Journal of Network and Computer Applications 32 (2009) 666–683

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

Principles and performance analysis of SeCond: A system for epidemic peer-to-peer content distribution

Oznur Ozkasap *, Mine Caglar, Ali Alagoz [1]

Department of Computer Engineering, Koc University, Rumeli Feneri Yolu, Sariyer 34450, Istanbul, Turkey

ARTICLE INFO

Article history:
Received 10 September 2007
Received in revised form
9 June 2008
Accepted 5 July 2008

Keywords:
Peer-to-peer
Epidemic
Content distribution
BitTorrent-like systems
Fluid model

ABSTRACT

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 manner. In contrast to prior work, it employs a self-organizing epidemic dissemination scheme for state propagation of available blocks and initiation of block transmissions. In order to exploit heterogeneity of peers, enhance the utilization of system resources and for the ease of deployment, scalability, and adaptivity to dynamic peer arrivals/departures, we propose mechanisms for adjusting protocol parameters dynamically according to the bandwidth usages. We describe design and analysis details of our protocol SeCond. Comprehensive performance evaluations and comparison with the BitTorrent system model have been accomplished for a wide range of scenarios. 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, and communication overhead. We show that SeCond is a scalable and adaptive protocol which takes the heterogeneity of the peers into account. The protocol is as fair as BitTorrent although it has no explicit strategy addressing free-riding. We also illustrate the applicability of an analytical fluid model to the behavior of SeCond.

1. Introduction

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. However, distribution of popular large content, such as software packages and popular movie files, may be very obstructive due to the bottleneck that may happen at the content source. As soon as a popular file is released, a flash-crowd scenario is expected in which many users strive to achieve a copy of the file suddenly. It is well known that traditional client–server based solutions are not appropriate in such a flash-crowd scenario. Even if it is assumed that the file server can handle large number of requests simultaneously, the time period required for a peer to obtain a full copy of the file increases linearly with respect to the system size. On the other hand, several P2P file sharing systems create a platform where users find many files to transfer, but generally they do not intend to disseminate a popular file or alleviate the load on the original source. Main goal of this class of P2P applications is locating sources for the desired files, not increasing number of the copies of the file as fast as possible. Hence, a well designed protocol addressing content distribution 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

* Corresponding author. Tel.: +90 212 338 1584; fax: +90 212 338 1548.
E-mail addresses: oozkasap@ku.edu.tr (O. Ozkasap), mcaglar@ku.edu.tr (M. Caglar), a.alagoz@ewi.utwente.nl (A. Alagoz).

[1] This work has been done when the author was at Koc University.

1084-8045/$ - see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.jnca.2008.07.005

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.

In this study, a new cooperative protocol SeCond, a *S*ystem for *E*pidemic P2P *Con*tent *D*istribution, is proposed for P2P distributed systems. As the novel features for content distribution, it consists of an efficient state propagation mechanism inspired by the epidemic methods, and an adaptive mechanism to utilize system resources of heterogeneous peers. SeCond's target is to distribute a large content to many users simultaneously in an efficient and effective way. We show that it distributes the load among all peers in the system. Cooperation among system participants is based on a P2P content trading paradigm (Cohen, 2003; Sherwood et al., 2004). That is, while peers are downloading blocks of the file, they also act as a source for the blocks they have downloaded. However, in contrast to prior work, informing other peers about available blocks is realized via epidemic dissemination. Moreover, views of the peers are continuously updated in order to increase utilization of the system resources. The epidemic approach is beneficial when managing dynamic peer arrivals and departures, is easy to implement, and inexpensive to run. In case of dynamic user arrivals and departures, it does not require an extra effort for reconfiguration. Most importantly, the approach is inherently scalable. A preliminary analysis of SeCond is given in Alagoz et al. (2007).

Using the discrete event simulation models, comprehensive performance evaluations of SeCond and its comparison with the BitTorrent system 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. SeCond peers utilize the system resources efficiently, achieve faster download times 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 the protocol.

The paper is organized as follows. Related work and comparison with our study are described in Section 2. Details of the SeCond protocol is explained Section 3. Simulation model description and extensive performance results are given in Sections 4 and 5, respectively. Section 6 discusses the analytical framework applicable to the behavior of SeCond. Finally, Section 7 includes concluding remarks and states the future work.

2. Related work

The general problem of getting popular content from heavily loaded servers is well studied. Some of the studies addressing content distribution require infrastructure support, which is usually an expensive supply. An *infrastructure-based* solution (Akamai) exploits mirroring or replication of the server. Along the same ideas, Squid stores the requested Internet objects on a system closer to the client sites in order to reduce the load on the server as well as the client return time. Other approaches can be classified under the title of *cooperative content distribution* solutions, which include *multicast*, *erasure codes* and *mesh cooperative architectures* as depicted in Fig. 1. In cooperative content distribution, every node in the system may be involved in the distribution. For instance, in multicast solutions, nodes other than the source are also involved in the relay of the content throughout the network as well as in supporting message loss recovery (Birman et al., 1999). On the other hand, erasure codes have been used actively to transfer bulk data (Byers et al., 2002, 1998; Kostic et al., 2003) where collaboration and reconciliation are required among the nodes. Mesh cooperative solutions are P2P protocols that create a mesh and aim to replicate the content rapidly. Fast replication leads to an increase in the number of seeds and hence improves performance.

Our study belongs to the class of mesh cooperative solutions. P2P systems create a platform where people find many files to transfer, but generally they do not intend to disseminate a popular file. Popular file sharing applications (KaZaA; Gnutella; E-donkey) are good examples of this kind of systems where peers are organized together so that they can exchange different files. However, the main goal of these applications is locating sources for the desired files. 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 (Cohen, 2003) and Slurpie (Sherwood et al., 2004). They 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 (Rodriguez and Biersack, 2002).

BitTorrent is a popular P2P file distribution protocol deployed on the Internet for several years. The idea behind BitTorrent is organizing the peers in such a way that the load of the seed is distributed to the entire system. We describe the principles of BitTorrent model in Section 4.1. In particular, it deploys a rate based tit-for-tat mechanism to avoid free riding (Adar and Huberman, 2000). However, as stated in Bharambe et al. (2006), that policy is not effective in preventing unfairness especially for heterogeneous systems. A fundamental drawback of BitTorrent is the waste of bandwidth and the possibility of causing network congestion. Since the tracker returns a list of randomly selected

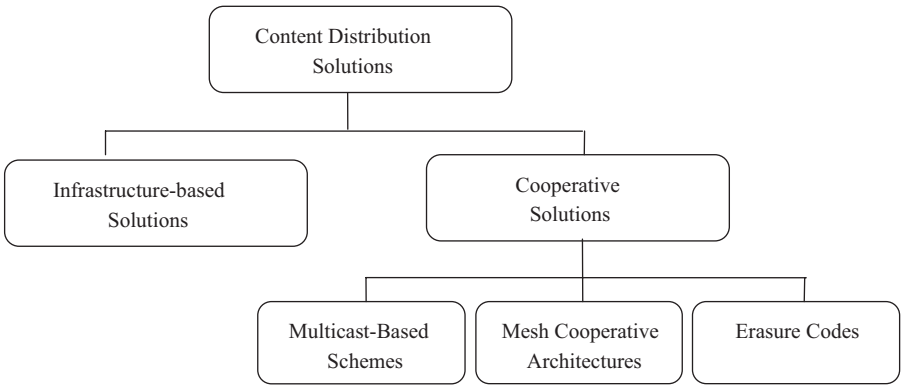

Fig. 1. Classification of content distribution solutions.

peers among the currently registered ones to a new peer, it is likely that some of these peers are physically far away from the new peer. Thus, transmitting the same content multiple times through wide area links would lead to redundancy in data traffic, and as a result waste of bandwidth and possible congestion. In this context, a recent study proposes an approach named biased neighbor selection to enhance BitTorrent traffic locality (Bindal et al., 2006). In addition, Guo et al. (2007) focuses on the limitations of poor service availability, fluctuation in download performance and unfair service to peers. Another problem with BitTorrent-like systems is that the control messages for the propagation of available block information among the peers may constitute large network overhead.

Slurpie is a P2P protocol based cooperative data transfer and designed for flash crowd scenarios (Sherwood et al., 2004). Major goals of the protocol are reducing the download times of large popular files for clients and reducing the load on the servers. The underlying idea of Slurpie is exploiting resources of clients during dissemination of the file as it is intended in BitTorrent. 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.

Another recent protocol for collaborative download of content and avoiding free riding is 2Fast (Garbacki et al., 2006) which is used within the Tribler (Pouwelse et al., 2007) social-based P2P file-sharing system. Its novel feature is to make use of social groups of peers that collaborate in downloading a file for the benefit of a single group member. As opposed to content trading model in BitTorrent-like systems, 2Fast model is based on bandwidth trading. In content trading, contents or files are traded between peers to impose fairness of sharing, whereas, in bandwidth trading, bandwidth resources are traded among peers in groups rather than content.

Julia content distribution protocol (Bickson and Malkhi, 2005) is another study addressing distribution of large files over P2P networks. Julia is a network aware protocol. It aims to reduce the overall network overhead and incur a balanced load on the network during dissemination. As the download progresses, the nodes gather statistics about the network. This knowledge is used to initiate transmissions from the closer nodes. Through simulations, it is shown that Julia reduces the network overhead and achieves slightly slower average finishing times relative to BitTorrent.

An epidemic based distribution is studied in Fernandess and Malkhi (2006). A gossip protocol that disseminates k blocks to the entire 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. Another protocol using epidemics in order to disseminate information and manage the membership is Newscast (Jelasity and van Steen, 2002). In Newscast, caches of peers hold neighbors' descriptors. A descriptor for a peer consists of peer address, creation time of that descriptor and information of the corresponding peer. At predefined time intervals, each peer creates its own descriptor. Since cache has a fixed size, that descriptor is replaced with the oldest one. A peer is selected from the cache and an exchange of states occurs with the selected peer. Then views are merged and old entries are removed to keep cache fresh. By this way, dynamic joins and leaves are handled easily by the protocol. However, this protocol does not directly target distribution of large files especially in flash crowd scenarios.

Our protocol differs from the prior work mentioned above, and in particular from BitTorrent-like systems, in a number of ways. Firstly, since the main goal of the protocol SeCond is to make copies of the blocks available in the system as fast as possible, we limit the number of parallel downloads. If the uplink bandwidth of the source is split among too many peers, this increases the time for the peers to download an entire block which can then be served to others. It may also cause the users to abort the download due to unacceptable download rates. Thus, file dissemination to the entire system may take longer. On the other hand, letting small number of peers to download simultaneously may lead to the 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 adjusted by each peer locally according to the utilization of uplink capacities. 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. Another point that SeCond differs from BitTorrent-like systems is the propagation of states during the dissemination of the file. In contrast to BitTorrent's way of multicasting information about the available blocks whenever a new block is obtained, we deploy a gossiping (epidemic) mechanism to propagate peers' state information. Finally, instead of forcing peers upload to peers from where they can download, each peer utilizes its upload bandwidth independently. This leads to an increase in utilization of system sources. In return, it enhances the performance of the system. However, peers give priorities to peers from where they have downloaded more. The details of this procedure are given in the description of our model in the next section.

