Computer Networks 53 (2009) 2259–2274

Contents lists available at ScienceDirect

Computer Networks

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

Stepwise fair-share buffering for gossip-based peer-to-peer data dissemination

Oznur Ozkasap [a,*], Mine Caglar [b], Emrah Cem [a], Emrah Ahi [c], Emre Iskender [a]

[a] Department of Computer Engineering, Koc University, Istanbul, Turkey
[b] Department of Mathematics, Koc University, Istanbul, Turkey
[c] Risk Software Technologies, Inc., ITU Technopark, Ayazaga, Istanbul, Turkey

ARTICLE INFO

Article history:
Received 1 September 2008
Received in revised form 25 March 2009
Accepted 25 March 2009
Available online 21 April 2009

keywords:
Distributed systems
Performance of systems
Buffering
Gossiping
Epidemic
Peer-to-peer
Data dissemination
Reliability
Scalability

ABSTRACT

We consider buffer management in support of large-scale gossip-based peer-to-peer data dissemination protocols. Coupled with an efficient buffering mechanism, system-wide buffer usage can be optimized while providing reliability and scalability in such protocols. We propose a novel approach, stepwise fair-share buffering, that provides uniform load distribution and reduces the overall buffer usage where every peer has a partial view of the system. We report and discuss the comparative performance results with existing buffering approaches as well as random buffering which serves as a benchmark. We present separate evaluations of bufferer selection and gossip-based data dissemination. Reliability, content dissemination time, message delay, buffering delay, and minimum buffer requirements are considered as the key metrics investigated through simulations. The performance of our approach in the case of multiple senders, link failures with multiple bufferers, and scalability to larger networks are investigated. Several power-law and hierarchical overlay topologies are considered. Analytical bounds for reliability of dissemination are also provided.

1. Introduction

Buffer management is a significant component that supports reliability in large-scale peer-to-peer (P2P) data dissemination protocols. A key feature of such large-scale protocols is the provision of all-or-none (that is, reliable) data delivery to all peers that need the data. In order to deal with problems such as failure of the sender and possible request implosion on the sender, other peers in the system buffer the data that they receive. *Buffering* refers to the approach of determining which data and how each peer keeps in its memory. It is used to offer data retransmissions when needed for loss recovery. However, a protocol having all peers buffer messages until the message has been delivered to all (that is, the message becomes stable) would not be scalable, since the buffering load on each peer grows with the system size. One reason is that the time to accomplish and detect message stability increases as the system size scales up [1]. Thus, an efficient mechanism for buffering is crucial to maintain both reliability and scalability of data dissemination protocols. Another benefit of efficient buffer management and having different bufferers for data messages is to balance the recovery overhead among peers. The available approaches for buffer management concentrate on several aspects of the problem such as flow control [2,3], reducing the memory usage [1,4,5], providing message stability [6,7], and the replacement of buffer items [8,9].

In this study, we couple the buffer management problem with a P2P gossip-based (or epidemic) data dissemination method. For large-scale P2P services, bio-inspired epidemic protocols have considerable benefits as they are

* Corresponding author. Tel.: +90 212 3381584; fax: +90 212 3381548.
E-mail addresses: oozkasap@ku.edu.tr (O. Ozkasap), mcaglar@ku.edu.tr (M. Caglar).

1389-1286/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2009.03.021

robust against network failures, scalable and provide probabilistic reliability guarantees. Several distributed services such as failure detection, data aggregation, resource discovery and monitoring [10], and database replication [11] utilize epidemic algorithms. A major aim of our approach is to be able to choose bufferers uniformly throughout the system so that the load of buffering will be well balanced among participating peers and the efficiency of content dissemination will be improved as a result.

We propose a robust and distributed buffering scheme named *stepwise fair-share buffering* and analyze its behavior by extending the preliminary results reported in [13]. Our approach needs only local information at each peer (that is, one-hop neighboring peers on the overlay). It is called *stepwise*, since it uses this local knowledge to search for and then determine bufferer(s) of a data message in a step-by-step (or, hop-by-hop) manner. Our buffering approach is referred as *fair-share*, since it is fair among the peers in terms of distributing the buffering load in a balanced way. This fairness is accomplished by determining bufferers uniformly throughout the system independent of the overlay topology.

Our simulation results show that stepwise fair-share buffering provides a uniform load distribution. It reduces the memory usage since only a small subset of the peers is chosen as bufferers for each message. Furthermore, stepwise fair-share buffering is applicable to large-scale scenarios, provides reliable delivery and is adaptable to dynamic join and leaves in the system. We show that it is scalable, simple and applicable to any kind of underlying network topology where each peer has only a partial view of the system. The performance of stepwise fair-share buffering is investigated also in the presence of link failures in which case multiple bufferers are crucial for reliability. The distribution of the bufferers among different domains is examined.

In our scheme, the buffer size can be adjusted to achieve message stability with a high probability. We derive an analytical model for computing the reliability of dissemination as a function of buffer sizes as well as the number of bufferers. These results are based on a Markov chain analysis and are evaluated numerically. Comparison with the simulation shows that the analytical approach yields a tight lower bound for higher reliability values.

The paper is organized as follows. Section 2 reviews the related work in comparison to our approach. In Section 3, we describe stepwise fair-share buffering approach in detail. Section 4 explains the simulation settings and other buffering approaches used for comparison. Section 5 is devoted to the analysis of the buffering approach only. Section 6 describes the analysis of data dissemination integrated with various buffering approaches. In Section 7, analytical evaluation of our approach is given. Finally, the concluding remarks are given in Section 8.

2. Related work

The existing approaches for buffer management are designed for various aspects of the problem, namely, flow control, optimization of the memory usage, providing message stability and the replacement of buffer items. In this section, we review the related work and compare with our approach.

2.1. Network flow control

Flow control is an adaptive mechanism that deals with varying resources such as CPU and bandwidth in the end hosts. In the NAK based retransmission control scheme given in [2], the sender reduces its transmission rate whenever it receives too many NAKs from the receivers. This mechanism helps to minimize the buffer overflows at the receivers.

A different idea explored in [3] requires every process to calculate the average buffer capacity among all processes it communicates with and transmit that information. When the rate is too high with respect to the average, the process reduces the rate locally. On the other hand, a source node reduces the rate of information production according to the process with the smallest buffer space.

2.2. Reducing the memory usage

The pioneering study [1] focuses on reducing the memory requirement by buffering each message only over a small set of members. During reliable multicast data dissemination, a member determines whether it should buffer a message it receives using an approximation of the membership information, and a hash function based on its network address and the identifier of the message. The hash function is devised so that the bufferers are chosen uniformly among the available peers. However, when a new peer joins the system it cannot become a bufferer, as dynamic redefinition of the hash table is not considered.

In [12], a probabilistic buffering algorithm is devised as a separate stage before epidemic data dissemination. It aims to distribute the load of buffering to the entire system and provides a fairly uniform distribution when each peer has a partial view of the system. However, the uniformity is observed only when the number of generated messages approaches the total long-term buffer capacity of the system. Stepwise fair-share buffering as well as the hash-based [1], and probabilistic approach [12] are all designed to reduce the memory usage since only a small subset of the peers is chosen as bufferers for each message. In contrast with the hash-based approach, a new member can become a bufferer in our buffering mechanism.

A multicast protocol that reduces buffer requirements is Randomized Reliable Multicast Protocol (RRMP) [4] which uses epidemic loss recovery. In particular, RRMP offers efficient loss recovery for large multicast groups. A data message is kept in the long-term buffer for a fixed amount of time. By distributing the responsibility of loss recovery among all group members, it aims to improve the efficiency and robustness of tree-based protocols.

Structured P2P networks such as Chord [24], CAN [25] and Tapestry [26] offer a management of participating

peers and published data items. Chord is based on a ring, in Pastry and Tapestry hypercube is used, Tornado uses a tree structure. These systems name the participating peers and available data items with a distributed hash function. Chord [24] assigns keys to nodes with consistent hashing. With a high probability this function balances the load imposed on peers: namely all nodes receive approximately the same amount of keys. Chord peers store a small amount of data and require partial membership information. A node resolves the hash function by communicating with other nodes because the hash function is distributed.

Another buffer management scheme which reduces memory usage is [5] where the members are organized into regions. In every region, the nodes with the most reliable links are responsible for buffering the data.

2.3. Achieving stability

A message is said to be stable when it is delivered to all members of the group. There are buffer management approaches which explicitly take stability into account. In [6], all members periodically exchange messages to inform each other about the messages they have received. When a member becomes aware of a message becoming stable, it safely discards the message. Thus, the system-wide buffer space is reduced. A drawback is the high traffic caused by frequent exchange of history messages.

Search Party [7] is another protocol in which contribution of a timer helps to discard packets from the buffers. All the members discard packets after a fixed amount of time to achieve stability.

A heuristic buffer management method based on both ACKs and NAKs is proposed in [5] to provide scalability and reliability. In every group of receivers, there are one or more members with higher error rates than the other members. These nodes are the ones with the least reliable and slowest links. The idea is that if a message is correctly received by these nodes, it has been probably received by all other nodes. In that case, the repair nodes that buffer the message can discard it.

Our protocol adjusts several parameters such as the number of bufferers and the buffer size to achieve stability with a high probability.

