Flash Data Dissemination in Unstructured Peer-to-Peer Networks

Antonis Papadimitriou and **Alex Delis**
University of Athens, Athens, 15784, Greece
{anthony, ad}@di.uoa.gr

Abstract

The problem of flash data dissemination refers to spreading dynamically-created medium-sized data to all members of a large group of users. In this paper, we explore a solution to the problem of flash data dissemination in unstructured P2P networks and propose a gossip-based protocol, termed Catalogue-Gossip. Our protocol alleviates the shortcomings of prior gossip-based dissemination approaches through the introduction of an efficient catalogue exchange scheme that helps reduce unnecessary interactions among nodes in the unstructured network. We provide deterministic guarantees for the termination of the protocol and suggest optimizations concerning the order with which pieces of flash data are assembled at receiving peers. Experimental results show that Catalogue-Gossip is significantly more efficient than existing solutions when it comes to delivery of flash data.

1. Introduction

The problem of rapid data dissemination in peer-to-peer (*P2P*) networks is now recognized as the key element to effectively address many applications including acquisition of real-time weather information, announcement of traffic-conditions, publication of stock quotes and proliferation of news and emergency alerts. Sizeable groups of users often organized in *P2P* overlays are now increasingly interested in acquiring dynamic information in order to make timely decisions [9]. In the aforementioned application domains, new data has to reach large numbers of users and/or customers as quickly as possible. This specialized form of dissemination is termed *flash data dissemination* and refers to quickly spreading dynamically-created medium-size data to all members of a potentially large group of users. Flash dissemination is very different from rudimentary file transfer approaches as the core requirement is that all participants have to obtain dynamic data fast and that the originator of the information might be any node in the *P2P*-network. Occasionally, flash data dissemination has to proceed under adverse conditions in which networks suffer from frequent node node arrivals/departures or feature nodes prone to failure with high probability during unexpected events. Such conditions appear when either emergency situations are in development or time-critical operations are involved including deployment of physical resources, handling of micro-payments among users of a community, and resumption of multi-player games.

In a scenario that entails fast dissemination of earthquake data, the objective is to efficiently propagate information regarding ongoing seismic activity to many sites so that data can be exploited for both scientific evaluation and for issuing necessary public alerts. Another equally important application is that of detecting heat and humidity while monitoring a forest area. The goal here would be to detect bush fires and gather information that would help design a strategy to combat them. The main challenge that these systems face is that timely data dissemination should be guaranteed even in light of catastrophic events that may damage computing equipment in multiple sites. In a multi-player game scenario, there is often a global game state that all peers should be aware of. This state information should be spread quickly by using a flash dissemination scheme, even if users constantly join or depart the game. Lately, unstructured *P2P*-networks are increasingly used to implement such systems [4, 12], due to their unique characteristic of ensuring reliable operation even in adverse network conditions with minor maintenance overhead. The main advantage of unstructured *P2P*-networks is that they impose no pre-fixed overlay on the participating nodes and by and large rely on random communication among peers for their operation. These two features allow for the flexible realization of the two fundamental operations –information search and dissemination– yielding an network capable of overcoming unexpected failures.

In this paper, we propose a new protocol for flash data dissemination in unstructured *P2P*-networks. The protocol features a novel gossip-based dissemination approach that reduces communication overheads among peers and thus, it achieves faster dissemination of data reports. Our dissemination algorithm termed *Catalogue-Gossip*, is based on a succinct structure called *Catalogue* that allows for efficient node interactions. Additionally, we develop a specialized termination discovery mechanism that provides deterministic guaranties for the reliability of the dissemination. We also discuss a more fault-tolerant variant of the protocol which implements a decentralized file chunk selection policy to increase resistance to failures on the originator of a disseminated resource.

2. Related Work

Overlay structure information is absent in unstructured *P2P*-networks and algorithms developed for such systems rely on random message exchanges. In this context, gossip-based protocols have been widely used in unstructured *P2P*-networks mainly due to their simplicity and increased fault tolerance. Moreover, results that have been obtained through probabilistic analysis point into the efficiency of gossip-based protocols to spread reports [8].

Simple-gossip has been applied in a number of fields including databases [3], network management [11] and information dissemination [5]. This protocol is initiated by a node that intends to disseminate a resource by selecting a fixed number –called fan-out– of random peers to forward the content in question. Every receiving peer does the same, unless it has already "received" the resource, in which case it ceases the forwarding. Unfortunately, simple-gossiping approaches focus on reliability and fault-tolerance and ignore the aspect of efficiency in terms of speed of dissemination. In fact, forwarding resources to randomly selected peers may significantly degrade performance, as peers may receive duplicate copies of resources. Another problem that there is always a possibility that a peer does not receive the disseminated information at all (i.e., the delivery is only probabilistic).