3. System description

SeCond's P2P network consists of peers and the index server agents. State propagation, view construction/update and upload/download decisions are the major properties of our system. In this section, we describe these agents, algorithms and properties of SeCond protocol.

3.1. Model of index server agent

The index server is the entrance point to the system which helps the peers to find each other. To initiate a file download, a peer registers 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. Since some peers may leave the system without notification or may not be willing to add the new peer to their mesh, the length of the returned list is greater than the *view size* of the peers. To update the 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 view is not active, it reports this situation as well.

The index server keeps two sets of information, namely Peer-Base and Content-Base. Peer-Base stores the data about the registered peers. This 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 information about the shared content (e.g. size of the file, number of blocks, publisher of the file).

There are four types of events associated with the index server. These are Registration Request Reception, Peer Request Reception, State Report Reception, and Dead Peer Detection Round. Algorithms of these events are summarized in Fig. 2.

Registration Request Reception: Reception of a registration request
-**View Selection** (List Size) // Function selecting peers randomly from the Peer-Base -Send selected list to requester peer -Add requester to Peer-Base
Peer Request Reception: Reception of a request for addresses of new peers
- Get number of the requested peers - **View Selection** (Number of requested peers) - Send selected list to requester peer
State Report Reception: Reception of a message stating the sender is alive
- Update the Report Time of the peer // Time of the last state report msg received from peer
Dead Peer Detection Round: Detection of dead peers registered to system
for all registered peers in the peer base If (Current time – Report Time of the peer > State Report Interval) - Remove the peer from the Peer-Base

Fig. 2. Events and algorithms for the index server.

3.2. Model of peer agent

A peer takes an important part in the distribution of the file. It keeps track of the peers in its view, and acts as a source for the blocks of the file it has downloaded. A peer may join the P2P network anytime and leave the system without a notification. A peer communicates with two sets of peers stored in *Gossip Receiver List* and *Gossip Sender List*. The first one includes the peers to which available blocks are uploaded. The latter consists of the peers from where the blocks are downloaded. Although these lists may include common peers, they may not be identical. The maximum size of the gossip receiver and gossip sender lists, *maximum connection size*, is a protocol parameter. Initially, the peers in the gossip receiver list are determined with the help of the list returned by the index server after registration.

A peer keeps the following information for each peer in its gossip sender and gossip receiver list:

Gossip Sender List: ⟨Agent id, Address, Peer state, Time⟩
Gossip Receiver List: ⟨Agent id, Address, Gossip order, Time⟩

Agent id states the order of the peer agent in the list. *Address* keeps the Internet address of the peer. For the gossip senders, states of the peer are also held. *Peer state* keeps a list of block ids downloaded by the peer. Density decisions of the blocks are given locally with respect to state information. *Time* field is used to determine the time of the last incoming block transmission from the given peer. For the gossip receiver list, *time* entry states time 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. *Gossip Order* keeps the index of the block (download order of the block) the peer is informed most recently.

The peers are categorized as *leechers* and *seeds* in accordance with the role they play in the dissemination. A peer with a complete copy of the file is said to be a seed. The peers striving to obtain a full copy of the file are called 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.

There are six types of events associated with the peer. These are Gossip Round, Gossip Message Reception, Block Request Reception, Gossip Sending Request, Gossip Receiving Request, and Completion of Block Download. Descriptions and algorithms of these events are summarized in Fig. 3. In the following sections, details on state propagation, view construction/update and upload/download decisions are described.

3.3. State propagation

The *state* of a peer stands for the blocks that have been downloaded recently. A peer propagates its state to the peers in the gossip receiver list. The 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 and propagates its state information. 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*. After selection of the gossip destinations, a gossip message is constructed per target. A gossip message carries the ids of the downloaded blocks that the receiver may not be aware of. It contains the ids of the recent blocks downloaded since the last time a gossip is sent to that particular receiver. The aim is to reduce the size of a gossip message, hence the protocol overhead.

The example in Fig. 4 illustrates the gossip message construction process for a destination peer 2 and update of its gossip order information. As shown in the figure, downloaded blocks are indexed according to their download times. During construction of a gossip message, the block about which destination peer is informed most recently is determined by using the gossip order entry held 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.

Upon receiving a gossip message, the receiver peer looks for the blocks that it does not have but the sender has. If there exist such blocks, the receiver peer requests the missing ones from the sender. It should be noted that sometimes requesting blocks may not initiate block downloads. Note that gossiping is not necessarily reciprocal.

3.4. View construction and update

Cooperation among peers should be maximized in order to enhance the performance of content distribution system. Some of the peers in gossip receiver and sender lists may have left the

Gossip Round: Periodic gossip dissemination

-Determine uplink utilization
-Apply smart-peer policy
-Select *fan-out* different peers randomly from the gossip receivers list.

Gossip Message Reception: Reception of a gossip message

-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) and transmission
If (Request is accepted)
-Initiate the block transmission from the source

Block Request Reception (Request Message): Reception of a block upload request

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

Gossip Sending Request: Reception of request to be added to gossip senders listt

If(Sender List Size < Max. Connection Size)
-Send an acceptation message and add the sender to gossip message senders
else
- **Passive Gossip Sender Selection**

Gossip Receiving Request: Reception of request to be added to gossip receivers list

If(Receiver List Size < Ma
-Send an acceptance message and add the sender to gossip message receivers list
else
- **Passive Gossip Receiver Selection**

Completion of Block Download: Completion of a block transmission

-Update state information
If (Sender has any other block in which the peer is interested)
- **Block Request Construction** (Gossip Sender) and transmission

Fig. 3. Events and algorithms for the peer.

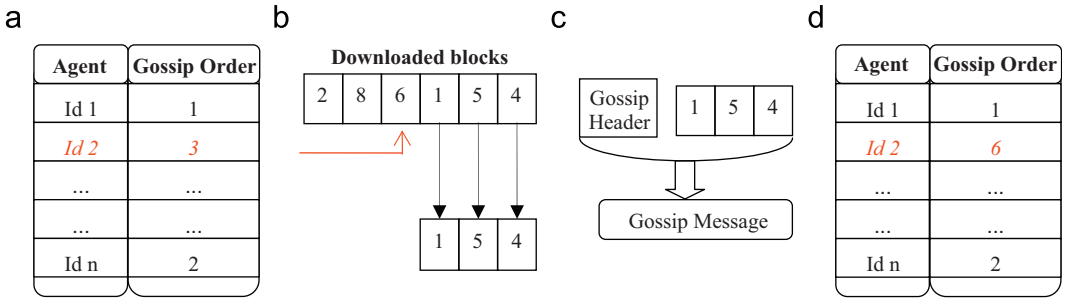

Fig. 4. Gossip message construction example for the peer with id 2: (a) index (3) of the block that the peer with id 2 has been informed most recently is determined, (b) blocks downloaded after block 6 (with index 3) are selected to be added to gossip message, (c) gossip message is formed by adding header to gossip data and (d) gossip order, index of the block (index 6) the destination peer is informed most recently, is updated.

system, or almost all the block transmissions among the peers might have been completed which could leave the link capacities of the peers idle. In such a case, a peer is forced to download blocks from the primary seed or stays idle. To avoid this situation, the peers update their views during dissemination.

Following the registration process, index server returns a list, to the newly joined peer, which consists of active peers (both seeds and leechers) randomly selected among the registered ones. The newly joined peer sends a request message to the peers in the returned list indicating that it wants to add the counterpart to the

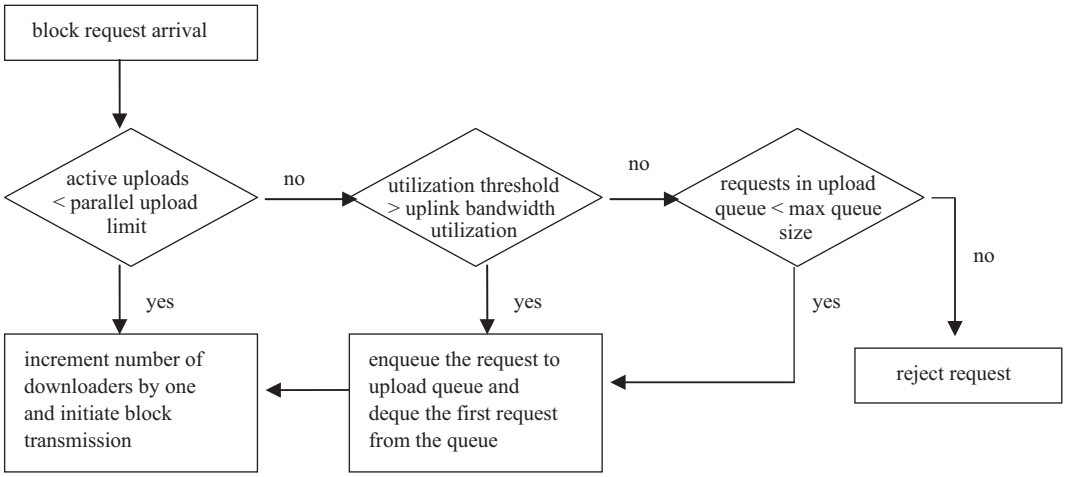

Fig. 5. Illustration of upload decision process.

gossip receiver list. If there is free space in the gossip sender list of the receiver, the new peer is directly added to the gossip sender list of the receiver. 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 to the new peer.

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. Both the passive peer interval and minimum connection size are protocol parameters.

3.5. Download/upload decisions

In the beginning, the primary seed is the only peer that has all blocks of the file. The number of peers owning blocks of the file increases as the dissemination progresses. 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. When a peer receives a gossip message from another peer, in addition to updating the state information of the gossip sender as described in Section 3.3, 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 missing block ids which are the least duplicated blocks, hence have the lowest density in the system. Peers hold the state information of the peers in the gossip sender list. The decision for the density of the blocks is given locally by the help of this information. The sender prefers to forward the least uploaded block among the requested ones.

Requesting some blocks from a source does not guarantee the initiation of a 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*. However, if the parallel uploading limit is reached and another upload request is received, the source checks the utilization of its uplink capacity. If this utilization is greater than *utilization threshold* parameter of the protocol, incoming request is rejected. Otherwise, it lets the requester to download without taking the parallel upload limit into consideration. In a heterogeneous network, the peers have different uplink and downlink capacities. Adjusting the number of parallel uploads dynamically according to utilization helps exploiting the uplink bandwidth of all sources and in particular of those with larger capacities. Moreover, we deploy a queue not to refuse requests and to increase utilization of the uplink capacities. The 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, the priorities may be given to peers that upload more to the owner of the queue. The flowchart for this decision process is given in Fig. 5.

