Wireless Ad Hoc Podcasting

Vincent Lenders
Department of
Electrical Engineering
Princeton University
Princeton, NJ 08512
vlenders@princeton.edu

Gunnar Karlsson
Laboratory for Communication
Networks
KTH, Royal Inst. of Technology
10044 Stockholm, Sweden
gk@ee.kth.se

Martin May
Computer Engineering and
Network Laboratory
ETH Zurich
8092 Zurich, Switzerland
may@tik.ee.ethz.ch

***Abstract*— Podcasting has become popular for dissemination of streaming contents over the Internet. It is based on software clients that query servers for updates of subscribed content feeds. Podcasting may be used for any media content, but it is most commonly associated with audio streams. It provides a simple, no-frills broadcasting system for delay tolerant contents. A main limitation with this system is the inflexible separation of downloading to a docked media player and expending of the data when on the move. We herein present how podcast could be supported by our previously proposed delay-tolerant broadcasting system in order to reduce the expected times between updates and to provide a new ad hoc podcasting mode among mobile nodes. Our system substitutes the client-server paradigm inherent in present podcasting by a peer-to-peer paradigm where mobile nodes provide each other with contents. We present the protocol for this, and an evaluation of solicitation and caching strategies that greatly affect the application-level throughput. Our design is aiming at simplicity in order to enable implementation in mobile phones with media players and other devices that communicate over short ranges by means of Bluetooth or wireless LAN.**

I. Introduction

Wireless broadcasting is not openly available to anyone who would like to provide programs for a general audience. Luckily, the Internet enables alternative broadcasting modes, such as web logs and podcasts, which quickly have become popular among both content producers and end users. Podcasting is used for downloading of contents to a mobile media player when it is docked to a high-speed Internet connection. The playback is then often done when the player is off line because of the user's movements. This separation of downloading and playback limits the usefulness of the service since there could be hours passing between the downloading opportunities.

This paper describes how the podcasting concept can be expanded for public (i.e., open and unrestricted) peer-to-peer delivery of contents amongst mobile nodes. Contents are provided in a wireless broadcasting area either by access points or by mobile nodes which have been filled with contents off line (Figure 1). In the first case, each access point fetches contents from podcast servers across the Internet and forwards them to mobile nodes within its range; this is done in the regular podcast mode. In the second case, a mobile node that has contents to share might provide data to another mobile node when they pass within radio range of one another. This is the new content distribution mode that we add to the existing one. Our ad hoc podcasting mode brings the following advantages. First, it provides nodes with contents when they are not connected to the Internet; second, it provides a new ad hoc broadcasting domain when also the sources of the data are mobile nodes (could be pictures or voice recordings from a mobile phone for instance). We will also refer to this wireless ad hoc mode of broadcasting as podcast in the hope of broadening the current concept (motivated by the use of the feet of a pod of users for content distribution).

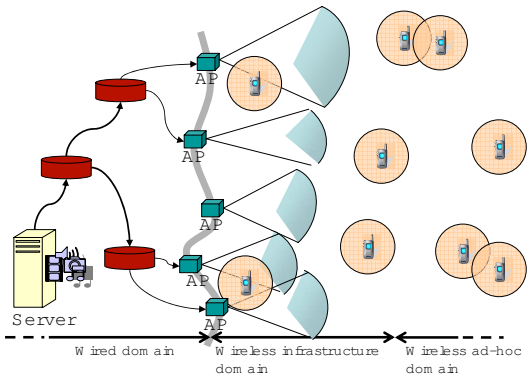


Fig. 1. Contents are provided by means of podcast via access points and they are re-distributed in the ad hoc domain.

Wireless ad hoc podcasting, as we present in this paper, is an application based on the *delay-tolerant broadcasting* concept we have proposed in an earlier paper [10]. The sharing of contents is based on a solicitation protocol by which a node asks a peer node for content. Hence, there is no flooding of contents in the broadcasting area. Contents are organized into *feed channels*, and nodes solicit *entries* for one or more feed channels (the emphasized words denote the terms we use in this paper).

The concept of feed channels allows for a higher hit rate of the queries than if they were for individual entries of contents. The entries of a particular feed channel will however reach a node in arbitrary order, and not all of them will be received; it is therefore important that all contents are provided in atomic units, which are short enough to be downloaded in a contact and which may be replayed without relation to other units. We believe that the podcasting application fulfills these requirements, since a feed channel could be composed of a mixture of entries of music, news items, weather updates and commentaries, such as the mix broadcast by many radio stations.