2.4. Replacement policy for buffer items

Network Friendly Epidemic Multicast [8] combines a standard epidemic protocol with a novel buffering technique that combines different selection techniques for discarding messages in case of a buffer overflow. The used selection strategies are random purging, age-based purging and semantic purging. Random purging refers to discarding an item from the buffer randomly. Age-based purging is simply discarding the oldest message and semantic purging means that a message which has been recognized as obsolete is discarded. Obsolescence relation is determined by the application.

Least recently used (LRU) buffer replacement scheme is considered in [9] for epidemic information dissemination. In LRU scheme, a new coming message is placed on the first position and the message at the rear is discarded as in our case. However, when a request arrives for a message in the buffer, that message is placed into the first place by moving the items in front one position down. Hence, the least used item stays at the rear of the stack possibly next to be discarded. In our approach, a first-in-first-out policy equivalent to age-based purging has been implemented in the case of a buffer overflow. We have run simulations also with LRU scheme and found no significant difference.

3. Stepwise fair-share buffering

In this section, we describe the system model, and explain details of stepwise fair-share buffering. We also describe the gossip-based data dissemination algorithm.

3.1. System model

We consider a P2P application consisting of a set of peers $P = \{p_1, p_2, \ldots, p_N\}$ where N is the system size. Each peer $p_i \in P$ has a unique identifier. Peers are connected through an overlay reflecting the properties of the underlying network topology. Each p_i has only a partial view of the system (its neighboring peers). This is a quite plausible assumption considering a large-scale distributed application scenario.

Each p_i has a *long-term buffer*. When a peer becomes a bufferer for a particular data message, it keeps the message in its long-term buffer. The long-term buffer is useful for achieving reliability in data dissemination. We use the term *bufferer* in the sense of long-term bufferer of a message. A peer may also have a *short-term buffer* which is useful during the data dissemination stage. Note that short-term buffers are not required for our approach to work. Once a data message is received by a peer during dissemination, it may be kept in the short-term buffer until it is replaced by a new message at a later time. For both short and long-term buffers, FIFO replacement policy is used.

3.2. Buffering algorithms

We describe stepwise fair-share buffering approach in detail through its algorithms. Abbreviations used in the algorithms are listed in Table 1.

The bufferer determination phase in stepwise fair-share buffering is initiated by a data message source (MS) towards its neighbors with a buffering request message

Table 1
Abbreviations used in algorithms.

d	Data message
MS	Message source
NH	Neighbor history
TTL	Time-to-live
BR	Buffering request message
LB	Long-term buffer
LB-count	Count of *d*s buffered in a peer's *LB* so far

(BR). Every peer maintains the number of data messages that its neighbors have ever buffered in their long-term buffers, which is collectively called neighbor history (NH) information. This local information is updated through our buffering mechanism and used for determining the bufferers of a data message.

Procedures for data generation at MS, NH information update and forwarding BR are given in Algorithm 1. When a data message is generated at a MS, the MS first updates its NH. Updating NH information at a peer is done as follows. The peer sends NH request message to its neighboring peers. Then, it collects responses (LB-counts) from its neighbors, and updates its NH information. A peer receiving an NH request sends requester the count of data messages it has buffered in its long-term buffer (LB) so far which we call LB-count. The MS, which updates its NH, then sets the TTL value of BR for the data, and forwards the BR. The TTL value attached to a BR indicates the maximum number of times (that is, steps) that the BR can be forwarded among peers. Forwarding BR is done to the neighboring peer with the minimum LB-count.

Handling a BR and accepting it are described in Algorithm 2. When a peer receives a BR for a data message for which it is not the MS, the peer first decreases the BR's TTL value. If the TTL becomes zero, then the peer accepts the buffering request. Accepting a BR is done as follows. The peer first inspects its LB, and if the LB is full then it removes the oldest data in LB according to FIFO policy. After that, it puts the data into its LB and increments its LB-count. Then, the peer informs the MS of data message that it has accepted the BR for the data and become a bufferer for it. In the other case (when the TTL value is greater than zero), the peer first updates its NH information. Then, it detects the peer among its neighbors (including itself) that has the minimum LB-count. If this is the peer itself, it accepts the buffering request, otherwise it forwards the BR (that is, sends it to the neighboring peer with the minimum LB-count). In the case of a tie among the minimum LB-counts, the peer chooses one of the neighbors randomly. Thus, the BR is propagated in steps from a peer to another neighboring peer until a bufferer for the data message is determined.

Algorithm 1. Data generation, neighbor history update and forwarding buffering request

Generation of data msg d at MS:
- *Update NH( )*
- set *TTL* of *BR* for *d*
- *Forward BR( )*

Update NH( ):
- send *NH request* to my neighbors
- get *LB-counts* from neighbors
- set local NH information of neighbors

On receipt of NH request from peer q:
- send peer *q* my *LB-count*

Forward BR( ):
- send *BR* to neighboring peer with the minimum *LB-count*

Algorithm 2. Handling a buffering request message, and accepting a buffering request

```
On receipt of BR for data msg d at a peer:
if I am not MS of d
    TTL=TTL-1
    if TTL == 0 then
        Accept BR( )
    else
        Update NH( )
        if my LB-count is the minimum among my neighbors' LB-counts then
            Accept BR( )
        else
            Forward BR( )
        end if
    end if
else
    TTL=TTL+1
    Forward BR( )
end if

Accept BR( ):
if my LB is full then
    remove oldest data from LB
end if
put d into my LB
LB-count=LB-count+1
send my peer ID and d.ID to MS
(I am the bufferer of d)
```

A major benefit of our approach is that it works with local information at each peer. There are four types of control messages, namely BR, Accept BR, NH request and NH response as described above. The number of hops that a BR can propagate is limited by its TTL value. After a bufferer is determined for a data message, identifiers of bufferer and data are sent back to MS in an Accept BR message. Note that, a BR can be piggybacked on a data message as well. Once a bufferer is found, the ID of the bufferer is then attached to the data during epidemic dissemination. Considering that NH request/response messages are exchanged with neighbors of a peer at most once per buffering request, the signaling cost associated with them is negligible. In fact, our performance analysis results show the low cost associated with finding bufferers, and also the effect of TTL parameter.

3.3. Improvements

We incorporate improvements to handle fast buffering request rates and also to deal with link/connection and peer failures. We consider a fail-stop model for peers and receive-omissions for message losses.

When a BR is received by a peer, it sends NH request messages to its neighbors (that is, its partial view members). Then, a certain time passes until all responses

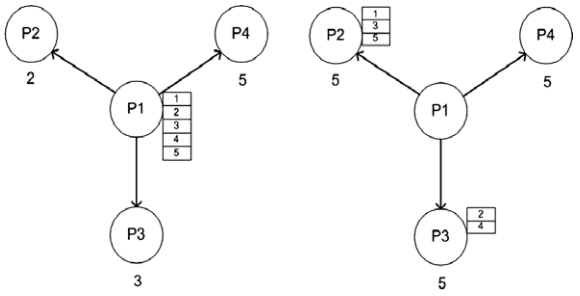

Fig. 1. Handling fast request rate: (a) five BRs accumulate at peer 1 and (b) load is distributed evenly to neighbors.

are received by the peer. Therefore, a key point in the mechanism is to make adjustments when the rate of receiving a BR is faster than the rate of updating the NH. In this case, before the peer receives the responses from its neighbors, multiple buffering requests accumulate in the buffering request list of the peer. Then, the peer *locally* updates its NH after processing each BR in the list. This mechanism, for *handling fast request rate*, balances the load in the case of faster reception of buffering requests.

The improvement for handling fast request rate is illustrated with a simple scenario in Fig. 1. In the network topology given in the figure, peer 1 has three neighbors. Peers 2, 3 and 4's LB-counts are 2, 3 and 5, respectively, as indicated in Fig. 1a. Assume that until peer 1 receives the NH responses from its neighbors, five messages accumulate in its buffering request list. Then, it sends message 1 to peer 2 because it has the minimum LB-count and increments peer 2's LB-count. As a result, LB-counts of peers 2 and 3 are equal to 3. After that, peer 1 selects randomly peer 3 and sends message 2 to it. Then, it sends message 3 to peer 2, message 4 to peer 3 and message 5 to peer 2 as shown in Fig. 1b. Consequently, every peer receives five BRs and the load is distributed evenly.

In the NH update process, every peer waits for responses from its neighbors. However, if there is a link or neighboring peer failure, the peer could wait indefinitely for a response. In order not to lead such a situation, a *timeout parameter* is used. The timeout can be set to a value greater than the maximum round-trip time to the neighbors.

3.4. Epidemic dissemination

A popular distribution model based on the theory of epidemics is the *anti-entropy* [19]. In the terminology of epidemiology, a peer holding information or an update it is willing to share is called *infectious*. A peer is called *susceptible* if it has not yet received an update. In the anti-entropy process, non-faulty peers are always either susceptible or infectious. For spreading data, our system uses a pull-based approach in which spreading is triggered by susceptible peers (by *pulling* data) when they are picked as gossip destinations by infectious peers.

Actions performed at each peer for the pull anti-entropy data dissemination is given in Algorithm 3. At each gossip round, every peer picks randomly *fan-out* number of peers from its partial view (or, neighbors) and sends its digest (containing the identifiers of recent data it has received and identifiers of their bufferers). In fact, each peer in the system performs a state exchange periodically and concurrently with the others.

On receiving a digest and comparing it with its local data, the receiving peer determines the data messages that it lacks. Then, it can request them from the bufferers indicated in the digest for retransmission, if the sending peer has dropped the message from its short-term buffer. If a bufferer has crashed or cannot retransmit the message, the request can be forwarded to another bufferer. We define three events that can occur in a round at a peer, namely digest receipt, request receipt and retransmission receipt. Corresponding actions for these events are given in the algorithm.

