FeedTree: Sharing Web micronews with peer-to-peer event notification

Dan Sandler Alan Mislove Ansley Post Peter Druschel

Department of Computer Science
Rice University, Houston (TX)
{dsandler,amislove,abpost,druschel}@cs.rice.edu

Abstract

Syndication of micronews*, frequently-updated content on the Web, is currently accomplished with RSS feeds and client applications that poll those feeds. However, providers of RSS content have recently become concerned about the escalating bandwidth demand of RSS readers. Current efforts to address this problem by optimizing the polling behavior of clients sacrifice timeliness without fundamentally improving the scalability of the system. In this paper, we argue for a micronews distribution system called FeedTree, which uses a peer-to-peer overlay network to distribute RSS feed data to subscribers promptly and efficiently. Peers in the network share the bandwidth costs, which reduces the load on the provider, and updated content is delivered to clients as soon as it is available.*

1 Introduction

In the early days of the Web, static HTML pages predominated; a handful of news-oriented Web sites of broad appeal updated their content once or twice a day. Users were by and large able to get all the news they needed by surfing to each site individually and pressing `Reload`. However, the Web today has experienced an explosion of *micronews:* highly focused chunks of content, appearing frequently and irregularly, scattered across scores of sites. The difference between a news site of 1994 and a weblog of 2004 is its flow: the sheer volume of timely information available from a modern Web site means that an interested user must return not just daily, but a dozen times daily, to get all the latest updates.

This surge of content has spurred the adoption of RSS, which marshals micronews into a common, convenient format. Instead of downloading entire web pages, clients download an RSS "feed" containing a list of recently posted articles. However, RSS specifies a polling-based retrieval architecture, and the scalability of that mechanism is now being tested. There is growing concern in the RSS community over these scalability issues and their impact on bandwidth usage, and providers of popular RSS feeds have begun to abbreviate or eliminate their feeds to reduce the bandwidth stress of polling clients.

The current RSS distribution architecture, in which all clients periodically poll a central server, has bandwidth requirements that scale linearly with the number of subscribers. We believe that this architecture has little hope of sustaining the phenomenal growth of RSS [10], and that a distributed approach is needed. The properties of peer-to-peer (p2p) overlays are a natural fit for this problem domain: p2p multicast systems scale logarithmically and should support millions of participating nodes. Therefore, we argue that RSS feeds can be distributed in a way that shares costs among all participants. By using p2p event notification to distribute micronews, we can reduce dramatically the load placed on publishers, while at the same time delivering even more timely service to clients than is currently possible. We sketch this system, called FeedTree, and go on to show how it can be deployed incrementally.

The remainder of this paper is organized as follows. Section 2 provides background on RSS and the RSS bandwidth problem. Section 3 discusses related work to improve RSS, and section 4 presents the design of FeedTree. Section 5 describes our prototype FeedTree implementation. Section 6 concludes.

2 Background

2.1 RSS

RSS[1] refers to a family of related XML document formats for encapsulating and summarizing timely Web content. Such documents (and those written in the Atom syndication format [1], a recent entry in the specification fray) are called *feeds*. A Web site makes its updates available to RSS client software (variously termed "readers" and

[1] There is some disagreement [4] over the exact expansion of this acronym. When Netscape first specified version 0.9 of RSS [19], it did so under the name "RDF Site Summary;" the acronym has since been taken to stand for "Rich Site Summary" or "Really Simple Syndication." The subtleties of the many debates over format versions, nomenclature, and ideology are omitted here.

"aggregators") by offering a feed to HTTP clients alongside its conventional HTML content. Because RSS feeds are designed for machines instead of people, client applications can organize, reformat, and present the latest content of a Web site—or many sites at once—for quick perusal by the user. The URL pointing to this feed is advertised on the main Web site.

By asking her RSS reader to *subscribe* to the URL of an RSS feed, a user instructs the application to begin fetching that URL at regular intervals. When it is retrieved, its XML payload is interpreted as a list of RSS *items* by the application. Items may be composed of just a headline, an article summary, or a complete story in HTML; each entry must have a unique ID, and is frequently accompanied by a permanent URL ("permalink") to a Web version of that entry. To the user, each item typically appears in a chronologically-sorted list; in this way, RSS client applications have become, for many users, a new kind of email program, every bit as indispensable as the original. An RSS aggregator is like an inbox for the entire Internet.

