This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the ICC 2007 proceedings.

Scheduling Feed Retrieval

Ward van Wanrooij
University of Twente
Enschede, The Netherlands
Email: w.a.h.c.vanwanrooij@student.utwente.nl

Aiko Pras
University of Twente
Enschede, The Netherlands
Email: a.pras@utwente.nl

***Abstract*— The popularity of RSS and similar feed formats is growing fast. This paper gives an overview of the standards and implementations in this field, and analyzes whether they allow scheduling the retrieval of feed updates. As will be shown, such support is very limited and current feed readers therefore poll providers at fixed rates. The measurements performed as part of our study show that in general a clear mismatch exists between such fixed polling rate of feed readers and the rate at which providers update their feeds; a significant performance gain is therefore possible by embedding scheduling information within feeds. This paper proposes a scheduling approach that both reduces lag in updates for active feeds and reduces wasted resources for less active feeds. Simulations show that our approach reduces the perceived lag by twenty percent, while having the same resource requirements as a fixed-rate algorithm.**

I. Introduction

In recent years a new type of site has emerged, structured like a file drawer: categories with items. Popular examples include weblogs ('blogs'), news sites and magazine sites. With this structure arrived the possibility to share recent site contents. Using standards like RDF, RSS and Atom, it is possible for a site owner to make the latest N entries (either abbreviated or in full) in a category ('feed') available at his website. Other site administrators can incorporate this content into their respective websites ('syndicate') and interested readers can stay up-to-date ('subscribe') with their favorite websites using dedicated feed readers. All of these standards share the basic principle of placing an XML formatted file conforming to some schema or description on a public webserver allowing for HTTP based pull synchronization. Unfortunately, these standards are mutually incompatible in naming and usage of elements. Some allow for embedding limited information pertaining to synchronization in the feed, others do not.

The inability to efficiently embed synchronization schedules in feeds causes two problems, both related to the fact that the client software, be it a web application for syndication or a desktop feed reader for subscription, is unaware of the update schedule of the feed provider. If the client fetches the file too often or at inappropriate times, resources like bandwidth and CPU cycles are wasted at both the provider and the subscriber. Conversely, if the client's update interval is too great, latency and lack of updates are introduced at the end-user.

With the rapidly increasing popularity of feeds (in August 2005 about 4% of end-users knowlingly use these [1]) the scale of these problems increases linearly to the number of users. Right now, feed providers are limited to politely asking web surfers to configure the feed reader appropriately (for example, updating once a day [2]) and end-users in general have to guess the right update interval for a news stream.

A. Related Work

Several attempts have been initiated in the Internet community to resolve these issues. The (proposed) solutions have been partial (RDF syndication module [3]), ambiguous (RSS ttl [4]) or unfinished (Atom refreshrate [5]). In addition two papers document a solution to the problems by a complete redesign of the feed protocols. Ka Cheung Sia and Junghoo Cho introduce a central middleman and self-learning update algorithms [6], while Dan Sandler et al. describe a peer-to-peer feed update distribution system [7]. We believe the first approach of requiring a central middleman that every feed publisher and reader adheres to is against the spirit of and does not scale for the Internet. Further we think that update settings should be entered manually by the feed publisher and not determined using algorithms, because:

- Manual specification allows for rapidly changing circumstances. If, for example, a major news event occurs, the feed can lower its update interval from 4 hours to 15 minutes.
- A feed provider can adapt the values dynamically based on server load.
- Ultimately the feed author knows best when he updates his feed, e.g. every Sunday evening or when Wall Street closes. The author not only knows the publishing schedule but also the impact of latency of feed updates (compare a feed providing stock trading tips with a feed announcing upcoming computer sciences conferences).

Moreover, we think that scheduling feed retrieval can and needs to be solved by extending the already in-use layer of protocols rather than creating a whole new distribution mechanism, like Sandler proposes.

B. Research Questions

In this paper we address the issues by answering the following research questions:

- What scheduling parameters are offered by the three major feed formats Atom, RDF and RSS?
- What is the level of support for these parameters in both feeds and feed reading software?
- How does the actual update rate relate to the real-world retrieval rate?