Our wireless ad hoc podcasting system is simple in its protocols, and lightweight in the computational requirements. Media players, including those in mobile phones, have ample storage so the system utilizes caching to increase the spread of data. In addition to the content that the user subscribes to, a node caches and carries contents unbeknownst to its user for the benefit of other nodes. We show in this paper that this mutual support is not merely altruistic: it increases the data throughput for everyone. We would like to emphasize that the sharing is an inseparable part of the implemented protocols and it is not governed by a social contract or an explicit incentive scheme.

An important issue is the strategy that should be used when selecting contents to cache in addition to contents of the user's interest. We would like the podcasting system to be insensitive to the relative user popularity for the channels, so that channels with low popularity also are distributed (it would be difficult to introduce new channels otherwise). Hence, contacts between nodes should not only be used to exchange entries for popular (i.e., highly requested) channels but also for rarely requested ones. This might however lead to distribution of channels that no end user receives, which wastes contacts and lowers the system throughput. This tradeoff between availability of contents and waste of resources is treated in this paper. The caching strategy may also be adapted to the node's storage and energy resources; this is however not included in this study.

This paper reports new research results regarding podcasting. The contribution is a proposal for a wireless ad hoc mode where mobile nodes exchange contents. We show how this mode may be implemented based on the Atom syndication protocol [13], and describe the protocols for soliciting and serving contents between nodes. Furthermore, we present caching strategies and evaluate their effect on the performance of the system. The paper is structured as follows: Traditional podcasting, as existing in the Internet, and the concept of delay-tolerant broadcasting are described in the next section. Section III introduces our new concept of wireless ad hoc podcasting and the necessary protocols. Section IV describes the caching strategies and the tradeoffs encountered when deciding on the most appropriate strategy. Section V contains the evaluations of the strategies and the design decisions. Section VI presents related work and Section VII concludes the paper.

II. Relevant Background Information

Our system generalizes the present podcasting concept by including a peer-to-peer mode for retrieval of contents. This section briefly describes how traditional podcasting works in the Internet, how the Atom syndication protocol is used in the regular client-server mode to retrieve podcasts, and the basic concept of delay-tolerant broadcasting.

A. Podcasting in the Internet

Podcasting has become popular in the Internet for distributing multimedia files such as audio songs or music videos. However, the concept could in principle be used to share any type of content. The procedure for podcasting is fairly simple. The only thing that a content provider has to do in order to create a new podcast channel, is to define a *feed* (an XML file), and make it accessible on a web server. This file contains a list of *entries*. An entry can contain the actual content, for instance a news clip, or it can provide a link (i.e., a URI) to the place where the content can be retrieved. The feed may contain all entries that have been published since the creation of the podcast channel, but it is typically limited to a short list of the most recent entries.

A user interested in a podcast channel subscribes to it by entering the feed URI into a software program called a *podcatcher*. This program retrieves and processes data from the feed URI on behalf of the user. The main difference between podcasting and traditional web content

```
<?xml version="1.0" encoding="utf-8"?>
 <feed xmlns="http://www.w3.org/2005/Atom">
  <title>ETH Life</title>
  <link href="http://example.org/"/>
  <updated>2003-12-13T18:30:02Z</updated>
  <author>
    <name>ETH press</name>
  </author>
  <id>urn:uuid:60a76c80-d399-b93C-0003939e0af6</id>
  <entry>
    <title>ETH starts podcasting service</title>
    <link href="http://ethz.ch/eth-life"/>
    <id>urn:uuid:1225c695-4ebb-aaaa-80da344efa6a</id>
    <updated>2003-12-13T18:30:02Z</updated>
    <summary>CN lectures avail. via podcast</summary>
  </entry>
 </feed>
```

Fig. 2. A typical Atom feed with one entry.

retrieval is that the podcatcher automatically downloads new entries as they are published. Thus, podcasting is particularly suited for contents that appears on a regular basis.

B. Brief description of Atom

Podcasting in the Internet uses either the RSS or the Atom syndication protocol. In our work, we consider the Atom syndication protocol which is on the standardization track of the Internet Engineering Task Force [13], and is planned to replace the various RSS protocols. Note however that the choice of syndication protocol is not important for the development we propose.

