SEEDS: The Social Internet Feed Caching and Dissemination Architecture

Ana Nunes[1], José Marques[2], and José Pereira[3]

[1] University of Minho aln@lsd.di.uminho.pt
[2] University of Minho trofa@lsd.di.uminho.pt
[3] University of Minho jop@di.uminho.pt

Abstract. Syndicated content in the Internet has been a huge success ever since the early days of RSS 0.9 and MyNetscape. Currently, it is the cornerstone of content push, ranging from podcasts to emerging Web 2.0 sites such as FriendFeed and Plexus. Unfortunately, the simple technology that makes publication and subscription very simple and flexible, thus explaining in part its success, is also limiting its usefulness in more demanding applications.
This paper proposes a novel distributed architecture for feed caching and dissemination. It leverages social networks as promoters of discovery and aggregation, and peer-to-peer protocols for content distribution, while providing an evolutionary upgrade path that does not disrupt current infrastructure or require changes to publishers' or consumers' habits.

1 Introduction

The Web has grown tremendously and content has evolved to be much more dynamic than in its early days. While some websites still feature essentially static content, others are updated daily, and in some cases (e.g., news oriented websites) the update rate is even higher, amounting to several updates a day.

Consider, for instance, that one tries to stay current with the content of a given website. It would be necessary to keep polling each of its pages to check for updates. If we also consider that instead of one client, a website will typically serve many and take into account the mere size of the Internet, bandwidth consumption and increased server load issues clearly emerge, as the number of clients (and therefore, the polling rate) go up. For example, according to Facebook statistics[4], more than 100 million users log on to Facebook each day.

Syndication[5] has changed the standard paradigm in order to cope with these events, by addressing the problem of knowing when an update is available. This is achieved in a content-push model, based on the publish/subscribe paradigm. Content publishers provide a Web feed: a XML file containing headlines and descriptions, which provides

[4] http://www.facebook.com/press/info.php?statistics
[5] Web syndication refers to how feeds are made available from a website, which is analog to how syndication works in broadcasting. For more information refer to http://www.internetcontentsyndication.org/downloads/whitepapers/content_creation.pdf

a summary of recently added or updated content, that users can subscribe to. However, the illusion of content-push is accomplished using hidden polling.

Web feed technology eases polling by collecting short descriptions of recent updates, with pointers, in a single file that can be polled more easily. Nonetheless, consider the following example. Each Wired Top Stories feed is sized at about 79KB, and from data available at Bloglines[6] it's subscribed to by at least 94,951 users. If separately polling the feed with an estimated refresh period of 2 hours, meaning 12 updates per day, daily polling-related traffic reaches 87.9 GB.

Current protocols such as RSS [1] and Atom [2] allow anyone to easily publish or subscribe to feeds by using simple tools. These protocols prove quite adequate for the large majority of feeds, featuring few subscribers and low data throughput. However, for high throughput or highly popular feeds, the polling mechanism becomes expensive for the publisher [3], making them less adequate.

Bandwidth consumption is aggravated by the generalized lack of support in both servers and clients for the retrieval of just new feed entries. Also, once a user subscribes to a feed, she'll likely maintain it for a long time and stemming from the use of feed readers and from the global nature of the Internet, polling will occur persistently. Some techniques have been proposed to try to minimize this problem [4], although with limited success.

Web feeds are also a major feature of social networking services, where they're being used to disseminate frequently updated information to interested users. For instance, Facebook uses feeds to highlight changes in social circles and also to disseminate information about any particular user; FriendFeed [7] condenses the content shared on multiple services by a group of selected users into a feed.

In order to mitigate bandwidth consumption at the publisher while achieving timely content delivery for the subscriber, we present an approach to feed dissemination using gossip in peer-to-peer groups. By leveraging social networks to aid in defining these groups, we address the issue of reducing the overhead associated with a generic gossip strategy and also derive additional functionality in the form of content discovery.

The rest of the paper is structured as follows: Section 2 describes the current feed dissemination architecture in detail. Section 3 states the goals of the current proposal, and in Section 4, an architecture is proposed to achieve these goals. Section 5 introduces the current prototype implementation. Section 6 compares this approach with previous work and Section 7 concludes the paper, underlining current research directions.