Crew [4] was the first gossip-based protocol which explicitly dealt with fast dissemination of data in unstructured *P2P*-networks. In order to avoid duplicate transfers, *Crew-Gossip* operates in a pull mode as it involves having all nodes eagerly request data they are missing, from random peers. To make all nodes aware of missing chunks, a phase of meta-data spreading via *Simple-Gossip* precedes the main dissemination process. As requesting peers know which files they need, duplicate data transfers are avoided. Nevertheless, *Crew-Gossip* has a weakness which degrades its performance. Specifically, in the beginning of the dissemination nearly all requests to random peers are unsuccessful, as no peer –apart from the originator– has any data resource to share. The existence of many unsuccessful messages offers ground for improvement when it comes to offering a more optimized flash data dissemination protocol. Our *Catalogue-Gossip* approach tackles the above shortcomings by introducing catalogue-based interactions among peers thus avoiding many redundant communications. In addition, we offer a decentralized implementation of a data chunk selection policy to increase fault-tolerance. To deal with the problem of probabilistic delivery, we suggest a mechanism to ensure deterministic delivery of resources to all peers in the network.

3. The Catalogue-Gossip Protocol

The key objective of the *Catalogue-Gossip* is to distribute content of arbitrary format and size to all peers constituting the network. *Catalogue-Gossip* relies on an underlying *Membership Protocol* [6, 7] for building up the unstructured network. A membership management protocol is responsible for creating consistent neighbor views at every peer, so that when a node selects randomly a peer from its view, it would be equivalent to randomly selecting from the entire set of network nodes. Moreover, gossip-based membership protocols are designed to deal efficiently with dynamic network conditions, so that the node views are consistent even in face of high churn or failure rate. Hence, membership protocols allow flash data dissemination approaches to focus on improving dissemination speed rather than handling dynamic aspects of the network.

With an unstructured network in place, *Catalogue-Gossip* disseminates resources by having all nodes spread a meta-data message called *Catalogue* to random peers in the system. A catalogue contains a list of data available for downloading from its owner. In the sections that follow, we outline the structures used to implement the protocol and explain how *Catalogue-Gossip* works towards achieving fast content dissemination.

3.1. Data Structures

To handle content of arbitrary size and format, the protocol treats every file in an uniform manner. It splits files into several parts or *data chunks* and disseminates each such chunk independently. A file is disseminated to a particular peer, if the peer in question has received all parts comprising the resource. The recipient peer is responsible for reconstructing the original file from its constituent components. To distinguish data chunks, the uploader of a file has to provide a unique *id* for each data chunk produced by using the *MD5*-algorithm [10].

Every peer maintains two structures which are necessary for the operation of the *Catalogue-Gossip* protocol as Figure 1 depicts. The first is a table containing all chunks that have been downloaded by a peer thus far. The table helps the node ascertain which chunks are still missing. To this end, the node can fetch missing parts from other peers. Moreover, the table of *available chunks* is used as an announcement (via the *Catalogue* messages) to other peers, TABLE of available chunks

LocalID	Filename	Chunk Sequence	ChunkID
1	map.jpg	1	md5{chunk data}
2	map.jpg	2	md5{chunk data}
3	map.jpg	8	md5{chunk data}
4	event.dat	3	md5{chunk data}

FREQUENCY COUNTERs

map.jpg

LocalID	Chunk Sequence	Frequence Estimate
1	1	f_1
2	2	f_2
⋮	⋮	⋮
missing	10	f_{10}

event.dat

LocalID	Chunk Sequence	Frequency Estimate
missing	1	f_1
missing	2	f_2
4	3	f_3

Figure 1. *Catalogue-Gossip* **Structures**

so that the latter may know which chunks are available for downloading from that specific peer. The second structure shown in Figure 1 is a set of of frequency counters whose objective is to offer an estimation on how frequent each specific chunk is in the entire *P2P*-system. In addition to the aforementioned two structures, *Catalogue-Gossip* peers trade *Catalogue* messages which contain information about the table of available chunks and a summary of the frequency estimates of the catalogue-sender at the moment of the *Catalogue* message dispatch.

The table of available chunks (Figure 1) associates each downloaded data chunk with the name of the file it belongs to and a sequence number denoting its order among the rest of the data chunks in that file. Moreover, for each separate chunk it stores the *MD5* key which has been produced by feeding the contents of the particular data chunk to the message digest algorithm. Each peer's table is updated every time a new chunk is downloaded at that node, by just appending an additional record which maps to the meta-data information of the newly acquired chunk.

