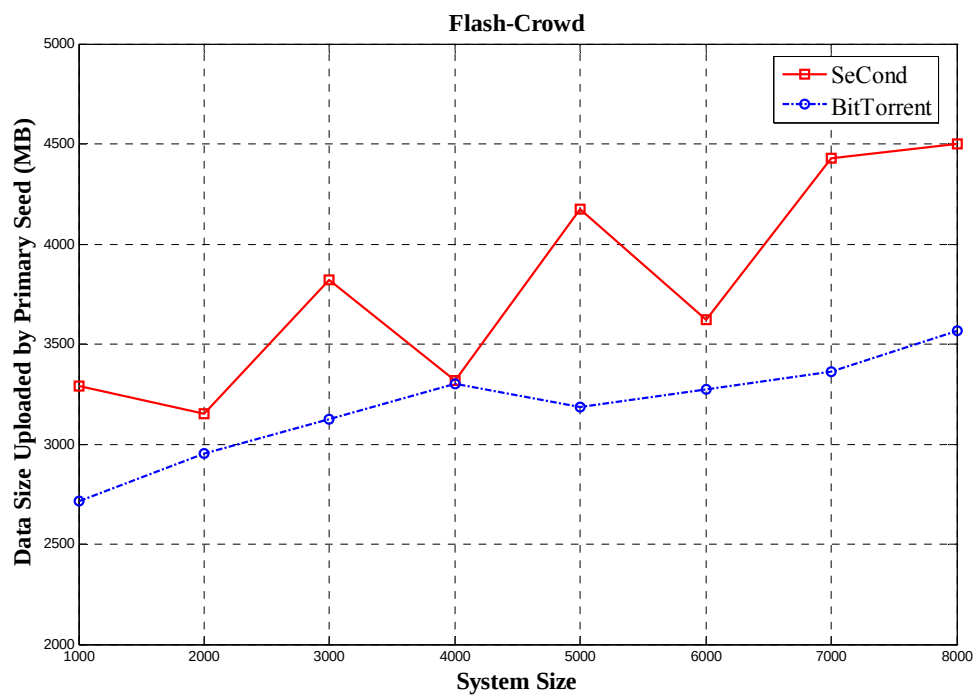


Figure 5.27 Load imposed on the primary seed vs. System Size

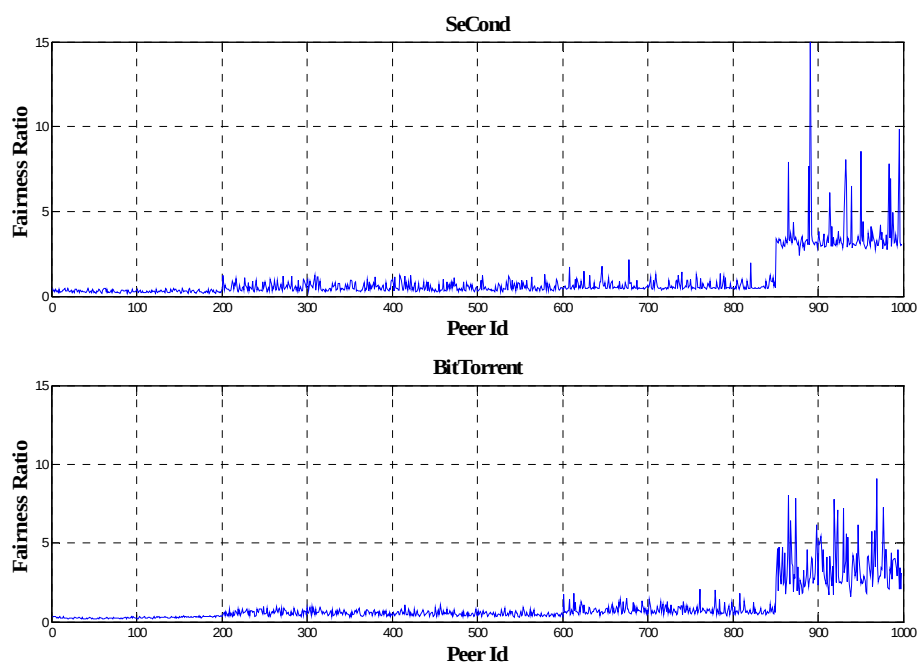


Figure 5.28 Fairness Ratio

5.2.5 Overhead Analysis

Control messages for propagation of available block information among the peers constitute the overhead of the cooperative protocols. In SeCond, at predefined time intervals, each peer sends gossip messages to inform other peers about newly downloaded blocks. On the other hand, each BitTorrent peer multicasts a message indicating the availability of a newly downloaded block to its neighbors immediately after completion of the block's transmission. Although a state update message in BitTorrent indicates the availability of a single block, SeCond's state update exchanges, namely gossip messages, can contain varying number of block identifiers. In order to make a comparison between BitTorrent's and SeCond's protocol overhead, we calculate the *weighted loads*. Weight of a state update message is determined by the number of blocks ids inserted to the message. We measure the state update messages for a system which consists of 1000 peers with flash crowd arrivals.