2.2 RSS bandwidth

Just as major news outlets have begun to discover RSS and to expose their audiences to this burgeoning technology [10, 11, 14], the RSS technical community is abuzz with weaknesses exposed by its runaway adoption. Chief among these is the so-called "RSS bandwidth problem." Essentially, Web servers which make RSS feeds available tend to observe substantially greater traffic loads as a result, out of proportion to any observable interactive visitor trend. Consequently, some sites have implemented self-defense mechanisms (*e.g.* smaller RSS feed sizes, or enforced limits on access) in an attempt to address the problem [12]. This situation is most likely the effect of many behaviors working in concert:

Polling. For each feed to which a user is subscribed, an RSS application must issue repeated HTTP requests for that feed according to some set schedule. Sites which offer RSS feeds must satisfy one request for every user, many times a day, even if there is no new content.

Superfluity. The RSS data format is essentially static; all entries are returned every time the feed is polled. By convention, feeds are limited to some N most recent entries, but those N entries are emitted for every request, regardless of which of them may be "new" to a client. While this bandwidth problem could be helped by introducing a diff-based polling scheme, all such requests would have to be processed by the RSS provider, which adds more processing load.

Stickiness. Once a user subscribes to an RSS feed, she is likely to retain that subscription for a very long time, so this polling traffic can be counted on for the foreseeable future. If a previously-obscure Web site becomes popular for a day, perhaps by being linked to from popular Web sites, its browsing traffic will spike and then drop off over time. However, if that site offers an RSS feed, users may decide to subscribe; in this case, the drop in direct Web browsing is replaced by a steady, unending load of RSS client fetches. Such a Web site might be popular for a day, but it may have to satisfy a crowd forever. [25, 23]

Twenty-four-hour traffic. RSS client applications are commonly running on desktop computers at all hours, even when a user is not present; the diurnal pattern of interactive Web browsing does not apply. While the global nature of Web users may generate "rolling" 24-hour traffic, global use of RSS readers generates persistent 24-hour traffic from all over the Earth.

It is easy to see how a website may suffer for publishing RSS feeds. The most popular feed on Bloglines[2] is Slashdot.org, which has about 17,700 subscribers as of this writing. If each of those subscribers were using personal aggregation software (desktop clients), Slashdot's headlines-only RSS feed (about 2 kilobytes for a day's worth of entries, and typically polled half-hourly) would be transferred 850,000 times a day, for a total of 1.7 GB of data daily. *The New York Times* recently introduced a suite of RSS feeds for its headlines; the front page alone claims 7,800 subscribers, but the sum of subscribers to all its feeds comes to 24,000. Feeds from the *Times* tend to be around 3 KB, or 3.5 GB of data per day with 30-minute polling. For websites wishing to provide their RSS readers with deeper content, the problem is worse still. Boing Boing, a popular weblog, chooses to publish complete HTML stories in RSS and Atom; 11,500 subscribers might receive 40 KB for each RSS request. To provide this service, Boing Boing must be able to accommodate 22 GB/day of RSS traffic alone. If the BBC News Web site is truly "updated every minute of every day,"[3] its RSS subscribers (18,000 to its various feeds on Bloglines) are unable to take advantage of it: the bandwidth demands of those subscribers polling every minute would be virtually insatiable.

[2] Bloglines (http://bloglines.com), a popular Web-based RSS reading application, offers subscription figures for the feeds it aggregates. We will use these figures (as of late October 2004) as a very crude approximation of reasonable RSS readership. Though Bloglines certainly polls RSS feeds only once for its thousands of subscribers, anecdotal evidence suggests that traditional desktop RSS client usage outweighs Web-based client usage, so we can regard these figures as a lower bound on overall RSS polling load.

[3] As advertised on http://news.bbc.co.uk.

3 Related Work

3.1 Improving the polling process

Several proposals have been submitted to ease the pain of RSS on webmasters. Many of these are described in detail in the RSS Feed State HOWTO [17]; examples include avoiding transmission of the feed content if it hasn't changed since the client's last request, `gzip` compression of feed data, and clever ways to shape the timetable by which clients may poll the RSS feed.

Unfortunately, because the schedule of micronews is essentially unpredictable, it is fundamentally impossible for clients to know *when* polling is necessary. Werner Vogels puts it succinctly: Uncontrolled Polling of RSS Resources Does Not Scale [24].