As mentioned above, each peer maintains a set of *frequency counters* shown in Figure 1. There is a frequency counter structure for every resource and/or file being disseminated in the network at any given moment. The structure in discussion helps maintain estimations of the relative frequency with which every chunk appears in the network. These estimations essentially reflect the proportion of the number of copies of each chunk to the total number of chunks of the file spread in the *P2P*-system. Evidently, estimations range between 0 and 1 and for each file in a peer, they sum up to 1. If a *frequency counter* estimation refers to a chunk that has been already downloaded, there is a pointer to the respective record at the table of *available chunks*. Otherwise, even if the chunk is missing, there is still an estimation derived from trading catalogue messages with other peers.

Peers exchange messages called *Catalogues* during the whole course of dissemination. These messages (Figure 2) are a critical component for the efficient operation of the proposed dissemination scheme. Dispatched by a peer, a *Catalogue* contains a summary of the frequency counter data it has estimated thus far and a list with all the data chunks that the peer has available for others. Each peer that receives a catalogue can select a chunk to fetch from the list and moreover it can use the frequency data to improve its own estimations. It is worth mentioning that the contents of this *Catalogue* are time-dependent. The content of *Catalogue* changes during the dissemination, according to the chunks available by that peer at any given moment as well as the current estimation of its *frequency counters*.

CATALOGUE = Frequency Counter Data + Available Chunks' IDs

Filename	Frequency Estimates	Total Chunks Counted
map.jpg	$\{(1,f_1),(2,f_2),\ldots,(10,f_{10})\}$	$Total_{map.jpg}$
event.dat	$\{(1,f_1),(2,f_2),(3,f_3)\}$	$Total_{event.dat}$

+

Filename	ChunkID
map.jpg	md5{chunk data}
map.jpg	md5{chunk data}
map.jpg	md5{chunk data}
event.dat	md5{chunk data}

Figure 2. The Catalogue message

3.2. Basic Protocol Operation

Every peer in *Catalogue-Gossip* runs four separate modules that synegistically implement the operations of the protocol. Figure 3 depicts how the components *Catalogue Processing Listener*, *Choker*, *Transfer Manager* and *Catalogue Dispatcher* cooperate. The *Catalogue Processing Listener* is the module which always waits for connections from other peers, so as to receive incoming catalogues and decide which chunk to possibly fetch. It operates in conjunction with the *Choker* module which works as a throttle mechanism used to reject connections if a node gets overloaded. The *Transfer Manager* is responsible for carrying out the data chunk transfers the peer is engaged in. Lastly, the *Catalogue Dispatcher* forwards updated versions of the *Catalogue* to randomly selected peers.

A node find itself in a passive state, if it does not forward its *Catalogue* to others. A peer in passive state may be either entirely idle simply waiting for connections or just taking part in some chunk transfers. A node is its active phase, if its *Catalogue Dispatcher* component has commenced gossiping the local *Catalogue*. The state of the node is turned into active the first time it downloads a chunk of a new file. The peer leaves its active state once the protocol has determined that the dissemination of the file is over.

Upon accepting a new connection, the *Catalogue Pro-*

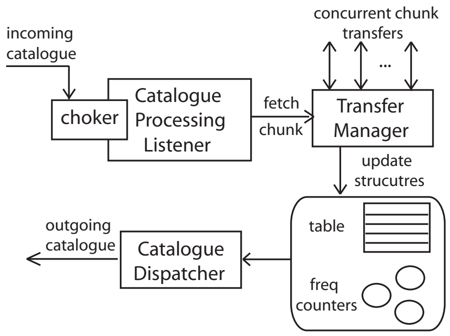

Figure 3. Logical Components of the Protocol

cessing Listener scans the incoming *Catalogue* to see which chunks are candidates for fetching from the remote peer. A chunk becomes a candidate for fetching, if it is contained in the incoming *Catalogue*, but missing from the local peer. The peer selects which chunks to fetch according to its own chunk selection policy. Once it has decided, it notifies the *Transfer Manager* about the chunk *id* and the remote peer's address, so that the transfer can be started. Li light of the fact that the *Catalogue* contains no chunk of interest to the receiving peer, the *Catalogue Processing Listener* responds with an "*UNSUCCESSFUL*" message.