2 Background

Fig. 1 depicts the common Internet Web feed architecture. Web feeds are XML files composed of entries, which may be headlines, full-text articles, excerpts, summaries, and/or links to content on a website, along with metadata. RSS and Atom are XML-based document formats, both used for feeds. For futher details on these concepts and on how they relate to each other see [5] and/or [6]. Feeds can be subscribed to directly

[6] `http://bloglines.com`, a popular Web based feed reader that provides usage statistics.
[7] `http://friendfeed.com`

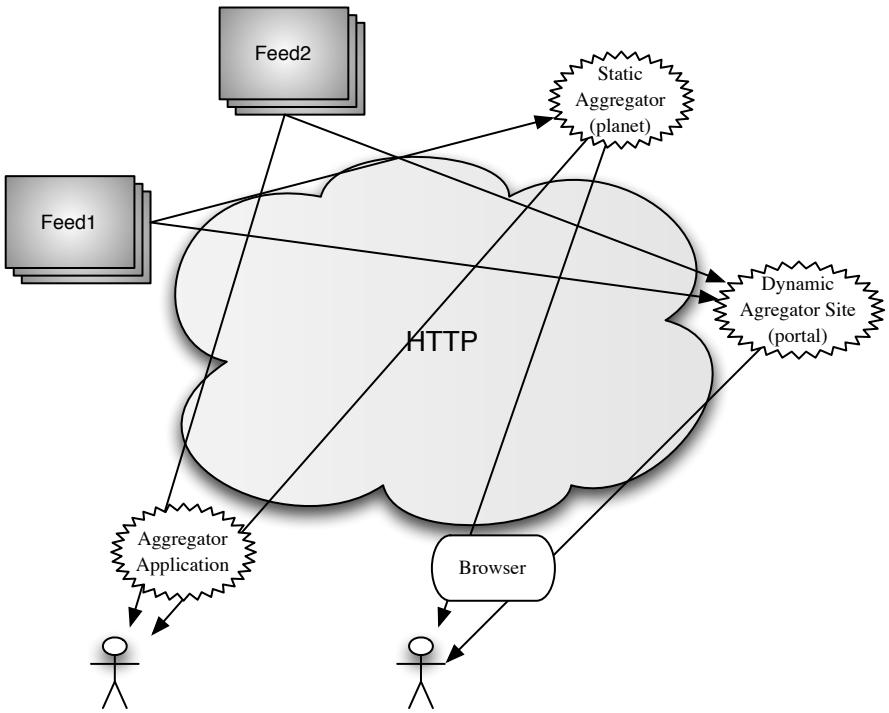

Fig. 1. Internet Web feed architecture.

by the end-user with an aggregator application, through a portal (the original intent), or by a planet. Aggregators, also known as feed readers or news readers, are applications that fetch feeds, usually in an automated manner, and present entries to the user (e.g. RSSOwl). Portals are websites featuring content from a static set of chosen feeds. Planets are websites featuring aggregated content of interest to a particular community of users. Notice that planets are a manual effort to identify communities with shared interests and provide them with commonly accepted content. Portals are much less prone to allow that sharing. On the other hand, planets don't allow any customization of the feeds by the end user. All aggregators perform caching, but only in portals and planets is the cache shared between multiple users.

Nowadays, Web feeds are being used to convey various types of content: from news headlines, to podcasts and video podcasts, to being a feature in Web 2.0 services. As an example from social networking services, more than 1 billion pieces of content (Web links, news stories, blog posts, notes, photos, etc.) are shared each week in Facebook, which are then published in Web feeds.

3 Goals

The proposed architecture aims at achieving the following goals:

Multiple Dissemination Protocols Content type and traffic patterns vary widely. For example, feeds can be mainly composed of text, normally for website related news, or feature richer content (feeds with audio/video attached), generating traffic with different characteristics. Broadcatching, which refers to automatic downloading of content published in feeds (e.g. TV series related feeds), imposes yet another traffic pattern. The number of participants (essentially subscribers) should also be considered. It makes sense not to use the same dissemination protocol for such different cases, so it is necessary to differentiate them and choose the right protocol. For example, small text based entries with a very high update rate could be disseminated via NeEM [7] and use BitTorrent [8] for audio and video content. This way, we are able to disseminate small content quickly and large content efficiently.

No Change to the Source One of the reasons why widespread adoption of other feed dissemination proposals (e.g. FeedTree [9]) hasn't occurred was that they relied heavily on the source to explicitly join the system. We should not expect the source to collaborate. Also, it should be possible to use the existing architecture and protocols without fundamentally altering them for this purpose.