The *smart peer policy* enables adjustment of the fan-out parameter f dynamically based on the current uplink utilization. When the uplink utilization of a peer is low, it will be likely to increment its fan-out value in the next gossip round. In general, 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.

4. Simulation model

We have developed discrete event simulation models for SeCond and BitTorrent protocols using Java JDK as the implementation platform. In order to verify the correctness of the simulation code, unit tests have been performed using the JUnit testing framework. Our simulations are based on application-layer models to be able to analyze protocol characteristics in particular for large peer populations. We consider access link bandwidths and model the block transfers as flows. As discussed in Eger et al. (2007), packet-level simulations are seldom used for P2P networks due to their high complexity. Differences between packet-level and flow-level (application-layer) simulation models for BitTorrent-like P2P systems are discussed and comparison is performed with the analytical models in Eger et al. (2007). The findings show that the results for the flow-level simulations of BitTorrent are near to the optimal values meaning that the flow-level protocol model works efficiently.

Our simulation model uses the uplink and downlink bandwidth of peers to calculate the delays associated with the transmission of data blocks. The delays are calculated by considering the number of flows using the uplink of the sender or downlink of the receiver. For scalability analysis, we have performed simulations for various population sizes up to 8000 peers. One reason that we need to limit the population size with

8000 peers is the cost of the delay computations for each data block. We performed our simulations on a Pentium-4 2.4 GHz, 1 GB RAM system. Each individual simulation is repeated five times and averages of the runs are reported in the performance results.

We have also made some simplifications regarding the underlying network model. These are needed to tackle the computational complexity of the simulation model in particular for large-scale scenarios. Similar simplifications were also introduced in Bharambe et al. (2006). Firstly, our model differentiates between control and data messages and assumes that the propagation delay is not applicable to the control messages. The control messages are small in size and exchanged among neighbors for requesting blocks. Therefore, the control messages are transmitted directly to the receivers. The incentive is that the file download time is mainly determined by the data block transmission delays. Thus, this simplification would not have a considerable effect on our results. Another assumption is that the bottleneck during block transmission arises either on uploader (uplink of the sender) or on downloader (downlink of the receiver) side. To be precise, the block download time is determined by the upload capacity of the sender and the download capacity of the receiver. This is a reasonable and common assumption as discussed in Eger et al. (2007) considering the fact that majority of the peers are home users connected to the Internet with DSL or cable modems. We also consider a fluid model for connections which assumes that the flows through a link share the link bandwidth equally.

Next, we describe our simulation model for the BitTorrent protocol and default values for protocol parameters. Then, block dissemination in SeCond is illustrated with an example and default values for protocol parameters are described. After that, we provide our simulation settings.

4.1. Bittorrent model

In order to reflect BitTorrent's behavior accurately in the simulation environment, we have integrated the protocol properties specified and the default values used in the BitTorrent official deployment (BitTorrent) into our simulation model. Tracker algorithm, choking/unchoking policy, optimistic unchoking and block selection policy stated below are implemented.

Tracker algorithm: Tracker is the server responsible for keeping track of the registered peers. It returns a list of randomly selected peers among the currently registered ones. The size of the returned list (*list size* parameter) is set to 50 peers (BitTorrent). However, 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 (*max connection size* parameter) is not directly stated in the official BitTorrent specification, we limit this number to 55 which is used in the real protocol implementations we investigated.

Choking/unchoking policy: BitTorrent employs a *tit-for-tat* policy to avoid free riding. Performance of BitTorrent-like systems are directly influenced by the level of cooperation among peers, and tit-for-tat strategy encourages peers to upload. In particular, a peer uploads to the peers providing the best download rates. Selection process of the peers to upload is called as *choking/unchoking*. This policy is employed (via *choking interval* parameter) once every 10 seconds. At every choking/unchoking round, a peer determines four 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 that 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 (via *optimistic choking interval* parameter) once every 30 s. 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.

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.

Key protocol parameters and their default values for the BitTorrent protocol in our simulations are summarized in Table 1. In relation to the tracker algorithm, the protocol parameters are *list size* and *maximum connection size*. For the choking/unchoking policy, the parameters are *choking interval* and *optimistic choking interval. Parallel upload limit* parameter defines the maximum number of concurrent uploads per peer and it is set to 5 as in the official deployment of the protocol. It includes the connection which is optimistically unchoked.

4.2. SeCond model

Simulation model for SeCond protocol is implemented based on the descriptions given in Section 3. An example illustrating block dissemination in SeCond is shown in Fig. 6. The list indicated as B[*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 [*upload capacity, 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 Fig. 6, shared file consists of 5 blocks. At round 1, peer 2 initiates transmission of block 1 from peer 1. Bandwidth initially reserved

Table 1
BitTorrent parameter settings

Parameter	Description	Default
List size	List size returned by tracker after registration	50
Maximum connection size	Maximum number of neighbors	55
Choking interval	Time interval at which choking policy is employed	10 sec
Optimistic choking interval	Time interval at which optimistic peer is selected	30 sec
Parallel upload limit	Maximum number of parallel uploads	5

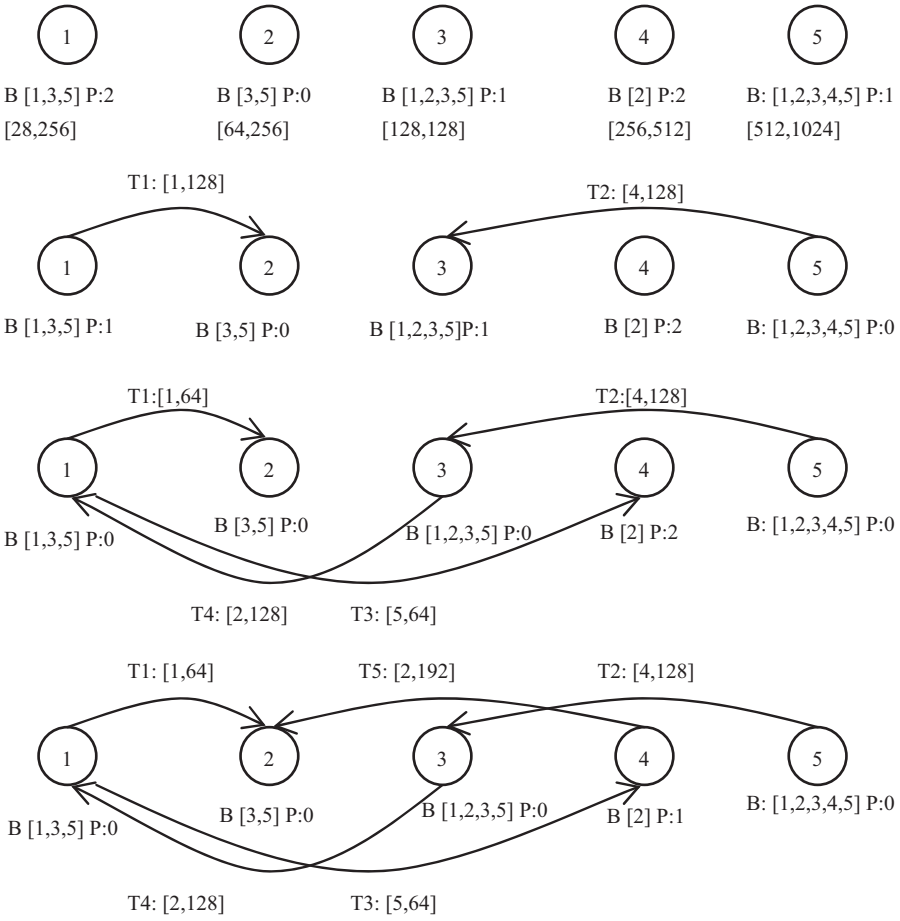

Fig. 6. Illustration of block dissemination in SeCond.

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.

Key protocol parameters and their default values for the SeCond protocol used in our simulations are given in Table 2. In parallel with the default values of the official BitTorrent deployment, *list size* returned by the index server to a peer is set to 50. Hence, the maximum size of the gossip receiver and gossip sender lists, *maximum connection size*, would be 50. The *fan-out* parameter of SeCond indicates the number of peers selected as gossip targets in a gossip round, and the *gossip interval* identifies the rate of gossip messages. *Parallel upload limit* parameter defines the maximum number of concurrent uploads per peer and it is set to 5 in consistent with the corresponding parameter in BitTorrent. However, recall that if the parallel upload limit of a peer is reached and a new upload request is received, the peer compares its uplink utilization with the parameter *utilization threshold*. The default value for the utilization threshold is 0.9 in our simulations. If the uplink utilization is greater than the utilization threshold, the request is rejected meaning that the peer's uplink is almost fully utilized. Otherwise, upload of the requested block is initiated as described in Section 3.5.

Table 2
SeCond parameter settings

Parameter	Description	Default
List size	List size returned by index server after registration	50
Maximum connection size	Maximum size of the gossip receiver and sender lists	50
Fan-out	Number of peers gossip msgs are sent to at each gossip round	3
Gossip interval	Time interval at which each peer gossips periodically	20 sec
Parallel upload limit	Maximum number of parallel uploads	5

4.3. Simulation settings

The shared file (content) is divided into blocks of size 256KB in order to enable parallel downloading or swarming (Rodriguez and Biersack, 2002). The size of the content is set to 200 MB (800 blocks) which indicates the typical size of a file such as software update, movie/audio clip and game patch. It is assumed that there is only one seed peer at the beginning which is the original content publisher. Whenever a peer completes downloading of a

Table 3
Heterogeneous bandwidth distribution

Downlink (Kbps)	Uplink (Kbps)	Fraction
784	128	0.2
1500	384	0.4
3000	1000	0.25
10 000	5000	0.15

block successfully, it starts sharing that block. Depending on the popularity of the published content, the arrival and departure patterns of peers may exhibit different characteristics. For this purpose, we have considered various arrival and departure patterns for the peers as described in Section 5.2.

A heterogeneous set of peers with different uplink and downlink capacities is considered. The uplink capacity of the primary seed is set to 6000 Kbps. For all other peers, the distribution of the uplink and downlink bandwidths is given in Table 3. This distribution is obtained from a real measurement study of the Gnutella network (Saroiu et al., 2002).

5. Performance results

In this section, our simulation results and analysis are given. We first analyze the key protocol parameters of SeCond. Then, we provide several comparative results with BitTorrent system model. These include the scalability analysis for different arrival/departure patterns, flash crowd scenario, overhead analysis and fairness ratio. The major metrics we study are as follows:

- *Average file download time:* The file download time of a peer is the time between the initiation and completion of the download. Average of file download times of all peers in the system is evaluated by 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:* The ratio of the uploaded data size to the downloaded data size by a peer.

