Publiy$^+$: A Peer-Assisted Publish/Subscribe Service for Timely Dissemination of Bulk Content

Reza Sherafat Kazemzadeh and Hans-Arno Jacobsen
{*reza, jacobsen*}*@eecg.utoronto.ca*

Middleware Systems Research Group (MSRG), University of Toronto

November 28, 2011
Toronto, Ontario, CANADA

Abstract

Publish/Subscribe (P/S) systems and file sharing applications traditionally share the common goal of disseminating data among large populations of users. Despite this similarity, the former focuses on timely dissemination of small-sized notification messages, while the latter presumes larger types of bulk content with less emphasis on the time needed between release and delivery of data. In this paper, we develop a peer-assisted content dissemination mechanism to bridge this gap by adopting the P/S model. We propose a hybrid two-layer architecture in which P/S brokers act as coordinators and guide their clients with interest in similar content to engage in direct exchange of data blocks in a peer-to-peer and cooperative fashion. Furthermore, we use network coding in order to facilitate data exchange among clients. Our peer-assisted scheme offloads the burden of disseminating huge volumes of data from P/S brokers to subscribers themselves. As an added advantage of our approach, brokers can employ strategies that helps shape traffic flows in multi-domain network settings. Finally, we have implemented our approach and carried out extensive large-scale experimental evaluation on a cluster with aggregate data transfers of up to 1 *TB and involving up to* 1000 *subscribers. Our results demonstrate good scalability and faster content delivery compared to file sharing protocols such as BitTorrent.*

1 Introduction

Recent advances in file sharing [11] and content dissemination applications [16, 23] have enabled large-scale distribution of bulk content among thousands of Internet users. As a common practice in these systems, a downloading client is required to actively seek the content by querying a tracker and initiating download sessions with other peers who possess the file. If the requested content is not readily available, *e.g.*, because it is not yet published, the client has to either abandon its quest for the file, or continuously poll the tracker until the file becomes available. Depending on whether the polling interval is short or long, continuous polling by many clients either imposes massive load on the trackers or increases the clients' download times. Moreover, popular file sharing protocols are not designed to and optimized for minimizing clients' download times.

In this paper, we address the first drawback by adopting the *Publish/Subscribe* (P/S) model that provides two major benefits: *(i)* dissemination is inherently *reactive* and is initiated as early as the content is released by a source; and *(ii)* published content is delivered to a *selected* subset of clients determined based on the clients' pre-specified subscription interests. Furthermore, to address the second deficiency, we develop content dissemination strategies that are specifically targeted to improve the clients' download times.

In recent years, the focus of P/S research community has been on efficient distribution and routing of *events* or notification messages which are typically small (tens of kilobytes) in size. These techniques, however, are not readily applicable to dissemination of bulk content (hundreds of megabytes). Hence, our objective in this paper is to bridge this gap and craft a *scalable* P/S scheme capable of disseminating bulk content. Before delving into the details of our approach, we present a few examples of potential application scenarios that demand fast distribution of content among many clients that all benefit from timeliness and selectivity of content delivery brought about by our scheme:

- Software security updates [15] are often released in periodic cycles. A P/S service enhances the timeliness of the delivery process via subscriptions that represent users' software configurations and pushing new updates to vulnerable machines accordingly.
- *Content Distribution Network*s (CDN) [3, 19] deploy hundreds of servers over the Internet. Timely replication of recently released content (*e.g.*, news clips published by a news source) is streamlined by a P/S system that allows CDN servers to subscribe to and receive the types of content that they are assigned to serve.
- Cloud-based storage services, *e.g.*, Dropbox [2], provide file synchronization facility among many clients. A file update in current approaches incurs costly network traffic (charged by cloud providers) from servers in the cloud to all synchronizing clients. Our peer-assisted scheme largely cuts these costs by enabling clients to contribute in dissemination of file updates.
- In *P2P* social networking applications, users who wish to share large content (*e.g.*, pictures and videos) with friends can use a P/S-based distribution platform to allow friends to directly subscribe to their content feeds.

The common denominators in the above application scenarios are timeliness and selectivity of content distribution. These traits make the P/S model an ideal choice.

In a conventional distributed P/S system, designated message routers (*a.k.a brokers*) form an application layer overlay network and provide an access point through which subscribing and publishing clients communicate [13, 10, 9, 12]. Brokers store clients' subscriptions and relay publication messages towards subscribers with matching subscriptions. Existing P/S systems that adopt such a broker-based architecture further presume that publications have a relatively small size (perhaps tens of kilobytes). As a result, brokers are unlikely to face an overwhelming volume of traffic. In contrast, our focus on distribution of bulk content (hundreds of megabytes) causes traffic-bearing brokers in such approaches to face serious limitations to scale to a large client population.

To overcome this challenge, we argue in favour of a *peer-assisted* dissemination scheme in which the system's available bandwidth grows organically with the number of peers. In recent years, such hybrid schemes have gained popularity among online music and video streaming industry and have been shown to lower dissemination costs, avoid server overload and maintain high Quality of Service (QoS) [5, 21]. Like many of these systems, we assume that peers are altruistic (possibly by running a proprietary piece of software) and wish to cooperatively download their content of interest as early as possible. These are reasonable assumptions in all usage scenarios we outlined above.

To this end, we propose a hybrid two-layer architecture that combines the benefits of P2P content distribution with those of broker-based P/S systems. In our approach, we rely on P/S brokers to store clients' subscription and act as coordinators that guide clients with similar interests to directly engage in exchange of segments of content. Direct peer exchange enables us to tap into the bandwidth capacity available at clients and relieve brokers from much of the burden of content forwarding. Henceforth, we refer to the layer in which P2P content exchange takes place as the *data layer*, and the layer where brokers

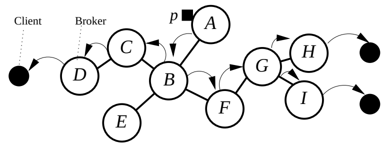

Figure 1. Publication *p* is routed to brokers with matching local subscribers and delivered to interested clients.

reside as the *control layer*. The two layers have a close coordination and act as one unified system. Hence, we have knowingly avoided having separate services, one for notifying clients of newly released content, and one for distribution of that content. We believe that a close marriage between these two functionalities improves content delivery times and eliminates the need to maintain separate infrastructures.