1-4244-0353-7/07/$25.00 ©2007 IEEE

- What are the shortcomings of current scheduling parameters and how would an improved scheduling mechanism be characterized?
- How does this improved scheduling mechanism compare to current scheduling approaches?

The structure of this paper is as follows. Section 2 describes the context of this research, namely the different feed standards and software. In section 3 we propose new scheduling facilities for feed retrieval. Section 4 shows the performance gain that could be obtained by using these facilities. Finally this paper finishes with the conclusions.

II. Context

This section describes the elements of the feed infrastructure relevant to this paper. It starts with an overview of existing feed publication standards and discusses the scheduling provisions provided by these standards. This section continues with analyzing other approaches to perform scheduling, and discusses how scheduling is supported by current feed aggregation software. This section concludes with the results of some measurements performed as part of our study to determine which standards and scheduling mechanisms are used by some of the most popular feeds.

A. Standards

Three different families of feed publication standards can be discerned:

- Resource Description Format Site Summary (hereafter referred to as RDF [8])
- Really Simple Syndication (referred to as RSS [4])
- IETF Atom [9]

The RDF format has originally (RDF 0.9) been developed by Netscape Communications [10] based on the language of the semantic web framework of the W3C [11]. After the initial release, this format has been reshaped [12] to be based on plain XML instead of the RDF XML wrapper (RSS 0.91). The RSS-DEV Working Group took over this work and finally created the RSS 1.0 standard. In this paper this version will be called RDF 1.0 however, to avoid confusion with RSS version 2.0, which is also based on 0.91 but created by UserLand Software [13], another leading company in this area. Since both competing formats, RDF 1.0 and RSS 2.0, are incompatible, political and technical reasons necessitated a new standard [14]. Political reasons included the struggle for the RSS trademark and the vacuum of ownership after Netscape abandoned its efforts. Technical reasons include the fact that RSS 2.0 has no required unique identification field (complicating update tracking and archiving) and ambiguous encoding of content. In cooperation with the Internet Engineering Task Force [15], a workgroup was formed that created two revisions of the new Atom standard: the informal 0.3 and the formal 1.0 version. The Atom 1.0 is published as RFC 4287 [9].

Each of these three groups has given some thought to scheduling update frequencies. The RDF 1.0 standard has optional modules, supported through XML namespaces, to incorporate additional functionality. One of these is the syndication module [3] that allows for setting a per feed update frequency. By setting the base date, period and frequency of updates, the client retrieval interval can be regulated by the feed publisher. The next lines show an example of specifying a 30 minute update schedule, starting in 2000, for RDF:

```
<sy:updatePeriod> hourly
 </sy:updatePeriod>
<sy:updateFrequency> 2
 </sy:updateFrequency>
<sy:updateBase> 2000-01-01T12:00+00:00
 </sy:updateBase>
```

In the RSS core specification, several provisions have been made to specify the lifetime of a copy and some limited scheduling. The `ttl` variable allows caching of the file for a specified number of minutes. However support for this parameter limited, because of a disagreement [16], [17] over its meaning. The dispute centers over whether the given ttl is the minimum (readers should not request the feed before the ttl expires) or a maximum (readers should request before the ttl expires) and if the ttl starts at the publication or retrieval time. Recently the arguments seem to have been settled by a clarification in a best practices profile. In addition to this interval, the parameters `skipHours` and `skipDays` can be used to specify at what times the feed should not be reloaded by the client. The next lines show an example of specifying a two hour caching value and no updates in the weekend during the nightly hours for RSS 2.0:

```
<ttl> 120 </ttl>
<skipHours>
 <hour> 2 </hour>
 <hour> 3 </hour>
</skipHours>
<skipDays>
 <day> Sunday </day>
 <day> Saturday </day>
</skipDays>
```

If desired the RDF 1.0 syndication module can also be used with RSS, Note, however, that the RDF 1.0 specification states that the syndication module ”supersedes the RSS 0.91 skipDays and skipHour elements” [3], which implies that skipHours and skipDays should not be used in conjunction with this module.