No Change to the Target Clients shouldn't need to change their habits and be able to use any aggregator application to interact with the system, thus removing one of the biggest barriers for the adoption of this proposal. Also, some content dissemination frameworks require a specific client application, and some support only specific-purpose social networks (eg. Tribler [10]). It should be possible to use pre-existent social networks (e.g. Hi5 [8]) and take advantage of different dissemination architectures without requiring a special client for each one, selecting the most appropriate method automatically.

Enhanced Discovery and Aggregation We enhance the subscribers' experience with recommendations of new entries or feeds. These could be content-related or stem from social connections.

4 Architecture

As stated in the previous section, it is our goal to keep changes to a minimum for all parties in the current Web feed architecture. To comply with the no-changes guideline, the interface to the proposed system is simply the subscriber's news aggregator application of choice. To this end, we provide a proxy server to be deployed locally, at the client, which intercepts and interprets the subscriber's requests. The subscriber only needs to configure a third-party aggregator application to use the local proxy server.

Fig. 2 shows the proposed SEEDS architecture.

[8] `http://www.hi5.com`

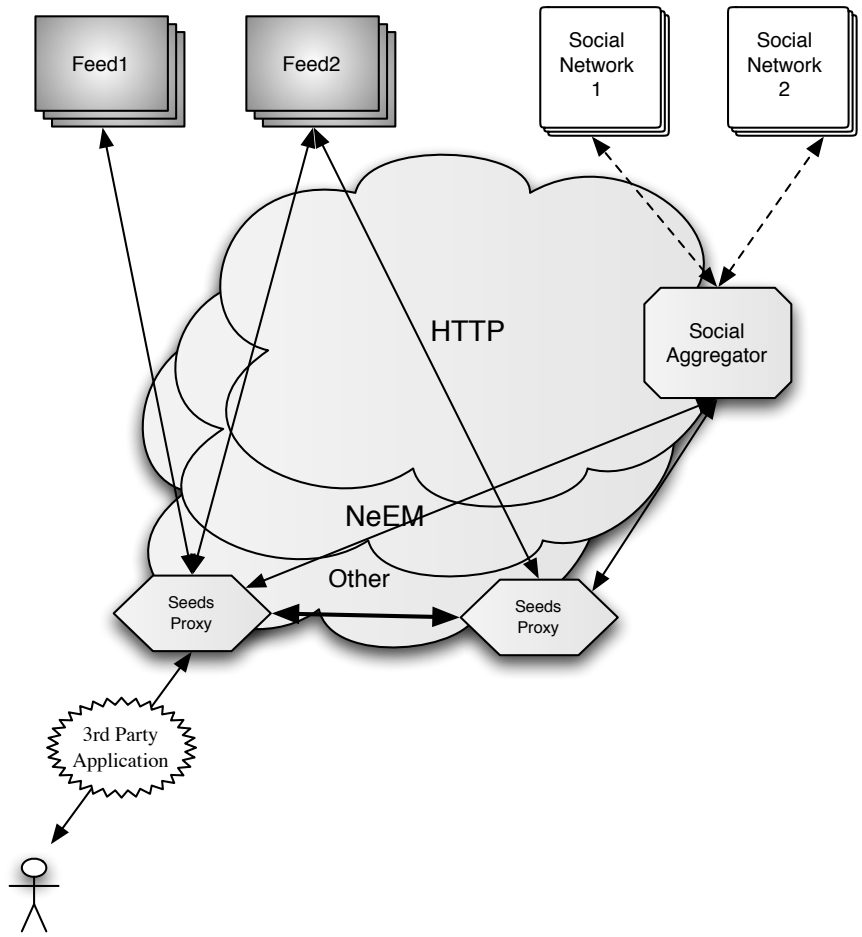

Fig. 2. SEEDS distribution and caching architecture.

The proxy identifies a feed request from a third-party application and either returns the corresponding cached feed, or requests it from the publisher. In case the feed is hitherto unknown, the proxy contacts a social aggregator to obtain instructions on how to handle the feed's caching and dissemination; if none is available, a local configuration is applied.

SEEDS proxies form peer-to-peer groups, translated into overlay networks. A multicast protocol (e.g. NeEM) instance is deployed in each group for entry dissemination. If deemed appropriate, other protocols can also be deployed. The feed-to-group mapping is defined by social aggregators, which gather information from social networking services along with perceived feed popularity. This information is also leveraged to provide Web feed recommendations. Social aggregators are separate, independently-deployed

modules. Each SEEDS proxy also provides a locally generated recommendation feed, which can easily be subscribed to by the user or simply ignored if desired.