Figure 5.29 and 5.30 show the histogram for the state update messages and gossip messages sent by the BitTorrent and SeCond peers respectively. It is observed that, more than half of the peers send approximately the same number of gossip messages for the SeCond system. However, the variation observed in the number of state update messages sent by peers is higher for the BitTorrent system. Likewise, the number of state update messages sent by the BitTorrent and SeCond peers are shown in Figure 5.31 and 5.32 respectively. In these simulations, the total number of state update messages sent by BitTorrent peers is measured as 35,944,579 which is the weighted load of BitTorrent peers at the same time. On the other hand, gossip message count sent by SeCond peers is measured as 410,843 and the weighted load imposed by the SeCond peers is calculated as 43,259,121. Although the computed weighted load of the SeCond is greater than BitTorrent's, the number of state update messages of BitTorrent is significantly larger than the SeCond's. The number of state update messages of BitTorrent is measured as 87 folds of the SeCond's. Furthermore, considering the fact that each individual message generated contains a header field that is not taken into

account when computing the weighted loads, this causes an additional large transmission overhead for BitTorrent. Hence, comparatively lower protocol overhead of SeCond is one of the major reasons behind the lower file download time of peers in the system. Since the size of the state update messages sent in bytes may vary depending on the implementations of the protocols, the communication load in bytes is not calculated. For example, instead of sending id of an available block as an integer, each state update message can contain a bit vector indicating the availability of the blocks.