3.2 Outsourcing aggregation

Several online RSS service providers (essentially, Web-based RSS readers) have proposed alternative solutions [2, 3]. In these "outsourced aggregation" scenarios, a centralized service provides a remote procedure interface which end-user applications may be built upon (or refactored to use). Such an application would store all its state—the set of subscribed feeds, the set of "old" and "new" entries—on the central server. It would then poll only this server to receive all updated data. The central RSS aggregation service would take responsibility for polling the authoritative RSS feeds in the wider Internet.

This addresses the bandwidth problem, in a way: A web site owner will certainly service fewer RSS requests as end users start polling the central service instead. The operators of these central services will definitely have bandwidth issues of their own: they will now be at the center of all RSS traffic.

There is a far more insidious danger inherent in this approach, however: a central point of control, failure, and censorship has now been established for all participating users. A central RSS aggregation service may: *(i)* experience unavailability or outright failure, rendering users unable to use their RSS readers, *(ii)* elect to discontinue or change the terms of its service at any time, or *(iii)* silently modify, omit, or augment RSS data without the user's knowledge or consent.

Modification of RSS data by the central aggregator may come in the form of optimized or normalized RSS formatting (a useful feature, since syndication formats found in the wild are frequently incompatible [21]), but might take more dangerous forms as well: it may modify or corrupt the entries in a feed, or it may add advertising or other supplemental yet non-indigenous content to those feeds.

In summary, a third party may not be a *reliable* or *trustworthy* entity, and so it cannot be guaranteed to proxy micronews for client applications. For these reasons, centralized RSS aggregation is most likely not a viable long-term solution.

4 FeedTree

4.1 Group communication with overlay networks

The obvious alternative to polling for data is to distribute that data, as it becomes available, to lists of subscribers. This approach may be adequate for small subscription lists (for example, e-mail lists), but it will not scale to accommodate the growing subscription demands of Web site syndication. Furthermore, while such an approach may reduce the overall bandwidth usage of RSS (by reducing unnecessary fetches), it does nothing to alleviate the per-update stress on network links close to the source.

To address these problems, we look to peer-to-peer overlay networks, which offer a compelling platform for self-organizing subscription systems. Several overlay-based group communication systems, including Scribe [7], offer distributed management of group membership and efficient routing of subscription events to interested parties in the overlay.

We propose FeedTree, an approach to RSS distribution based on peer-to-peer subscription technologies. In FeedTree, timely Web content is distributed to interested parties via Scribe, a subscription-based event notification architecture. Although we chose to base this design on Scribe, there is no reason it could not be deployed on any group communication system that provides similar performance characteristics. In such a system, content may be distributed as soon as it becomes available; interested parties receive these information bursts immediately, without polling the source or stressing network links close to the source.

4.2 Scribe

Scribe [7] is a scalable group communication system built on top of a peer-to-peer overlay such as Pastry. Each Scribe group has a 160 bit *groupId* which serves as the address of the group. The nodes subscribed to each group form a multicast tree, consisting of the union of Pastry routes from all group members to the node with nodeId numerically closest to the groupId. Membership management is decentralized and requires less than $\log n$ messages on average, where n is the number of nodes in the overlay.

Scribe has been shown to provide cooperative multicast that is efficient and low overhead [7]. The delay stretch is approximately double that of IP multicast and comparable to other end system multicast systems such as ESM [8] and Overcast [13]. Link stress is also low and less than twice that of IP muliticast. When there are

a large number of groups in the system, as is expected in FeedTree, the load is naturally balanced among the participating nodes. Scribe uses a periodic heartbeat mechanism to detect broken edges in the tree; this mechanism is lightweight and is only invoked when there are no messages being published to a group. It has been shown to scale well to both large groups and to a large number of groups. These properties make it a good fit for building large scale event notification systems like FeedTree.