Algorithm 3. Pull Anti-entropy Gossiping

Algorithm executed periodically once per gossip round at each peer p:
for *fan-out* number of randomly selected peers q **do**
  Send *Digest* (containing list of p's recent data IDs and data bufferers' IDs) to q
end for

Event(**Digest Receipt**) from peer r:
Compare *Digest* of r with p's local data
if r has a data d that p is missing **then**
  Request d from r (or from a bufferer of d)
end if

Event(**Request Receipt**)
for data d from peer q:
Retransmit d to q (if d is in local buffer)

Event(**Retransmission Receipt**) for data d:
Update p's local data with d

4. Simulation settings

We evaluate the performance of stepwise fair-share buffering through simulations. The simulation software is implemented in Java where a discrete time event based model is used. We generate two types of topologies, namely power-law and hierarchical, with equal mean degree and mean link delay. In this section, we describe the topology settings and give an overview of the buffering approaches that will be compared with the stepwise fair-share buffering.

4.1. Topology properties

Power-law and hierarchical network topologies are considered in this study as models of the Internet overlays. Power-law graphs have attracted great interest on the basis that the Internet AS level graph exhibits a power-law degree distribution [14]. On the other hand, the hierarchical structure of the topology of the Internet is reproduced by the transit-stub model.

A *power-law* graph is one where the number of nodes with degree k is proportional to $k^{-\beta}$ for some $\beta > 1$. We

generate the power-law topologies using the BRITE topology generator together with the coordinates of the nodes on the plane [15,16]. We assign link delays using the Euclidian distances as in [17]. The topology is generated according to the Barabasi–Albert model with incremental growth which is suggested as one possible cause for the emergence of a power-law degree distribution in the Internet.

We use the gt-itm tool [18] for generating *transit-stub* topologies and the corresponding link delays. The transit-stub model is a hierarchical approach which views the Internet as a set of interconnected routing domains. Each domain can be classified as either a stub or transit domain. Stub domains correspond to interconnected local area networks and the transit domains model wide or metropolitan area networks. A transit domain is composed of backbone nodes which are well connected to each other with high bandwidth links. Every transit node is connected to one or more stub domains.

We set the mean degree and mean link delay the same in both topologies. However, their degree and delay distributions are quite different. The degree distribution is approximately normal for the hierarchical topology, whereas it is power-law for the power-law topology. For a realization with 1000 nodes, about 9000 edges and average link delay of 2.5 ms, the delay distributions are illustrated in Figs. 2 and 3. The delay distributions inherit the characteristics of the respective degree distributions.

4.2. Other buffering approaches

There are two comparable approaches to stepwise fair-share buffering developed in connection with epidemic data dissemination. These are hash-based [1] and probabilistic approaches [12]. As a third and baseline approach, we also consider random buffering [20] which assumes full membership information on the source side. In this case, the bufferer selection occurs at once as well as it can be expected to be uniform due to completely random selection of the bufferers.

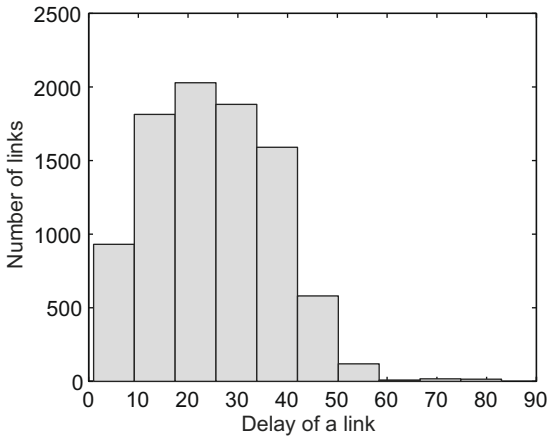

Fig. 2. The distribution of link delays (in ×.1 ms) for hierarchical topology.

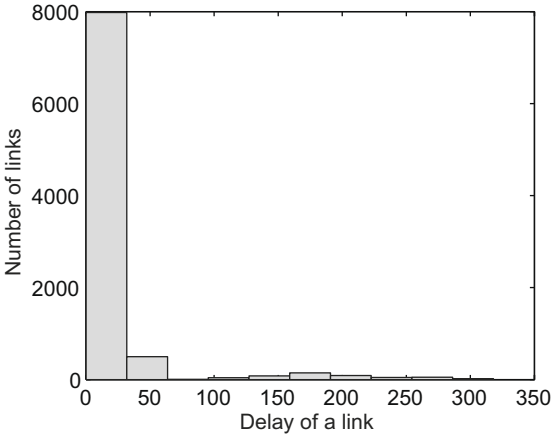

Fig. 3. The distribution of link delays (in ×.1 ms) for power-law topology.

The hash-based approach proposed in [1] focuses on reducing the buffer requirement by buffering each message only over a small set of members. All members are assumed to have an approximation of full membership in the form of a set of member addresses. Upon receiving a data message through epidemic dissemination, a member can determine whether it should buffer the data. For this purpose, it uses its approximation of the entire membership, and a hash function based on its network address and the identifier of the data message. A commonly used identifier for a data message is [*source address, sequence number*]. The hash function is devised so that the bufferers are chosen uniformly among the members. Note that a member can become a bufferer of a data message *only* when it receives the data message through gossiping eventually. Thus, there is no immediate bufferer selection phase separate from the data dissemination in contrast with stepwise fair-share buffering.

A drawback of the hash-based approach is that it requires members to have an approximation of the entire membership (in the form of a set of member addresses) which is not practical to achieve in large-scale systems. Furthermore, since updates to the full membership information are not considered in the case of dynamic peer arrivals, new peers cannot become a bufferer. This may cause uneven buffering of messages over the membership, and hence unfairness in balancing the load of buffering over members. In terms of signaling cost of the approach, message exchanges among members are needed to form an approximation of the full membership. This information is required to determine bufferers of data messages. Furthermore, all members should agree on the hash function, and then use it locally to determine for any data message which members are the bufferers. There are no other control messages specifically needed to determine bufferers.

Probabilistic buffering [12] provides a fairly uniform distribution with partial views of peers. For determining the bufferers of a data message, the source sends buffering request messages to randomly selected b peers in its partial view. Parameter b is the number of bufferers per message. For a data message, if $b > 1$ then its bufferers are determined in parallel. The buffer fullness ratio of a peer

(BF) is the ratio of the number of messages that are stored in the peer's buffer to its long-term buffer capacity. When a peer receives a buffering request message for a particular piece of data, it accepts the request with probability $(1 - BF)$. Otherwise, it forwards the message to a randomly selected peer from its partial view with a probability equal to BF. If a member receives a buffering request, the member writes its ID to the buffering request and sends it to a neighbor. The IDs of the last n forwarders are included in the buffering request message. Via this information, a bufferer request is not resent to the last n forwarders and the TTL mechanism is used more efficiently.

5. Buffering results

In this section, the simulation results for our buffering approach are evaluated separately to demonstrate its efficiency. In particular, uniformity of buffering load, its scalability, the effect of the TTL parameter and multiple-sender scenarios are investigated.

5.1. Uniformity of buffering load

We evaluate the performance of stepwise fair-share buffering in terms of distributing the buffering load. In the first group of experiments, 1000-node power-law and hierarchical topologies are used. Long-term buffer capacity of the nodes is 10 messages, and 50,000 messages are disseminated from a single source with rate of 20 msgs/s. The total messages disseminated in the system is greater than the total long-term buffer capacity of all nodes. The TTL value is set to 20. Each message is buffered only by a single bufferer. In Fig. 4, the buffering load of the nodes is given for various dissemination percentages (20–100%) in power-law and hierarchical topologies. It is observed that stepwise fair-share buffering provides uniformity over time which would be helpful for reliable data dissemination. In particular, it achieves a more uniform distribution with power-law topology in comparison to hierarchical, for all dissemination percentages. Likewise, the comparison for reliable data dissemination to the entire system for both topologies is given in Fig. 5. In comparison to random buffering, the uniformity of the distribution of buffering load is significantly better in the fair-share scheme as given in Fig. 6 for a power-law 1000-node network.

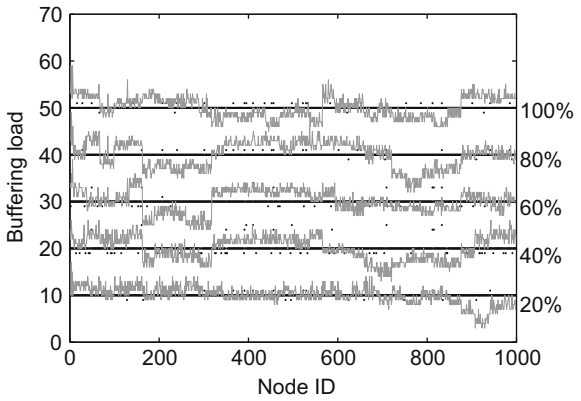

Fig. 4. Uniformity of the fair-share scheme in time for power-law (black dots) and hierarchical (gray dots) topologies.