Previous research has demonstrated network coding as a viable new direction [24, 21, 25, 16, 14]. Network coding improves scheduling and exchange of packets and enables peers to participate in content dissemination as early as having received a single coded block of data. Another desirable property of network coding is that it provides a virtually unlimited stream of coded packets each of which is equally useful (*a.k.a. innovative*) to reconstruct the original content. A client seeking to receive a missing piece of content can thus receive this data from *any* other peer who is also interested in the same content. In absence of network coding, however, the client's request could only be fulfilled by those peers who have previously received the *exact same piece* of the file. The search for such a peer can be time consuming and adversely impact the time to complete the download. Furthermore, in comparison to the well-known BitTorrent system [11], our solution includes strategies specifically tailored to lower dissemination delay by avoiding peer coordination problems, and mitigate the issues caused during flash-crowd scenarios [8]. As a matter of fact, *due to the reactive nature of the P/S model the flash-crowd phenomenon is expected to be a highly recurring scenario in our system.*

To address these challenges, this paper makes the following contributions: *(i)* In Section 3, we devise a two-layer hybrid architectural framework that brings together the benefits of timely push-based P/S communication with the scalability of P2P bulk content dissemination applications; *(ii)* In Section 4, we elaborate on how network coding can be incorporated to facilitate the exchange of data blocks in an efficient manner within our framework; *(iii)* In Section 5, we devise a number of strategies to reduce dissemination delay and improve fault-tolerance; and finally, *(iv)* in Section 6, we report on our evaluation results based on a fully functional Java-based implementation of our approach, called *Publiy*$^+$ [18, 4, 17].

2 Background and Related Work

In this section, we review concepts in the areas of distributed P/S as well as network coding research.

2.1 Distributed P/S Systems

A distributed P/S system consists of a set of brokers that form an *application layer overlay network*. Brokers are the system's points of contact where subscribing clients register their subscriptions and publishing clients inject their publications (*i.e.,* published content). A subscription specifies a client's interest in specific types of content using predicate-based expressions. For example, `sub=[sftw:IE, ver:6.0, build:1300,os:vista]` represents a user's interest for updates of Internet Explorer version 6 installed on Windows Vista. Once a publication (*i.e.,* a new security update) is released, its *content descriptor* is matched against subscription predicates to determine which clients it must be delivered to. Brokers then forward publications in the overlay towards matching subscribers.

To set up forwarding paths in the overlay, client subscriptions are injected into the system at their local brokers and sent throughout the network. Brokers record the subscription's predicates as well as a reference pointer to a neighboring broker from whom the subscription was received. These reference pointers construct routing paths that are used for forwarding of publications. For example, Figure 1 illustrates a simple P/S network where a publication is routed towards interested subscribers along the overlay links designated by arrows. In the P/S literature, many optimization techniques such as use of advertisements and subscription covering [20] are proposed to avoid flooding of subscriptions and constructing concise routing tables at brokers. In our approach, we use *Publiy*$^+$ [18, 4, 17], our distributed P/S System as our underlying distribution network that also supports many of these techniques including subscription covering and efficient multipath publication forwarding.

Original data matrix $\mathbb{Y}$:	A coded block $\mathbb{X}_\mathbf{i}$:
$\mathbb{Y} = \begin{bmatrix} b_{1,1} \cdots b_{1,n} \\ \vdots \quad \vdots \\ b_{k,1} \cdots b_{k,n} \end{bmatrix} = \begin{bmatrix} \mathbb{B}_\mathbf{1} \\ \vdots \\ \mathbb{B}_\mathbf{k} \end{bmatrix}$	$\mathbb{X}_\mathbf{i} = \mathbb{C}_\mathbf{i}\mathbb{Y} = \begin{bmatrix} c_{i,1} \cdots c_{i,k} \end{bmatrix} \begin{bmatrix} \mathbb{B}_\mathbf{1} \\ \vdots \\ \mathbb{B}_\mathbf{k} \end{bmatrix}$
Decoding of matrix $\mathbb{Y}$: $\mathbb{C}^{-1}\mathbb{X} = \mathbb{C}^{-1} \begin{bmatrix} x_{1,1} \cdots x_{1,n} \\ \vdots \quad \vdots \\ x_{k,1} \cdots x_{k,n} \end{bmatrix} = \begin{bmatrix} \mathbb{B}_\mathbf{1} \\ \vdots \\ \mathbb{B}_\mathbf{k} \end{bmatrix} = \mathbb{Y}$	

Table 1. Coding and decoding of data matrix $\mathbb{Y}$ in $GF(2^8)$.

2.2 Random Linear Network Coding

Random linear network coding was is an active research topic in information theory and has proved to be a practical approach to improve the network's bandwidth utilization [25]. This is particularly important for our data dissemination system in which scalability demands effective use of all peers' available bandwidth resources. Several systems adopt linear network codes in a variety of applications ranging from video transmission and playback over the network [24, 21, 22] to file sharing applications [16].

In this paper, we use random linear network codes in order to facilitate exchange of blocks of data between clients that have mutual interest in the same content. To code a piece of data of size N bytes, the source breaks it up into k equally sized *blocks* of n bytes ($N = kn$[1]) to form a 2-dimensional k by n byte matrix, $\mathbb{Y} = [\mathbb{B}_\mathbf{1}, \mathbb{B}_\mathbf{2}, \cdots, \mathbb{B}_\mathbf{k}]^T$ (illustrated in Figure 1). Each row of $\mathbb{Y}$ corresponds to one *data block* $\mathbb{B}_\mathbf{i}$. To generate a *coded block*, $\mathbb{X}_\mathbf{i}$, the clients first randomly chooses k coefficients, $C_i = [c_{i1}, \cdots, c_{ik}]$ and compute the linear combination of $\mathbb{X}_\mathbf{i} = \mathbb{C}_\mathbf{i}\mathbb{Y}$ by performing a matrix multiplication in the finite *Galois Field* of $GF(2^8)$. This results in a coded block, $\mathbb{X}_\mathbf{i}$, which is packaged along with random coefficient matrix of C_i and sent over the network as one *data packet*. A receiver can reconstruct the original content simply by "holding up a bucket" and collecting *any* k of the coded blocks, $\mathbb{X} = [\mathbb{X}_\mathbf{1}, \cdots, \mathbb{X}_\mathbf{k}]^T$ and their corresponding coefficients, $\mathbb{C} = [\mathbb{C}_\mathbf{1}, \cdots, \mathbb{C}_\mathbf{k}]^T$. If the k rows of the coefficient matrix, C are linearly independent with a high probability and the receiver can perform a Gaussian (or Gauss-Jordan) elimination to compute the inverse matrix $\mathbb{C}^{-1}$. As the last step, the original data matrix is *decoded* by computing $\mathbb{Y} = \mathbb{C}^{-1}\mathbb{X}$ in $GF(2^8)$. A client can also start to seed (serve) a file after only partially receiving the coded blocks necessary to decode the original content. In this case, the sending peer may only possess $k' < k$ blocks of coded data and thus uses a coefficient matrix of size k' to code both the currently available coded blocks ($[\mathbb{X}_\mathbf{1}, \cdots, \mathbb{X}_{\mathbf{k'}}]^T$) as well as their corresponding coefficient matrix $\mathbb{C} = [\mathbb{C}_\mathbf{1}, \cdots, \mathbb{C}_{\mathbf{k'}}]^T$. The resulting coefficients and coded blocks are then packaged and sent in a similar manner to when the client the original data matrix $\mathbb{Y}$.