The *Choker* module acts as a filter on the connections accepted by the *Catalogue Processing Listener*. More specifically, it computes the node's available bandwidth in order to decide whether to allow additional connections. If the peer does not have adequate spare bandwidth to accommodate another chunk transfer, the incoming connection is "choked" and a "*BUSY*" message is returned to the sender. The spare bandwidth is calculated by subtracting the bandwidth used by ongoing transfers from the total bandwidth of the node. The latter can be estimated by having the *Transfer Manager* keep track of the highest downloading bandwidth rate ever appearing during the node's operation. Because of the potentially vast number of connections established in this gossip-based environment, this measured value quickly approximates the actual maximum bandwidth of a node [4].

During the dissemination of a file, any peer can take part in multiple data chunk transfers. The *Transfer Manager* is a multi-threaded module able of handling concurrent connections to remote hosts. Besides that, its main responsibility is to update the structures maintained by the peer any time a chunk transfer is complete. The *Transfer Manager* does so by appending a record to the table of available chunks and by updating the estimates of the frequency counters. If the peer is in its passive state, the *Transfer Manager* initiates the *Catalogue Dispatcher* module to start gossiping the node's *Catalogue*.

As soon as a node enters its active state, the *Catalogue Dispatcher* begins gossiping the local *Catalogue* to other peers. The node selects a peer at random from its local view of neighbors. As mentioned earlier, this is equivalent to selecting a random peer from the entire system. The node can now construct an updated version of the local *Catalogue* and forward it to the selected remote peer. This procedure is constantly repeated during the dissemination of a file. The peer leaves its active state when the protocol detects that the dissemination is over. A system parameter regarding the active state of a node is the inter-gossip time interval. This parameter denotes the time period elapsed between successive pushes of the *Catalogue* and may designate the degree of aggressiveness in the behavior of each node. Aggressive behavior as exemplified by short intervals may cause more load imbalances across nodes, with the originator suffering the highest load. This is due to the fact that the initiator may send more *Catalogues* during a single chunk transfer which in turn results in more peers fetching chunks from the initiator. The same holds for all the nodes that start gossiping their *Catalogue* early during the dissemination, thus, leading to uneven load distribution.

3.3. Protocol Enhancements

3.3.1 Dissemination Termination Mechanism

A potential disadvantage of gossip-based systems is that there are no guarantees that a disseminated resource will ultimately reach all network nodes. In [8], the fan-out (i.e. number of gossip targets) required at each node is computed, in order to achieve reliable dissemination of a resource, with very high probability. Essentially, only probabilistic guaranties can be provided in a simple gossip-based setting. Hence, an additional mechanism is required to provide deterministic guaranties. In *Catalogue-Gossip*, deterministic delivery is attained by having all peers continuing to forward their *Catalogue* until the dissemination completes. The question which emerges is how each node will know that the dissemination of a file is complete across the entire network, so that it can stop forwarding its *Catalogue*.

To address the issue we develop a termination mechanism. Our protocol instructs peers to leave their active phase by using a heuristic condition. Each node that forwards its *Catalogue* expects either the beginning of a chunk transfer or an "*UNSUCCESSFUL*" response. Receiving many successive unsuccessful responses is an indication that the dissemination has either concluded or is almost complete. The exact number of responses (r) that would indicate the termination of the dissemination is a system parameter.

To ease the proper tuning of this parameter, we provide the following analysis. Consider a network which consists of N nodes, M of which have completed (i.e. received all file chunks) and K are still pending. For providing the analysis we make the assumption that the dissemination proceeds in steps. We say that a set of K nodes is isolated for one round, if none of the M completed nodes sends a *Catalogue* to any of the K nodes. The probability of a one-round isolation is equal to the probability that all M nodes select a completed node [1] to forward the *Catalogue* to. So we have:

$$q = Pr(a\ node\ selects\ a\ completed\ node\ to\ send\ to)$$

$$q = \frac{M}{N} = 1 - \frac{K}{N}$$

and,

$$p = Pr(all\ M\ nodes\ select\ a\ completed\ node)$$

$$p = q^M = (1 - \frac{K}{N})^{N-K}$$

finally,

$$P_r = Pr(p\ happens\ in\ r\ successive\ rounds)$$

$$P_r = p^r = (1 - \frac{K}{N})^{r(N-K)}$$

Hence, according to the anticipated size of a system, one can adjust system parameter r so that a node becomes isolated with as low P_r as desired. A system with low P_r is more accurate as in this case nodes keep on forwarding their catalogue, until the dissemination is actually completed. Allowing a higher P_r means that more nodes might infer that the dissemination is over before this actually becomes true. Such a setting is suitable for systems with constraint resources (e.g., battery consumption in mobile networks). Here, it would be better for most nodes to stop expending resources while having their *Catalogue* forwarded and let a small portion of remaining nodes pull their missing chunks.