The comparatively new Atom standard does not define any no rate-limiting or scheduling parameters, despite the fact that a proposal [5] was made several years ago to include a `refreshRate` parameter. This parameter would specify the minimum interval in milliseconds between feed refreshes by a client. The next lines show an example of a 15 minute update schedule using this parameter:

```
<atom:refreshRate> 900000
 </atom:refreshRate>
```

B. Other Approaches

Two other approaches exist for retrieving feeds more intelligently.

The RSS specification allows for a push/pull hybrid model by introducing the `cloud` element. By connecting to the SOAP webservice specified by this parameter, a client registers at the service to receive update notifications. When the service detects a change (either by polling the feed or by integration in the publication process), it connects to a SOAP service at the client's workstation and transmits the URL of the changed feed. The client can now retrieve the updated feed. This approach seems to introduce more problems than solutions however. These problems relate to security (traversing firewalls and NATs, listening port on workstation) and scalability (the cloud protocol does not allow the client to stop the subscription; it instead relies on expiration after 24 hours). A similar idea [18] has been proposed for the Atom standard.

One particular header in the HTTP protocol deserves some further explanation since its existence is sometimes used to justify the lack of support for ttl in aggregators or standards: the *expires header* (see paragraph 14.21 [19]). Using this header a webserver can specify the maximum time-to-live of a document. Because HTTP is also used for transferring feed files, it could be interpreted as a caching hint by feed readers. One problem however is that the expiration time is part of the HTTP server configuration and can not be changed dynamically from a document (contrary to e.g. the last-modified header). This means that ordinary users can not influence this timer and making it multivalued (dependent on publication schedule) is difficult. This problem and the fact that from an architectural point of view such usage would break the layering principle, makes this header not suitable for our purposes. This conclusion has also been drawn in the Atom caching proposal, which initially considered [20] to reuse the HTTP expires header for scheduling purposes, but later reversed its stand [21].

The HTTP protocol also allows for the server and client to reduce resource bandwidth usage by employing gzip encoding and header fields like *if-modified-since*. Because our scope is broader than just reduction of content bandwidth consumption (for example server load during compression or database access and number of TCP connections) and these possibilities can be combined with our proposal without altering the results, we do not discuss these further.

C. Feed Aggregation Software

Feeds can be consumed by desktop applications known as feed aggregators, named so because they aggregate news from several sources and display it in a uniform interface. Many webbrowsers feature a built-in aggregator, but standalone software packages are also common. Some require a subscription to an internet service that acts as a central proxy; these are outside the scope of this paper, because their retrieval/synchronization mechanisms are not solely based on pulling feeds over HTTP.

TABLE I
PROPERTIES OF POPULAR FEED READERS

Application	Share[1]	Refresh (min.)		Support		
		def	min	update*	ttl	skip*
Firefox	20%	60	1	No	No	No
NetNewsWire	17%	240	30	No	No	No
FeedDemon	7%	60	1	No	No	No
SharpReader	4%	60	15	No	No	No
Thunderbird	4%	100	1	No	No	No
Safari	3%	30	30	No	No	No

Table I provides an overview of the six most popular [22] that do not connect to an intermediate service. This table also details the default and minimum refresh rate in minutes and the level of support for the three scheduling elements (RDF 1.0 updateFrequency, RSS 2.0 ttl and RSS 2.0 skipHours/skipDays). Remarkable are the spread of the default update rate (ranging from 30 minutes to 4 hours) and the lack of support for any scheduling mechanism in the popular packages[2].

Due to the very limited support for the scheduling features in the feed formats, feeds will be refreshed according to the wildly varying default synchronization settings for the diverse software packages, unless the user manually chooses a different interval.

D. Usage

To investigate which feed standards and scheduling mechanisms are used in practice, we have analyzed the two hundred most popular feeds, as identified by a large online feed aggregator [23]. After removing invalid, inactive and duplicate feeds, 193 unique feeds were left. If a feed is available in multiple formats, then in our analysis the RSS version is preferred to RDF, and RDF to Atom. These feeds have been retrieved a fifteen minutes interval for a period of eight weeks (between July 14th 2006 and September 10th 2006). These files have been parsed and inserted into a database. Table II shows the distribution of formats/versions and the use of scheduling parameters among the feeds. The data shows that few sites still use one of the RDF based formats, with the majority using the latest RSS 2.0 standards. The rise of the open standard Atom, backed by giants like IBM and Google, has just begun and is likely to continue in the following years as its familiarity increases.