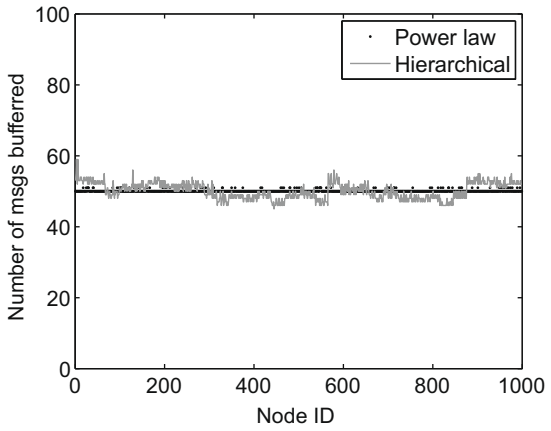

Fig. 5. Comparison of buffering load distribution: hierarchical and power-law topologies.

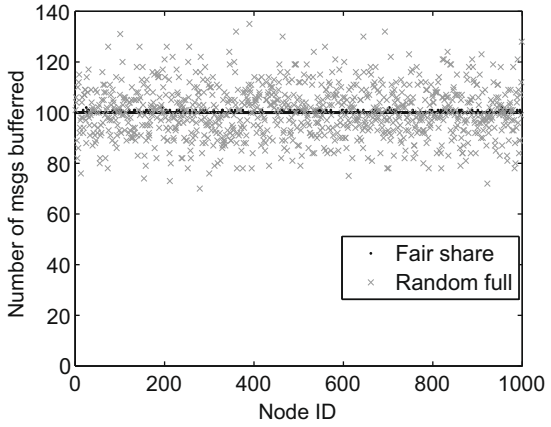

Fig. 6. Comparison of buffering load distribution in large scale: fair-share and random-full approaches.

5.2. Scalability of buffering load

In this part, we investigate the performance of stepwise fair-share buffering as the system size scales up. We compare the buffering load for both hierarchical and power-law topologies. The size of the network is increased from 1000 to 10,000 nodes. A total of 100,000 messages are generated from a single source with rate of 20 msgs/s. The average link delay is 2.5 ms. The short-term buffer size is set to 0, the long-term buffer size is taken to be 10, the TTL parameter is 20 and the fan-out is 5.

The number of messages buffered by each node for the power-law topology is given in Fig. 7 up to system size 6000 for the sake of visual clarity. The analogous result for hierarchical topology is similar and not shown here. As the system size increases, each peer buffers fewer messages. Clearly, this is due to the increase in the total

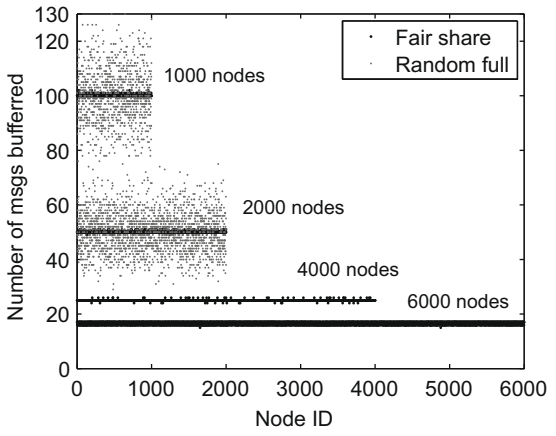

Fig. 7. The buffering load versus peer id for various group sizes in power-law topology.

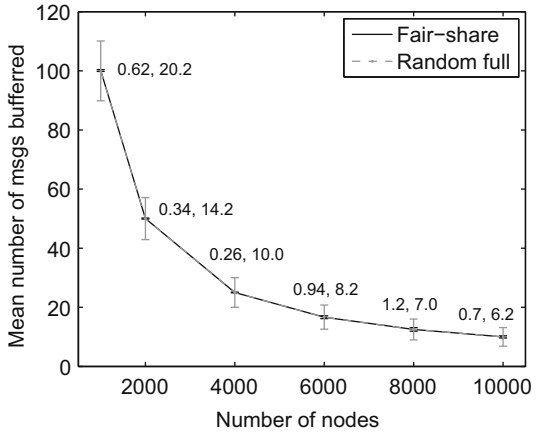

Fig. 8. Mean number of messages buffered versus group size in power-law topology. The pair of numbers denote the width of the error bars for fair-share and random approaches, respectively.

buffering capacity of the network. The uniformity of fair-share holds in all system sizes in both topologies as opposed to random buffering shown for 1000 and 2000 only. The results for the larger sizes with random buffering are similar and are not graphed. What is more, the stepwise fair-share algorithm leads to a more uniform distribution in power-law topology consistently throughout various network sizes compared to the hierarchical topology. This confirms the results of Fig. 5.

In Fig. 8, the mean number of messages buffered per peer is shown up to group size 10,000 for different approaches in power-law topology. Since the mean values are very close, the two line graphs coincide. However, the standard deviations are different and the error bars in the figure show two standard deviations spread of the buffering load distribution among all peers. The hash-based approach results are found to be similar to those of random buffering, but are not shown in the figure for the sake of visual clarity. The hash-based approach does not specifically aim at uniformity of the buffering load. Originally, it is designed for several bufferers. A parameter of the algorithm guarantees at least one bufferer for each message. For comparison purposes, we choose only one of those randomly in our implementation. Thus, the results of hash-based approach turn out to be similar to those of random buffering. The spread of the buffering load is much lower for the stepwise fair-share algorithm. This is true also for the hierarchical topology in view of our simulations.

5.3. The effect of TTL

For the stepwise fair-share algorithm to achieve uniform buffering load, the buffering request must be forwarded sufficiently many steps over the network. The parameter that controls the number of steps is TTL, which is therefore crucial in the implementation. For a relatively large network of size 4000, we analyze its effect on the uniformity of the buffering load. The long-term buffer size is set to 10, and 80,000 messages are sent from the source.

In Fig. 9, the buffering load of each node for various TTL values are given for the power-law topology. The load becomes more uniform as the TTL value increases. The standard deviation of the load on the peers is plotted against TTL values for both hierarchical and power-law topologies in Fig. 10. The standard deviation for hierarchical topology for TTL = 10 is not shown as it is about 100 which is much larger than the other cases. Although the results for hierarchical topology are similar for the behavior of TTL, there is a difference between the topologies for optimizing this parameter. The uniformity of the buffering load increases significantly for TTL value 20 for power-law topology as the standard deviation decreases to about 1 or 2 messages for TTL = 20, 25, 30. On the other hand, such a low value of the standard deviation is obtained only for TTL = 30 in the case of hierarchical topology. The reason for this is that different topologies have different diameters. The diameter is effective in the efficiency of the buffering request delivery to sufficiently far distances in the network in connection to TTL. Also note that there is connection to the whole topology, not just the diameter, as the standard deviation is lower for TTL = 15 and much larger for TTL = 10 in the

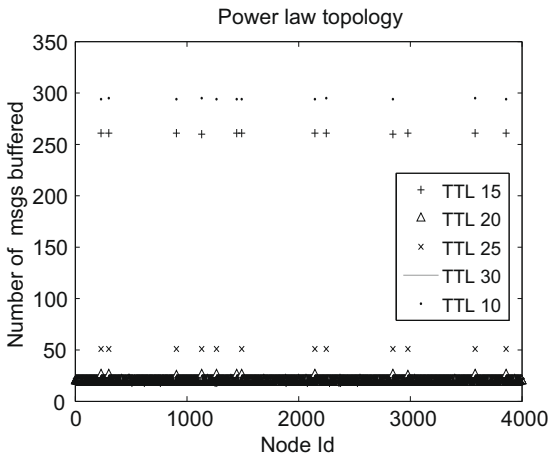

Fig. 9. The buffering load of each node for various TTL values.

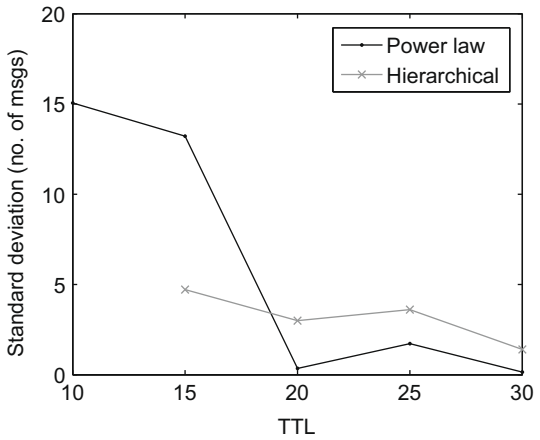

Fig. 10. The standard deviation of buffering load versus TTL values.

hierarchical topology. As a result, our fair-share algorithm balances the buffering load uniformly depending on the topology and the adjustment of the TTL parameter.

5.4. Multiple senders

In this section, we investigate the performance of the stepwise fair-share algorithm in presence of several senders. The power-law topology is considered with 4000 peers. The long-term buffer size is set to 15 and a total of 90,000 messages are sent from all sources. The senders are uniformly distributed over the network. The single sender case is also included for comparison.

We have seen that the buffering load is quite uniform for a single sender. When the number of senders increases, the variance of the buffering load is expected to increase as well. If the buffering algorithm proceeds independently for all sources, then the standard deviation is expected to increase as the square root of the number of sources. This follows from the fact that the variance of the sum of the buffers contributed from all sources is the sum of the variances of the buffering load from each source. Then, the variance would grow linearly with the number of sources and the standard deviation as the square root of it.