Atom is based on XML. Figure 2 shows an example (based on an example in [13]) of an Atom feed with a single entry. The first line defines the version and encoding to use. The second line starts the definition of the feed channel and refers to the XML name space for Atom. The third line is the clear text title of the feed channel. The link (URL) associated to this feed is given on the fourth line. This is the web resource where it can be found. `Updated` is the time and date when the feed channel was last changed (there is also an optional Atom element for specifying when it was first published), and `author` is a data structure with the name, URI and other data of the person creating the feed. The identity of the feed is an internationalized resource identifier (IRI) [7], and it may be de-referenced into either a universal resource locator or a universal resource name.

The remaining lines describe the entries of the feed (there is just one entry in this specific example). The data-structure elements of an entry are defined analogously to the feed itself.

C. Delay-tolerant broadcasting

The podcasting protocol in the ad hoc mode relies on our delay-tolerant broadcasting system introduced in [10]. The broadcasting concept means that there is no receiver group to be maintained, as with multicast, and there is no explicit routing in this communication mode; routing is in fact replaced by mobility of the nodes.

The system functions as follows: Every node scans periodically the network neighborhood for the presence of other nodes. When a delay-tolerant broadcast-enabled node is discovered, the two nodes associate. Indeed, nodes only associate pairwise, even if more than two are within reach of one another. The reason is that the contact durations may be short and it is better to get high throughput by only sharing the transmission capacity between two parties than to get high connectivity. Furthermore, following the pairwise-only rule, the system leverages all types of wireless data-link protocols, such as IEEE 802.11, Bluetooth and even IrDA, which only supports point-to-point communication. The link layer resolves contention for the data transmission between the nodes.

Instead of flooding the network with unwanted data, information is only exchanged when requested by one of the two nodes. The resulting behavior of the system can be described as a receiver-driven broadcasting scheme; there is no flooding of contents as in epidemic dissemination schemes [14].

III. The ad hoc podcasting mode

This section describes our actual enhancement of podcasting to operate in the wireless ad hoc mode. The application software in a mobile node is aware of the mode in which it should operate: if it is connected to the Internet directly, then it solicits contents according to the standard podcast procedure, as sketched in Section II. Otherwise, it operates in the ad hoc mode as described in this section.

A. Peer-to-peer distribution of content

To motivate our design, consider the two problems that are encountered when a node is unconnected from the Internet. First, it has no means to search for podcast feed channels. Second, should a channel feed or its ID be known a priori, then the node would nevertheless not be able to resolve the URIs or URLs and fetch entries for it.

To solve these two fundamental problems, we enhance the podcatcher software with new features. First, each node has a designated *discovery* channel that lists all

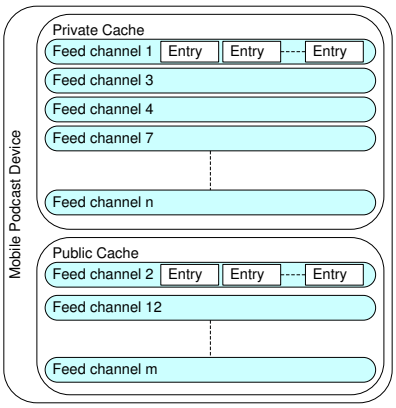


Fig. 3. Cache structure of a mobile podcasting device. The storage consists of a private cache for subscribed feed channels and a public cache for any other channels.

feed channels for which the node may provide entries; the list could be empty. A simple scenario is as follows: Two nodes make contact. A node searching for channels asks for the *discovery* channel and thus receives a listing of the feed channels held by the corresponding node. The node may then search the discovery channel it obtained for feeds that match the search criteria given by the user. This simple mechanism allows a mobile node to bootstrap into the wireless ad hoc podcasting system as soon as it encounters a corresponding node that has a listing of one or more channels that are of interest. It could of course take several encounters before a suitable channel has been identified.

In addition to the *discovery* channel, each node also stores the actual feed files for all the channels that it has entries for. The entries of these feed files must constantly be updated by the nodes to reflect the latest status (when encountering new nodes or when docking to the Internet). The feed channels are of two types: the *subscribed channels* the user receives and the *public channels* that are solely for redistribution. The cache structure of a node is depicted in Figure 3. The private and the public caches are of arbitrary length. Their relative sizes depend on the storage capacity of the mobile devices and the storage requirements for all entries of the subscribed channels (i.e., the public cache size might be reduced if a user needs the storage for the channels of its own interest).