3 Content Dissemination Framework

We distinguish between two types of nodes, namely clients and brokers. While the primary goal of clients is to publish or to receive some content, brokers are deployed as part of the infrastructure to help subscribing clients in their quest to receive their content of interest. For this purpose, brokers are placed at strategic locations, called *regions*, around the Internet or within administrative domains to provide content dissemination service. As part of this task, brokers guide subscribers and facilitate exchange of data blocks between clients with mutual interest. Based on the distinctive roles of clients and brokers, we develop a two-layer architecture consisting of data and control layers (depicted in Figure **??**). We next describe the layers in detail.

Control Layer: The control layer consists of an overlay of P/S brokers that acts as the core architectural components in our system. Brokers maintain client subscriptions and guide clients with interest in the same file to engage in direct data exchange. In a simple scenario, a client connects to a broker that is deployed by its ISP or belongs to the same administrative domain. We thus say that clients connected to a broker belongs to its *region*. Subscribing clients apply for a *subscription lease* (or many leases) from their regional broker. The subscription lease has an expiry date that is agreed upon by both client and broker and thus needs to be renewed periodically. The lease renewal process gives the broker the assurance of the client's presence in the system as well as its continued interest in the subscription over a long period of time. Once a subscription

[1]The source may perform some padding in order to achieve this.

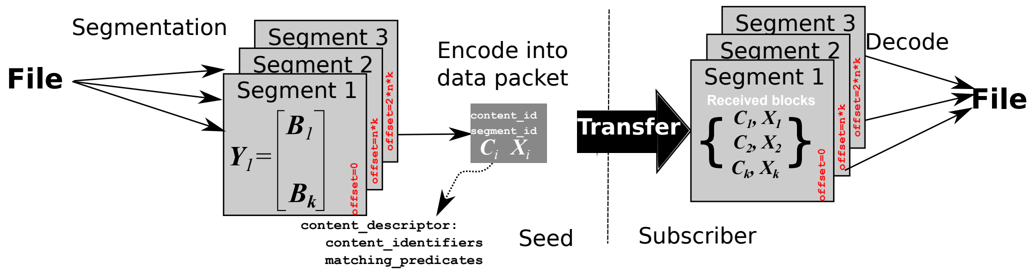

Figure 2. Content dissemination involves segmentation, coding, transfer and decoding.

lease request is granted by the broker, the client receives a lease acknowledgement message confirming that its subscription is now registered.

At the broker side, a subscription is inserted into the broker's local subscription routing table along with its lease expiry timestamp as well as the identity of the issuing client. We refer to these pieces of information collectively as a *subscription entry*. Furthermore, the subscription entry is propagated to neighboring brokers as described in Section 2.

Data Layer: The data layer consists of all subscribing clients in all regions regardless of their specific subscription interests. In general, clients in this layer are completely oblivious to each other and purely rely on brokers (in the control layer) for instructions on how and when to communicate with one another. More concretely, a source client who wishes to publish its content first contacts its own broker for instructions on how to proceed. We refer to such a broker that the content source is connected to as the content's *home broker*. As part of this interaction, the source receives a list of other peers who are interested in its published content and to whom it must send the content to. Likewise, non-source clients who have received the content and want to contribute in the distribution process also contact their regional brokers in order to receive similar lists of peers interested in the content. In this regard, our system's control layer acts as a *distributed registry service* that coordinates dissemination of the content in the data layer by supplying the source and other seeding clients with lists of peers interested the content. This is in contrast to conventional P/S designs in which brokers are directly involved in relaying publication messages. Finally, our approach to engage clients in exchange of data packets allows us to tap into the otherwise unused clients' available bandwidth. This relieves the brokers from shouldering the heavy burden of traffic associated with forwarding of bulk content.

4 Content Distribution with Network Coding

In this section, we elaborate on the details of our content dissemination protocol that uses network coding to facilitate exchange of blocks of data among clients.

4.1 Content Publishing Procedure

A file that is to be published by a source may come in different sizes ranging from a few megabytes to hundreds of megabytes. The source breaks up the file into a number of fixed sized units called *segments* which are handled by the system separately. Each segment is associated with the content descriptor of the original file and matches the same set of subscriptions. A segment is uniquely identified by its *metadata* information which consists of the unique identifier of the original file, as well as the segment's byte offset within the original file. We distinguish between two types of closely related messages: A *content notification* message is a publication message that carries metadata information associated with a segment. On the other hand, a *content data message* (or simply data message for brevity) contains both metadata information and a coded data block of a segment. In our two-layer architecture, notification messages flow between brokers in the control layer while data messages are transmitted only between clients in the data layer.

4.2 Content Distribution with Push-Lists

We use the term *regional seeding* and *cross-regional seeding* to refer to sending of data messages by a client to other peers in the same region, and in different regions, respectively. Since the source is located only within one region, full system-wide content dissemination requires a combination of both regional as well as cross-regional seeding. Furthermore, since seeding

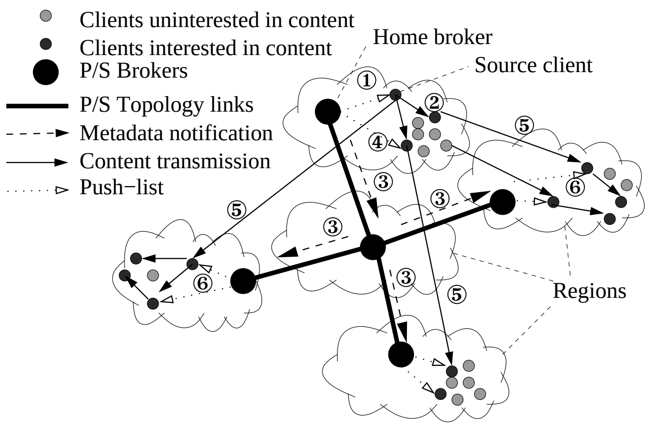

Figure 3. Data dissemination within/between regions. ①: source receives first push-list from home broker; ②: source pushes coded blocks to regional peers; ③: home broker routes notification to other brokers; ④: home broker's local clients request new push-lists; ⑤: home broker replies with a mix of regional and cross-regional peers; ⑥: peers who want to seed content request push-lists from their own brokers.

clients are unaware of other peers who are interested in the content they possess, they send a request message to their broker and include the metadata of the segments they are offering. The broker replies with a *push-list* message containing a list of interested clients to which the seed can push data messages to.