5.1. Parameter analysis for SeCond

We examine smart peer policy, parallel upload limit and gossip interval parameters of our system. When the smart peer policy is activated, the peers reconfigure their fan-out parameter. In this case, increasing the fan-out parameter as necessary increases the probability of finding a peer for cooperation. Although decreasing the frequency of gossiping reduces the control message overhead, it decreases the average uplink/downlink utilization. Infrequent gossiping is also expected to increase the average file download time. Fig. 7 shows the effect of gossip interval on utilization both with a fixed fan-out throughout dissemination and with smart-peer policy where fan-out is adjusted. Clearly, smart peers increase the utilization. Likewise, the average download time of the file is improved with the smart peer policy as shown in Fig. 8 across various gossip interval values. Recall that smart peers increment their fan-out parameter by 1 with a probability $P(1-\text{uplink utilization})$ at each gossip round. Otherwise, the fan-out is decreased by 1. Based on the data of Figs. 7 and 8, the improvement resulting from smart peer policy is quantified in Fig. 9, which is more visible as the gossiping becomes less

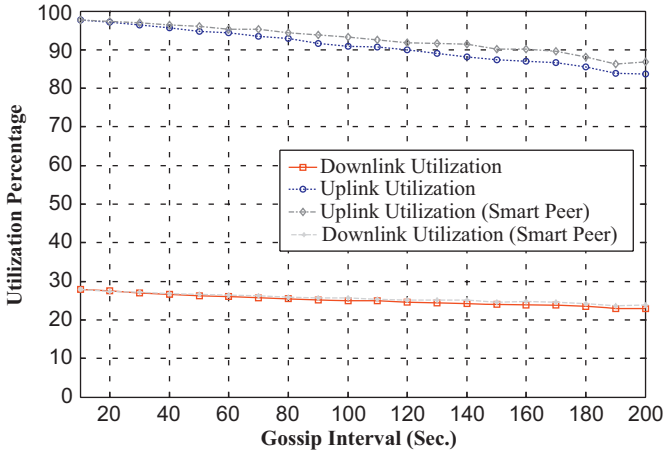

Fig. 7. Effect of gossip interval on uplink/downlink utilizations.

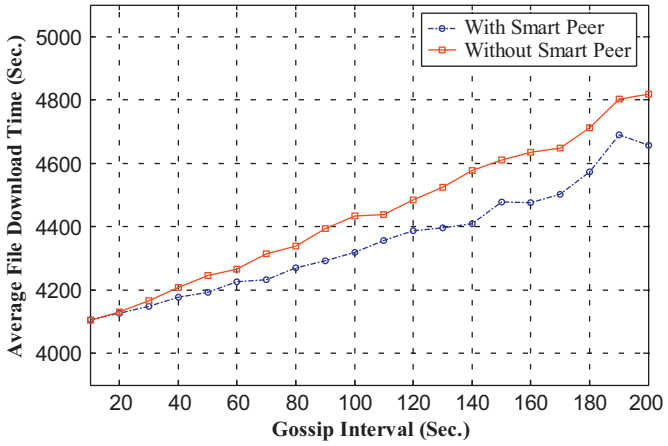

Fig. 8. Effect of gossip interval on average file download time.

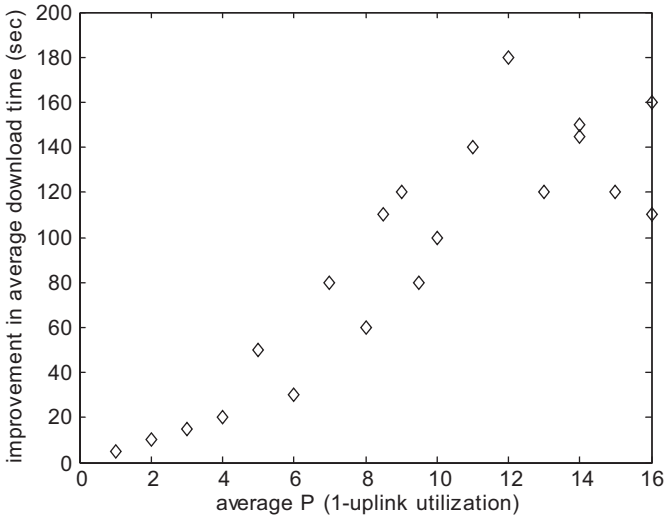

Fig. 9. Improvement in average download time as a function of average P(1-uplink utilization).

frequent. In this figure, improvement in average download time is plotted against average probability obtained from uplink utilization percentage without smart peer policy.

Fig. 10 shows that increasing parallel upload limit has a positive effect on the utilization of downlink capacities up to a

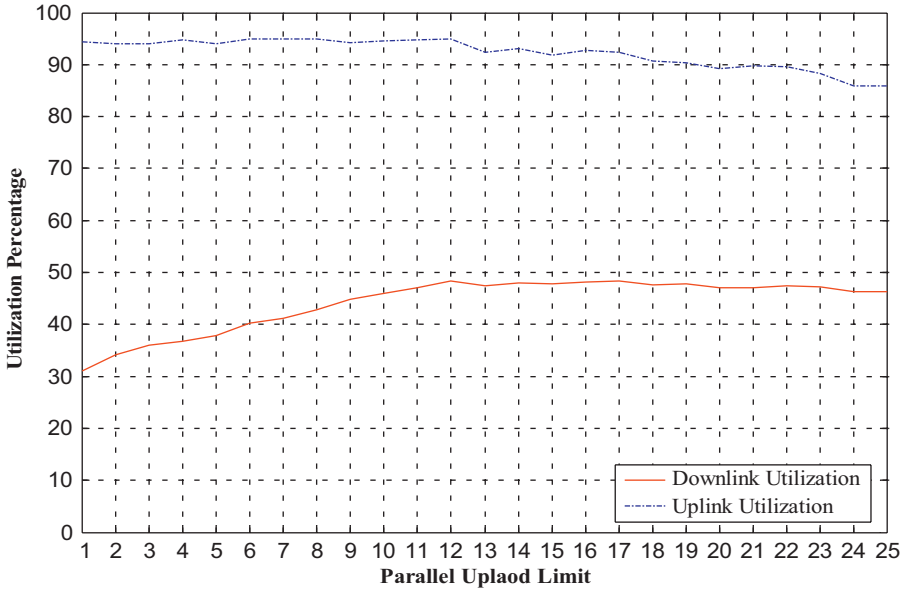

Fig. 10. Parallel upload limit vs. downlink and uplink utilizations.

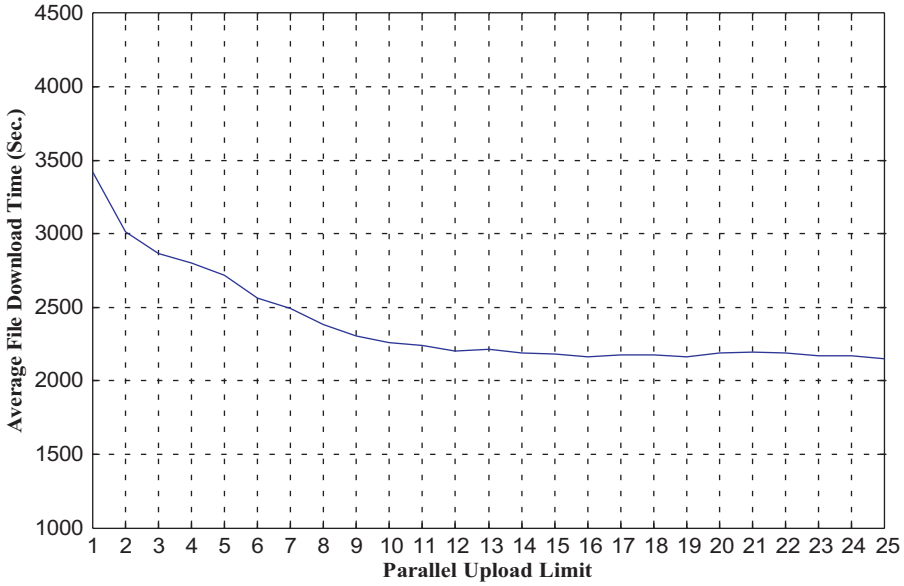

Fig. 11. Effect of parallel upload limit on average file download time.

certain value. After that point, downlink utilization stays almost constant. However, the uplink utilization does not change significantly with the increase of parallel upload limit. When parallel upload limit is set to a small number, peers handle few transmissions each with high upload rates instead of serving more peers each with 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. In Fig. 11, we observe that the average file download time decreases as the parallel upload limit increases up to a certain value in the simulations. Increasing this parameter after this threshold does not lead to further decrease in the download time.

Another significant result is the data size uploaded by the primary seed. As shown in Fig. 12, 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.

5.2. *Scalability analysis and impact of arrival and departure patterns*

According to the popularity of the published content, the arrival and departure of peers can follow different patterns. For

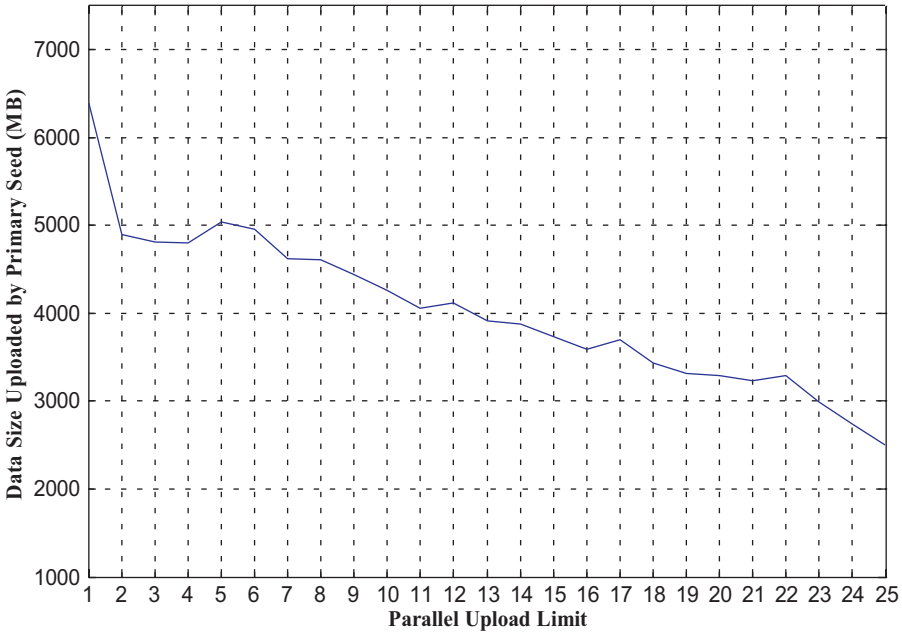

Fig. 12. Parallel upload limit vs. load imposed on the primary seed.