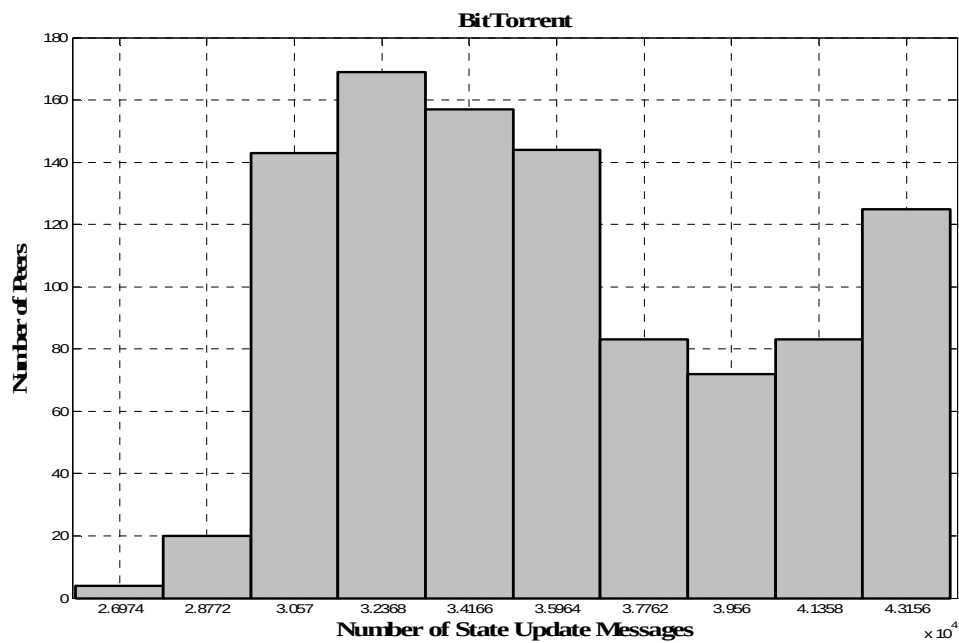


Figure 5.29 Histogram for BitTorrent state update messages

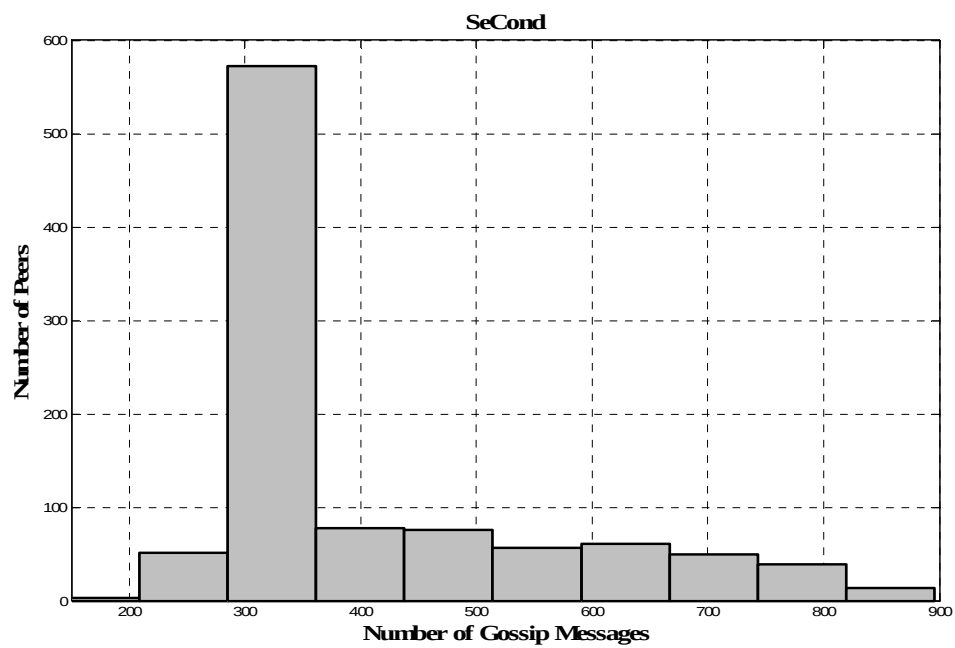


Figure 5.30 Histogram for SeCond gossip messages

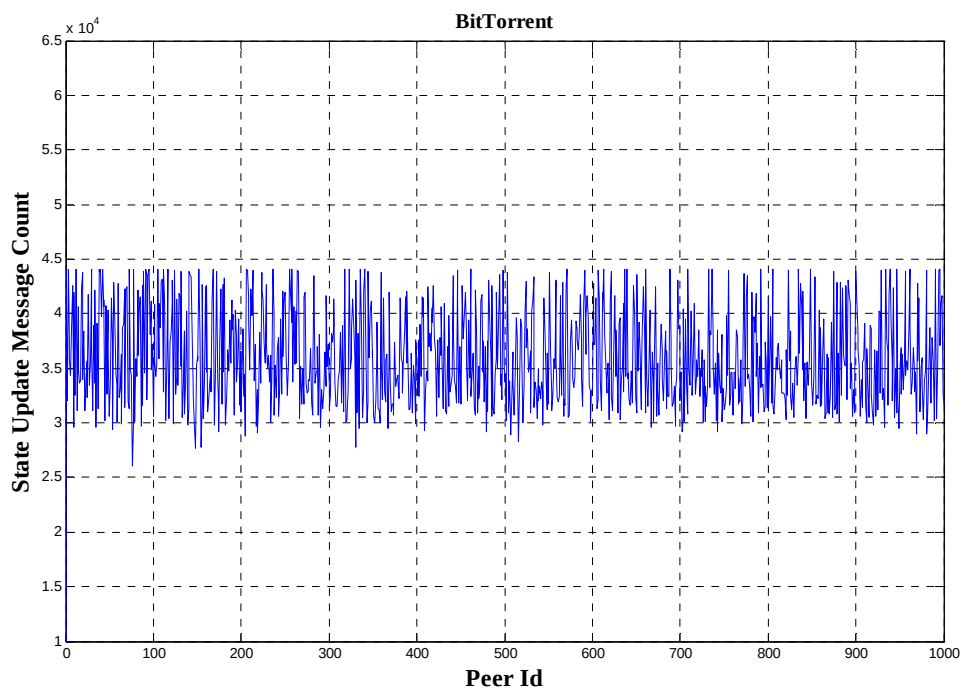


Figure 5.31 Number of state update messages sent by BitTorrent peers

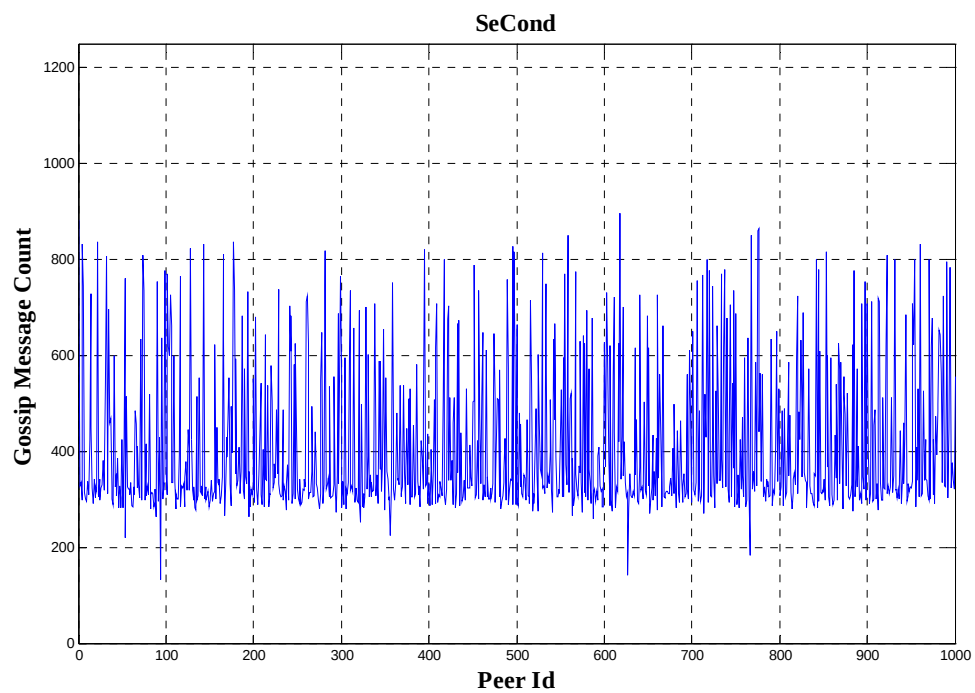


Figure 5.32 Number of gossip messages sent by SeCond peers

Chapter 6

ANALYTICAL APPROACHES ON COOPERATIVE P2P SYSTEMS

Many studies so far have focused on design, traffic measurement, and workload analysis of P2P file sharing systems. In addition to these, there are several analytical studies providing better understanding in design and analysis of such systems. In this chapter, we review the analytical models investigating the cooperative content distribution protocols in several aspects.

There is a tradeoff between the simplicity of the proposed analytical model and the ability to reflect the aspects of the protocol. Emplacement of many protocol parameters into the model may result in analytical results that are hard to follow. Hence it is indispensable to make assumptions and simplifications while constructing the analytical models.

The characteristics of the cooperative P2P systems are studied for both *transient* and *steady state* regimes. Transient regime stands for a flash-crowd arrival which may arise when a popular file is first released. As the replication of the content proceeds, the intensity of the requests stabilizes and the system enters steady state regime.