Because of the overhead in group formation and maintenance, only popular feeds are allocated to groups. Feeds featuring few subscribers are simply cached locally, but its entries are not disseminated. The popularity threshold is determined by the social aggregator and dynamically adjusted.

Social aggregators can be promoted by social networking services as a service to its users. Thus, different services deploy different aggregators, leading to different group mappings for the same feeds.

5 Implementation

To illustrate our architecture, we developed a software prototype that implements the features outlined in the last section. The SEEDS architecture advocates a two module configuration. A proxy is deployed in the subscribers machine, which connects to a social aggregator on the internet. We examine each of these modules in detail.

5.1 Seeds Proxy

To ease the process of using SEEDS, the SEEDS proxy is provided as a Java WebStart application. The user can thus deploy it on her machine by, e.g. following a link in a social networking site of her choosing. This proxy is then able to use UPnP to configure itself in most home networking environments, even when behind firewalls.

The SEEDS proxy internal architecture is shown in Fig. 3. The proxy is composed of three main modules: the HTTP Servlet which interprets the requests received by third-party applications and publishes a local recommendation feed; the Cache Manager which handles feed caching, dissemination to groups and recommendation feed generation; and an HTTP Client, responsible for fetching feeds from the publishers and interacting with social aggregators to, among other interactions, obtain feed-related configuration.

The issue of discovering a social aggregator's URL is outside the scope of this work. Here is a step-by-step feed retrieval example:

1. A third-party aggregator application makes a feed request.
2. The request is interpreted by the HTTP Servlet module to check whether the URL effectively points to a feed and if so passes it on to the Cache Manager module.
3. The Cache Manager module does a lookup for the URL in the cache. If the URL is being cached, it retrieves the cached feed; otherwise, the HTTP Client retrieves the feed from the publisher and its configuration from a social aggregator. Then, the Cache Manager module deploys the dissemination protocol module, configuring it with a peer-to-peer group according to the configuration information it received. The relation between feeds and groups is many-to-one.
4. In the background, the Cache Manager is both sending updates to peer-to-peer groups and listening for other updates from them.

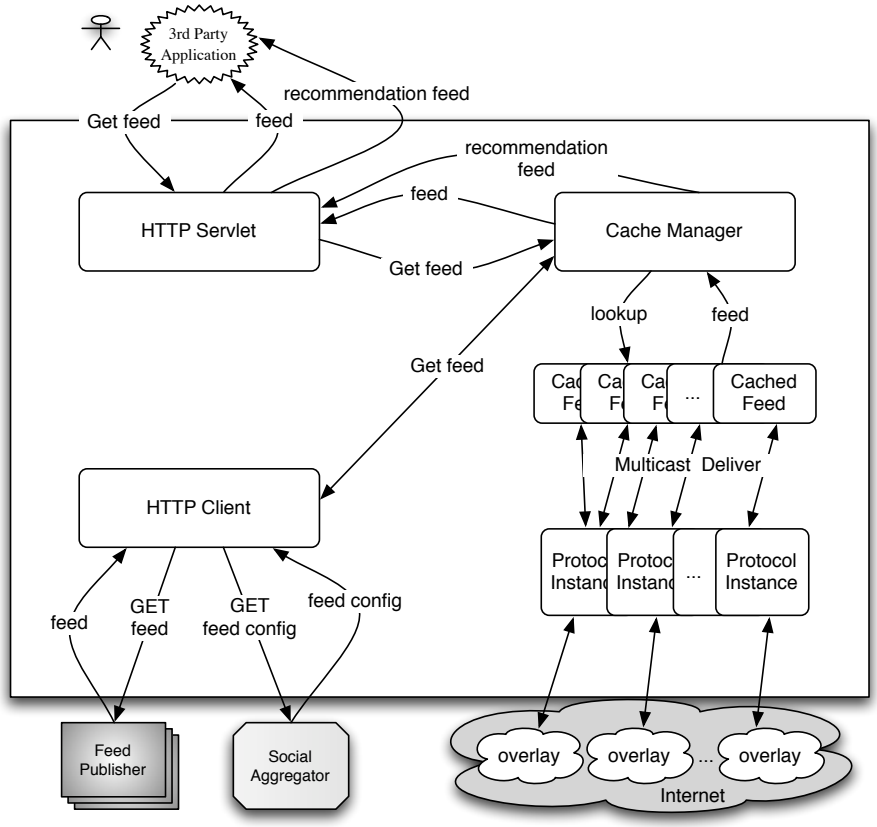

Fig. 3. SEEDS proxy implementation.