We now use Figure 3 to elaborate on details of the interactions between clients and brokers, as well as between brokers and brokers. A newly published content is initially seeded by its source. The source starts the dissemination process by sending the content's metadata information to the broker it is connected to, *i.e.*, the content's *home broker*. The home broker uses the content descriptor and its locally stored subscription entries to identify its set of *local* client subscription leases that are active (i.e., not yet expired) and also match the content descriptor. This information is readily available at the home broker and allows it to immediately reply back to the source with a push-list containing interested peers for regional seeding. This interaction between the source and home broker is marked by ① in Figure 3.

When the source receives a push-list, it contacts the peers referenced therein and starts to send, i.e., *push*, coded data messages. This is marked by ② in Figure 3. Each data message includes one coded block of data, the random coefficient used for coding as well as the unique content identifier and the segment's byte offset that the coded data block corresponds to. The receiving clients accumulate the coded blocks sent from all the seeds until k randomly independent coded blocks are available. At this point, the client decodes the blocks and reconstructs the original file segment. Furthermore, the client sends a *break message* to the seeds in order to stop the data push and also notifies its own regional broker about successful reconstruction of the segment. The broker notes this information and refrains from referencing the clients in any future push-list for that particular segment.

So far, the dissemination process can only accomplish regional seeding of the content segments. To achieve system-wide distribution the content must be pushed to interested clients attached to other brokers via cross-regional seeding. However, information about non-local subscription leases is not readily available at the home broker. To retrieve the list of these clients, the home broker uses the P/S overlay network (i.e., control layer) in order to inquire other P/S brokers for the list of their local matching subscribers. This is done by issuing a publication notification message containing the metadata of the content. The notification is routed in the overlay and delivered to P/S brokers who have at least one client interested in the content (marked by ③ in Figure 3). In contrast to flooding of the notification in the P/S overlay, the use of P/S routing techniques enables efficient distribution of the notification message to only the brokers who have matching regional subscribers (see Section 2). These brokers compile push-lists consisting of a *small subset* of their interested regional subscribers and reply back to the home broker. The home broker uses the push-lists in future interactions with its regional seeds and in order to direct them to push content to peers in other regions. This process is marked by ④ and ⑤ in Figure 3.

We now elaborate on how a non-source client seeds content segments. Depending on the client's *serving policy* it either waits until successfully decoding the whole segment or starts to seed when a large enough (but not complete) subset of the required coded blocks have been received, i.e., $k' < k$ coded blocks out of the required k. At this point, the seed proceeds to contact its regional broker and requests a list of peers with matching subscriptions. This is marked by ④ and ⑥ in Figure 3.

```
procedure SEED-SEGMENT()
    $PushList \leftarrow$ Query regional broker for push-list
    while $PushList \neq \emptyset$ do
        Push coded packets to peers in $PushList$
        if Receive break from $breakingPeer$ then
            $PushList \leftarrow PushList - breakingPeer$
            $PushList \leftarrow PushList \cup$ Query regional broker
procedure SUBSCRIBER-RECEIVE-LOOP
    Receive coded blocks
    if $k$ linearly independent blocks are available then
        Decode segment
        Inform broker of successful segment completion
        Send break to pushing peers
    if Sufficient coded blocks available for seeding then
        $SEED - SEGMENT()$
```

Figure 4. Algorithm executed by clients.

As we mentioned earlier, if the contacted broker is the content's home broker, the push-lists retrieved from other regions are also incorporated in the replies. This enables cross-regional seeding to take place. On the other hand, if the contacted broker is not the content's home broker, only matching subscribers within the same region are included in the push-list replies. Figures 4 and 5 illustrate the pseudo code of the dissemination algorithm executed by the clients and brokers, respectively.

5 Dissemination Strategies

5.1 Initial Seeding Strategy

Bharambe *et al.* [8] show that during a flash-crowd scenario, the source's uplink bandwidth plays a crucial role in driving fast download times in the BitTorrent file sharing application [11]. Particularly if the source's bandwidth is less or comparable to that of downloaders, it is unable to provide file segments quick enough to fully utilize the capacity available in the network. As a result, in the early hours after release, some segments become rare thus lengthening the clients' download time. As a matter of fact, reactive content push using the P/S model in our approach also creates flash-crowd scenarios. This is due to the fact that at the time fresh content is released, all matching subscribers are present in the system and wishing to receive the content as soon as possible. Any delay at this stage hinders this goal and lengthens the download times. Thus, effective handling of flash-crowd scenarios must be a key aspect of our system.

To mitigate this issue, we devise an *initial seeding strategy*. The idea is to boost a source's uplink capacity by delegating the task of seeding newly released segments to a small cluster of subscribers, drawn from all regions in the network. The bandwidth available to this cluster largely frees up the source's scarce capacity. This strategy can be incorporated in our approach by the following modification to the protocols: When a home broker replies to a push-list request from the source, it also sends the same push-list to the clients referenced therein (note that without this strategy, push-lists were only sent after explicit requests from these subscribers). The receivers immediately participate in the dissemination process by re-coding the packets as they arrive from the source. This creates a cluster of subscribers who supply each other with re-coded data packets concurrently as the source pushes data packets into the cluster.

An immediate benefit of this strategy (as observed in our experimental evaluation reported in Section 6.4) is that almost the time it takes for the source to push an equivalent of one segment of content into the cluster, all participants in the cluster retrieve and decode the entire segment (considering that all peers and the source have the same uplink bandwidth). This has two advantages: First, dissemination of the segment from this point on takes place in a multi-source (rather than single-source) fashion and minimizes the problems associated with a limited uplink capacity of a single seed. Second, the clustered peers participate in dissemination since receiving the initial packets from the source, leading to more effective bandwidth utilization.

```
procedure BROKERS INFINITE LOOP
    if Received push-list request msg from client or broker then
        Match content descriptor against local subscribers
        push-list ← $m$ matching subscribers
        Send push-list to source
        if Request is from content's source then
            Route content notification in P/S network
            Collect push-lists from other brokers
```

Figure 5. Algorithm executed by brokers.