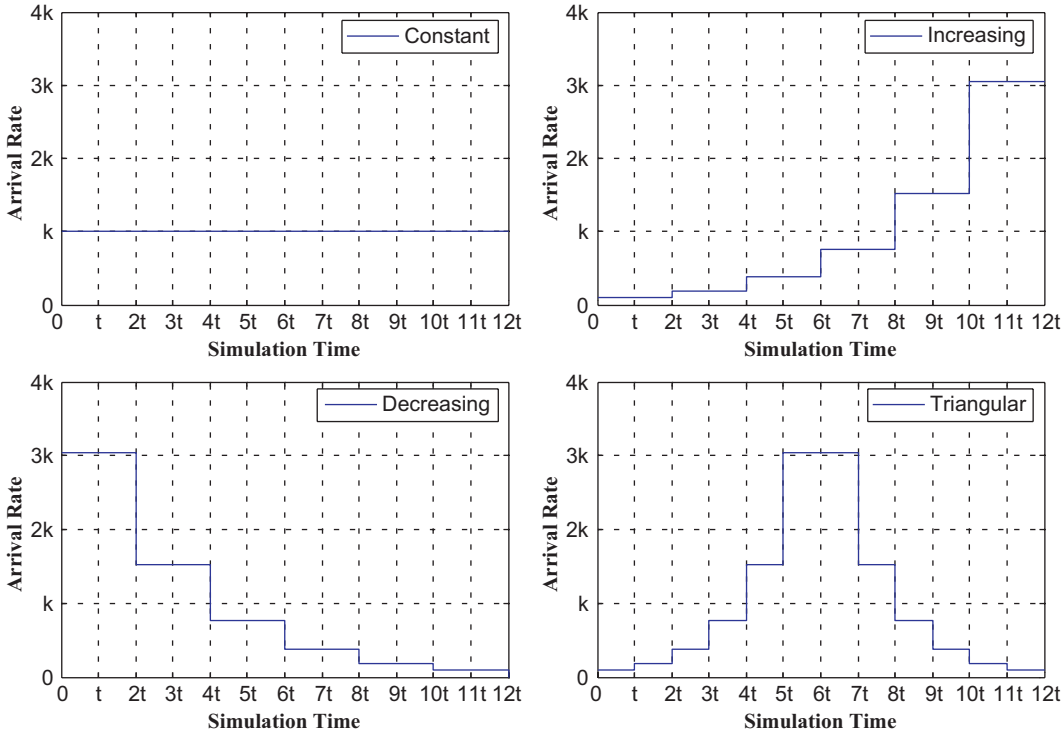

Fig. 13. Arrival patterns.

this purpose, we analyze five different arrival patterns, namely *constant*, *increasing*, *decreasing*, *triangular*, and *flash crowd*. While the constant arrival corresponds to a Poisson process with constant rate for the generation of file requests, the increasing and decreasing arrival rates stand 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, which is first increasing then decreasing, models such a case. Flash crowd stands for joining the system at the same time which can be thought as a storm. Due to its uniqueness, we analyze it separately together with the associated protocol overhead in the next section.

Fig. 13 illustrates our implementation of the first four arrival scenarios. 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

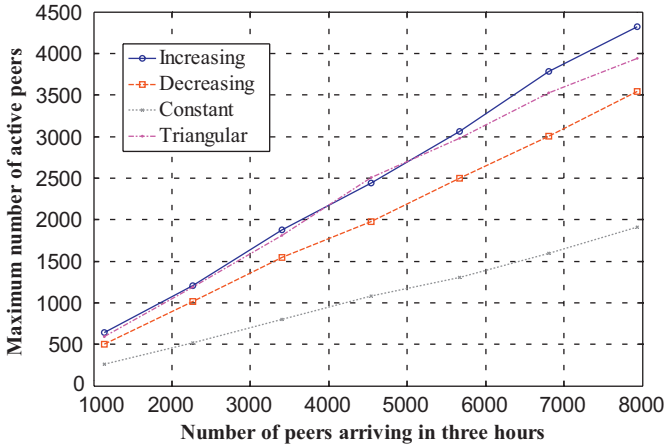

Fig. 14. Maximum number of active peers vs. system size.

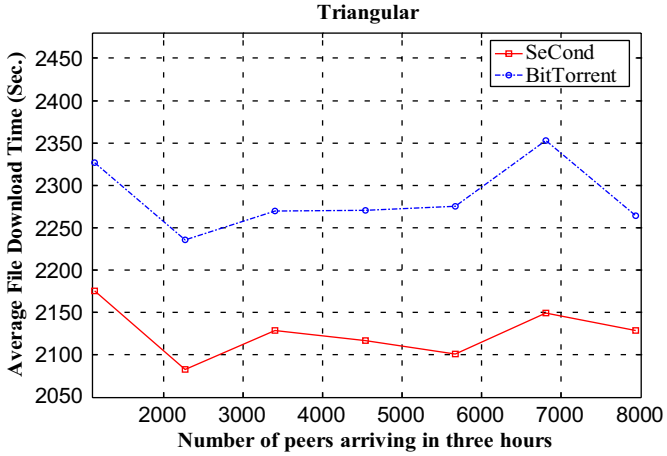

Fig. 16. Average file download time for triangular arrival pattern vs. system size.

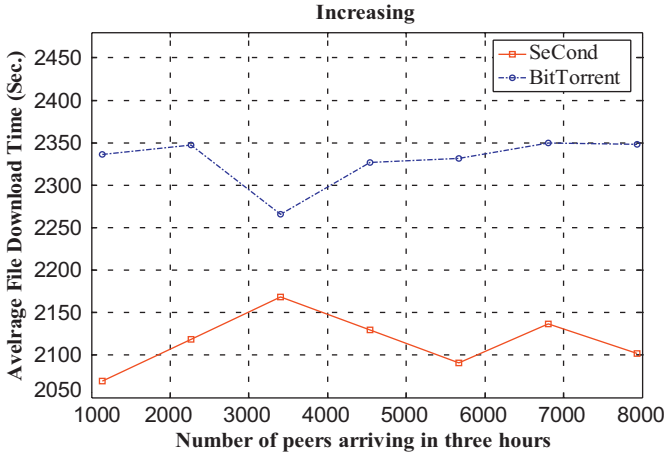

Fig. 15. Average file download time for increasing arrival pattern vs. system size.

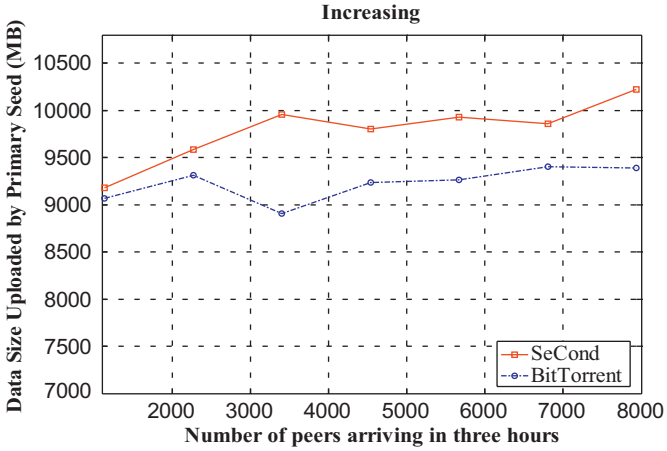

Fig. 17. Load imposed on the primary seed for increasing arrival pattern vs. system size.

arrival period. In the triangular 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.

For departure patterns, we assume two different cases, namely, *immediate* and *random* departure. Immediate departure stands for the case in which the peers leave the system as soon as their file download is completed. In random departure, the peers continue participating in the system as a seed for a random amount of time after the download completion, which follows an exponential distribution.

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 the previous section. For scalability analysis, we vary the peer population joining the system from 1000 to 8000. Fig. 14 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 constant arrival rate.

The effect of system size on the average file download times are given in Figs. 15 and 16 for increasing and triangular arrival patterns, respectively. The other two arrival patterns yield similar results that are not shown here. Our results indicate that the average file download time is almost constant even if the system size is scaled up for both Second and BitTorrent models. There is not much difference in the average download times corresponding to arrival patterns in SeCond simulation results. On the other hand, for the BitTorrent, the increasing arrival pattern causes the largest average file download time. For these arrival scenarios, SeCond peers download the file in shorter time periods in comparison to BitTorrent peers. In order to exploit the system resources, SeCond proposes strategies such as smart peer and view update policies. The results endorse that these policies increase the utilization of the system resources and lead to smaller download periods.

If peers download a large portion of the file from the primary seed, the download time is expected to increase since the primary seed would be overloaded and become a single point of failure. Figs. 17 and 18 show the data sizes uploaded by the primary seed as the system size scales up. These results indicate that in both of the protocols, peers cooperate well among themselves instead of requesting the blocks from the primary seed. 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. Because of the fact that 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 the reason for the slight difference between the two protocols in terms of the loads imposed on the primary seed.

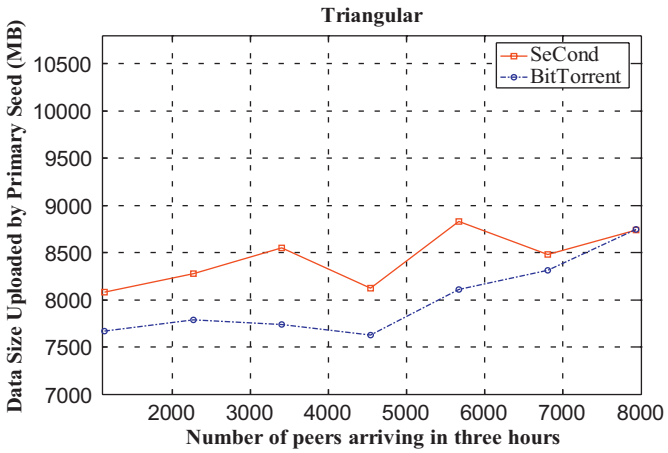

Fig. 18. Load imposed on the primary seed for triangular arrival pattern vs. system size.

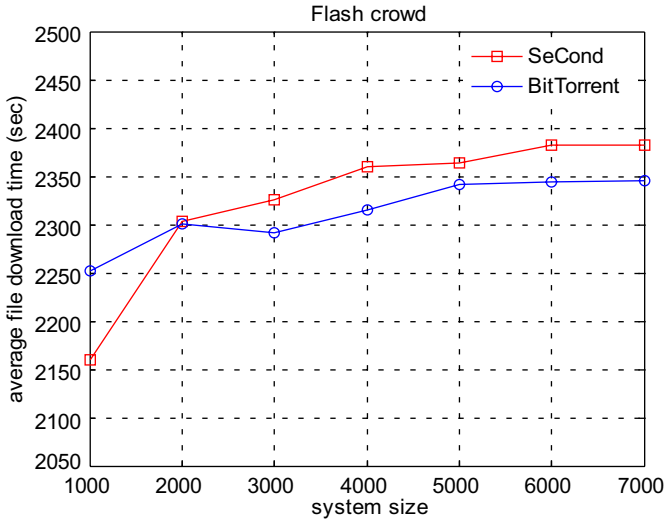

Fig. 19. Flash crowd scenario: average file download time vs. system size.