A node checks the heuristic condition only after it has collected all the chunks of a file to ensure that an unsuccessful response means that the remote node has completed the transfer of all the chunks. As soon as a peer has received r successive unsuccessful responses, it leaves its active phase, stops forwarding its *Catalogue* and assumes that the dissemination is over.

To handle the extreme case of an isolation of r rounds, we include a timeout mechanism in the protocol. This timer expires if no *Catalogue* is sent to a peer for a long time. In this case, the peer stops gossiping its catalogue and starts gossiping direct requests for the missing chunks to random peers. In other words, this mechanism actually allows a peer to unilaterally make a transition from the push to the pull model of operation, in order to deal with isolation.

[1] one of the M nodes that have received all chunks.

3.3.2 Resilience to Initiator Faults

Gossip-based protocols are in general resilient to node faults and their performance degrades gracefully with crashing peers. However, they have an inherent weakness when it comes to resilience to faults on the initiator of a dissemination. It is crucial that the initiator can forward all the data chunks quickly, so that the entire file becomes replicated in the network as fast as possible. This is required so that the system can complete the dissemination even if the initiator fails early on in the process.

To reach this increased level of fault tolerance, we designed a chunk selection policy which favors rare chunks, so that all chunks can be replicated across the peers quickly. The other policy we investigated is having the recipient of a *Catalogue* select one chunk at random. More specifically the two policies we use are:

- *Rarest-First:* ideally, the rarest-first policy selects to fetch the chunk which is less common among the nodes of the *P2P*-network. This can facilitate speeding up the degree of data diffusion in the network earlier during the dissemination.

- *Random-First:* this policy instructs the peer to randomly fetch any of the available chunks, as announced by the *Catalogue*. Note that this approach does not incur the extra communication overhead of Rarest-First policy (the protocol could be implemented without *frequency counters*).

To implement a policy such as Rarest-First, there is a need to maintain a global view of the system to be able to tell which chunks are rare and which are more common. A centralized approach could readily achieve this by having every downloader register at the origin site which chunks it has already downloaded [2]. This is not applicable in the case of data dissemination in fully-decentralized *P2P*-networks, where no global state can be maintained at the initiator. We thus propose a decentralized implementation of the Rarest-First policy, so that it can be used in pure P2P environments.

Decentralized Rarest-First policy is implemented by using the frequency counter structures of Figure 1. Each peer maintains a frequency counter for every different file being currently disseminated in the system. This counter provides an estimation of the percentage of each chunk's copies relative to the total number of chunk copies belonging to the particular file in the system. The originator of a resource creates a counter which assigns the frequency $\frac{1}{L}$ to all L chunks of the disseminated file.

When dispatching its *Catalogue*, the initiator includes the data of the frequency counter as illustrated in Figure 2. The recipient of the *Catalogue* selects one of the chunks based on its current local frequency counter and fetches the chunk. The data of the frequency counters of a peer has to be updated in two cases: when a new chunk is fetched and when it receives a frequency estimation from another peer.

The update in the first case is accomplished by increasing the *Total* variable of the file (Figure 2) by one and then adding $\frac{1}{Total}$ to the frequency of the chunk fetched. To retain the assertion that the sum of frequency estimations over the L chunks of a file is equal to one ($\sum_{i=1}^{L} f_i = 1$), the added fraction has to be subtracted from the rest of frequency estimates. So it is split into $L-1$ equal parts, each of which is subtracted from the respective estimates.

For the second case, the recipient of the *Catalogue* has to compute the new estimation of frequencies by aggregating the information sent by the remote peer. To do so, we propose a weighted mean aggregator operator. To compute the respective weights it uses the "*Total Chunks Counted*" information shown in the *Catalogue* of Figure 2. This information is initially set to L (the number of the chunks) by the originator and 0 by all other peers, and is updated each time a peer fetches a chunk or aggregates the estimates of other peers. Formally:

$$f_i^A = M_i(f_i^A, f_i^B) = w_A f_i^A + w_B f_i^B$$

with

$$w_k = \frac{Total_k}{Total_A + Total_B}, k = \{A, B\}$$

where A is the recipient of the *Catalogue*, B is the sender and f_i is the estimate of the frequency of the i-th chunk. Although intuitively Random-First and Rarest-First policies have comparable dissemination speed performance in the long term, Rarest-First is faster in replicating all data chunks in the system, whereas Random-First incurs less overhead in the network.

4. Experimental Results

In order to evaluate the efficiency of the *Catalogue-Gossip*, we created a simulation environment based on the *JiST* simulation engine [1]. *JiST* is a Java-based runtime environment for discrete event simulations. Our assessment included a number of protocols used for flash data dissemination. Specifically, the compared protocols were *Catalogue-Gossip*, *Crew* and *Simple-Gossip*. Our comparison mainly focuses on the scalability of the protocols for diverse network and file sizes while taking into account other performance aspects of the protocols.