5.2 Early Seeding Strategy

One advantage of using network coding is to allow clients to start seeding a segment without having it in its entirety (i.e., early seed). Enabling early seeding, however, substantially increases the risk of arrival of unuseful linearly dependant blocks at subscribers. As a result, an early seed who chooses to participate in the dissemination process without having a decoded copy of the segment must be cautious about whether its coded packets are innovative for other peers. One way is to have the sender check the independence of the coding coefficients at the receiver prior to transmitting a packet. To avoid this hurdle, we use a simple scheme and restrict the maximum number of blocks sent by an early seed to any other client to be the number of coded blocks it has received from peers who possess the decoded segments. This is done as follows: Coded packets include a flag indicating whether the sender possesses the decoded segment. An early seed stores the number of packets with the flag set and is allowed to send to each of other peers at most the same number of new coded packets.

Before implementing this strategy, we observed significant number of packet dependencies in our executions when early seeding was enabled. After implementing this strategy, the issue was entirely resolved even when peers start to seed after receiving a single data block of each segment.

5.3 Push-List Compilation Strategies

While all subscribers interested in the content must ultimately be referenced in at least one push-list, the order and frequency in which they appear on push-lists during the dissemination process has an impact on the overall performance of the system. To clarify this point consider a client who repeatedly appears on push-lists of several segments during a short interval in the early stages of dissemination. As a result, many seeds will concurrently push the data messages and the client will get to the point to serve all those segments at about the same time. At this point, the client's uplink bandwidth is likely to become a bottleneck preventing all its segments to be effectively served. A more desirable approach would have considered the client's remaining bandwidth capacity when compiling the push-lists. This has the advantage of distributing load evenly across all peers.

To accomplish this goal, we require brokers to maintain their clients' remaining capacity and take this information into account while compiling push-lists. More specifically, a broker sorts its *regional* clients in ascending order of their remaining capacity and includes a client in a push-list only when no other peer (interested in the segment) has higher remaining upload bandwidth. This way, only clients with more bandwidth resources are included in push-lists during the early stages of dissemination. Once these clients reach the point to act as seeds, they can serve the segments to more peers and help boost the system's throughput.

5.4 System Operation over a Long Run

During the dissemination process, brokers must keep track of the progress of their *regional* subscribers in downloading the content. This information is required to be maintained per *segment* in order to avoid referencing a client who has already received a subset of the segments in future push-lists sent to seeds. However, once all segments of the content are downloaded by a client the broker can simply *forget* the details about individual segments and only maintain knowledge of the fact that the client possesses the entire content. This reduces the bookkeeping information to a minimum. Furthermore, since our approach is more concerned with timely dissemination of newly published content, brokers can gradually forget bookkeeping information associated with old content. This further improves the operation of our system over an extended period of time.

5.5 Break and Deceleration Policy

A good breaking policy improves the system's bandwidth efficiency by preventing arrival of data messages after a subscriber has successfully received a decodable set of data blocks. In our system, we use a *deceleration* technique for this purpose that allows receiving clients to carefully throttle arrival of the final blocks of data that are needed to decode each segment. For this purpose, the receiving client initially welcomes all other sending clients to unrestrictedly contribute and push data messages. However, once it approaches the final few blocks of data, it issues *break messages* to the seeding peers and enters a request/reply mode in which it specifically requests a specified number of coded data blocks from select seeds. If any of such blocks do not arrive within a desired time interval, the client simply requests data blocks from other seeding peers until all required blocks are available.

5.6 Traffic Shaping Strategy: Minimizing Cross-Regional Traffic

Our use of application layer brokers gives great flexibility in coordination of clients and provides opportunities for shaping the network's traffic patterns. To underscore the significance of this capability, consider the case of P2P file sharing applications. A major portion of ISPs' operational costs associated with file sharing applications is due to the large volume of traffic that crosses ISP domain boundaries for which ISPs must pay each other. This is largely due to the nature of P2P file sharing applications which are oblivious to the clients' residing domain and grant unrestricted file download and upload between all peers in the system. We believe that our broker-based solution can help mitigate such problems. For this purpose, clients connect to brokers within their ISPs (domain-specific brokers may be looked up using DNS names). In this setting, the regional content seeding which represents the bulk of data transfer takes place entirely within one domain. Furthermore, the cross-regional seeding of content can be brought to a minimum by limiting the size of the push-lists sent between brokers in different regions.

5.7 Size of Push-Lists

An important factor affecting system performance is the size of the push-lists compiled by P/S brokers. Smaller push-lists are greatly advantageous in the early stages of the dissemination since a seeding client (say the source) can dedicate its available upload bandwidth to push content to only a few other peers instead of breaking down its effort among many clients. As a result, smaller push-lists allow receiving peers to be ready to seed the content earlier. By contributing their resources, these fresh seeds boost the system's throughput and shorten the time required to push the content throughout the system. On the other hand, larger push-list have the advantage of reducing the number of interactions between brokers and seeds.

5.8 Broker Failure and Client Churn

Each broker monitors its neighbors in the topology by periodically exchanging heartbeat messages. Once a neighbor fails, our brokers use δ-fault-tolerance algorithms [17] to reconstruct the topology and avoid interruptions to P/S notifications routing. Detailed explanation of our fault-tolerance and recovery approach is outside the scope of this paper and can be found in [17].

Furthermore, a subscriber who leaves the system due to a crash failure, network partitioning, or simply becomes uninterested in its old subscriptions but fails to appropriately unsubscribe may leave behind obsolete subscription entries at P/S brokers. Our use of subscription leases enables brokers to identify and purge these entries from their routing tables once they timeout and are not renewed by the clients.

6 Evaluation

In this section, we report on our experimental evaluation results carried out using *Raccoon* and *Publiy*$^+$, our Java-based prototype implementations of a multi-threaded coding engine and the P/S system that uses *Raccoon* and the algorithms presented herein for content dissemination (both projects are distributed online as open-source).

Configuration	Clients	Brokers	Upload bandwidth
C-120	120	1	100 KB/s
C-300	300	5	100/200 KB/s
C-1000	1,000	5	200 KB/s

Table 2. Evaluation setup.