Notice that interaction with the social aggregator is only necessary when the proxy is presented with a new feed. Therefore if no social aggregator can be contacted, existing groups and known feeds are not affected. New feeds that are subscribed to in this situation, or for which the social aggregator simply does not provide a group, are cached locally, without dissemination, periodically refreshed from the publisher.

Although both the SEEDS architecture and the proxy's implementation are dissemination protocol agnostic, currently, we provide only an epidemic multicast protocol module. Epidemic multicast[11, 12], also dubbed probabilistic or gossip-based, is based on the simple procedure of relaying each received item to a small random subset of other nodes. This mechanism ensures these protocols are well suited to disseminate events to a large number of interested parties, while achieving stable high throughput, and coping with node or network faults. It has been shown that the probability that all nodes are informed can be made as large as desired, without reaching 1, and that messages spread exponentially fast. NeEM[7] is an implementation of epidemic multicast in wide-area

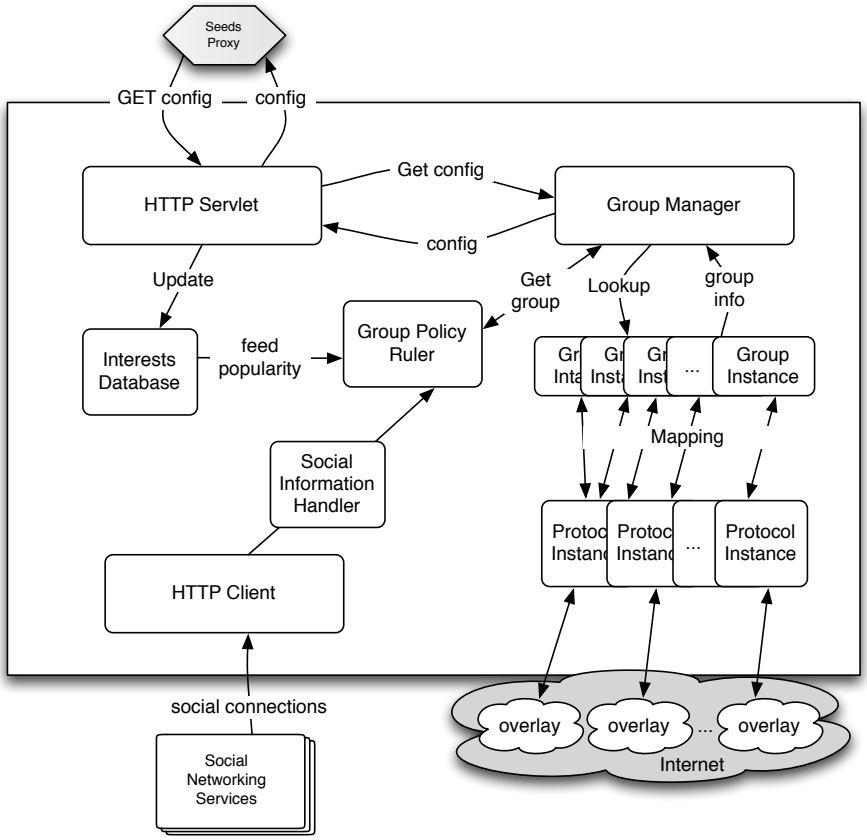

Fig. 4. SEEDS social aggregator server implementation.

networks relying on connection-oriented transport connections and on specific mechanisms to avoid network congestion. The resulting overlay network is automatically managed by the protocol. Its scalability and reliability make this protocol ideal for disseminating feed entries in groups where most peers are interested in the same set of feeds, and with high update rates. A protocol directed at a very large number of feeds with a small number of interested parties should also be provided as in FeedTree [9].

The recommendation feed is generated by the Cache Manager by collecting the feed URLs that are being disseminated in the groups but that are not yet subscribed to by the user and made available through the HTTP Servlet.

The SEEDS proxy implementation can be found at `http://seeds.sf.net`.