In the sections 6.1 and 6.2, we investigate the studies based on Markovian models and fluid models respectively. The studies modeling heterogeneous systems are given in section 6.3. Finally, we compare one of the fluid models with the simulation results obtained for SeCond.

6.1 Markovian Models

Several approaches have been used for the analysis of cooperative P2P file sharing systems. One of the complementary approaches is modeling the system by Markov

chains. The analysis given in [40], [41] uses a branching process to investigate service capacity of the system in transient regime. Branching process model for file replication shows that in transient regime cooperative P2P protocols scale well. The average file download time grows logarithmically in system size. In fact, our simulation results given in Figure 5.26 are consistent with this induction. For the steady state analysis, the system is described by a continuous time Markov chain. The system state is a pair (x, y) where x and y are the number of leechers and seeds respectively. A Poisson process with a constant arrival rate λ is assumed for the arrival of new peers. Leecher peers contribute the service capacity of the system with a fraction η taking values between 0 and 1. If $\eta=0$, then the downloader peers do not upload any data to others and only download from the seeds. The total service capacity of the system is determined by the equation $\mu(\eta x + y)$. The parameter μ is the rate of exponential distribution which determines the upload time of the file by a single peer. It is assumed that the time until a peer leaves the system after becoming a seed is exponentially distributed with rate γ . The transition matrix Q of the Markov chain is given by

$$\begin{aligned} q((x, y), (x+1, y)) &= \lambda, \\ q((x, y), (x-1, y+1)) &= \mu(\eta x + y), \\ q((x, y), (x, y-1)) &= \gamma y \end{aligned} \tag{1}$$

In this study, it is shown that the service capacity of a P2P system scale and peer's performance does not degrade significantly. However, although the stationary distribution for the Markov chain is computed numerically, there are no closed form solutions to study the steady state.

In [42], a probabilistic model of coupon replication systems is introduced. The detailed Markovian model with huge state space is used to investigate the effectiveness and scalability of a P2P system. It is analytically shown that file swarming performs well under flash crowds, even when rarest first is replaced by a random selection strategy and the performance of file swarming systems does not depend critically on altruistic peer behavior.

6.2 Fluid Models

However, using the same model of [40], a deterministic fluid model is developed in [43] to determine the steady state solution. The authors assume that each peer has the same uplink μ and downlink c bandwidth capacities. Abandonment of file downloads by peers is integrated in the model. θ is the rate at which downloaders leave the system. The total uploading rate of the system is given as $\min\{cx(t), \mu(\eta x(t) + y(t))\}$. The deterministic fluid model for the estimation of number of leechers and seeds is given by

$$\begin{aligned}\frac{dx}{dt} &= \lambda - \theta x(t) - \min\{cx(t), \mu(\eta x(t) + y(t))\}, \\ \frac{dy}{dt} &= \min\{cx(t), \mu(\eta x(t) + y(t))\} - \gamma y(t)\end{aligned}\quad (2)$$

For the steady state analysis, the rate of changes in the number of seeds and the number of leechers are assumed to be zero. Namely, $\frac{dx(t)}{dt} = \frac{dy(t)}{dt} = 0$. It is shown that average file download time T in steady state regime is given by

$$T = \frac{1}{\theta + \beta} \quad (3)$$

where $1/\beta = \max\{1/c, 1/\eta(1/\mu - 1/\gamma)\}$. Equation (3) shows that the average download time T is independent of the arrival rate λ which agrees well with the result given in Figure 5.12. T is inversely proportional to the effectiveness of the file sharing η . The download bandwidth c has a positive effect on T if $1/c \leq 1/\eta(1/\mu - 1/\gamma)$. Otherwise, the download bandwidth is not the bottleneck. In the paper, a simple model to calculate the value of η is stated. In the analysis, η is estimated as $1 - (\log M/M)^k$ where $k = \min\{x-1, K\}$ and M is the number of blocks of the shared file. K is the maximum number of connections to the leecher peers that a peer can maintain. A stochastic model addressing the variability of the number of leechers and seeds predicted by the deterministic fluid model is given in the study. It is shown that the stochastic process

converges to deterministic equation limits as the arrival rate λ goes to infinity. Finally, the incentive mechanism and optimistic choking properties of BitTorrent are discussed.

The study in [44] proposes a fluid model based on the stochastic differential equations. In the model, it is assumed that each peer maintains a connection with a probability of ρ to other peers in the system. Hence, the expected number of connections that a peer maintains is approximately $\rho N(t)$ where $N(t)$ is the system size at time t . The average download rate between two peers is denoted by ς . Maximum transfer bandwidth denoted by B is supposed to be a constant for all the peers. The file is divided into M blocks of size s . Apart from the models given above, this model defines three states for a peer. The peers in the first state are leecher peers holding blocks less than half of the M blocks while the second state peers are leechers holding more than half of the file but not all the blocks. The last state indicates a seed. Let $X_1(t)$, $X_2(t)$, $Y(t)$ denote the number of peers at time t in these stages respectively.