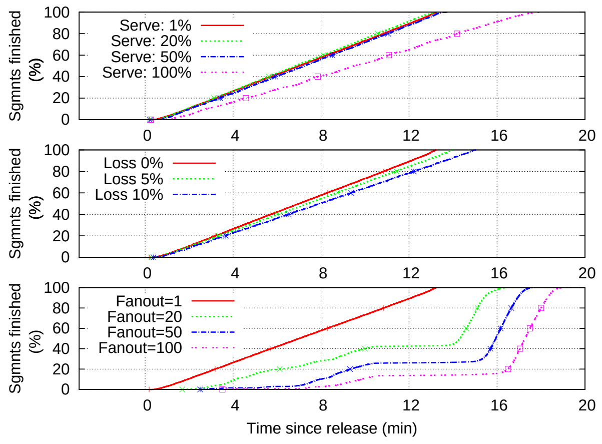

Figure 6. Impact of fanout (top), packet loss (middle) and content serving policy (bottom) on segment completion time (graph best viewed in color).

6.1 Evaluation Setup

For our evaluation, we used the SciNet High Performance Cluster [1]. Each computing node on the cluster is equipped with 8 Intel Xeon 2.53 GHz CPU cores with 8 MB of L2 cache. In our deployments, each client and broker has exclusive access to one dedicated CPU core. Machines are connected using gigabit switches but our clients are configured with a *capped* upload bandwidth of 100 – 200 KB/s. This is enforced at the client's socket communication layer and creates a realistic scenario of limited bandwidth availability over the Internet. Our physical network also has a very low latency, but we believe that this is not a major issue in interpretation of our experimental results, especially since in our throughput-intensive usage scenario bandwidth is *the far more significant determining factor*. We used three different network configurations with different client population sizes. The P/S overlay network in each configuration is composed of 1 – 5 brokers while the client population ranges from 120 – 1000 peers. This choice of small number of brokers is chosen to demonstrate that only a handful of P/S brokers can coordinate content dissemination among a much larger population of subscribers. Clients in each configuration are evenly distributed among brokers in the network. Furthermore, for the purpose of all our experiments we used block size of 10 KB and segment size of 1 MB. This implies that each block is coded with a 100 Bytes coding coefficient. This way, each data packet with the packet's header information, the coding coefficients and the coded data blocks have a combined maximum size of 10140 bytes and transfers 10000 bytes of useful data. This brings the overhead of network coded packets to about 1.4%. In our implementation, content descriptors are not included in every data messages and are sent to subscribers out-of-band only once. This is to avoid redundancy and improve efficiency. As we demonstrate in later sections, we believe that this overhead is negligible compared to the benefits of using network coding. Finally, unless otherwise specified all clients subscriptions match the published content. Table 2 summarizes the evaluation setups.

6.2 Impact of Source Fanout, Packet Loss and Content Serving Policy

Figure 6 illustrates the impact of three key factors, namely, content serving policy, packet loss, and source fanout (*i.e.*, the number of segments that the source is offering at any point in time). For all executions, we used C-300 with 200 KB/s uplink bandwidth allocation per peer. At time 0, a source publishes 100 MB of data that is delivered to all 300 clients.

The top graph in Figure 6 shows segment completion times of different content serving policies. The fastest completion time is when peers start to offer coded blocks as early as having received 1 data packet. This proves that early seeding is an

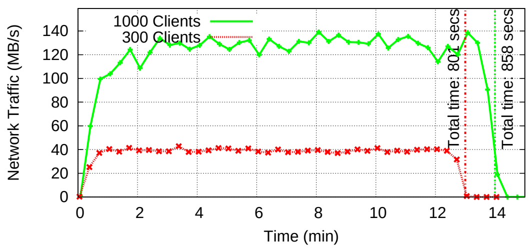

Figure 7. Aggregate network traffic due to dissemination of 100 MB of published content among 300 and 1000 peers.

effective way to hasten the dissemination process.

The middle graph in Figure 6 depicts the impact of packet loss on segment completion times. An advantage of network coding is that data transfer is inherently resilient to packet loss: If a receiver misses a packet, it simply waits for a future packet from the same sender or another peer. We experimented by subjecting communication links to 5% and 10% of message loss. The graph shows that the impact on the completion times is proportional to the loss injected.

The bottom graph in Figure 6 illustrates the impact of the source fanout parameter which controls the rate at which the publisher injects new segments into the network. More specifically, using a fanout of *x*, the source fully uploads *x* segments concurrently before moving to the next *x* segments. It can be seen from the graph that more restricted offering with lower fanout values perform better. Particularly interesting, is when all the segments are offered together (Fanout=100). In this situation, the source's resources is split among too many segments leading to a situation where none (or few) of the segments are completely downloaded at peers. Remember that a peer who fully possesses segment is free to send as many packets as necessary to others. On the other hand, if the peer's download is incomplete (a situation that is more likely when all segments are offered together), then number of blocks it can offer to each of the downloading peers is restricted (see Section 5.1).

Based on these results, we continue with the rest of our evaluation by only using the best performing values for the parameters: fanout of 1 and content serving policy that allows clients with only one data block to seed the segment Also, the injected loss for all future experiments is 0.

6.3 System Scalability and Throughput

An important advantage of our peer-assisted data dissemination approach is its scalability with the population of subscribers. This is due to the fact that the bandwidth available in the system grows organically as more subscribers specify mutual interest in some published content. Figure 7 illustrates the network-wide aggregate throughput of the system in which a 100 MB file is published by one publisher and disseminated among clients in configurations C-300 and C-1000. In the graph, configuration C-1000 achieves three times network-wide throughput compared to C-300 (from 40 MB/s to 130 MB/s). Furthermore, despite more than three-fold increase in the number of subscribers and aggregate volume of transferred data (from 30 GB to 100 GB), the total dissemination time increases from 801 to 858 seconds, less than 8% increase. This demonstrates that the client population size has a marginal impact on timely delivery of content to subscribers.

6.4 Source Contribution in Dissemination

A source must upload each segment of its published content at least once. A direct consequence of our initial seeding dissemination strategy is that a source (with a limited uplink bandwidth) attempts to maintain this lowerbound by delegating dissemination of each segment to cluster of initial seeding subscribers (see Section 5.1). This allows the source to utilize its available bandwidth more effectively, *i.e.*, to upload new segments as opposed to re-upload segments that it already injected in the system before. This improves the dissemination process, especially in our usage which resembles an ultimate flash-crowd scenario.

Figure 8 illustrates the source's number of uploaded blocks per content segment (the network configuration is C-1000). The average uploaded blocks per segment is 136 which is just about 30% higher than the minimum. Furthermore, the standard deviation is also very small (about 21 blocks) which indicates a smooth and even effort on part of the source to disseminate each data segment.