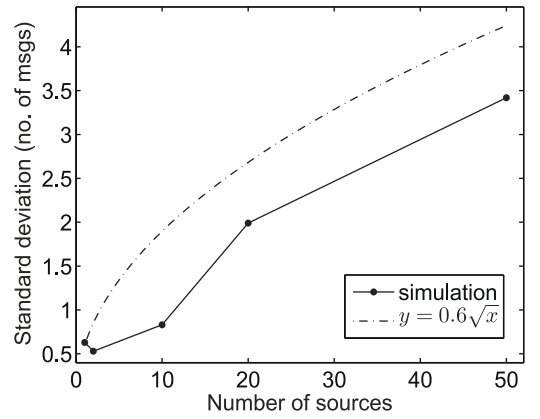

Fig. 11. The standard deviation of number of messages buffered with multiple sources.

The stepwise fair-share buffering algorithm is implemented in its original form also with multiple senders. The results are shown in Fig. 11 where the number of senders is increased from 1 to 50. For a single sender the standard deviation is about 0.6. The standard deviation would be $0.6\sqrt{n}$ if there were n senders behaving independently in terms of buffering. Both this curve and the standard deviation of the buffering load in our simulations are plotted on the same graph. The standard deviation with fair-share increases slowly when compared to the independent behavior case. The algorithm proceeds as before by monitoring the neighbor history information to achieve uniformity regardless of the source of a received message. The peers at the vicinity of a sender will quickly exhaust their long-term buffers. The request is propagated further in the graph at each transmission as long as the TTL parameter permits. However, the number of buffering requests at a given time could be larger in some nodes with multiple senders in comparison to the single source case, depending on the topology and the TTL parameter. This clearly causes the standard deviation to increase, but not as much as in the independent case. The neighbor history check modulates the number of messages buffered for each peer as up and down, which keeps the covariances negative over time and hence lowers the total variance.

6. Gossip-based dissemination results

In this section, evaluation of stepwise fair-share buffering and its comparison with other approaches are given in terms of data dissemination metrics, scalability and link failures. The metrics investigated are reliability, content dissemination time, buffering delay and message delay of the system. These metrics have shown minimal variance through several independent replications of the simulations. That is why we report only the mean values.

6.1. Reliability

Reliability is defined as the ratio of the total number of received messages by peers to the total number of generated messages. Less than full reliability means that some messages have been discarded from all buffers before they are delivered to some of the peers.

The simulations are done on the hierarchical topologies of various sizes and the other parameters are kept the same. In these simulations, short-term buffer size per node is zero, that is, only the long-term buffer is used as the storage for received messages. The message generation rate is 100 msgs/s and the gossip interval is 200 ms. Simulation results show that the minimum buffer requirement decreases as the system size scales up from 500 to 2000 peers as given in [13]. Since the number of nodes increases, the rate of being bufferer per node decreases and the waiting time of a message in the buffer increases. Thus, smaller buffer sizes begin to be sufficient for a message to be delivered by all members if the size of the network gets larger. The results in Fig. 12 indicate the reliability of the data

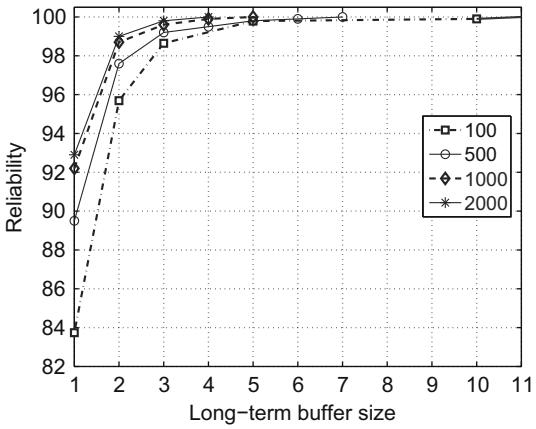

Fig. 12. Reliability as a function of long-term buffer size.

dissemination as a function of the long-term buffer size for different system sizes N from 100 to 2000. The minimum buffer sizes for full reliability can be observed from this figure as well.

6.2. Dissemination time

Content dissemination time is the time that passes for dissemination of the content to all peers from the start to end including the buffering phase.

Stepwise fair-share scheme is compared with the hash-based approach [1], probabilistic buffering [12] as well as random buffering [20]. The dissemination times in a 1000 node hierarchical network scenario are given in Fig. 13. All 50,000 messages are generated from a single source and the message generation rate is 20 msgs/s so that all messages are generated in 2500 s. The gossip interval is set to 200 ms. In the random and hash-based buffering methods, every peer has the full view of the system. As inferred from Fig. 13, dissemination times of stepwise fair-share and probabilistic buffering are close to that of random buffering, even though in the first two every peer has only partial membership information on which a buffering stage is based. In random buffering, the bufferers are determined right away, among all members, when a message is generated and the message is directly sent to the bufferers. Therefore, we infer that the buffering stage is relatively fast in stepwise fair-share and probabilistic buffering approaches on the average. On the other hand, the hash-based approach has a higher dissemination time. In this approach, a peer decides to be a bufferer for a message when it receives the message through gossiping eventually. The dissemination time is larger probably due to quite delayed delivery of some messages to some of the peers. There is no significant difference between the probabilistic buffering and fair-share buffering in terms of dissemination time. Basically, the last message is sent out from the source at time 2500 s, and is received by all nodes in the next few gossip rounds for both approaches.

For topology comparison, Fig. 14 depicts the content dissemination times for fair-share buffering in the case of power-law and hierarchical networks. Consistent with the distribution of buffering load reported in the previous section, fair-share buffering achieves a more uniform distribution of dissemination times at peers with power-law topology in comparison to hierarchical.

6.3. Message delay and buffering delay

Message delay is the duration between the generation of the message from the data source and the delivery of it by a receiver node. *Buffering delay* is the time required to find a bufferer that indicates the signaling cost associated with our approach.

Comparison of the average message delays on 1000-node hierarchical topologies is given in Fig. 15. Stepwise fair-share and probabilistic buffering approaches lead to slightly higher average message delays per node, in comparison to hash-based and random buffering. This is due to the fact that the former approaches use additional time to determine the bufferer of each data message disseminated. However, when distributing a large content consisting of thousands of messages, bufferer determination and

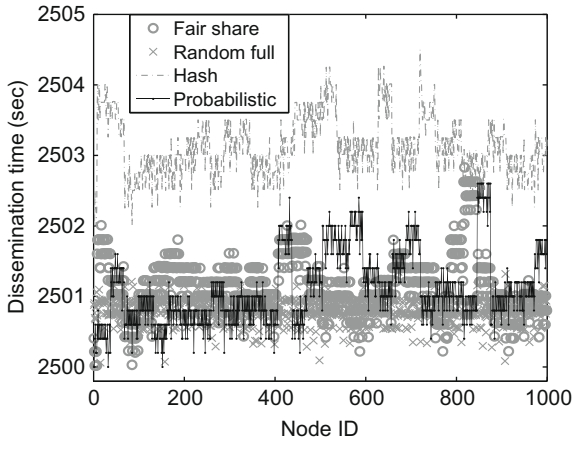

Fig. 13. Comparison of content dissemination times.

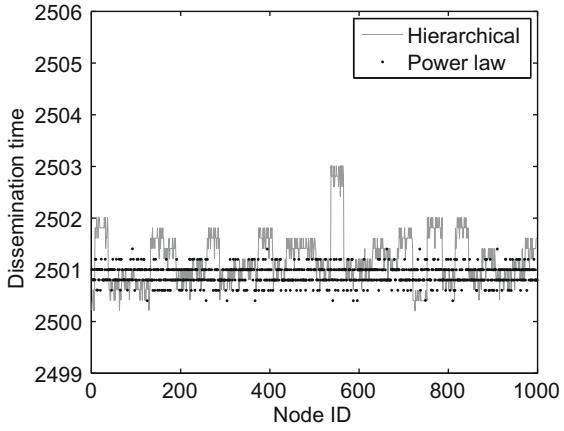

Fig. 14. Content dissemination times: power-law and hierarchical topologies.

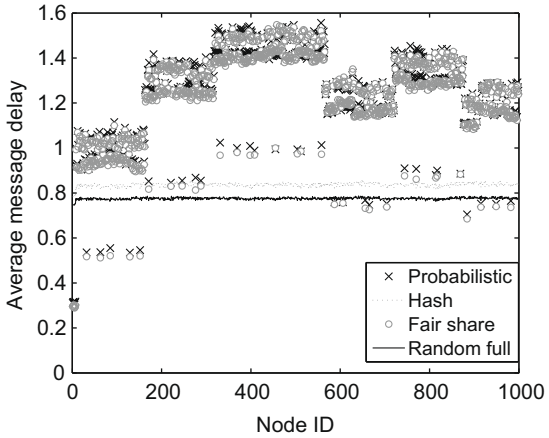

Fig. 15. Comparison of average message delays.

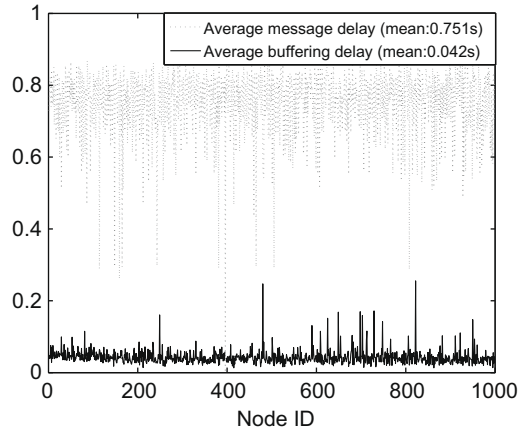

Fig. 17. Average buffering delay in comparison with message delay in power-law topology.

message dissemination phases take place concurrently, and total dissemination time for the content is not affected adversely as discussed for the results of Fig. 13. We also conclude in the hash-based approach that the last delivered messages that cause the dissemination time to be high in Fig. 13 must be quite rare because the mean delay is relatively low as shown in Fig. 15 for this approach. We have also observed that uniform buffering load distribution of fair-share leads to a uniform long-term buffering time distribution among nodes. On the other hand, message delays and content dissemination times vary from node to node as shown in Figs. 13 and 15 due to the partial view of each peer during data dissemination.