The analogous results for random departure pattern are also analyzed and the comparative conclusions are similar. As the mean time that the peers remain connected after download increases, the average file download time decreases dramatically as expected. Likewise, the load of the primary seed is alleviated when the mean time to departure is increased.

5.3. Flash crowd scenario and overhead analysis

We analyze the flash crowd arrival scenario in terms of average file download time, the load on the primary seed and the overhead associated with SeCond and BitTorrent. Fig. 19 shows that even for this worst case scenario, SeCond scales well and SeCond peers complete their download in a comparable time to BitTorrent peers. However, the primary seed uploads a little more for SeCond due to the immediate departure characteristics described in the previous section.

Control messages used for the propagation of available block information among the peers form 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 block identifiers inserted to the message. We measure the state update messages for a system which consists of 1000 peers with flash crowd arrivals.

Figs. 20(a) and (b) 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, Figs. 21(a) and (b) show the number of state update messages sent by the BitTorrent and SeCond peers 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 SeCond's. The number of state update messages of BitTorrent is measured as 87 folds of 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.

Note that, although gossiping leads to lower number of state update messages, it may affect the density equalization of each data block among neighboring peers as explained in Section 3.5. SeCond approximates the rarest-first policy of BitTorrent in much smaller scope by collecting density information via gossiping to only a subset of the neighbors. However, we have not observed degrading in performance in terms of average download time. SeCond peers are able to download faster than BitTorrent peers for the arrival patterns except flash-crowd. Even for the worst case scenario of flash-crowd arrivals and immediate departures, SeCond peers complete their download in a comparable time to BitTorrent peers. In support of efficiency of gossiping mechanism, the recent theoretical study of Sanghavi et al. (2007) investigates the performance of gossip-based block selection protocols in the context of file sharing.

5.4. 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 the *tit-for-tat* strategy

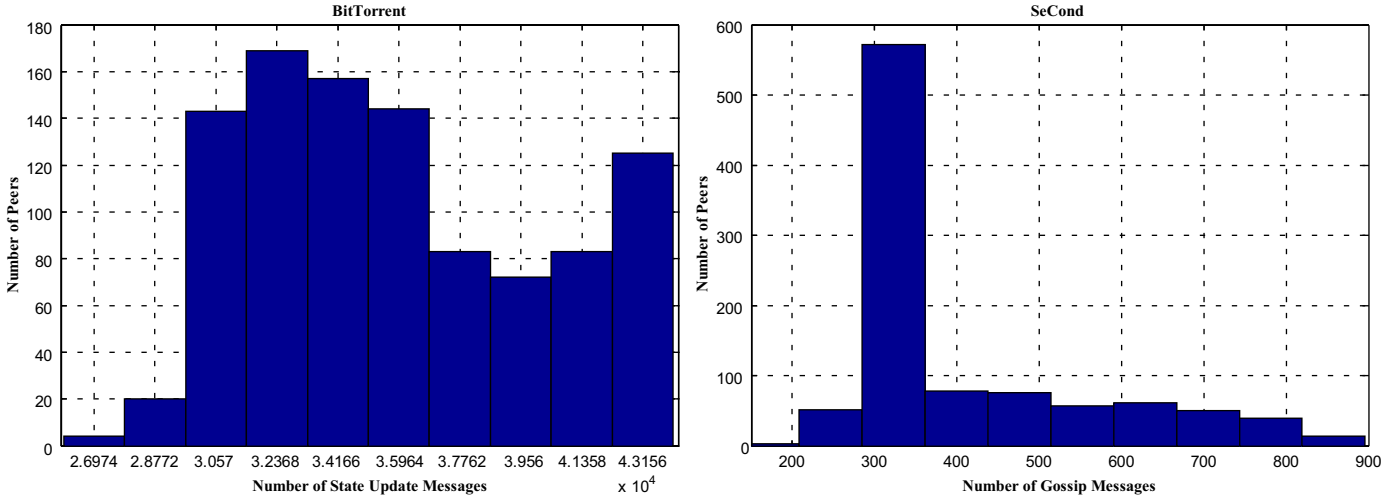

Fig. 20. Histograms for (a) BitTorrent state update messages and (b) SeCond gossip messages.

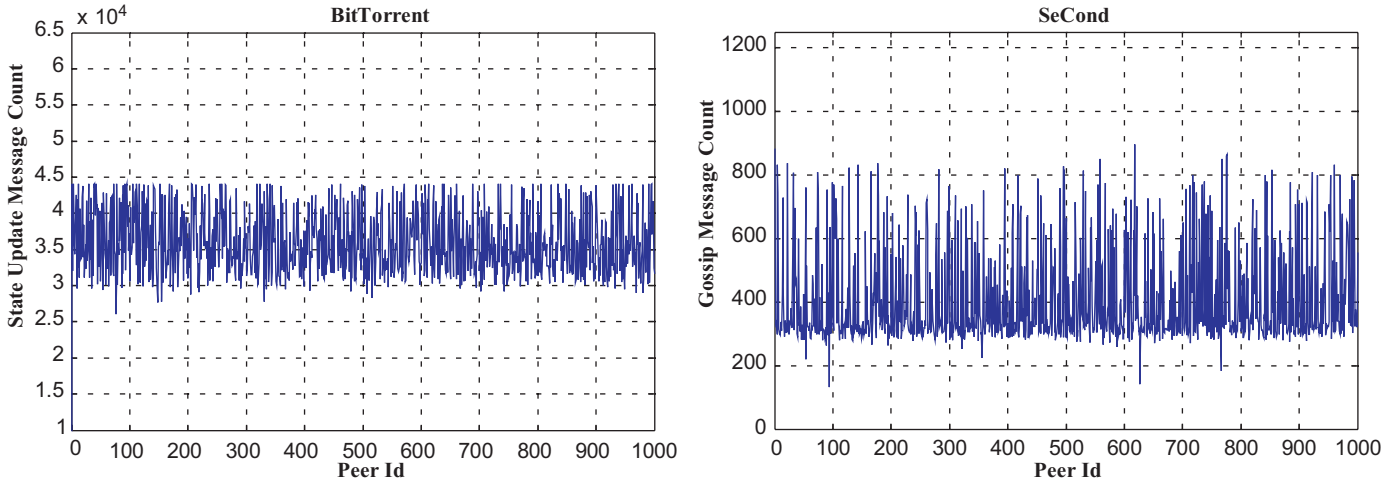

Fig. 21. Messages sent: (a) state updates by BitTorrent peers and (b) gossips by SeCond peers.

Table 4
Bandwidth distribution according to peer identifiers

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

which encourages peers to upload. A peer uploads the peers providing the best download rates. On the other hand, SeCond aims to increase the performance of the overall system. Each peer updates its gossip sender and receiver lists dynamically. Moreover, smart peer policy is employed to adjust fan-out parameter based on the utilization of uplink bandwidths.

In order to compare BitTorrent and SeCond, we study fairness ratio for a system of size 1000. Bandwidth distribution according to peer identifiers is given in Table 4. Fig. 22 shows that SeCond is as fair as BitTorrent. The average fairness ratios are recorded as 0.97 and 0.98 for SeCond and BitTorrent, respectively. Despite the fact that SeCond has no explicit strategy addressing free-riding, the closeness of the fairness ratios is expected due to the form of the peer selection in upload decisions. As presented in Section 3.2, each peer's Gossip Senders List consists of the peers from where

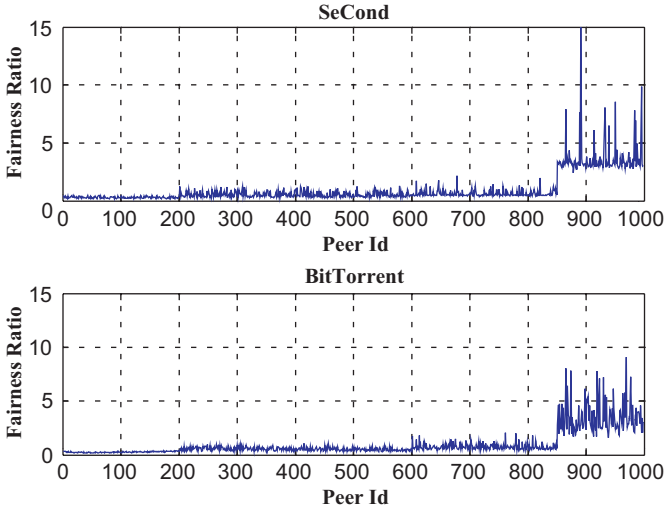

Fig. 22. Fairness ratio.

the blocks are downloaded, as well as the peers' state which is a list of block ids downloaded. During upload decisions, each peer gives priority to the peers from which it has downloaded more.

This strategy seems to compensate for the tit-for-tat strategy of BitTorrent as observed in comparable fairness results.

Fairness is especially important in heterogeneous environments for giving high bandwidth users an incentive to participate. 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 Fig. 22, the peers having identifiers between 850 and 1000 have larger fairness ratio values in comparison to the peers with lower capacities implying that their uplink capacity is exploited by several peers downloading concurrently. Such a high bandwidth peer may be getting a low download rate itself, yet letting other peers consume its uplink bandwidth at the same time. Fairness ratio should ideally be 1, which is not achieved for such peers even with BitTorrent's tit-for-tat mechanism. The study of Bharambe et al. (2006) confirms this observation with comparable values for the maximum fairness ratio to those in Fig. 22. It further indicates that matching peers according to their bandwidths during upload/download decisions ensures fairness. Such a mechanism decreases uplink utilization hence degrades performance as a drawback. On the other hand, it is observed that groups of disconnected peers emerge if the tracker matches the peers according to their bandwidths. It is shown that a hybrid of bandwidth matching and random selection of peers by the tracker performs well for both fairness and utilization metrics.

6. Analytical framework

In this section, we discuss an analytical framework applicable to SeCond's behavior in order to clarify the effect of the network parameters and various arrival/departure patterns on the file dissemination time. Several analytical models of BitTorrent-like networks have been proposed over the recent years. Yang and De Veciana (2004) use branching processes and numerically solved Markov chains to study the transient and steady state behavior of the file sharing system, respectively. Inspired by this study, the simple fluid model of Qiu and Srikant (2004) represents a homogeneous network through deterministic differential equations. Clevenot-Perronnin and Nain (2005) equip the latter model with multiple classes in order to represent a heterogeneous network. More recently, Tian et al. (2006) construct a continuous time Markov chain involving different states for the download job completeness. This provides a detailed analysis of the number of peers in the system with various fractions of the file in BitTorrent-like systems. However, we focus on the total number of downloading peers and seeds along the lines of the former models since our aim is to identify the role of the main parameters in dissemination time. Although the fluid models contain few parameters as a mathematical abstraction, they serve as an estimate of the average behavior observed in simulations. The following results complement the detailed performance analysis accomplished through simulations in the preceding section where the link bandwidth is indeed shared among the peers equally as in a fluid flow.