After parsing the individual items in each file are compared to determine the update time of a feed. An update is defined as "the set of computed CRC32 for the items of a file is not equal to the set of the previous file". Figure 1 shows a graphical

[1]The marketshare is based on data [22] provided by a feed management provider and is adjusted to exclude online aggregators and podcast based applications.

[2]Some less popular or upcoming programs like Opera 9 and Microsoft Internet Explorer 7 do support the ttl element.

TABLE II
FEED FORMAT DISTRIBUTION

Version	Total number	Using		
		update*	ttl	skip*
RDF 0.9	1	n/a	n/a	n/a
RDF 1.0	28	6	n/a	n/a
RSS 0.91	8	0	0	0
RSS 0.92	4	0	1	0
RSS 2.0	133	7	38	1
Atom 0.3	2	n/a	n/a	n/a
Atom 1.0	17	n/a	n/a	n/a

distribution of the average update interval for all feeds (X axis) on a logarithmic scale (Y axis, number of updates in eight week timeframe). At the extreme left are feeds that update only a few times per month while at the right are sites that update almost continuously. Fifty seven percent of the feeds update at most twice a day and only seven percent of these feeds are updated every hour. The median update interval for these feeds is 16 hours. These statistics do not match the default refresh rate in most popular feed readers, where hourly is the most prevalent choice. The advantage of being able to specify the retrieval interval is evident from this mismatch.

III. SPECIFICATION

This section describes the shortcomings in current scheduling facilities and introduces our proposed improvements.

A. Problems

Because the RSS 2.0 solution to scheduling is feature wise the most complete of the three feed families, we examine the ttl/skipHours/skipDays approach for shortcomings that prevent it from offering true synchronization scheduling.

- Only one ttl value can be set; separate ttl values for different times of the day are not possible[3].

[3]Technically it is possible to modify the ttl value at each update of the feed to the appropriate value, but this creates problems for feed readers that are not online continuously and this usage is also not intended.

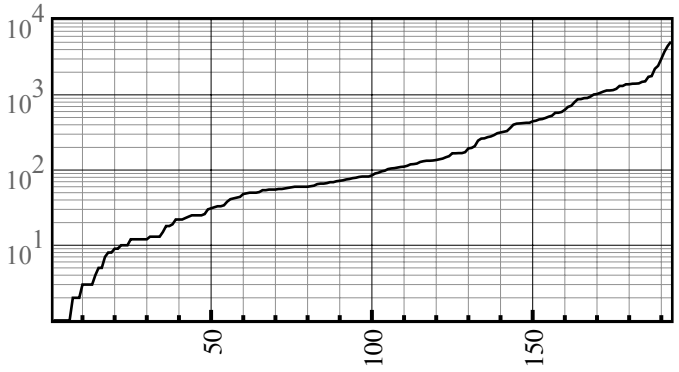

Fig. 1. Distribution of Update Frequency

- The ttl value not only suffers from the ambiguity of its specification but also from specifying an inappropriate key. We think the feed should not specify a minimum or maximum time-to-live but a recommended value: the final decision should be made by the feed reader/user (following an optional extension should not restrict a user more than ignoring the extension).
- The skipHours and skipDays mechanism is inefficient at specifying combinations of both when the interval of permissible hours passes the date line due to the shift from GMT, for example only on weekdays between nine A.M. and nine P.M. Pacific time.
- The skipHours and skipDays elements assume a reader to be online at all times, because the reader is not supposed to reload the feed during these times, even if the reader was offline when the skip restrictions were not in place.

Though the RSS standard has limited support for scheduling, the Atom version is even shorter on functionality. Considering its chances for being a dominant player, this is also a serious problem.