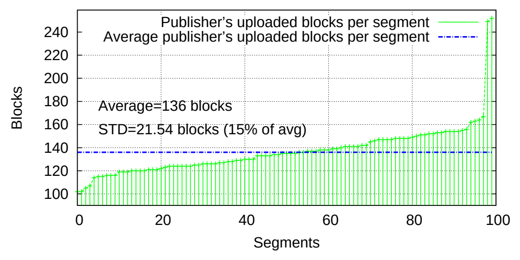

Figure 8. Source's contribution in terms of number of uploaded blocks per segment of published content (configuration C-1000, and 100 MB of published content).

6.5 Clients Contribution in Dissemination

Next, we studied the client's contribution in the dissemination process. Figure 9 illustrates the number of uploaded blocks of data per peer in configuration C-1000 with 10 publishers distributed uniformly among the 5 regions. For this case, also, the graph is largely smooth and evenly distributed (note that in our setup all subscribers have equal uplink bandwidth). In other words, full dissemination of 1 GB of published content among 1000 subscribers results in 1 TB of traffic and requires 100 million coded blocks in total. In this process, each peer participates with it fair share of offering 103 thousand blocks on average. Note that the extra 3 blocks are either due to sending of data blocks after a break is requested by the recipient but arrives late, or due to linear dependencies among the received data blocks which necessitates a retransmission.

6.6 Effectiveness of Traffic Shaping

A key objective in our design is to shape the traffic that flows within and between regions (see Section 5.6). This is an important requirement for many practical scenarios. For example, file sharing traffic that crosses ISP boundaries amounts to a significant portion of ISP costs whereas the traffic that flows within an ISP domain is virtually free.

We verified the effectiveness of our scheme in accomplishing this goal using configuration C-1000 with 5 distinct regions. There were one source that was located in one region. The source publishes 100 MB of content that is disseminated among all subscribers in all regions. Note that in our approach, the cross-regional traffic is confined to the blocks of data sent to the initial seeds of each segment. This is set to 2 peers for each segment which compared to the large population of subscribers in each region (about 200 peers) represents a tiny percentage. Furthermore, since these peers also exchange data as part of the cluster, this further reduces their reliance on the source.

Table 3 shows the results in terms of the percentage of network-wide aggregate traffic that flows between a sending and receiving region. It can be observed that the vast majority of data messages are sent within regions and this constitutes roughly about 20% for each region.

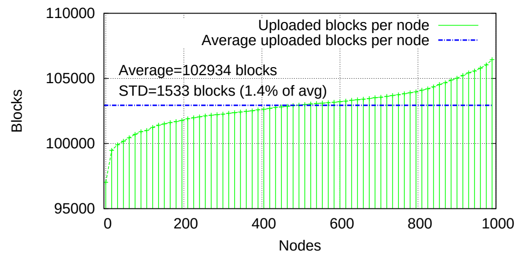

Figure 9. Subscribers contribution in terms of number of blocks offered for all content segments (configuration C-1000, and 1 GB of published content).

	Reg-1	Reg-2	Reg-3	Reg-4	Reg-5
Reg-1	19.57%	0.109%	0.109%	0.109%	0.108%
Reg-2	0.110%	19.57%	0.109%	0.109%	0.109%
Reg-3	0.110%	0.110%	19.55%	0.108%	0.108%
Reg-4	0.114%	0.114%	0.114%	19.57%	0.114%
Reg-5	0.110%	0.110%	0.110%	0.111%	19.49%

Table 3. Proportion of traffic within (shaded cells) and between (white cells) regions.

6.7 Shared, Multi-Source Environments

In a real deployment, the P/S system is shared among many sources and subscribers have varying interests. An important aspect of operation in such an environment is how the traffic flows due to different content sources coexist together. Ideally, content flows with overlapping subscribers compete but do not distort each other. To investigate this desirable behaviour, we experimented with a number of scenarios: In the first case, we had 10 sources each publish content that is of uniform interest to all 1000 subscribers. We observed that all content flows use an equal share of the system's bandwidth. In the second (and more interesting) scenario we grouped the publishers into groups of 5 such that content from the first, second and third group was of interest to one third, two thirds and all of the clients, respectively. Figure 10 illustrates the proportion of traffic due to each content with different popularity. The results were gathered using configuration C-1000 and involves dissemination of 1 TB of data with the above matching distribution among 1000 clients. The graph confirms that the traffic due to each content is proportional to its popularity. This implies that competing flows in a shared system are friendly to each other and do not inflict one another.

6.8 Comparison with BitTorrent

We now report on the experimental comparison results with the BitTorrent (BT) file sharing protocol [11]. The standard BT protocol does not provide native support for clients to receive newly released content unless they explicitly query the BT tracker where the requested content's ".torrent" file is registered. Despite this, many BT client applications provide a workaround to use RSS polling to accommodate automatic download of newly released content. After the content is released the client's next query results in a hit and the client subsequently proceeds to download the file in a regular manner. Furthermore, Bharambe *et al.* [8] observe that if the seed of a popular download has low uplink capacity (*i.e.*, comparable to other peers' bandwidth) the performance of BT degrades in a flash-crowd scenario.

We have dealt with both these issues as follows: First, the reactive push-based nature of the P/S model provides a readily available mechanism to initiate downloads at subscribers and also to shepherd them into direct exchange of data. This is reasonable, since in our usage scenarios companies that offer the data dissemination service also provide the broker network infrastructure. Second, the dissemination strategy of Section 5.1 boosts the bandwidth available to seed a given segment. This is especially effective in the early stages of dissemination of each segment. Finally, the use of network coding allows

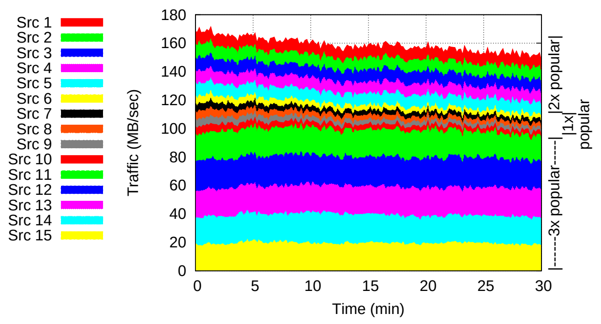

Figure 10. Sharing of network bandwidth is proportional to content popularity: Each colored stack represents the number of blocks sent for different content per second.