To study steady state performance, three cases are defined as

$$\begin{aligned} E(X_1(t)) + E(X_2(t)) + E(Y(t)) &< B / 2\varsigma\rho && \text{Case 1,} \\ E(X_2(t)) + E(Y(t)) &< B / 2\varsigma\rho < E(X_1(t)) + E(X_2(t)) + E(Y(t)) && \text{Case 2} \\ B / 2\varsigma\rho &< E(X_2(t)) + E(Y(t)) && \text{Case 3} \end{aligned} \quad (4)$$

In the model, $E(X_1(t))$, $E(X_2(t))$, $E(Y(t))$, and the average download time T are found as follows:

$$\begin{aligned} E(X_1(t)) &= \begin{cases} \frac{\sqrt{5}-1}{2} \sqrt{\frac{sM}{2\varsigma\rho}} - \frac{\lambda}{4\gamma} & \text{Case 1,} \\ sM\lambda/B & \text{Case 2 and 3} \end{cases} \\ E(X_2(t)) &= \begin{cases} \sqrt{\frac{sM\lambda}{2\varsigma\rho}} - \frac{\lambda}{2\gamma} & \text{Case 1 and 2,} \\ sM\lambda/B & \text{Case 3} \end{cases} \\ E(Y(t)) &= \lambda/\gamma && \text{Case 1,2, and 3} \end{aligned} \quad (5)$$

$$T = \begin{cases} \frac{1+\sqrt{5}}{2} \sqrt{\frac{sM}{2\varsigma\rho\lambda}} - \frac{3}{4\gamma} & \text{Case 1,} \\ \sqrt{\frac{sM}{2\varsigma\rho\lambda}} + \frac{sM}{B} - \frac{1}{2\gamma} & \text{Case 2,} \\ \frac{2sM}{B} & \text{Case 3.} \end{cases} \quad (6)$$

From the equations given above, the authors conclude that the system scales well and for the steady state the average download time decreases when the arrival rate λ increases. Unless peers are saturated due to the bandwidth capacity, having more seeds has a positive effect on the performance of the peers.

In [45], a system where peer arrival occurs with an exponentially decreasing rate is analyzed. The arrival rate is given by $\lambda(t) = \lambda_0 e^{-\frac{t}{\tau}}$ where λ_0 is the initial arrival rate when the file is released and τ is the attenuation parameter of the evaluation. In case of lack of seeds, a peer cannot complete the file download due to a single missing block and the download may fail. The downloading failure ratio is represented by

$$R_{fail} = \frac{\gamma}{\lambda_0} \quad (7)$$

where γ is departure rate of seeds. The authors develop a fluid model following the idea in [43] to analyze peer performance variations. However, the arrival rate is assumed to be as it is given above. It is shown that average download performance fluctuates considerably when the system size is small.

6.3 Studies on Heterogeneous Systems

All the models given above are designed for homogeneous systems. Namely, peers have the same uplink and downlink bandwidths. [46] is an extended study of [43] which models the heterogeneous systems. The multi-class fluid model takes the application-level differentiated services into account such as first class peers gets better service than second class peers. Two classes of peers are defined. The fluid model results in a system of nonlinear differential equations given by

$$\begin{aligned} \frac{dx_i(t)}{dt} &= \lambda_i - \theta_i x_i(t) - \min(c_i x_i(t), \alpha_i \mu_i (\eta_i x_i(t) + y_i(t)) + (1 - \alpha_k) \mu_k (\eta_k x_k(t) + y_k(t))) \\ \frac{dy_i(t)}{dt} &= \min(c_i x_i(t), \alpha_i \mu_i (\eta_i x_i(t) + y_i(t)) + (1 - \alpha_k) \mu_k (\eta_k x_k(t) + y_k(t))) - \gamma_i y_i(t) \quad (8) \\ &\text{for } i = 1, 2 \text{ and } k = 3 - i \end{aligned}$$

where subscript i defines the type of the peer. Although these equations are complex and difficult to solve, the closed form solutions are obtained for some special cases. The model is applied to service differentiation and bandwidth diversity problems where it is assumed that peer leaves the system immediately after their download. The parameter α determines how a peer allocates its upload bandwidth between two classes of peers. In the study, it is shown that how the parameter α can be chosen so as to achieve a target quality of service ratio and minimize the highest download time between two classes of peers.

Another study addressing the effect of different links capacities on the performance of cooperative systems is given in [47]. In the model, it is assumed that there are two classes of bandwidths, low-rate and high-rate. The study claims that heterogeneity has a positive effect on the content distribution.

6.4 Analytical Comparison

We compare the results obtained through simulations with simple fluid model given in [43]. As shown in Figure 6.1 a-d, the simulation results agree well with solution of the deterministic model given in equation (2). Here, normalized number of seeds (or leechers) indicates the ratio of the number of seeds (or leechers) to the arrival rate λ . It is especially useful when simulations with different arrival rates are shown on the same graph. Equation (2) is limit model of Markov models for BitTorrent-like systems for large arrival rates. That is why λ is chosen relatively large in this particular simulation.

For the results given in 6.1.a and 6.1.b, the parameters are chosen as $\mu = 0.00125$, $c = 0.002$, $\gamma = 0.001$ all with the units of 1/min, $\theta = 0$ and $\lambda = 0.1$ with the units of peers/min, equivalently 1 peer per 10 minutes. Finally, the effectiveness parameter η is set to 1, which indicates that the uplink capacity of the leechers is fully used. In figures 6.1.c and 6.1.d, we have the same setting as the first experiment, except that we set $\gamma = 0.005$. The first departure rate $\gamma = 0.001$ stands for a system where the downloading

bandwidth is the bottleneck, namely $\gamma < \mu$. When the departure rate of seeds γ is set to 0.005, now the uploading bandwidth of the peers becomes the bottleneck.