In the simple fluid model of Qiu and Srikant (2004), the dynamics is described by the deterministic equations

$$\frac{dx}{dt} = \lambda - \theta x(t) - \min\{cx(t), \quad \mu(\eta x(t) + y(t))\},$$
$$\frac{dy}{dt} = \min\{cx(t), \quad \mu(\eta x(t) + y(t))\} - \gamma y(t) \tag{1}$$

where $x(t)$ and $y(t)$ represent the number of leechers (downloaders) and the seeds at time t, respectively. Each peer is assumed to have the same uplink μ and downlink c bandwidth capacities. The parameter λ is the constant arrival rate of new requests, and μ and c are the uplink and downlink capacities of a peer, respectively. The parameter θ is the rate at which the leechers abort the download and γ is the rate at which the seeds leave the system. Lastly, η indicates the effectiveness of the file sharing. The total upload rate of the system is given as $\min\{cx(t), \mu(\eta x(t) + y(t))\}$. The solution (x, y) of the deterministic model (1) represents the expected value of the leechers and the seeds, respectively, in the limit as λ goes to infinity in a stochastic fluid model described by

$$x(t) + \sqrt{\lambda}\hat{x}(t), \quad y(t) + \sqrt{\lambda}\hat{y}(t) \tag{2}$$

where $(\hat{x}, \hat{y})$ is a two-dimensional Ornstein-Uhlenbeck process. That is, the variability around the mean number of peers is described by Gaussian random variables. In the steady state, these are mean zero random variables with variance that can be explicitly computed in terms of the above parameters (Qiu and Srikant, 2004, p. 371). In fact, model (2) is itself a stochastic fluid limit for a continuous time Markov chain describing the number of leechers and the seeds given in Yang and De Veciana (2004) as λ goes to infinity.

Clevenot-Perronnin and Nain (2005) propose a multiclass fluid model (x_i, y_i), $i = 1, 2$, generalizing (1) to describe heterogeneous bandwidth capacities as well as service differentiation. The system of differential equations is given by

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

where $k = 3-i$, and i denotes the type of the peer, according to its bandwidth capacity for example. The parameter α_i determines how a peer allocates its upload bandwidth between two classes of peers. The system of differential equations is a switched system, the analysis of which gets relatively complicated even for two classes and no seeds, that is, $y_1(t) = y_2(t) = 0$ for all $t > 0$. It is especially complex to study the existence and stability of these solutions when the seeds are included (Clevenot-Perronnin and Nain, 2005). Therefore, the authors study only the no seed case with only two equations and single α, which nevertheless serves as the worst case scenario where the downloaders depart as soon as their file is complete.

In SeCond simulations, we have considered a heterogeneous network with four levels of bandwidth as indicated in Table 1. Even in the no seed case, this would require four classes of downloaders x_i, equivalently a system of four differential equations generalizing (3) to include all other values of k. Such an analysis could reveal the coupling between the peers with varying bandwidth capacity. However, we leave it as future work due to its complexity and adopt the homogeneous model (1) for illustrating the ballpark effect of download and upload capacities on the download time below. The other aspects have been considered only through simulations in the preceding section.

For the steady state analysis of (1), the rate of changes in the number of seeds and the number of leechers are assumed to be zero. Namely, $dx/dt = dy/dt = 0$. It is shown that average file download time in the steady state regime is given by $T = 1/(\theta + \beta)$, where $1/\beta = \max\{1/c, \ 1/\eta\,(1/\mu - 1/\gamma)\}$. This shows that the average download time T is independent of the arrival rate λ confirming scalability from another viewpoint (Qiu and Srikant, 2004). The simulations with a constant deterministic

arrival rate given in Alagoz (2006) show that the average file download time stays constant as λ increases in the heterogeneous case as well.

In the following analysis, we run simulations over a homogeneous network according to the bandwidth parameters chosen in Qiu and Srikant (2004) to demonstrate their effect on the download time in the steady state. As shown in Fig. 23, the simulation results agree well with the solution of the deterministic model given in (1). The level of agreement is similar to Fig. 8 of Qiu and Srikant (2004) where a real experiment with BitTorrent is compared with the analytical model. This is due to the fact that there are several parameters that the analytical model does not take into account, but appears in the real system and its simulation. The agreement with model (2), which is yet a very close analytical model to (1), is almost perfect for larger arrival rates as predicted by the theory (Qiu and Srikant, 2004). As the analytical models are good approximations for greater values of λ, it is chosen relatively large in the following. For the results given in Figs. 23(a) and (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 min. The bandwidth values are rates with respect to the file size, which is assumed to be 1 in the analytical model. In simulations, they are adjusted according to the actual file size. Finally, the effectiveness parameter η is set to 1, which indicates that the uplink capacity of the leechers is fully used. It is taken as 0 only when the leechers in the system is 1. In Figs. 23(c) and (d), we have the same setting as the first experiment, except that we set $\gamma = 0.005$. In these results, when the departure rate is set as $\gamma = 0.001$, it refers to a system where the downloading bandwidth is the bottleneck, namely $\gamma < \mu$. When the departure rate of seeds γ is set to 0.005, the uploading bandwidth of the peers now becomes the bottleneck (Qiu and Srikant, 2004). Here, normalized number of seeds (or leechers) indicates the ratio of the number of seeds (or leechers) to the arrival rate λ. According to Little's law, this ratio is equal to the download time in the steady state. Indeed, the simulations seem to fluctuate around a steady state value in all graphs. For $\gamma = 0.001$, the number of leechers are 50 on the average and hence the average download time is 500 min. The model predicts the download time by $T = 1/c = 500$ since $\mu > \gamma$ in this case. On the other hand, we observe that the average number of seeds stabilize at 100 which coincides with the arrival rate λ with $1/\gamma = 1000$ as expected from Little's law since γ is the departure rate for the seeds. For $\gamma = 0.005$, similar calculations hold except that $\mu < \gamma$ this time yielding $T = 1/\mu - 1/\gamma = 600$ min. The average number of leechers is 60 and the average number of seeds is 20 which appear as 600 and 200, respectively, in Figs. 23(c) and (d) after normalization.

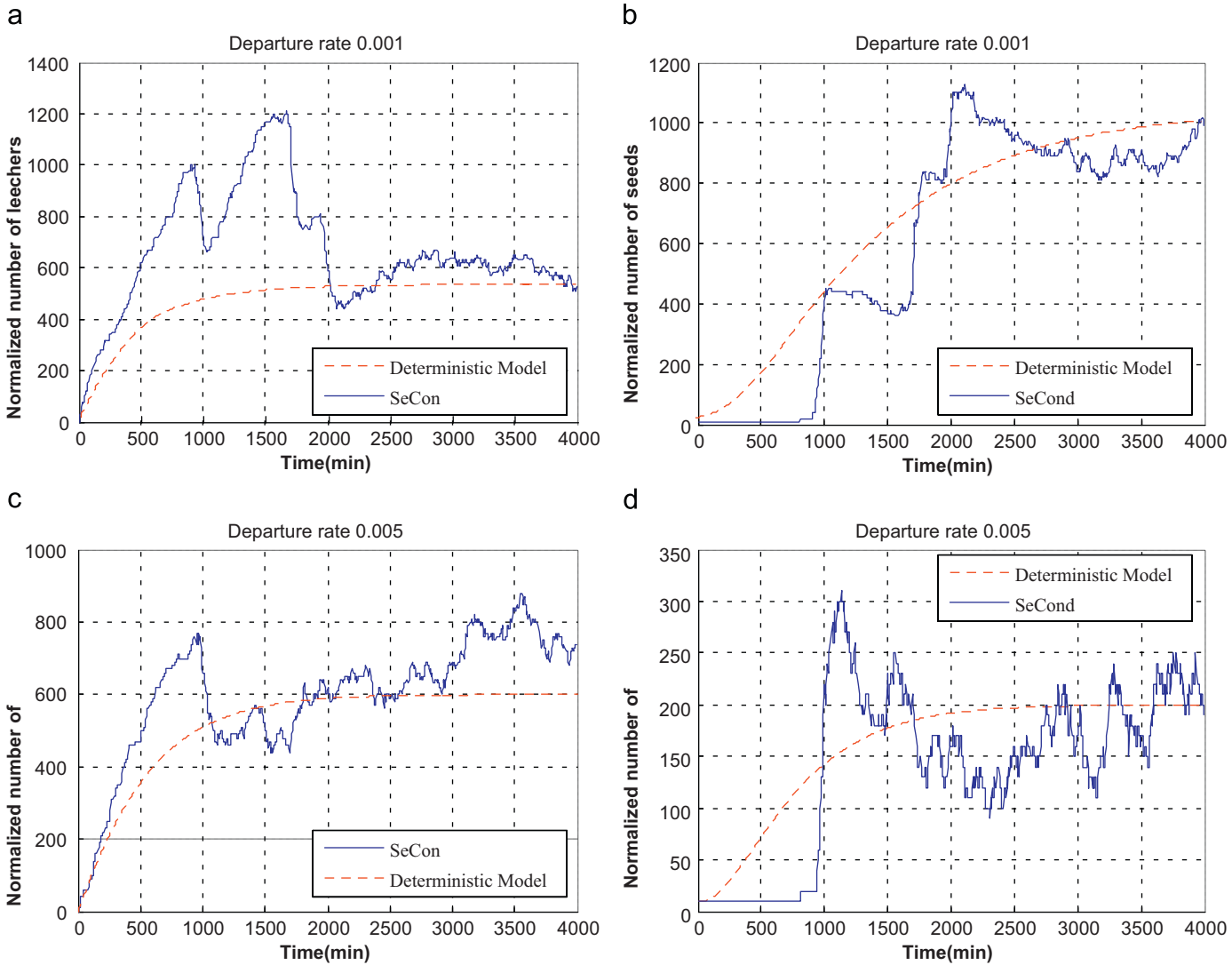

Fig. 23. Deterministic model vs. SeCond: normalized number of leechers/seeds as a function of time.