B. Proposed Scheduling Algorithm

Our proposal consists of set of scheduling rules to be embedded by the author in the XML file. The specification of these rules should be convenient, indicating hours and/or days when the rule should be active by defining start and end values. When a rule is active, the interval associated with it is the recommended retrieval rate for the client. Depending on the settings in the client, this interval may be adopted: a program can offer a user the choice of a minimum and maximum interval and whether the recommendation should be followed ('normal') or the value should be multiplied/divided by two ('more frequent', 'less frequent').

As an example of the proposed syntax, picture a feed that has a half hour retrieval rate during business hours and at four in the morning when scheduled updates are posted and a four hour rate at all other times. This can[4] be expressed in XML as:

```
<interval> 240 </interval>
<interval starthour="9" endhour="17"
 startday="1" endday="5"> 30 </interval>
<interval starthour="4" endhour="4" > 30
 </interval>
```

Using the simple algorithm in figure 3, a client can determine the next exact update time given the last update time/start time and the set of rules. Before actually using the value provided by this function, the feed reader should add a random value in the range -duration/2 ... duration/2 to avoid that all readers retrieve the feed at exactly the same time and so overload the server.

[4]The precise syntax definition, including timezone choice, namespace and schema, is not specified, because this depends on the respective feed standard and is not relevant for this paper.

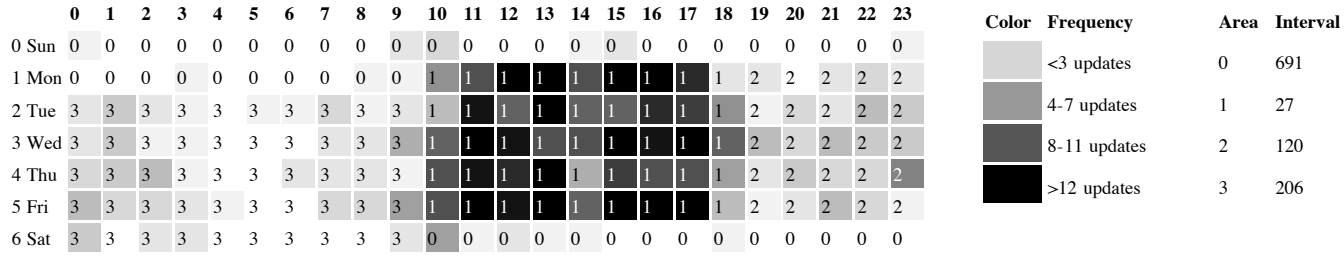

Fig. 2. Visual Display of Updates of The Register

IV. Performance

This section details the results of using the proposed algorithm.

A. Determining Rules

As stated before the schedules are determined manually for each of the 193 feeds. Before analyzing the update behavior, the available data is split in a six week training set and a two week test set. We have developed a program to visualize change activity for different times (horizontal axis) and days (vertical axis) of the week during the training period for each feed, see the left-hand side of figure 2. This programs sets the background color of each cell of the matrix according to the update frequency in that particular hour for that feed. Next we create the rules by marking similarly colored rectangular areas and assigning a number (zero is reserved for the unmarked area); each area represents a different interval and thereby a unique rule. Afterwards another program calculates the average interval between updates for each of these rules (see the right-hand side of the figure). Thirty percent of the feeds have just the default rule, because no publication patterns are discernible.

B. Results of Using Rules

We have developed a small simulator for feed retrieval that can request feeds using two algorithms: either using the scheduling mechanism that allows different intervals or by employing a linear mechanism. This linear algorithm using one fixed interval represents a traditional feed reader and is used as a benchmark. For the scheduling algorithm, the simulator is set to follow the calculated average interval per rule, if it is between 15 and 240 minutes; otherwise one of these boundaries is used. The minimum was chosen to be equal to the retrieval interval of our test data and the maximum was chosen to be equal to the maximum default retrieval value for popular feed readers. The linear algorithm runs after the other one and uses the timeframe divided by the number of downloads by the smarter algorithm to determine the interval. This way the performance of both can be justly analyzed, because the amounts of consumed resources are equal.