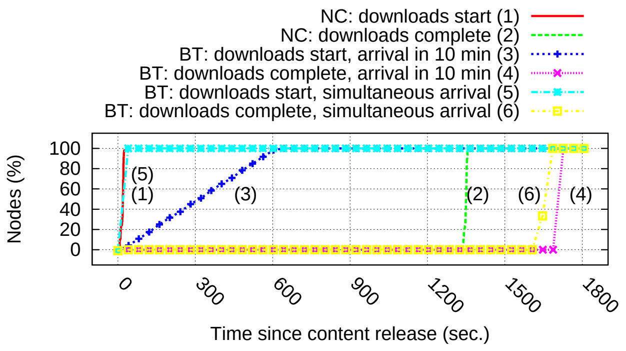

Figure 11. Comparison of start and completion times between network coding assisted push-based dissemination with network coding (NC) and BitTorrent (BT) [6].

subscribing peers to participate in content dissemination as early as having received a single 10 KB data block. This is a much smaller size than the minimum transferable data units in BT (also called segments) which is commonly 256 KB – 1 MB in size.

We now study the impact of these factors experimentally using configuration C-120. We first ran the system using *Publiy*$^+$ to disseminate 100 MB of published content from one source. Next, we ran the system with the Transmission BT [6] and OpenTracker [7], an open source BT client and tracker, respectively. The uplink bandwidth of the peers in all executions were capped at 100 KB/s. Figure 11 illustrates the results. The x-axis is the time since the release of the file and the y-axis is the percentage of peers who have started or finished download of the *entire* 100 MB of the file. The graphs marked by (1) and (2) indicate the start and completion of client downloads in our approach. It can be seen that all clients start their download almost immediately after the content becomes available.

On the other hand, the graph marked (3) shows the start of downloads for clients using BT configured with the standard RSS poll internal of 10 minutes. In this execution, the completion of all downloads takes about 1750 seconds (mark (4) in the figure). It is worth mentioning that a client who uses the default RSS poll interval of 10 minutes for files that are released weekly, say, a weekly TV show, will unsuccessfully query the tracker about 140 times a day or about 1000 times during a week. All but one of these queries results in an actual hit. Use of a shorter poll interval for better responsiveness increases this load at the tracker.

It might seem that the RSS poll interval is the source of this long completion times. In order to investigate this, we carried out another execution using BT in which all clients arrive immediately after the content is released (marked by (5) in the figure). Despite this fast arrival rate which requires an impractically low RSS polling interval, the downloads still took around 1700 seconds to finish (marked by (6) in the figure). This implies that our approach still outperforms BT by about 30% lower download times even when there is a hypothetical mechanism that somehow trigger immediate downloads at the clients. This gain is due to a combination of our initial segment seeding strategy and use of network coding for data dissemination.

7 Conclusions

Our goal in this paper was to bring the benefits of timely and selective publication distribution available in P/S systems to the field of bulk content dissemination and file sharing. To this end, we developed a two-layer architecture in which P/S brokers take the role of coordinators and guide subscribers with similar interests to engage in exchange of coded data blocks. Furthermore, this process is reactive as it is initiated as early as the content is published by the source. We also demonstrated that a variety of usage scenarios can benefit from our peer-assisted approach to accomplish timeliness and mass distribution of bulk content while avoiding the costs of deploying dedicated large-scale content replication servers. Finally, we have implemented and experimentally evaluated our algorithms. Our results confirm that by tapping into the clients available resources, a handful of deployed P/S brokers can oversee and coordinate timely distribution of large volumes of published content at scale.

References

[1] SciNet High Performance Computing Consortium. `http://www.scinet.utoronto.ca`.
[2] Dropbox. `http://www.dropbox.com`.
[3] Akamai technologies. `http://www.akamai.com`.
[4] Publiy publish/subscribe system. `http://www.eecg.utoronto.ca/~reza`.
[5] Spotify. `http://www.spotify.com`.
[6] Transmission bittorrent client. `http://www.transmissionbt.com`.
[7] Open tracker. `http://erdgeist.org/arts/software/opentracker`.
[8] A. R. Bharambe, C. Herley, and V. N. Padmanabhan. Analyzing and improving a bittorrent networks performance mechanisms. In *INFOCOM'06*, pages –1–1, 2006.
[9] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 2001.
[10] R. Chand and P. Felber. XNET: a reliable content-based publish/subscribe system. In *SRDS*, 2004.
[11] B. Cohen. Incentives build robustness in bittorrent. Technical report, bittorrent.org, 2003.
[12] G. Cugola, E. D. Nitto, and A. Fuggetta. The JEDI event-based infrastructure and its application to the development of the OPSS WFMS. *IEEE Trans. on Software Eng.*, 2001.
[13] E. Fidler, H.-A. Jacobsen, G. Li, and S. Mankovski. The PADRES distributed publish/subscribe system. *ICFI*, 2005.
[14] C. Fragouli, J.-Y. Le Boudec, and J. Widmer. Network coding: an instant primer. *SIGCOMM*, 2006.
[15] C. Gkantsidis, T. Karagiannis, P. Rodriguez, and M. Vojnovi. Planet scale software updates. In *SIGCOMM*, 2006.
[16] C. Gkantsidis and P. Rodriguez. Network coding for large scale content distribution. In *INFOCOM*, 2005.
[17] R. S. Kazemzadeh and H.-A. Jacobsen. Reliable and highly available distributed publish/subscribe system. In *SRDS'09*.
[18] R. S. Kazemzadeh and H.-A. Jacobsen. Adaptive multi-path publication forwarding in distributed publish/subscribe systems, 2011. Available at: `http://msrg.org`.
[19] J. Lee and G. de Veciana. On application-level load balancing in fastreplica. *Computer Communications*, 2007.
[20] G. Li, S. Hou, and H.-A. Jacobsen. A unified approach to routing, covering and merging in publish/subscribe systems based on modified binary decision diagrams. In *ICDCS'05*.
[21] Z. Liu, C. Wu, B. Li, and S. Zhao. Uusee: Large-scale operational on-demand streaming with random network coding. In *INFOCOM*, 2010.
[22] A. T. Nguyen, B. Li, and F. Eliassen. Chameleon: Adaptive peer-to-peer streaming with network coding. In *INFOCOM*, 2010.
[23] R. S. Peterson and E. G. Sirer. Antfarm: efficient content distribution with managed swarms. In *NSDI*, 2009.
[24] M. Wang and B. Li. R2: Random push with random network coding in live peer-to-peer streaming. *IEEE Journal on Selected Areas in Communications*, (9), 2007.
[25] P. C. Yunnan, P. A. Chou, Y. Wu, and K. Jain. Practical network coding, 2003.