B. Request for contents

A node that has feed channels for which it would like contents, simply solicits entries from the corresponding nodes it encounters. There are several types of requests allowed with various degree of detail. The more details, the lower the hit rate will be in general.

- **Any content:** The request is matched by any content that the corresponding node has. The discovery channel is provided as well as any selection of entries picked by the serving peer.
- **Feed channel:** Any entry for the specified feed channel. The feed channel may be specified by its ID, title, or meta data.
- **Entry:** An entry specified by ID, title or meta data from any feed channel. (N.b. only a request by ID provides a unique match.)
- **Feed channel+entry:**
  Request for contents from a specified feed channel. The entries are specified according to:
  Time: matches any entry with a publication date more recent than the specified time.
  Uniqueness: matches any entry not in a provided list of entry ID's.

A list of entry ID's may be specified as a Bloom filter computed from the ID's [4]. The corresponding node compares its ID's with the filter. The soliciting node has not seen an entry if its ID does not match with the filter; the entry may then be provided since it is unique with respect to the Bloom filter. False positives are possible, which means that entries will not be provided even though it would have been possible; false negatives of providing contents already seen are not possible. Hence, the use of a Bloom filter reduces the hit rate but avoids wasting communication time on supplying unwanted entries. The Bloom filter may greatly save on solicitation overhead when a long list of ID's is specified for the sake of uniqueness.

An important part of the ad hoc podcasting is the caching of contents in which the user of a node might not have any interest. This is done in two requests. First, the node solicits contents for its own subscribed channels. Second, it solicits contents for redistribution. The selection of feed channels for this depends on the caching strategy, whether it should cache contents for as many feed channels as possible for example. These strategies are further described in section IV.

C. Data structures

The information in the feeder file for a podcast channel is defined by a producer (identified in the atom:author element). It is desirable that the feed channels have distinct identifiers since the meta data might not be sufficient to distinguish two or more channels (the meta data could be absent). Hence, the feed channel IDs are

chosen as uniform resource names from the name space called universally unique identifiers (UUID) [11], and they are generated to ensure global uniqueness by the fully distributed algorithm specified in the RFC. The ID in the feed does not change when it is updated.

The meta data and identifier for an entry is generated by either the producer of the channel, the rights holder of the contents, or a third party that provides entries for syndication. This means that, for example, a given piece of music in a specified recording might be distributed under different identifiers which were generated by independent producers. This is unfortunate since it means that a request for one of the IDs would not result in the music being provided if it is listed under the other ID in the corresponding node. Use of rich, descriptive meta data will eliminate this problem. Different identifiers should be used when the piece of music is supplemented by different commentaries in the element summary, for instance, which makes the entries different from one another.

We order entries by their publication times, if specified. Note that this does not allow a node to ascertain whether there exists an entry published in between any two entries that it has seen. The purpose for the enumeration is not to determine missing entries, but to have a simple means of asking for new entries.

The media player needs a few internal data structures. One is a list of feed channels. For each feed channel, it marks whether it is a subscribed channel or a channel kept for the public cache. It stores a count of the number of requests for each channel, to be used in the solicitation decisions. For each entry, the node must keep track of the downloading status so that it can resume downloading of an incomplete entry at a later contact.

D. Protocol design

The proposed podcasting system is entirely receiver-driven, i.e., all actions are initiated by the requesting node. When two devices meet, they associate and start exchanging messages. The requesting node is querying for entries from its subscribed feed channels and the serving node replies to the incoming requests.

The state machine for a wireless podcasting device that handles incoming messages is illustrated in Figure 4. We distinguish two modes: an associated and a disassociated mode.

As soon as two devices associate, they start two processes where they accept solicitations (casting state), and where they generate request messages (solicitation state). Hence, each device incorporates two roles at the same time: a content provider (server) and a content solicitor (client). The device changes to being client only when there are no more requests to process, or server only when the device has no more content to solicit.