We further examine the average message delay behavior of stepwise fair-share buffering in the case of power-law topology for comparison with the hierarchical case. As shown in Fig. 16, average message delays in power-law networks are lower and follow a more uniform distribution compared to hierarchical networks.

We also investigate the signaling cost to find a bufferer for each data message in our approach. In 1000-node network scenarios, 50,000 messages are generated from a single source and the message generation rate is 20 msgs/s. We explicitly measure the time required to find a bufferer (buffering delay) in comparison to data message delay. Fig. 17 shows average buffering and message delays for all 50,000 messages disseminated to 1000 peers in a power-law topology. We measure average message delay over all peers as 0.751 s, and average buffering delay as 0.042 s. Based on these, percentage of buffering delay within data message delay is 5.7% on average for all peers. Similar results are also obtained for hierarchical networks. These results indicate the low cost associated with finding bufferers in stepwise fair-share approach.

6.4. Scalability of dissemination

We investigate data dissemination metrics for the buffering approaches as the system size scales up. The hierarchical topology of size 1000 to 10,000 nodes is used to compare with the other buffering approaches. The number of transit nodes and the stub domains is increased with the system size while the average number of stub nodes in

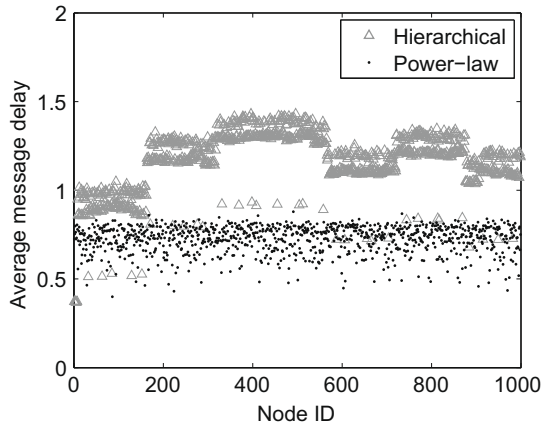

Fig. 16. Average message delays: power-law and hierarchical topologies.

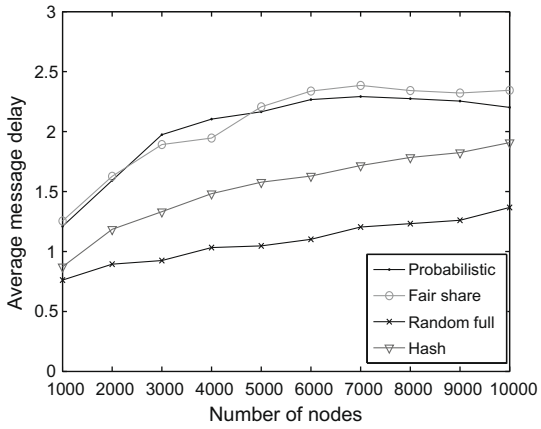

Fig. 18. Average message delays as a function of group size.

each domain is 30. In these simulations, the message generation rate is 100 msgs/s, gossip interval is 200 ms and 5,000,000 messages are disseminated to the whole network.

Fig. 18 shows the comparison of average message delays for stepwise fair-share buffering with the other approaches as a function of network size. Stepwise fair-share and probabilistic buffering approaches result in higher average message delays per node when compared to hash-based and random buffering. This is due to the fact that fair-share and probabilistic buffering work with local neighbor information (that is, each peer has only a partial view of the system) and they use a bufferer selection phase separate from data dissemination. However, bufferer determination and data dissemination phases take place *concurrently* for several data messages, and total dissemination time for the content is not affected adversely as depicted in Fig. 19 for all network sizes.

In Fig. 19, the lowest content dissemination time occurs with the random approach which serves as a baseline for comparison. In this approach, since the sender is assumed to have the full membership knowledge, it immediately selects bufferers at random among all peers. On the other hand, there is no need to have full membership information in stepwise fair-share as well as probabilistic buffering at the expense of only slightly higher average message delays. Hash-based method leads to a higher dissemination time since a peer becomes a bufferer only when it receives the message through gossiping eventually. In other words, there is no immediate bufferer selection phase separate from the data dissemination in contrast with stepwise fair-share buffering. This may lead to the set off in time due to dissemination of the last fraction of the messages in hash-based buffering. In all cases, the dissemination time increases only logarithmicly with the system size due to the epidemic dissemination algorithm. The delay increases in the same fashion also for power-law topology.

Note that, stepwise fair-share buffering works with local information at each peer. This makes it applicable to large-scale systems at low cost. In contrast, hash-based and random buffering need an approximation of the full membership information that may be difficult to achieve in dynamic peer arrivals and large-scale systems due to the cost of maintaining this information. In our simulation models for hash-based and random buffering, we assume that peers have the full view of the system, and we also do not include the cost associated with maintaining an approximation of the entire membership.

6.5. Failure cases and multiple bufferers

In this group of experiments, the performance of epidemic dissemination with stepwise fair-share buffering is investigated under congestion and/or link failures in the network. In either case, a data message may not reach its destination and the nodes could simply request a retransmission from the sender if the single bufferer is not sufficient. Since this could have an implosion effect at the sender, multiple bufferers are used to ensure the recovery.

Certain link drop probabilities are assigned for each message traversing the network to simulate congestion and failures. We assume a uniform link drop probability in the network. Explicitly, any message traveling on a given link has a chance of not being delivered to the destination node with probability p. In the results given in Figs. 20 and 21, a 1000 node hierarchical network topology is used for message dissemination and 50,000 messages are disseminated from a single source. The short-term buffer size of a peer is set to zero in order to observe the long-term buffer performance. The message generation rate is 100 msgs/s and the gossip interval is 200 ms. In the first result given in Fig. 20, the link drop probability of the network is increased from 0.01 to 0.05. Clearly, the minimum number of bufferers increases as the link drop probability increases. Fig. 21 gives the minimum buffer size needed for reliable dissemination with the number of bufferers b of a message set to 6. The minimum buffer size is 5 messages if the drop probability is 0.01 which is consistent with Fig. 20. On the other hand, this increases to 11 if the drop probability goes up to 0.05. Note that the minimum buffer size remained constant for link drop probabilities 0.04 and

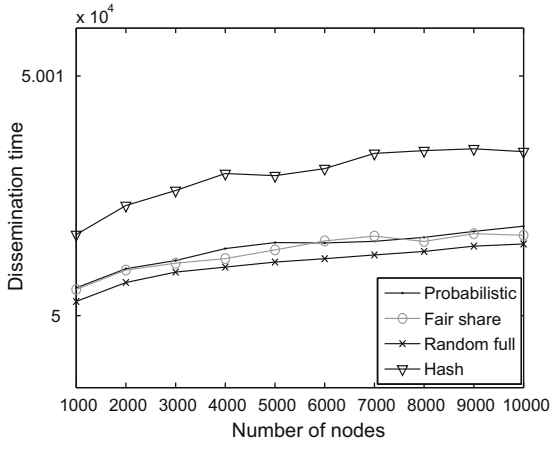

Fig. 19. Dissemination time as a function of group size.

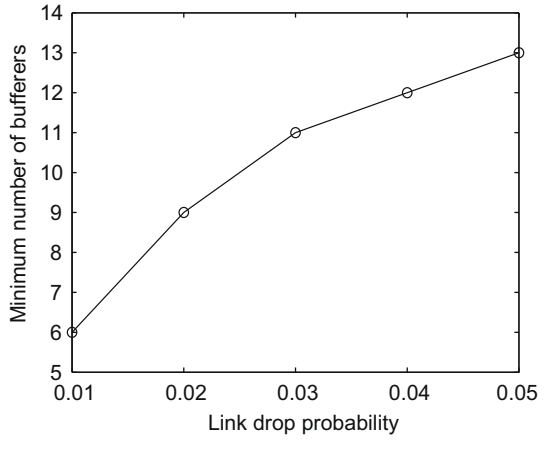

Fig. 20. Minimum number of bufferers for reliability as a function of link drop probability.

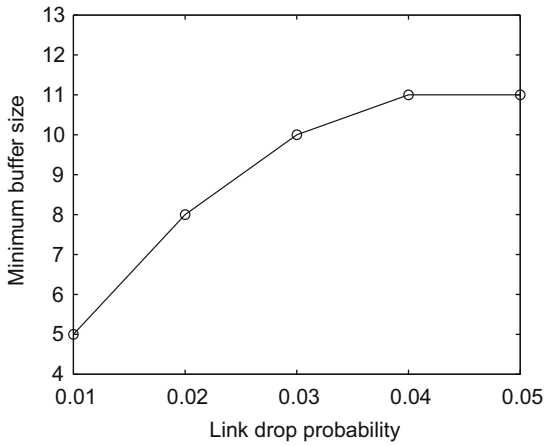

Fig. 21. Minimum buffer size needed for reliability as a function of link drop probability.