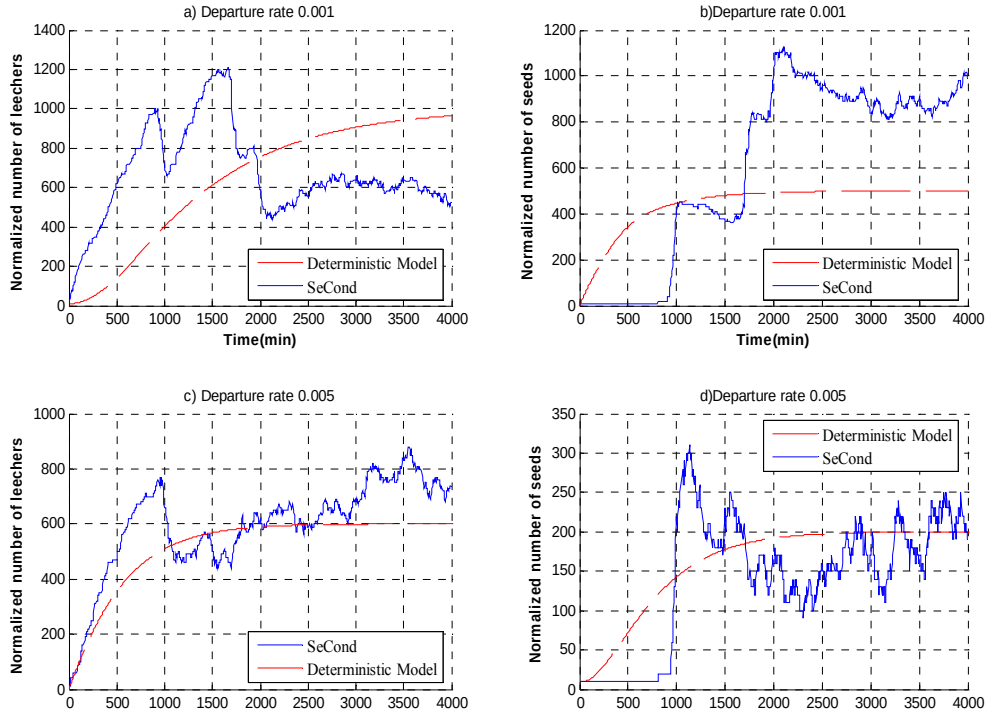


Figure 6.1. a-d) Normalized number of leechers and seeds as a function of time

In this chapter, we examined main analytical studies and their approaches on cooperative content distribution. All the studies verify that cooperative content distribution protocols scale well. Moreover, the analytical results coincide with the simulation results we present in chapter 5.

Chapter 7

CONCLUSION AND FUTURE WORK

This thesis has studied the problem of distribution of large sized data to large number of end systems in an efficient and effective way. We have started by discussing several requirements that should be satisfied by a well designed content distribution protocol:

- Handling large set of users at the same time
- Being able to operate under dynamic conditions
- Being easy to deploy
- Supporting heterogeneity of peers

We have first examined the existing solutions on content distribution and found that multicast and infrastructure-based solutions do not have the ability to meet the requirements given above. Similarly, although erasure codes have been used to transfer bulk data, it is hard to deploy them in a large scale dynamic environment. On the other hand, peer-to-peer cooperative protocols let the end systems decide the source for the data they strive to achieve locally. This locality increases the utilization of the system resources and enables parallel downloading. Two important examples of this category are BitTorrent and Slurpie. However, the control messages for the propagation of available block information among the peers constitute large network overhead in both protocols.

Our proposal in this thesis, SeCond, is a peer-to-peer protocol addressing the distribution of large sized content to large number of end systems in an efficient and effective manner. In order to support heterogeneity of peers, ease of deployment, scalability, and adaptivity to dynamic peer arrivals/departures, and also to increase the utilization of the system resources, it employs mechanisms such as adjusting protocol

parameters according to the bandwidth usages dynamically. Moreover, SeCond peers continuously update their views to maximize the cooperation among themselves. Another distinguishing feature of SeCond is the deployment of epidemic dissemination scheme for state propagation of available blocks and initiation of block transmissions. In SeCond, at each predefined time interval, each peer sends gossip messages to inform other peers about newly downloaded blocks.

In order to analyze the performance of SeCond, we have developed a discrete event based simulator modeling SeCond in Java programming language. Performance metrics such as download time, peer arrival patterns, load imposed on the original file source; scalability and communication overhead have been studied in terms of protocol parameters. Smart peer policy and parallel upload limit parameters are the key features of our system. We examine these features together with the gossip interval parameter. Although increasing the gossip interval parameter reduces the overhead, it decreases uplink/downlink utilization. It is shown that smart seed policy alleviates the negative effect of increasing the gossip interval on the utilization. Likewise the smart peer policy reduces the average download time of the file.

In addition to the SeCond simulator, in order to make a comparison with a real life application, a discrete event based simulator has been implemented for the BitTorrent model. A comprehensive performance evaluation of the protocol and its comparison with the BitTorrent system model have been accomplished for a wide range of scenarios. Performance analysis results include scalability analysis for different arrival/departure patterns, flash-crowd scenario, overhead analysis and fairness ratio. SeCond peers download the file faster compared to BitTorrent peers for most of the scenarios and the protocol is as fair as BitTorrent although it has no explicit strategy addressing free-riding. We show that SeCond is a scalable and adaptive protocol which takes the heterogeneity of the peers into account. Furthermore, comparative overhead analysis of SeCond and BitTorrent protocols have been carried out. It is shown that SeCond has comparatively lower protocol overhead than

BitTorrent which is one of the major reasons behind the lower file download time of SeCond peers.