5.2 Social Aggregator

The social aggregator module, is presented in Fig. 4. Here, there are six main modules: the HTTP Servlet which receives group requests from SEEDS proxies; the Group

Manager which creates and manages dissemination groups; the Group Policy Ruler that determines feed-to-group mappings; the Interests Database which tallies feed configuration requests; the Social Information Handler which handles social networking and bookmarking information; and the HTTP Client which interfaces with social networking services. Both the Interests Database and the Social Information Handles support the Group Policy Ruler.

Here is a step-by-step group retrieval for a feed:

1. The HTTP Servlet interprets the configuration request and passes it on to the Group Manager.
2. The Group Manager queries the Group Policy Ruler for a feed-to-group mapping. The Group Policy Ruler considers both the data present at the Interests Database and the information afforded by the Social Information Handler to decide on the most convenient mapping. In fact, the Group Policy Ruler might not assign a group to the feed, if it finds that the its low popularity doesn't warrant it. The relation between groups and protocol instances is one-to-one.
3. Finally, the Group Manager returns a configuration stating the dissemination protocol to be used, the group to which the feed belongs to and other relevant parameters.

The Interests Database provides the Group Policy Ruler with accurate data related to the number of subscribers for each feed, enabling an efficient management of the dissemination groups.

The social networking information gathered by the Social Information Handler makes it possible to create groups according to social connections. In some social networking services, such as FriendFeed, social connections represent a similarity of interests. Therefore, the feeds mapped to groups formed in this manner should be of interest to the majority of participants. Besides reducing the overhead of disseminating entries to non-interested users, this mechanism also provides a source for recommendation: the feeds that are disseminated in a group but that are not yet subscribed by a given user will probably be of interest to him. Also, the tagging information found at social bookmarking services might be leveraged to categorize either feeds or specific entries in such a way that recommendations of related content can be generated.

Currently, feed-to-group mappings are statically determined, with a configuration drawn from existing Planets. Work is in progress to do it dynamically and provide a full-featured implementation of a social aggregator.

6 Related Work

There are a number of technologies that approximate either the caching and dissemination or the social aggregation and discovery aspects of SEEDS. For instance, Planet sites are used to aggregate syndicated Web content for online communities and to display it on a single page. There is, without question a social component to this, albeit a static one, as viewers do not control which feeds are aggregated. Also, for different areas of interest, a client has to visit different planets which impose sharp boundaries on communities.

FeedBurner [13] provides several services to feed publishers. To do so, it works like a centralized cache, in that the "burned" version of the feed is the one publicized. The original feed is checked for updates every 30 minutes. Although server load is moved away from the publisher's server to FeedBurner's servers, the bandwidth consumption problem remains the same. Also, the published "burned" feed's timeliness is limited.

FeedTree [9] is an approach directed at collaborative micronews delivery, and thus the closest to our proposal in terms of dissemination strategy. It does however use a single communication protocol (Scribe [14]) built on top of a peer-to-peer overlay network (Pastry [15]) and does not promote discovery and aggregation based on interests or social connections.

Tribler [10] is an open source BitTorrent client that leverages a social network to provide content recommendations and cooperative downloading, using a mechanism of bandwidth donation. Tribler users can create profiles, groups, add friends and tag content. Tribler's decentralized recommendation uses an algorithm based on an epidemic protocol, by exchanging download histories either with peers known to have similar tastes but also with peers chosen at random. Tribler is geared towards file-sharing, particularly video content. However, Tribler is focused only on multimedia content and requires custom tools to participate.

Last.fm [16] is a service that can be used for listening to music and that provides content recommendation based on user's profiles. When a user listens to a music, this information is added to a profile page, visible to other users. With this data, Last.fm suggests music to users, also allowing them to create personalized radio stations. In this social network, the notion of friendship exists along side the notion of neighborhood, taken from the similarity in musical taste. This service is designed to work with musical content only and is restricted to centrally supplied content.

FeedEx[17] is a news feed exchange system, where peers cooperate by exchanging feed documents with their neighbours. In FeedEx, content recommendation is provided, stemming from two sources: from entries rating (or votes) or by viewing a read as an implicit endorsement; from the feed subscription sets, determining similar interests. Neighbour selection is done at each peer, based on the overlap in subscription sets, considering also other factors like topological proximity.
FeedEx does provide interesting features, such as the construction of a distributed news archive and an incentive mechanism to minimize selfish behaviour. But, in this proposal, a specifically tailored peer-to-peer protocol is used, and it is heavily intertwined with the application itself. Thus, FeedEx does not support different dissemination protocols.