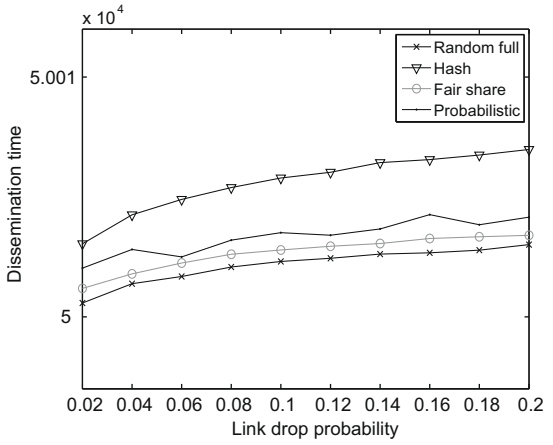

Fig. 22. Comparison of dissemination time as a function of link drop probability.

0.05. Although this is somewhat misleading, the graph would show an increase for larger drop probabilities.

Fig. 22 shows the comparative behavior of the buffering approaches as the drop probability of the links increases. In these simulations, 500,000 messages are disseminated to the network, network size is 1000 peers, message generation rate is 100 message/s, gossip interval is 200 ms. Short-term buffer size is 10, long-term buffer size is 20 and the number of bufferers per message is five so that a reliable dissemination is achieved for all the scenarios. Fig. 22 shows that the dissemination time increases as p increases and the different approaches behave analogously as in Fig. 19. Likewise, we have observed that the average message delay increases slightly as a function of the link failure rate.

7. Analytical bound for reliability

If the buffering time of a message is longer than its dissemination time, it can be safely discarded. In a random environment, the probability of this event represents the reliability of dissemination. In this section, a Markov chain formulation is considered for finding the distribution of the dissemination time. On the basis of this, we find an analytical expression for reliability for each buffer size and compare it with simulations.

Let α denote the message generation rate of the source node, λ the rate of receiving a new message to be buffered, n the number of nodes in the system, B the size of the long-term buffer of a node (namely the number of messages in the long-term buffer when the buffer is full) and T the time that passes for one message to reach all the nodes. Our aim is to find the minimum buffer size B that guarantees reliable delivery of a message to all nodes. We can approximate the expected time between two buffer updates by $\frac{1}{\lambda}$. As a result, the average waiting time of one message in the long-term buffer of a node is $W = \frac{B}{\lambda}$ when the buffer is full as in the steady state. By virtue of the results obtained in Section 3, it can be assumed that the load of being a bufferer is distributed uniformly to all n nodes. Thus, the rate of being a bufferer λ can be approximated as $\frac{\alpha}{n}$. Therefore, the average waiting time becomes

$$W = \frac{Bn}{\alpha}. \tag{1}$$

To provide perfectly reliable dissemination, the waiting time of one message in the long-term buffer of a node should be greater than the time T that passes for one message to reach all the nodes. That is, we must have $T \leqslant W$ in order to have a reliable dissemination. By approximating the waiting time in the buffer as a deterministic quantity given by its mean (1), we consider $P(T \leqslant W) = P\left(T \leqslant \frac{Bn}{\alpha}\right)$. That is, the cumulative distribution function (cdf) of T given by $F\left(\frac{Bn}{\alpha}\right) = P\left(T \leqslant \frac{Bn}{\alpha}\right)$ is computed and we require this value to be close to 1 for high reliability.

In order to find F, a Markov chain model is used for epidemic dissemination of messages as given in [21]. We use the push-based model rather than pull as its analytical model can be used for larger fan-out values as well. That is why the simulations in this section are performed with the push mechanism for comparison purposes. In the pull model, an infectious peer selects a susceptible peer randomly and sends its digest message to a susceptible peer. In the push model, the process is the reverse namely a susceptible peer selects an infectious peer randomly and sends its digest message to the infectious peer. The qualitative behaviors are the same, and hence the analysis of this section is based on push approach for convenience.

The states of the Markov chain $(X_t : t = 1, 2, \ldots)$ are defined as the number of infected nodes for one fixed message in the system at time t, taking integer values between 1 and n. This is an absorbing Markov chain and the absorbing state is equal to the total number of nodes n. Therefore, we need to find the time to absorption which is the time that passes for the Markov chain to reach from an initial state to an absorbing state. The cdf of the time to absorption is given by

$$F(k) = \mu_0 + 1 - \mu Q^k e, \tag{2}$$

where k denotes the time to absorption μ_n and μ denote the probability and the probability (row) vector that the

Markov chain starts at the absorbing and transient states, respectively, e is the vector consisting of all 1's and Q is the portion of the transition matrix P of X that corresponds to the transient states [22].

In the push model, the probability that there will be j infected nodes at the next stage when there are k infectious peers at present is found as [21,23]:

$$P_{kj} = \binom{n-k}{j-k} \times \left(1 - \frac{\binom{n-k-1}{f}}{\binom{n-1}{f}}\right)^{j-k} \times \left(\frac{\binom{n-k-1}{f}}{\binom{n-1}{f}}\right)^{n-j}$$

for $j = 1, 2, \ldots, n-k$ and fixed f where $n-1$ is the number of nodes a peer sends gossip to, excluding itself. Using P, we construct the matrix Q. The cdf F of (2) is evaluated at $k = \lfloor \frac{Bn}{\alpha} \rfloor$ because k should be an integer and the floor function $\lfloor \cdot \rfloor$ gives the correct lower bound for reliability rather than the ceiling function $\lceil \cdot \rceil$. In our model, μ is a unit vector as the chain starts at state 1. Then, the following result is obtained:

$$r = F\left(\left\lfloor \frac{Bn}{\alpha} \right\rfloor\right) = 1 - [10\ldots0]Q^{\lfloor \frac{Bn}{\alpha} \rfloor}[1\ldots1]^T = 1 - \sum_{j=1}^{n-1} Q_1 j^{\lfloor \frac{Bn}{\alpha} \rfloor}.$$

Using this information the minimum buffer size B needed for reliable dissemination is computed for each level of reliability r.

The results obtained from the analytical model are compared with the results obtained from the simulation of stepwise fair-share buffering scheme coupled with epidemic dissemination. Reliability is estimated from the simulations as the ratio of the total number of received messages by peers to the total number of generated messages. There are 100 nodes in the system ($n = 100$) and the message generation rate is 100 msgs/s. The average of 100 independent replications are reported as the estimate of $P(T \leqslant W)$.

In Fig. 23, the reliability of the scheme with different buffer sizes in analytical and simulation results is compared when the fan-out is 1. It can be inferred that the analytical values give a lower bound for reliability. In other words, the simulations achieve higher reliability for a given buffer size. This may be due to the discrepancy between the simulation approach and the analytical model. In the simulations, there is partial view and the peers have a limited short-term buffer. Some messages are obtained from the short-term buffers and the others are recovered from the long-term buffers. In the analytical model, the infection is assumed to occur in a similar fashion when a susceptible contacts with an infectious peer. The missing information is identified and it is assumed to be recovered from the infectious peer which has an infinite size (short-term) buffer. In terms of the required time, this is no different from obtaining the message from the long-term bufferer. However, the difference with the analytical model is that it is based on full knowledge of the system. Partial membership knowledge and digest exchange seem to speed up the dissemination process in the simulations. On the other hand, the analytical results are very close to the simulation results for the larger reliability values. Therefore, the analytical model can be used for designing a highly reliable system.

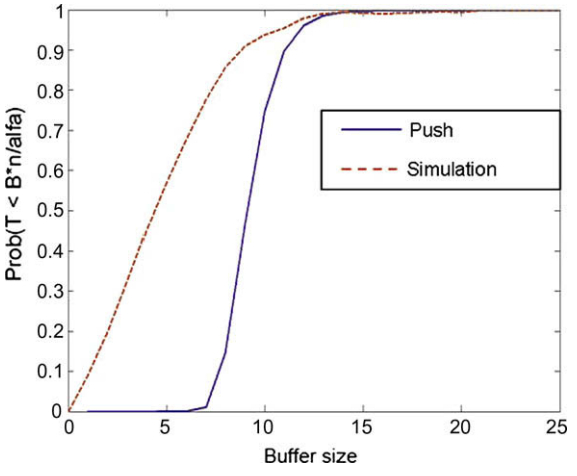

Fig. 23. Reliability versus buffer size for model and simulation ($f = 1$).

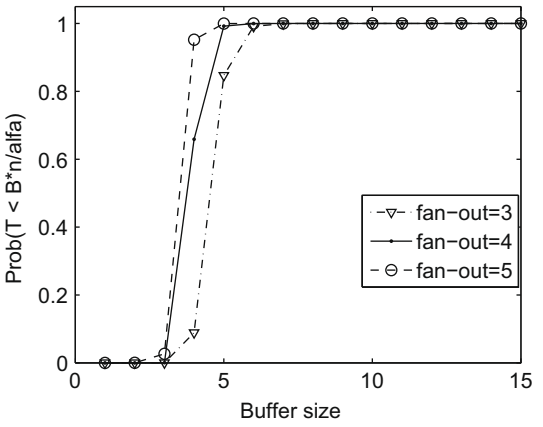

Fig. 24. Reliability versus buffer size: analytical results for different fan-out values.

When the comparison is done with fan-out 3, the results are similar to the $f = 1$ case. However, the reliability is achieved with a smaller buffer size since increasing the fan-out hastens the epidemic spread. So, a message reaches all nodes in a shorter time period and smaller buffer size becomes enough in this case. In Fig. 24, the reliability computed by the analytical model is compared for different fan-out parameters. As expected, if fan-out increases, the same buffer size provides more reliable dissemination.