4.3 Architecture

When FeedTree publishing software wishes to deliver an update to subscribers, the following steps are taken (in addition to refreshing a conventional RSS feed URL):

- **A complete RSS document is created** to contain one or more pieces of timely micronews. Each item is assigned a timestamp and a sequence number, to aid clients in the detection of omitted or delayed events.
- The RSS data is then **signed with the publisher's private key.** This is essential to establishing the authenticity of each published item.
- The signed RSS document is **multicast in the overlay** to those peers who have subscribed to a Scribe group whose topic is (a hash of) the feed's **globally unique ID**, trivially defined to be the canonical URL of the advertised RSS feed.
- Peers receiving the message verify its signature, parse the RSS data, and add it to the local RSS application state as if it were a conventional, polled RSS feed. **The user can be notified immediately** of the new entries.

FeedTree aware client applications should be able to examine conventional RSS feed data to discover if updates to that feed will be published through FeedTree. To do this, FeedTree metadata can be added to the RSS document structure to signal that it is available for subscription in the overlay. In this way, a FeedTree application bootstraps the subscription process with a one-time HTTP request of the conventional feed. All future updates are distributed through incremental RSS items published in FeedTree.

Each RSS feed to be published through FeedTree should advertise a *time-to-live* value, the maximum interval between FeedTree events. (Many RSS feeds already include such a value, to indicate the minimum allowed polling period for clients.) If the publisher observes that no new FeedTree events were generated during this interval, the publisher must generate a heartbeat event. These heartbeats allow subscribers to know conclusively that no published items were lost during the time-to-live period.

It is desirable for all publishers to cryptographically sign their published RSS data, so that clients may be able to trust the Scribe events they receive.[4] The conventional RSS feed should also include the URL and fingerprint of the publisher's certificate, so that clients may retrieve (and cache) the credentials necessary to validate the integrity of signed RSS data.

4.4 Adoption and deployment

The proliferation of conventional RSS has depended largely on the availability of quality tools to generate RSS data; FeedTree will be no different. Developers have several opportunities to provide support for this system. We break down the deployment scenarios into those that support FeedTree fully, and those that serve as "adapters" to ease transition for legacy RSS systems.

4.4.1 Full FeedTree support

Publishers. Web content management systems (such as weblog publishing packages or traditional workflow-based CMS software) join the overlay by becoming long-lived FeedTree nodes. When new content is posted, the publishing software automatically creates a new FeedTree message and publishes it to multicast tree.

Readers. RSS-reading applications join the FeedTree peer-to-peer network as well. By doing so, they become part of the global FeedTree service, distributing the network and processing loads of RSS event forwarding. The user interface for an RSS client should remain unchanged; the user subscribes to RSS feeds as she would do ordinarily, and the software takes care of detecting and bootstrapping a FeedTree subscription if it is available. New RSS items are made available to users as soon as the FeedTree events are received by the application.

4.4.2 Incremental FeedTree support

Publishers. Legacy publishing software that currently emits valid RSS can be adapted to FeedTree with a "republishing" engine running on (or near) the Web server. This tool would poll the legacy RSS feed on an aggressive schedule, sifting out new content and distributing it via FeedTree. Such a republishing tool might even be operated by a third party, in case the owner is slow to deploy FeedTree. This is already a common emergent behavior of the RSS community; several Web sites currently "scrape" the HTML of popular sites and redistribute that content in RSS format. It is up to a user to decide whether or not to trust this third-party proxy feed.

[4] Even though the general benefits of signed content are independent of the FeedTree architecture, we believe our design offers both an excellent opportunity and a compelling need to introduce signed RSS.

Readers. Until RSS applications support FeedTree natively, users can still contribute to the RSS bandwidth solution by running a local FeedTree proxy. The proxy would listen receive RSS data through FeedTree instead of through conventional means. Existing end-user RSS tools could poll a local FeedTree proxy as often as desired without unnecessary bandwidth usage. Users would then see new FeedTree items sooner than they would under a more conservative polling policy.

4.5 Discussion

4.5.1 Benefits for participants

The system we propose offers substantial benefits for both producers and consumers of RSS data. The chief incentive for content providers is the lower cost associated with publishing micronews: large Web sites with many readers may offer large volumes of timely content to FeedTree clients without fear of saturating their network links, and a smaller Web site need not fear sudden popularity when publishing a FeedTree feed. FeedTree also offers publishers an opportunity to provide differentiated RSS services, perhaps by publishing simple (low-bandwidth) headlines in a conventional RSS feed, while delivering full HTML stories in FeedTree.