In our evaluation, we used the following metrics: *i) Completion-Time* defined as the time taken for the successful update of all peers, *ii) Coverage-Speed* designated as the number of successfully updated peers at any point during the dissemination, *iii) Data-Overhead* measured as the number of unsuccessful messages and finally, *iv) Chunk-Replication* defined as the time taken by the originator to send all chunks. In our experimentation, we vary the following key factors:

- *Network Size (N)*: The largest the network is in terms of peer population, the longer the dissemination takes to be accomplished. It is also critical to establish out how well the different protocols scale as the network size increases.
- *Content Size (S)*: Flash dissemination in unstructured networks usually refers to medium-sized data (of $50\,KB$ or less). Nevertheless it may refer to the rapid dissemination of image data captured by security cameras or to the fast spreading of alerts. Hence, it is crucial to determine which protocols work best for different content sizes.

Next, we briefly outline results from our experimentation while focusing on the above-mentioned four metrics.

Completion-Time (CT): In order to ascertain how the protocols efficiently scale in large *P2P*-networks we carried out experiments to assess the completion time of each protocol under varying content sizes for the communicated resources. Figure 4 depicts the behavior of *Simple*, *Crew* and *Catalogue* protocols in terms of CT under varying peer population assessing their applicability for flash dissemination. Simple-Gossip has the worst performance among the

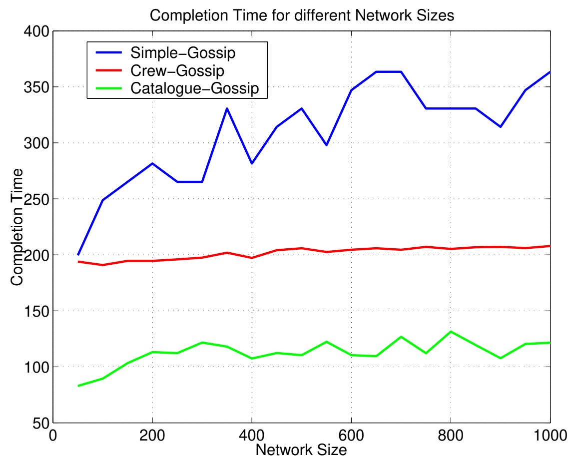

Figure 4. CTs for different network sizes.

three protocols. It takes longer to disseminate the data in every case and consequently appears to scale poorly with increasing network size. This is not the case with *Crew* and *Catalogue-Gossip* which show more robust behavior in large-scale networks. *Catalogue-Gossip* however achieves better completion times as it completes in about half the time it takes for *Crew*-Gossip to do so. The same behavior appeared in our experiments even for extremely high number of nodes of up to $50\,K$ peers where as *Catalogue-Gossip* still outperformed *Crew*.

Figure 5 depicts how the protocols react to different sizes of disseminated file for a 500-node network. We vary the size of the data resources from 5 to 50 KB. Apparently, *Simple-Gossip* performs poorly whereas both *Crew* and *Catalogue* seem to scale roughly the same. We should mention here that the setting of this experiment favors *Crew* and is the worst case for *Catalogue* as disseminated data resources were split into a small number of chunks (five).

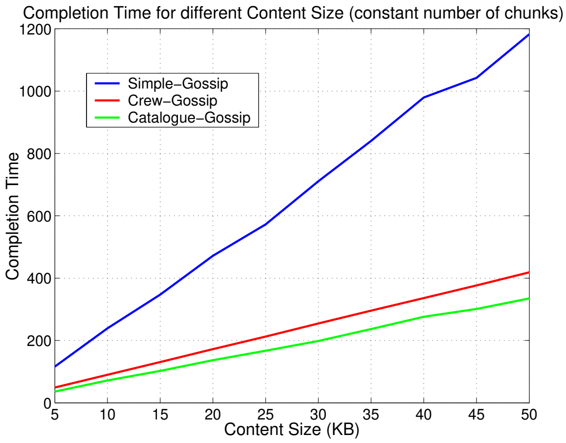

Figure 5. CT**s vs. varying content size.**

Catalogue performs much better than *Crew* when resources are split in many chunks. For instance in the experimental setting of Figure 4, resources were fragmented into 10 chunks and the resulting performance of *Catalogue* was twice as good as *Crew*'s. In cases of small chunks, the overhead of the numerous unsuccessful *Crew* pull requests increases and renders *Crew* inefficient when compared to *Catalogue*. Our proposed protocol scales well regardless of the splitting of resources into chunks and this further contributes to its robustness.