Fig. 3. Algorithm to Determine Next Update Time

```
DETERMINE-NEXT-UPDATE(time, schedules)
 1  oldinterval ← NIL
 2  previousupdate ← time
 3  repeat
 4        (interval, duration) ←
            GET-SCHEDULE(time, schedules)
 5        if oldinterval = NIL or oldinterval > interval
 6          then nextupdate ← max(time,
                    previousupdate + interval)
 7        time ← time + min(interval, duration)
 8        oldinterval ← interval
 9    until time ≥ nextupdate
10  return nextupdate
```

The simulator has two sets of data: the already studied training data and the test data to gauge the possible efficiency of the rules. For each mechanism, each feed and each dataset, the simulators starts 240 runs that each have a starting time that is one minute later than the previous run in order to eliminate any coincidences in timing. In total 185,280 simulations are performed. After the simulations have been performed, the simulator outputs grouped statistics that are used to evaluate the scheduling. The training run revealed that 58 of the feeds with multiple rules (42.33%) benefitted significantly, meaning a reduction in average delay by at least five percent, from this system. Table III details the statistics of both algorithms and both runs for those 58 feeds. These statistics include the average delay between publication and retrieval of an updated file (delay), the arithmetic mean of the standard deviation of the average delay (stddev delay) and the average number of downloads that returned an updated file (hits) or an unchanged file (misses).

Several observations can be made from this table. First the manual scheduling rules are also applicable to the unseen test data. Secondly, all averaged indicators improve when using the scheduling algorithm versus linear retrieval. The arguably most important statistic, namely average delay, improves by 22.42%. If we calculate the improvement in average delay on a per feed basis and then calculate the mean, so filtering out extreme absolute improvements, this value equals 20.07%. Obviously, this improvement in latency can also be traded off in full or partially for a decrease in resource usage. The 79 feeds that did not benefit from multiple rules did not have a clear publication pattern or, in most cases, had too few updates and the algorithm would default to its maximum value,

TABLE III
PERFORMANCE COMPARISON FOR SCHEDULING ALGORITHMS

Average	Training		Test	
	Linear	Schedule	Linear	Schedule
Delay (min.)	75.46	56.78	75.90	58.88
Stddev delay (min.)	41.76	37.09	40.15	37.47
Hits (%)	37.56	45.28	37.15	44.72
Misses (%)	62.44	54.72	62.85	55.28

realizing equal values for both mechanisms.

The maximum gain from using this rule based scheduling mechanism instead of a linear algorithm is capped by setting the maximum the interval parameter to 240 minutes. Especially feeds having few updates at some times of the days and many updates at others but also feeds not having many updates at all, would have decreased lag and decreased resource usage when raising this parameter. Still we maintain a value of four hours, so consumers know that the feed is updated when applicable but at least every four hours.

V. CONCLUSION

We have discussed the scheduling possibilities of the three major feed syndication formats RDF, RSS and Atom. Both RDF and RSS allow for limited specification of the update interval. Atom, while rising in popularity, currently offers no way of schedule specification at all. Although a quarter of our test feeds did specify such interval, support for it is extremely poor in feed reading software, so feed retrieval will almost always be done using the current interval settings of the software. We speculate that this entails that most feeds are retrieved using the default interval of the used software package, ranging anywhere from half an hour to four hours. This contrasts sharply with the observed median update interval of sixteen hours.

We have proposed a scheduling specification applicable to all three formats that allows for setting a recommended retrieval rate for different times and days of the week. Using a training and verification data set we have established that active feeds would benefit significantly from using this algorithm by decreasing the perceived lag between updates and retrieval while using the same bandwidth as a purely linear algorithm on average by over twenty percent. Less active feeds also have an advantage by being able to specify the schedule, they can conserve resources by using an appropriate schedule.

The main conclusion of this paper is that support for the specification of feed update schedules should increase, in both standards and software. For the Atom standard to have at least equivalent functionality to its main competitor RSS, it has to be extended by a scheduling mechanism, preferably based on a rule system such as our proposal. The developers of the next generation of the RSS format should also evaluate the merits of including such functionality. Software publishers that incorporate pull based synchronization of feeds should also support the scheduling parameters present in feed standards. Only then are wasted resources by clients updating too often and unnecessary lag and lack of updates due to clients updating not often enough a thing of the past.