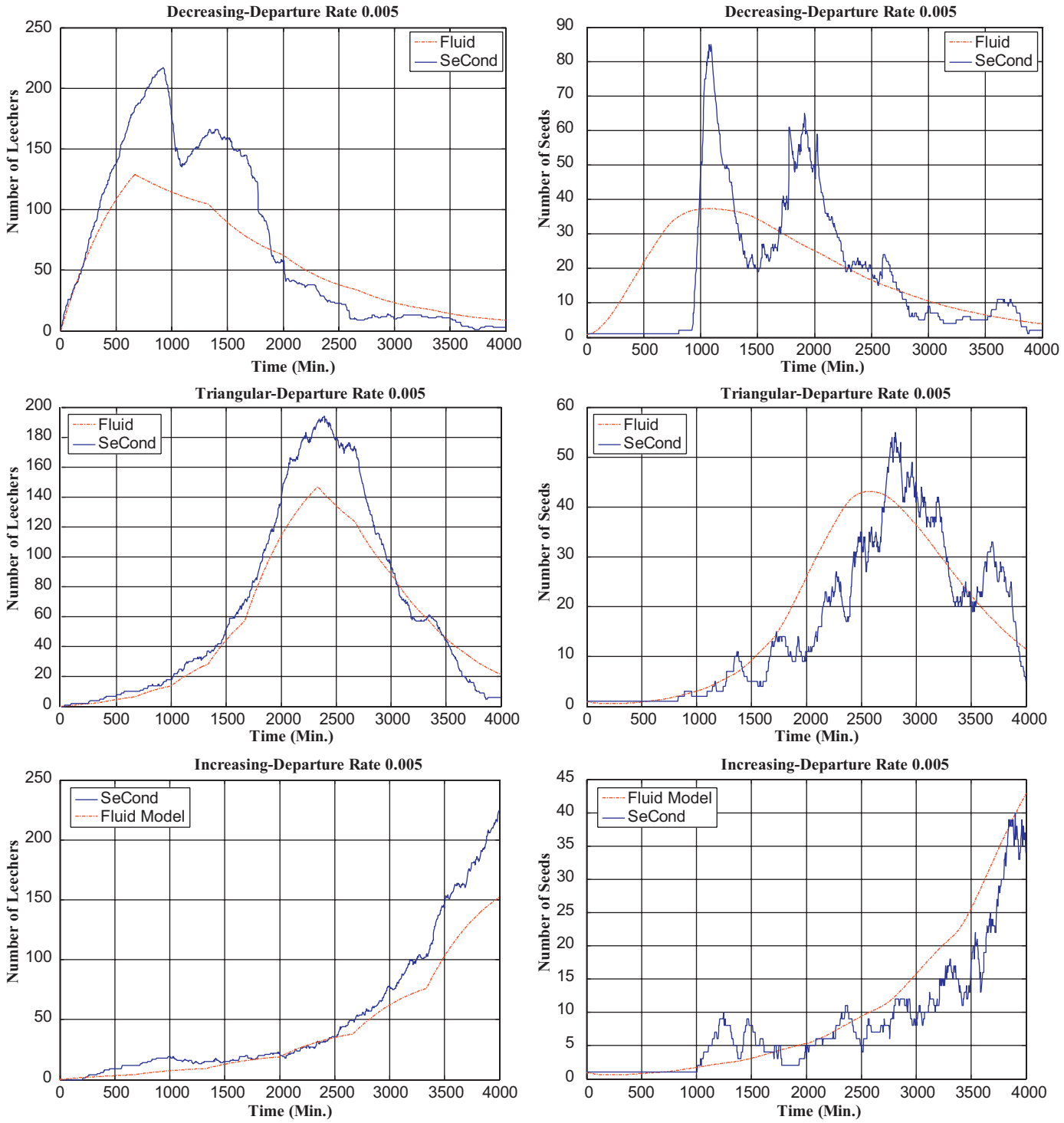

Fig. 24. Deterministic model vs. SeCond for (a) decreasing, (b) triangular and (c) increasing arrival pattern.

For the varying arrival rates considered in the previous section, we modify model (1) by making λ depend on time as in Fig. 13 and solve for the number of leechers x and seeds y. Realizations for the decreasing, triangular and increasing arrival patterns are compared with the analytical model in Figs. 24(a)–(c), respectively, for the departure rate $\gamma = 0.005$. Note that the model represents the average behavior in the transient state as the arrivals are nonstationary. The numbers of leechers and seeds in the system reflect the arrival pattern they stem from in each case. These results indicate that the analytical model can approximate the expected number in the system for all arrival patterns. For the flash crowd scenario, the minimal time to disseminate the file to all peers can be computed using the bound given in Mundinger et al. (2007) as a benchmark.

7. Conclusions

Our proposal in this article, SeCond, is a P2P protocol addressing the distribution of large sized data to several end

systems in an efficient manner. For the ease of deployment, scalability, adaptivity to dynamic peer arrivals/departures, and also increasing 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.

We have developed a simulation model of SeCond and analyzed its behavior via several simulations for different peer arrival patterns. Performance metrics such as average download time, load on the primary seed, uplink/downlink utilization, 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. Smart peer policy increases the utilization dramatically.

A well known and widely used P2P content distribution system is BitTorrent which we also model and compare as a benchmark. A comprehensive performance evaluation of our system, 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, SeCond has comparatively lower protocol overhead than BitTorrent which is one of the major reasons behind the lower file download time of SeCond peers.

An analytical fluid model is used to approximate the behavior of SeCond. We have compared the results obtained through simulations with the fluid model and shown that simulation results are consistent with the analytical predictions for various arrival patterns. For the flash crowd scenario, analytical studies of epidemic dissemination can provide a lower bound for the expected dissemination time and average delay.

As future work, we consider enhancing our protocol with a dynamic gossip interval adjustment mechanism to alter the rate of gossip messages. 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.

Acknowledgements

We would like to thank the anonymous reviewers for their constructive and valuable comments. We also wish to thank Hasan Tuncer for his help in generating the data points of Fig. 19. This work is partially supported by TUBITAK (The Scientific and Technical Research Council of Turkey) and COST (European Cooperation in the field of Scientific and Technical Research) Action 279 "Analysis and Design of Advanced Multiservice Networks supporting Mobility, Multimedia, and Internetworking", and TUBITAK CAREER Award Grant 104E064.

References

Adar E, Huberman B. Free riding on gnutella. First Monday 5(10), 2000.
Alagoz A, Ozkasap O, Caglar M. On epidemic peer-to-peer content distribution. WorldComp-PDPTA'07. International conference on parallel and distributed processing techniques and applications. Las Vegas, June 2007.
Akamai: 〈http://www.akamai.com〉.
Alagoz A. Design and performance evaluation of a system for epidemic peer-to-peer content distribution. M.Sc. thesis, Koc University, August 2006.
Bharambe AR, Herley C, Padmanabhan VN. Analyzing and improving a bittorrent network's performance mechanisms. IEEE Infocom, 2006.
Bickson D, Malkhi D. The Julia content distribution network. In: Second Usenix workshop on real, large distributed systems (WORLDS '05), December 2005.
Bindal R, Cao P, Chan W, Medved J, Suwala G, Bates T, et al. Improving traffic locality in BitTorrent via biased neighbor selection. In: 26th IEEE international conference on distributed computing systems (ICDCS'06), 2006.
Birman KP, Hayden M, Ozkasap O, Xiao Z, Budiu M, Minsky Y. Bimodal multicast. ACM Trans Comput Syst 1999;17:41–88.
BitTorrent: 〈http://www.bittorrent.com/〉.
Byers JW, Considine J, Mitzenmacher M, Rost S. Informed content delivery across adaptive overlay networks. In: Proceedings of the ACM SIGCOMM 2002.
Byers JW, Luby M, Mitzenmacher M, Rege A. A digital fountain approach to reliable distribution of bulk data. In: Proceedings of the ACM SIGCOMM'98, pp. 56–67, 1998.
Clevenot-Perronnin F, Nain KRP. Multiclass p2p networks: static resource allocation for service differentiation and bandwidth diversity. Performance Evaluation, vol. 62, October 2005.
Cohen B. Incentives build robustness in BitTorrent. P2P economics workshop, Berkeley, CA, 2003.
E-donkey: 〈http://edonkey2000.com/〉.
Eger K, Hoßfeld T, Binzenhöfer A, Kunzmann G. Efficient simulation of large-scale p2p networks: packet-level vs. flow-level simulations. In: Second workshop on use of P2P, GRID and agents for the development of content networks, UPGRADE '07, Monterey, CA, 2007.
Fernandess C, Malkhi D. On collaborative content distribution using multi-message gossip. In: 20th IEEE international parallel and distributed processing symposium (IPDPS 2006), April 2006.
Garbacki P, Iosup A, Epema D, van Steen M. 2Fast: collaborative downloads in p2p networks. In: Sixth IEEE international conference on peer-to-peer computing (P2P'06), 2006. p. 23–30.
Gnutella: 〈http://gnutelliums.com/〉.
Guo L, Chen S, Xiao Z, Tan E, Ding X, Zhang X. A performance study of BitTorrent-like peer-to-peer systems. IEEE J Select Areas Commun (JSAC) 2007;25(1):155–69.
Jelasity M, van Steen M. Large-scale newscast computing on the internet. Technical Report IR-503, Vrije Universiteit, Department of Computer Science, October 2002.
JUnit: 〈http://www.junit.org/〉.
KaZaA: 〈http://kazaa.com/〉.
Kostic D, Rodriguez A, Albrecht J, Vahdat A. Bullet: high bandwidth data dissemination using an overlay mesh. In: Symposium on operating systems principles (SOSP), 2003.
Mundinger J, Weber R, Weiss G. Optimal scheduling of peer-to-peer file dissemination. J Schedul 2007.
Pouwelse JA, Garbacki P, Wang J, Bakker A, Yang J, Iosup A, et al. Tribler: a social-based peer-to-peer system. Concurr Comp: Pract E 2008;20(2):127–38.
Qiu D, Srikant R. Modeling and performance analysis of bittorrent-like peer-to-peer networks. In: Proceedings of ACM SIGCOMM, 2004.
Rodriguez P, Biersack E. Dynamic parallel access to replicated content in the internet. IEEE/ACM Trans Network 2002;10(4):455–65.
Sanghavi S, Hajek B, Massoulie L. Gossiping with multiple messages. IEEE Trans Inform Theory 2007;53:4640–54.
Saroiu S, Gummadi PK, Gribble SD. A measurement study of peer-to-peer file sharing systems. In: Proceedings of the multimedia computing and networking 2002 (MMCN'02), San Jose CA, USA, January; 2002.
Sherwood R, Braud R, Bhattacharjee B. Slurpie: a cooperative bulk data transfer protocol. IEEE Infocom, Hong Kong 2004.
Squid: 〈http://www.squid-cache.org〉.
Tian Y, Wu D, Ng KW. Modeling, analysis and improvement for BitTorrent-like file sharing networks. IEEE Infocom 2006.
Yang X, De Veciana G. Service capacity of peer to peer networks. IEEE Infocom, Hong Kong 2004.