Coverage-Speed(CS): this measurement offers a view on how quickly a resource is disseminated throughout the network. Figure 6 depicts the rate of peer completion during the entire period of the dissemination for the three protocols. Although initially *Simple* closely follows *Crew*, it finally demonstrates much inferior rates as soon as about 80% of the peers have received the entire data resource, because the probability of selecting a random peer which has not completed yet falls dramatically. Moreover, *Catalogue* completes much earlier than *Crew*. This is because in *Crew*, the probability of selecting a peer with available chunks during the first steps of the dissemination is very small. This means that there is a significant delay in spreading the chunks in the network at the beginning of the dissemination, a fact reflected by the *Crew* curve in Figure 6.

Data-Overhead(DO): The reason behind the good performance of *Catalogue-Gossip* is that it manages to reduce data overhead. The protocol's design has focused on avoid-

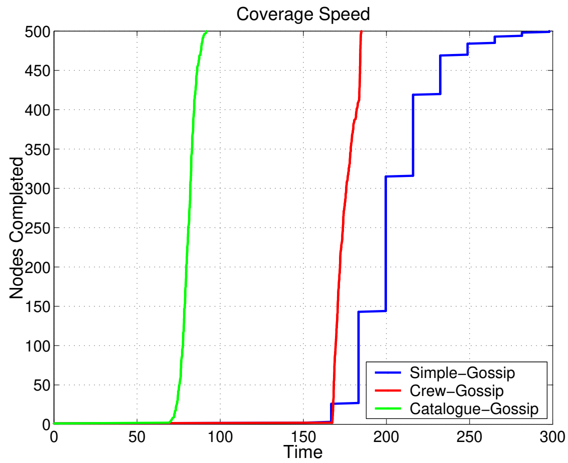

Figure 6. CS**s for the three protocols.**

ing redundant transfers and unsuccessful messages. This section confirms that *Catalogue-Gossip* is more efficient in peer communication. Before continuing, it is useful to define what an "unsuccessful" message means in each protocol. First of all, an unsuccessful message in Simple-Gossip is a transfer of a duplicate chunk. As already analyzed, duplicate chunk transfers in Simple-Gossip are the source of high data overhead. In Crew-Gossip, an unsuccessful message is a pull request to a peer which has no chunk that is of interest to the requesting peer. Unsuccessful messages in *Catalogue-Gossip* reflect the event of a peer pushing its *Catalogue* to another node, which has all the chunks in the *Catalogue*. Figure 7 illustrates the number *Catalogue-Gossip* has about 80% less redundant messages than Crew). of redundant messages for the three protocols. *Catalogue-Gossip* has the smallest redundancy, an order of magnitude less than in Crew-Gossip (the vertical axis is logarithmic).

Chunk-Replication(CR): Chunk-replication examines how well chunk selection policies do in the context of *Catalogue-Gossip*. Figure 8 displays all three chunk selection policies evaluated in this paper, namely Random-First, Centralized Rarest-First and Decentralized Rarest-First policy. As Figure 8 shows, the centralized Rarest-First replicates the chunks of the original file the fastest in the system. This is because every peer selects the optimal chunk to fetch, as if there was a global state maintained about the frequency of chunks in the *P2P* network. On the contrary, the Random-First policy takes much longer to achieve the same degree of replication. Moreover, difference between the above two policies becomes wider when it comes to files consisting of more chunks (Figure 8 refers to a resource of 10 chunks). This is because the expected number of random samples needed to draw all N chunks increases rapidly with N. The Decentralized Rarest-First is a much better approximation to its centralized version, than

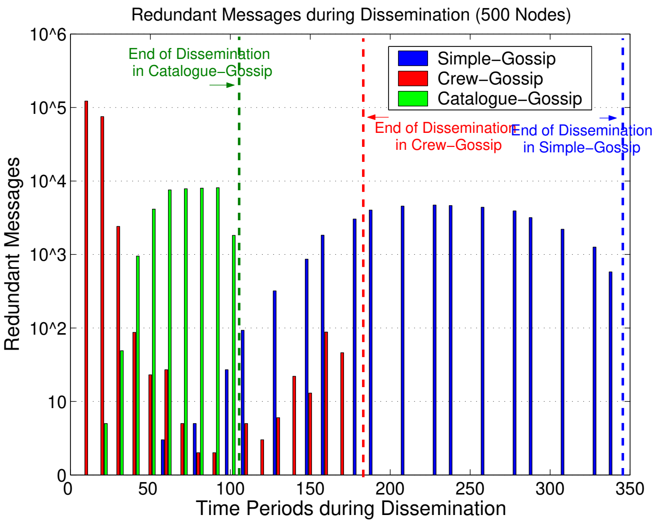

Figure 7. Redundant messages in Simple-Gossip, Crew-Gossip and Catalogue-Gossip.

the naive Random-First policy.