Finally, the analytical approaches investigating the effectiveness of cooperative peer to peer content distribution protocols are studied. It is shown that results inferred from the analytical models agree well qualitatively with the SeCond's simulation outcomes. We have also compared the results obtained through simulations with a fluid model. We observe from the comparison that simulation results are consistent with the fluid model.

As future work, we consider enhancing our protocol with a dynamic gossip interval adjustment mechanism to alter the rate of gossip messages. We have compared the simulation results with a fluid model. However, in the comparison, the arrival rate of the peers is assumed to be constant. This comparison may be extended for the arrival patterns: increasing, decreasing, and triangular. Furthermore, more precise stochastic models accounting various arrival patterns can be sought. In the long term, we plan to implement SeCond's peer software to deploy on the Internet in order to evaluate the performance of the protocol in real network conditions.

BIBLIOGRAPHY

- [1] L. Roberts, T. Merril, "Toward a Cooperative Network of Time-Shared Computers", *AFIPS Fall Conference*, Oct 1966
- [2] www.napster.com
- [3] <http://gnutelliums.com/>
- [4] <http://kazaa.com/>.
- [5] <http://www.itu.int/>
- [6] B. Cohen, "Incentives build robustness in bittorrent" *P2P Economics Workshop*, Berkley, CA, 2003
- [7] Rob Sherwood, Ryan Braud, Bobby Bhattacharjee, "Slurpie: A Cooperative Bulk Data Transfer Protocol", *IEEE Infocom 2004*
- [8] <http://www.akamai.com>
- [9] Squid, <http://squid-cache.org>
- [10] S. Deering and D. Cheriton, "Multicast Routing in Datagram Internetworks and Extended LANs," in *ACM Transactions on Computer Systems*, May 1990.
- [11] Y.-H. Chu, S. G. Rao, and H. Zhang, "A Case for End System Multicast," in *Proceedings of ACM SIGMETRICS*, June 2000.
- [12] S. Banerjee, B. Bhattacharjee, and C. Kommreddy, "Scalable Application Layer Multicast," in *Proceedings of ACM SIGCOMM*, 2002.
- [13] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "Application-level multicast using content-addressable networks," in *Proceedings of 3rd International Workshop on Networked Group Communications*, Nov. 2001.
- [14] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron, "SCRIBE: A large-scale and decentralized application-level multicast infrastructure," *IEEE Journal on Selected Areas in communications (JSAC)*, 2002.
- [15] Van Renesse, R., "Why bother with CATOCS?," *ACM SIGOPS Oper. Syst. Rev.* 28, 1 (Jan. 1994), 22–27, 1994..

- [16] Cheriton, D. R. and Skeen, D., "Understanding the limitations of causally and totally ordered communication," *ACM SIGOPS Oper. Syst. Rev.* 27, 5 (Dec. 1993), 44–57, 1993.
- [17] Birman, K., "A response to Cheriton and Skeen's criticism of causal and totally ordered communication," *ACM SIGOPS Oper. Syst. Rev.* 28, 1 (Jan. 1994), 11–21, 1994.
- [18] Lidl, K., Osborne, J., and Malcome, J., "Drinking from the firehose: Multicast USENET news," In *Proceedings of USENIX Winter* (Jan.). USENIX Assoc., Berkeley, CA, 33–45, 1994.
- [19] Paul, S., Sabnani, K. K., Lin, J. C., and Bhattacharyya, S., "Reliable Multicast Transport Protocol," *IEEE J. Sel. Areas Commun.* 15, 3 (Apr.), 1997.
- [20] Floyd, S., Jacobson, V., Mccanne, S., Liu, C.-G., and Zhang, L., "A reliable multicast framework for light-weight sessions and application level framing," *SIGCOMM Comput. Commun. Rev.* 25, 4 (Oct.), 342–356, 1995.
- [21] Birman, K., Hayden, M., Ozkasap, O., Xiao, Z., Budiu, M., and Minsky, Y., "Bimodal Multicast," *ACM Trans. Comput. Syst.* 17, 2 (May), 41–88, 1999.
- [22] Danny Bickson, Dahlia Malkhi, "The Julia Content Distribution Network," 2nd Usenix Workshop on Real, Large Distributed Systems (WORLDS '05), Dec. 2005.
- [23] Coby Fernandess, Dahlia Malkhi, "On Collaborative Content Distribution Using Multi-Message Gossip," Twentieth IEEE International Parallel and Distributed Processing Symposium (IPDPS 2006), Greece, April 2006
- [24] J. Byers, J. Considine, M. Mitzenmacher, and S. Rost, "Informed content delivery across adaptive overlay networks," in *Proceedings of the ACM SIGCOMM 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM-02)*, ser. Computer Communication Review, J. Wroclawski, Ed., vol. 32, 4. New York: ACM Press, Aug. 19–23 2002, pp. 47–60.
- [25] J. W. Byers, M. Luby, M. Mitzenmacher, and A. Rege, "A digital fountain approach to reliable distribution of bulk data," in *Proceedings of the ACM SIGCOMM*