VI. ACKNOWLEDGMENTS

The work reported in this paper was supported in part by the EC IST-EMANICS Network of Excellence (#26854).

REFERENCES

[1] J. Grossnickle, T. Board, B. Pickens, M. Bellmont, *RSS - Crossing into the Mainstream* October 2005, http://publisher.yahoo.com/rss/RSS_whitePaper1004.pdf

[2] M. Moncur, *Use our Quotations on Your Site* July 2006, http://www.quotationspage.com/useourquotes.php

[3] RSS-DEV Working Group, *RDF Site Summary 1.0 Modules: Syndication* December 2000, http://web.resource.org/rss/1.0/modules/syndication/

[4] RSS Advisory Board, *Really Simple Syndication: RSS 2.0 Specification* (version 2.08) August 2006, http://www.rssboard.org/rss-specification

[5] J. Snell, *Proposal: PaceFeedRefreshRate* June 2004, http://www.imc.org/atom-syntax/mail-archive/msg05846.html

[6] K. Cheung Sia, J. Cho, *Efficient Monitoring Algorithm for Fast News Alert* June 2005, http://oak.cs.ucla.edu/~cho/papers/sia-blog.pdf

[7] D. Sandler, A. Mislove, A. Post, P. Druschel, *FeedTree: Sharing Web micronews with peer-to-peer event notification* January 2005, http://iptps05.cs.cornell.edu/PDFs/CameraReady_221.pdf

[8] RSS-DEV Working Group, *RDF Site Summary (RSS) 1.0* May 2001, http://web.resource.org/rss/1.0/spec

[9] M. Nottingham, R. Sayre, *RFC 4287: The Atom Syndication Format* December 2005, http://www.ietf.org/rfc/rfc4287

[10] Netscape, *My Netscape Network - RSS 0.90 Specification* May 2001, http://my.netscape.com/publish/help/quickstart.html

[11] I. Herman, R. Swick, D. Brickley, *Resource Description Framework (RDF) / W3C Semantic Web Activity* July 2006, http://www.w3.org/RDF/

[12] M. Pilgri, *The Myth of RSS Compatibility* February 2004, http://diveintomark.org/archives/2004/02/04/incompatible-rss

[13] UserLand Software, *UserLand RSS Central* September 2006, http://rss.userland.com/

[14] R. Sayre, *Atom: The Standard in Syndication* IEEE Internet Computing, volume 9, number 2, July 2005, pages 71-78.

[15] IETF Secretariat, *Atom Publishing Format and Protocol (atompub) Charter* March 2006, http://www.ietf.org/html.charters/atompub-charter.html

[16] RSS Advisory Board, *Really Simple Syndication: Best Practices Profile* (Proposed) September 2006, http://www.rssboard.org/rss-profile#element-channel-ttl

[17] M. Woodman, *RSS: Time to Live (TTL)* August 2006, http://inkblots.markwoodman.com/2006/08/31/rss-time-to-live-ttl/

[18] P. Saint-Andre, J. Hilderband, B. Wyman, *Transporting Atom Notifications over the Extensible Messaging and Presence Protocol (XMPP)* June 2006, http://tools.ietf.org/wg/atompub/draft-saintandre-atompub-notify-05.txt

[19] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, *RFC 2616: Hypertext Transfer Protocol – HTTP 1.1* June 1999, http://www.w3.org/Protocols/rfc2616/rfc2616.html

[20] W. Underwoord, *PaceCaching - Atom Wiki* February 2005, http://www.intertwingly.net/wiki/pie/PaceCaching

[21] J. Snell, *Atom Metadata Expiration: Specifying Timestamps for Atom Feed and Entry metadata* Internet Draft, work in progress, December 2005, http://tools.ietf.org/wg/atompub/draft-snell-atompub-feed-expires-06.txt

[22] B. Livingston, *RSS Readers: Narrowing Down Your Choices* July 2005, http://itmanagement.earthweb.com/columns/executive_tech/article.php/3517646

[23] IAC Search & Media, *Bloglines — Most Popular Feeds* July 2006, http://www.bloglines.com/topblogs