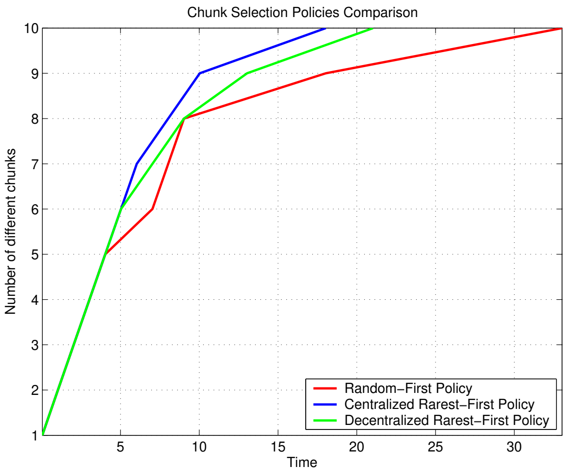

Figure 8. Time required by initiator to send all 10 **chunks to other peers.**

5. Conclusions and Future Work

In this paper, we proposed the *Catalogue-Gossip* protocol for addressing the problem of flash data dissemination in unstructured *P2P*-networks. *Catalogue-Gossip* experimentally shows improved performance behavior over existing counterparts as it features reduced data overheads and accelerated speed with which data reach their destination nodes during the course of the dissemination process. *Catalogue-Gossip* also enforces deterministic delivery which is a strong requirement for data dissemination systems. Finally, *Catalogue-Gossip* implements a decentralized policy ensuring short time periods elapsed between initiation of a dissemination and the full replication of the data throughout the unstructured network. We plan to enhance the protocol by incorporating compression techniques to further reduce the volume of traded-data chunks and by exploiting specialized Bloom-filters to more efficiently represent protocol structures. We also intend to pursue an analytical approach in establishing bounds of feasible performance gains with respect to prior approaches.

References

[1] R. Barr, Z. J. Haas, and R. van Renesse. *JiST*: an Efficient Approach to Simulation using Virtual Machines. *Software: Practice and Experience*, 35(6):539–576, May 2005.

[2] B. Cohen. Incentives Build Robustness in BitTorrent. In *Workshop on the Economics of Peer-to-Peer Systems*, Berkeley, CA, June 2003.

[3] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic Algorithms for Replicated Database Maintenance. In *6th ACM Symposium on Principles of Distributed Computing*, Vancouver, Canada, 1987.

[4] M. Deshpande, B. Xing, I. Lazaridis, B. Hore, N. Venkatasubramanian, and S. Mehrotra. CREW: A Gossip-based Flash-Dissemination System. In *26th IEEE Int. Conf. on Distributed Computing Systems*, 2006.

[5] P. T. Eugster, R. Guerraoui, A.-M. Kermarrec, and L. Massoulie. Epidemic Information Dissemination in Distributed Systems. *IEEE Computer*, 37(5):60–67, May 2004.

[6] A. J. Ganesh, A.-M. Kermarrec, and L. Massoulie. Peer-to-Peer Membership Management for Gossip-based Protocols. *IEEE Transactions on Computers*, 52(2):139–149, February 2003.

[7] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. van Steen. Gossip-based Peer Sampling. *ACM Transactions on Computer Systems*, 25(3):8, 2007.

[8] A.-M. Kermarrec, L. Masoulie, and A. J. Ganesh. Probabilistic Reliable Dissemination in Large-Scale Systems. *IEEE Transactions on Parallel and Distributed Systems*, 14(3):248–258, March 2003.

[9] K. Ramamritham and P. Shenoy. Dynamic Information Dissemination. *IEEE Internet Computing*, 11(4):14–15, 2007.

[10] R.Rivest. The *MD5* Message-Digest Algorithm. RFC1321, 1992.

[11] S. Voulgaris and M. van Steen. An Epidemic Protocol for Managing Routing Tables in Very Large Peer-to-peer Networks. In *14th IFIP/IEEE Int. Workshop on Distributed Systems: Operations and Management (DSOM'03)*, 2003.

[12] A. Yu and S. T. Vuong. MOPAR: a Mobile Peer-to-Peer Overlay Architecture for Interest Management of Massively Multiplayer Online Games. In *Int. Workshop on Network and Operating Systems Support For Digital Audio and Video (NOSSDAV'05)*, Stevenson, WA, 2005.