- '98 conference on Applications, technologies, architectures, and protocols for computer communication. ACM Press, 1998, pp. 56–67
- [26] D. Kostić, A. Rodríguez, J. Albrecht, and A. Vahdat, “Bullet: High bandwidth data dissemination using an overlay mesh,” in *Symposium on Operating Systems Principles (SOSP)*, 2003.
- [27] Christos Gkantsidis, Pablo Rodríguez. “Network Coding for Large Scale Content Distribution”, *IEEE Infocom 2005*
- [28] BitTorrent, <http://www.bittorrent.com>
- [29] Pablo Rodríguez, Ernst W. Biersack, “Dynamic Parallel Access to Replicated Content in the Internet”. *IEEE/ACM Transactions on Networking* 2002
- [30] E. Adar and B. A. Huberman, “Free Riding on Gnutella”, *First Monday*, 5(10), October 2000.
- [31] D. Hughes, G. Coulson, and J. Walkerdine. “Free riding on Gnutella Revisited: the Bell Tolls?,” *IEEE Distributed Systems Online*, 6(6), 2005
- [32] S. Saroiu, P. K. Gummadi, and S. D. Gribble, “A Measurement Study of Peer-to-Peer File Sharing Systems,” In *Proc. Multimedia Computing and Networking 2002 (MMCN'02)*, San Jose CA, USA, January 2002.
- [33] A. Bharambe, C. Herley and V. Padmanabhan, “Analyzing and Improving a BitTorrent Network's Performance Mechanisms,” *Proc. InfoComm 2006*.
- [34] M. Izal, G. Urvoy-Keller, E. W. Biersack, P. Felber, A. Al Hamra, and L. Garcés-Erice, “Dissecting bittorrent: Five months in a torrent's lifetime,” in *Passive and Active Measurements (PAM)*, 2004.
- [35] J. Pouwelse, P. Garbacki, D. Epema, and H. Sips, “The Bittorrent P2P File-sharing System: Measurements and Analysis,” In *4th International Workshop on Peer-to-Peer Systems (IPTPS'05)*, Feb. 2005.
- [36] A.J. Demers et al., “Epidemic Algorithms for Replicated Database Maintenance,” *Proc. 6th Ann. ACM Symp. Principles of Distributed Computing*, ACM Press, 1987, pp. 1-12.

- [37] R. van Renesse, Y. Minsky, and M. Hayden, "A Gossip-Style Failure Detection Service," *Middleware 98:IFIP Int'l Conf. Distributed Systems and Platforms and Open Distributed Processing*, N. Davies, K. Raymond, and J. Seitz, eds., Springer, 1998, pp. 55-70.
- [38] R. van Renesse, K.P. Birman, and W. Vogels, "Astrolabe: A Robust and Scalable Technology for Distributed Systems Monitoring, Management, and DataMining," *ACM Trans. Computer Systems*, vol. 21, no. 2, 2003, pp. 164-206.
- [39] www.junit.org
- [40] G. de Veciana and X. Yang, "Fairness, incentives and performance in peer-to-peer networks," in *the Forty-first Annual Allerton Conference on Communication, Control and Computing*, 2003
- [41] X. Yang and G. de Veciana, "Service capacity of peer to peer networks," in *Proceedings of IEEE INFOCOM*, 2004.
- [42] L. Massoulie and M. Vojnovic, "Coupon replication systems," in *Proc. ACM SIGMETRICS*, 2005.
- [43] D. Qiu and R. Srikant, "Modeling and performance analysis of bittorrent-like peer-to-peer networks," in *Proc. ACM SIGCOMM*, 2004.
- [44] Fan Bin, Dah-Ming Chiu and John C.S. Lui. "Stochastic Differential Equation Approach to Model BitTorrent-like P2P Systems," IEEE International Conference on Communications (ICC 2006).
- [45] L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding, X. Zhang, "Measurements, Analysis, and Modeling of BitTorrent-like Systems", IMC 2005.
- [46] F. Clvenot-Perronnin and K. R. P. Nain, "Multiclass p2p networks: Static resource allocation for service differentiation and bandwidth diversity," in *Performance*, 2005.
- [47] G. B. F. Lo Piccolo, G. Neglia, "The effect of heterogeneous link capacities in bittorrent-like file sharing system," in *IEEE International Workshop on Hot Topics in Peer-to-Peer Systems (HOT-P2P)*, Oct. 2004.

VITA

Ali Alagöz was born in Ankara, Turkey on April 13, 1980. He received his B.Sc. degree from Mathematics Department of Middle East Technical University, Ankara, Turkey in 2004. From January to September of 2004, he worked as software specialist in Odesa Information Technologies, Ankara. From September 2004 to August 2006, he worked as a teaching and research assistant in Koç University, Istanbul, Turkey and participated in the “TUBITAK-COST Action: 279, Analysis and Design of Multiservice Networks Supporting Mobility, Multimedia, Internetworking” project. He presented a technical report at COST-279 meeting Action’05, Antalya, Turkey, a poster paper at ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC 2005), July, 2005, Las Vegas, USA, and a conference paper at ISCN’06, Istanbul, Turkey. He has currently one conference paper submission. At the time of press, he had a Ph.D. admission from the Software Engineering Group at the Department of Computer Science in the Faculty of Electrical Engineering, Mathematics and Computer Science at the University of Twente.