End users will receive even *better* news service with FeedTree than is currently possible. While users currently punish Web sites with increasingly aggressive polling schedules in order to get fresh news, no such schedule will match the timeliness of FeedTree, in which users will see new items within seconds—not minutes or hours. If publishers begin to offer richer micronews through FeedTree, we believe users will be even more likely to use the system. Finally, since RSS readers are generally long-running processes, building FeedTree into the RSS clients will likely result in a stable overlay network for the dissemination of micronews.

4.5.2 Recovery of lost data

Because Scribe offers a best-effort service, failures and node departures within the multicast tree may result in FeedTree clients missing events. In this case, the client will detect a gap in the sequence numbers or an overdue heartbeat. A client may query its parent to recover the missing items; in order to satisfy such a request, each member of the system will keep a small fixed buffer with the last n items in the feed. As a fallback, missing items may be recovered by retrieving the conventional RSS feed by HTTP as in the bootstrapping phase. FeedTree clients may also be offline for periods, during which time they will miss update events. Clients coming online should "catch up" by examining the HTTP-based RSS feed for previously-unseen items during bootstrapping.

A malicious node acting as an interior node in a Scribe tree can suppress events. This attack can be addressed by distributing the responsibility of the Scribe root among several nodes and by routing around non-root interior nodes that fail to forward events. We omit the details due to space limitations.

4.5.3 Overhead

The bandwidth demands made on any individual participant in each multicast tree are quite innocuous. For example, an RSS feed generating 4 KB/hour of updates will cause an interior tree node with 16 children to forward less than 20 bytes per second of outbound traffic. Due to the extremely low forwarding overhead, we believe that the motivation for freeloading is very small. In the future, we expect richer content feeds, and consequently, the potential incentive for freeloading may increase. Incentives-compatible mechanisms to ensure fair sharing of bandwidth [20] can be applied if most users subscribe to several feeds, which is a common model of RSS usage. We intend to explore integrating these techniques with FeedTree in future work.

5 Development Status

In order to validate our design for FeedTree, we have developed a software prototype which follows the design outlined in Section 4. The `ftproxy` daemon serves as an intermediary for conventional RSS client software; an HTTP request for a given RSS feed is satisfied by `ftproxy`, which constructs a new ad-hoc RSS document from recent FeedTree messages received for that feed.

When subscribing to a new RSS feed, the proxy first checks to see if that feed is already being published through FeedTree. If the feed is not being published, `ftproxy` will "volunteer" to republish the RSS feed: it begins polling the RSS feed as if it were a conventional RSS reader. New items are published through FeedTree; if a polling interval yields no new items, the proxy publishes a "no news" heartbeat event. This event informs other listening `ftproxy` instances that the feed is already being polled by another volunteer.

In the current implementation, this mechanism is generalized to allow multiple instances of `ftproxy` to poll a single RSS feed cooperatively, providing updates to FeedTree with higher frequency than conventional RSS polling. To "overload" a feed by a factor of N, `ftproxy` will choose to volunteer if it observes fewer than N FeedTree events for that feed during its polling interval. On average, an RSS feed with a minimum polling period of T will have an effective FeedTree refresh period of T/N. The polling schedule for volunteers is jittered to help avoid synchronicity.

At the time of this writing, we are running a small FreeTree deployment internally at Rice. We plan to soon

expand the distribution to the PlanetLab testbed for further experimentation and validation.

6 Conclusions and Future Work

The current RSS polling mechanism has been said to scale well because "its cost is almost directly proportional to the number of subscribers" [5]. In fact, linear cost is typically an indicator of poor scaling properties, especially when that cost is focused on one member of a distributed system. It is likely that the further growth of RSS adoption will be badly stunted, without substantial change to the way micronews is distributed.

The proposed FeedTree subscription system for RSS takes advantage of the properties of peer-to-peer event notification to address the bandwidth problem suffered by Web content providers, while at the same time bringing micronews to end users even more promptly than is currently possible. Self-organizing subscription systems like Scribe offer scalability that cannot be matched by any system designed around resource polling.

Building upon the FeedTree distribution system, we foresee a potential for entirely new services based on RSS which cannot be accomplished today. By using single-writer logs [18] in combination with a distributed storage mechanism such as a DHT [22, 15, 9], we can record permanently every RSS item published, allowing a distributed archival store of micronews across the Internet. Clients of such a system would easily be able to find out what they "missed" if they had been offline for so long that old RSS items are no longer available in any conventional, static RSS feed. Another area for future work is anonymous RSS feeds involving an anonymizing peer-to-peer routing system, such as AP3 [16]. Finally, we can envision the use of cooperative multicast (such as SplitStream [6]) to distribute large files—such as software, audio, and video—as part of FeedTree feeds.