8. Conclusions

We have studied the buffer management problem in support of large-scale gossip-based peer-to-peer data dissemination services. A robust scheme named stepwise fair-share buffering has been proposed, modeled and analyzed. It achieves a uniform load of buffering throughout

the network with only partial knowledge of peers about the system. It is simple and functions independently of the underlying network topology. Our approach reduces the memory usage since only a small subset of the peers is chosen as bufferers for each message. As a result, the efficiency of content dissemination is improved. Furthermore, it is applicable to large-scale scenarios, provides reliable delivery and is adaptable to dynamic join and leaves to the system.

Separate evaluations of bufferer selection and gossip-based dissemination are presented to demonstrate the buffering efficiency through simulations. For the buffering analysis, uniformity of the buffering load, its scalability, the effect of TTL parameter and multiple senders scenarios have been investigated. Furthermore, the evaluation of our approach in comparison with other buffering schemes are given in terms of data dissemination metrics, scalability and link failures. The metrics investigated are reliability, content dissemination time, buffering delay and message delay of the system.

The hash-based buffering scheme, probabilistic scheme and a completely randomized approach with full membership information have been used for comparison. Several power-law and hierarchical overlay topologies were considered. We have found that stepwise fair-share buffering performs well and is scalable for large networks also in the case of failures in the links. The buffer sizes and number of bufferers have been determined to guarantee reliable delivery.

Analytical results for reliability of epidemic dissemination as a function of buffer sizes and the number of bufferers have been derived. These results are based on a Markov chain analysis and are evaluated numerically. Comparison with simulations of stepwise fair-share scheme shows that the analytical model provides a good lower bound for reliability. For high levels of reliability values, the bounds are very close to the simulation results.

As future work, one can include link failures in the underlying network topology to the analytical model. The minimum number of bufferers required for reliability would be of interest. On the other hand, identifying the true overlay topologies more precisely would be of interest. It has been shown recently in [27] that Gnutella network exhibits the clustering and short path lengths of a small world network rather than power-law scaling. Finally, trace-driven data could be used instead of synthetic topologies which can be overly simplified.

Acknowledgement

Research of the first author is supported by TUBITAK (The Scientific and Technical Research Council of Turkey) under CAREER Award Grant 104E064.

References

[1] Ö. Özkasap, R. van Renesse, K.P.Birman, Z. Xiao, Efficient buffering in reliable multicast protocols, in: Proceedings of the First International Workshop on Networked Group Communication (NGC' 99), Pisa, Italy, November 1999, pp. 188–203.
[2] K. Yamamoto, Y. Sawa, M. Yamamoto, H. Ikeda, Performance evaluation of ACK-based and NAK-based flow control mechanisms for reliable multicast communication, IEICE Trans. Commun. E84-B (8) (2001) 2313–2316K.
[3] L. Rodrigues, S. Handurukande, J. Orlando, R. Guerraoui, A.M. Kermarrec, Adaptive gossip-based broadcast, in: IEEE International Conference on Dependable Systems and Networks (DSN), 2003.
[4] Z. Xiao, K.P. Birman, R. van Renesse, Optimizing buffer management for reliable multicast, in: Proceedings of the International Conference on Dependable Systems and Networks (DSN'02), Washington, DC.
[5] J.F. Paris, J. Baek, A heuristic buffer management and retransmission control scheme for tree-based reliable multicast, ETRI J. 27 (1) (2005).
[6] G. Rhee, I. Rhee, Message stability detection for reliable multicast, in: Proceedins of the 19th IEEE Conference on Computer Comm. (INFOCOM 2000), New York, USA, March 2000, pp. 814–823.
[7] M. Costello, S. McCanne, M. Yamamoto, H. Ikeda, Search party: using randomcast for reliable multicast with local recovery, in: Proceedings of the 18th IEEE Conference on Computer Comm. (INFOCOM'99), New York, USA, March 1999, pp. 1256–1264.
[8] J. Pereira, L. Rodrigues, M. Monteiro, R. Oliviera, A.M., Kermarrec, Network friendly epidemic multicast, in: IEEE International Symposium on Reliable Distributed Systems, 2003.
[9] C. Lindemann, O. Waldhorst, Modelling epidemic information dissemination on mobile devices with finite buffers, in: SIGMETRICS, 2005.
[10] R. van Renesse, K.P. Birman, W. Vogels, Astrolabe: a robust and scalable technology for distributed systems monitoring, management, and data mining, ACM Trans. Comput. Syst. 21 (2) (2003) 164–206.
[11] A.J. Demers et al., Epidemic algorithms for replicated database maintenance, in: Proceedigns of the 6th Annual ACM Symposium on Principles of Distributed Computing, ACM Press, 1987, pp. 1–12.
[12] E. Ahi, M. Çağlar, Ö. Özkasap, Stepwise probabilistic buffering, in: International Conference on Bio Inspired Models of Network, Information and Computing Systems (Bionetics), December 11–13, 2006, Cavalese, Italy.
[13] E. Ahi, M. Çağlar, Ö. Özkasap, Stepwise fair-share buffering underneath bio-inspired P2P data dissemination, in: 6th International Symposium on Parallel and Distributed Computing, July 2007, Hagenberg, Austria.
[14] M. Faloutsos, P. Faloutsos, C. Faloutsos, On power-law relationships of the Internet topology, in: Proceedings of ACM SIGCOMM, 1999.
[15] A. Medina, A. Lakhina, I. Matta, J. Byers, BRITE: Universal topology Generator from a Users Perspective, Technical Report, BUCS-TR-2001-003, April 12, 2001, Boston University. <http://www.cs.bu.edu/brite/publications/usermanual.pdf>).
[16] A. Medina, A. Lakhina, I. Matta, J. Byers, BRITE: an approach to universal topology generation, in: Proceedings of MASCOTS, Cincinnati, OH, August 2001.
[17] B. Zhang, T.S.E. Ng, A. Nandi, R. Riedi, P. Druschel, G. Wang, Measurement based analysis, modeling, and synthesis of the internet delay space, in: Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement, October 2006.
[18] gt-itm, <http://www-static.cc.gatech.edu/fac/Ellen.Zegura/graphs.html>.
[19] N.T.J. Bailey, The Mathematical Theory of Infectious Diseases and its Applications, second ed., Hafner Press, 1975.
[20] A. Alagöz, E. Ahi, Ö. Özkasap, Network awareness and buffer management in epidemic information dissemination (poster paper), in: ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC), July 2005, Las Vegas.
[21] M. Çağlar, Ö. Özkasap, A chain-binomial model for pull and push-based information diffusion, in: IEEE ICC, June 2006, Istanbul.
[22] M.F. Neuts, Matrix-geometric Solutions in Stochastic Models, The John Hopkins University Press, 1981.
[23] Ö. Özkasap, E. Ş.Yazıcı, S. Küçükçifçi, M. Çağlar, Exact performance measures for peer-to-peer epidemic information diffusion, in: Lecture Notes in Computer Science, vol. 4263, ISCIS 2006.
[24] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, H. Balakrishnan, Chord: a scalable peer-to-peer lookup service for internet applications, in: Proceedings of the ACM SIGCOMM Conference, San Diego, CA, USA, August 2001.
[25] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker, A scalable content-addressable network, in: Proceedings of the ACM SIGCOMM Conference, San Diego, CA, USA, August 2001.
[26] B. Zhao, J. Kubiatowicz, A. Joseph, Tapestry: an infrastructure for fault-tolerant wide-area location and routing, Comput. Sci. Div., Univ. California, Berkeley, Tech. Rep. UCB/CSD-01-1141, 2001.

[27] D. Stutzbach, R. Rejaie, S. Sen, Characterizing unstructured overlay topologies in modern P2P file-sharing systems, IEEE-ACM Trans. Network. 16 (2) (2008) 267–280.

Oznur Ozkasap received her M.S. and Ph.D. degrees in Computer Engineering from Ege University, in 1994 and 2000, respectively. From 1997 to 1999, she was a graduate research assistant at Cornell University, Department of Computer Science, where she completed her Ph.D. dissertation. She is currently an assistant professor in Department of Computer Engineering at Koc University which she joined in 2000. Her research interests include distributed computing systems, multicast protocols, peer-to-peer systems, bio-inspired distributed algorithms and computer networks.

Mine Caglar received her B.S. and M.S. degrees in Industrial Engineering from Middle East Technical University and Bilkent University, respectively. She received a Ph.D. degree in Statistics and Operations Research from Princeton University in 1997. She worked as a post-doctoral research scientist at Bellcore in Morristown, in Network Design and Traffic Research Group during 1997–1998. She is currently an associate professor in Department of Mathematics at Koc University which she joined in 1999. Her current research interests include stochastic modeling in telecommunication networks; in particular traffic modeling, epidemic algorithms and queuing.

Emrah Cem received his B.S. in Computer Engineering from Koc University in 2008. He is currently working towards the M.S. degree in Computer Engineering at Koc University. His research interests include mobile ad hoc networks, peer-to-peer and distributed systems.

Emrah Ahi received his B.S. in Mathematics from Middle East Technical University in 2004 and M.S. degree in Computational Sciences and Engineering from Koc University in 2007. He currently works at Risk Software Technologies, Inc., in Istanbul.

Emre Iskender received his B.S. in Electrical and Electronics Engineering from Bilkent University in 2006. He is currently working towards the M.S. degree in Computer Engineering at Koc University. His research interests include peer-to-peer and distributed systems.