Content distribution networks attempt to optimize content delivery, implementing Web caches to store popular content closer to the user, thus reducing bandwidth requirements and server load, increasing reliability and also improving client response times. For example, the Coral Content Distribution Network [18] is an academic, free, peer-to-peer content distribution network that leverages the bandwidth of participating nodes as proxy servers, making it similar to a distributed Web proxy. However, network nodes are not users and there is no social component associated.

7 Conclusions and Future Work

In this paper we have introduced the architecture and the initial prototype implementation of SEEDS, an evolutionary proposal for Internet feeds infrastructure. While no changes are required to either the subscriber or the publisher, it should reduce server bandwidth usage when a large number of subscribers choose to participate. In contrast with previous proposals, SEEDS provides the means to foster the participation of such large number of participants by leveraging social networking and bookmarking sites. Finally, information that can be obtained from tagging performed within social bookmarking services can be used for group policy and suggestions. A full-featured implementation of the social aggregator is to be developed. Also, the issue of discovering social aggregators is to be addressed.

The current SEEDS prototype enables several interesting research problems. First, we are integrating SEEDS with a social networking service and working on obtaining traffic statistics to validate current hypotheses. On the other hand, we are also working on optimizing the multicast protocol and evaluating different protocols for different traffic patterns.

Finally, it is interesting to speculate whether improving Internet feeds such that (i) a very large number of subscribers to fast changing content can be supported without vast server resources; and (ii) aggregation and discovery is much easier and flexible, will enable radically new uses for this technology.

References

1. Rss protocol (specification). `http://www.rssboard.org/rss-specification`.
2. Atom protocol (ietf draft). `http://bitworking.org/projects/atom/draft-ietf-atompub-protocol-04.html`.
3. Matthew Hicks. Rss comes with bandwidth price tag. http://www.eweek.com/c/a/Messaging-and-Collaboration/RSS-Comes-with-Bandwi dth-Price-Tag/.
4. Randy Charles Morin. Howto rss feed state. http://www.kbcafe.com/rss/rssfeedstate.html.
5. S. Powers. What are Syndication Feeds. O'Reilly, 2005.
6. H. Wittenbrink. RSS And Atom: Understanding And Implementing Content Feeds And Syndication. Packt Publishing, 2005.
7. J. Pereira, L. Rodrigues, M. J. Monteiro, R. Oliveira, and A.-M. Kermarrec. NEEM: Network-friendly epidemic multicast. In SRDS, pages 15–24. IEEE Computer Society, 2003.
8. Bram Cohen. Incentives build robustness in bittorrent, May 2003.
9. D. Sandler, A. Mislove, A. Post, and P. Druschel. Feedtree: Sharing web micronews with peer-to-peer event notification, 2005.
10. J.A. Pouwelse, P. Garbacki, J. Wangand A. Bakker, J. Yang, A. Iosup, D. Epema, M.Reinders, M.R. van Steen, and H.J. Sips. Tribler: A social-based based peer to peer system. In 5th Int'l Workshop on Peer-to-Peer Systems (IPTPS), Feb 2006.
11. J. Pereira, L. Rodrigues, M. J. Monteiro, R. Oliveira, and A.-M. Kermarrec. Neem: Network-friendly epidemic multicast. Reliable Distributed Systems, IEEE Symposium on, 0:15, 2003.
12. P.T. Eugster, R. Guerraoui, A.M. Kermarrec, L. Massoulie, and AJ Ganesh. From epidemics to distributed computing. IEEE Computer, 37(5):60–67, 2004.
13. Feedburner. `http://www.feedburner.com/fb/a/home`.

14. M. Castro, P. Druschel, A.-M. Kermarrec, and A.I.T. Rowstron. Scribe: a large-scale and decentralized application-level multicast infrastructure. Selected Areas in Communications, IEEE Journal on, 20(8):1489–1499, Oct 2002.
15. Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. Lecture Notes in Computer Science, 2218, 2001.
16. Last.fm. `http://www.last.fm/`.
17. Seung Jun and Mustaque Ahamad. Feedex: collaborative exchange of news feeds. In WWW '06: Proceedings of the 15th international conference on World Wide Web, pages 113–122, New York, NY, USA, 2006. ACM.
18. M. Freedman, E. Freudenthal, and D. Mazi. Democratizing content publication with coral, 2004.