References

[1] Atom Syndication Format. http://www.atomenabled.org/developers/syndication/.

[2] Bloglines Web Services. http://www.bloglines.com/services/.

[3] NewsGator Online Service. http://www.newsgator.com/ngs/.

[4] RSS protocol (Wikipedia entry). http://en.wikipedia.org/wiki/RSS_(protocol).

[5] RSS for Mac OS X Roundtable. http://www.drunkenblog.com/drunkenblog-archives/000337.html, Oct. 2004.

[6] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: High-bandwidth multicast in cooperative environments. In *Proc. SOSP'03*, Oct. 2003.

[7] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. SCRIBE: A large-scale and decentralized application-level multicast infrastructure. *IEEE JSAC*, 20(8), Oct. 2002.

[8] Y. Chu, S. Rao, and H. Zhang. A case for end system multicast. In *ACM Sigmetrics*, pages 1–12, June 2000.

[9] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *Proc. ACM SOSP'01*, Banff, Canada, Oct. 2001.

[10] L. Gomes. How the next big thing in technology morphed into a really big thing. *The Wall Street Journal*, Oct. 2004.

[11] H. Green. All the news you choose – on one page. *BusinessWeek*, Oct. 2004. http://www.businessweek.com/magazine/content/04_43/b3905055_mz011.htm.

[12] M. Hicks. RSS comes with bandwidth price tag. *eWeek*, Sept. 2004. http://www.eweek.com/article2/0,1759,1648625,00.asp.

[13] J. Jannotti, D. Gifford, K. Johnson, M. Kaashoek, and J. O'Toole. Overcast: Reliable multicasting with an overlay network. In *OSDI 2000*, San Diego, CA, 2000.

[14] V. Kopytoff. One-stop way to read news, blogs online: RSS allows users to get free, automatic feeds. *The San Francisco Chronicle*, Oct. 2004. http://www.sfgate.com/cgi-bin/article.cgi?file=/chronicle/archive/2004/10/25/BUG1U9ES301.DTL.

[15] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. OceanStore: An architecture for global-scale persistent store. In *Proc. ASPLOS'2000*, Cambridge, MA, November 2000.

[16] A. Mislove, G. Oberoi, A. Post, C. Reis, P. Druschel, and D. S. Wallach. AP3: Cooperative, decentralized anonymous communication. In *Proc. SIGOPS-EW*, Leuven, Belgium, Sept. 2004.

[17] R. C. Morin. HowTo RSS Feed State. http://www.kbcafe.com/rss/rssfeedstate.html, Sept. 2004.

[18] A. Muthitacharoen, R. Morris, T. Gil, and B. Chen. Ivy: A read/write peer-to-peer file system. In *Proc. OSDI'02*, Boston, MA, December 2002.

[19] Netscape Communications Corp. *My Netscape Network*, Mar. 1999. http://www.purplepages.ie/RSS/netscape/rss0.90.html.

[20] T.-W. J. Ngan, A. Nandi, A. Singh, D. S. Wallach, and P. Druschel. On designing incentives-compatible peer-to-peer systems. In *Proc. FuDiCo'04*, Bertinoro, Italy, June 2004.

[21] M. Pilgrim. The myth of RSS compatibility. http://diveintomark.org/archives/2004/02/04/incompatible-rss, Feb. 2004.

[22] A. Rowstron and P. Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *Proc. ACM SOSP'01*, Banff, Canada, Oct. 2001.

[23] R. Scoble. A theory on why RSS traffic is growing out of control. http://radio.weblogs.com/0001011/2004/09/08.html#a8200, Sept. 2004.

[24] W. Vogels. Once more: Polling does not scale. http://weblogs.cs.cornell.edu/AllThingsDistributed/archives/000511.html, Sept. 2004.

[25] N. Wallace. RSS is sticky traffic. http://www.synop.com/Weblogs/Nathan/PermaLink.aspx?guid=db37ec96-9271-4e4a-ad8d-6547f27fc1cb, July 2004.