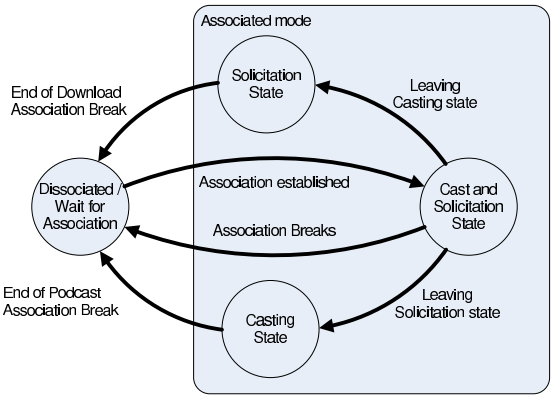


Fig. 4. Finite state machine of a podcasting device in ad hoc mode.

The solicitation process distinguishes two phases: During the first phase, the node satisfies the user's requests for content of subscribed channels. When all available entries of the subscribed channels are downloaded, the device may use the remaining connection time to update and download content that it caches for public use. An open question is which unsubscribed feeds a node should solicit entries from in order to best serve its potential future encounters. As the nodes do not know which nodes they will encounter in the future and what their interest will be, we propose to use history-based solicitation strategies that keep track of the requested feeds in the past. The different strategies are described in details in the next section.

IV. Solicitation Strategies

In this section, we describe different solicitation strategies for public feeds and explain briefly how to implement them. Note that our focus is not on particular incentive mechanisms since, by design, we always prioritize downloads that are of a users's interest. Hence, a node always downloads entries for feeds that the user is subscribed to first. Only when the remote peer does not have contents of interest to the user, will a node start soliciting contents to fill its public cache.

A resource that is directly affected by the solicitation of public content is the battery of the mobile devices. However, in our application scenario, we assume that the users have the opportunity to recharge their batteries on a frequent basis (e.g., during the night) and that the battery lifetime is high enough to support additional transfers for the public cache.

Another problem often encountered in file sharing systems are free-riders (nodes which only download contents without participating in the distribution of the contents). This problem is solved by not making the sharing controllable by the users.

The considered solicitation strategies are:

- **Most Solicited:** This strategy aims at increasing the probability that a next encounter will be interested in a feed from a node's public cache. This is achieved by fetching entries from feeds that are most popular. For this, a node keeps track of the feeds that were requested by past nodes and maintains an history-based ranking. Only the requests for the feeds that the requester is actually subscribed to, are counted. The requests to fill the public cache of the requesting nodes are neglected to avoid feedback loops. With this strategy, a node always first solicits entries for the most popular feed. If the associated node has no entries to share for this feed, the second most popular feed is solicited, then the third most popular feed, and so on. Note that the feed popularity is determined locally which means that the popularity distribution at different nodes might be slightly different depending on the sequence of contacts each individual node has made.
- **Least Solicited:** This strategy aims at the opposite goal than *most solicited*: it favors less popular feeds. Since entries from popular feeds are available anyway in the private cache of many devices, it might be more useful to fill the public cache of devices with rare entries in order to improve the performance of those feeds. The implementation of this strategy is similar to *most solicited*. Each node keeps track of the requested feeds and maintains an history ranking, but this time, a node starts by soliciting entries from the less popular channel, then the second less popular channel, and so on.
- **Uniform:** With this strategy all feed channels are treated equally, independent of their popularity. A node records all feed channels it has seen in the past and solicits entries from those channels randomly. The *uniform* strategy has the advantage of an easy implementation, since it requires no information about feed popularity.
- **Inverse Proportional:** To mitigate the effect that with *least solicited*, the public cache of the nodes could be filled with unpopular feeds that might never be requested in the future, we propose this additional strategy. *Inverse proportional* consists of maintaining a history list and soliciting a feed with a probability which is inverse proportional to its popularity. As a result, entries from popular channels might be solicited from time to time even when new entries from an unpopular feed are available.
- **No Caching:** This is not a solicitation strategy in itself, but rather a benchmark for the other strategies. It can be viewed as the behavior of nodes which only download and share entries of their interest. With *no caching*, a device has no public cache and stores or distributes only content for the feeds it is subscribed to.

These five different strategies are different in implementation complexity and information requirements. While the *no caching* and *uniform* strategy are straightforward to implement and do not require knowledge about former requests and popularity, the other three approaches require history and popularity information. However, such popularity information can easily be obtained and maintained locally by exploiting the information from the requests of other peers. The performance of these different strategies is evaluated in the next section.

V. Evaluation

This section evaluates the performance of the different solicitations strategies with mobile users.

A. Network and Simulation Model

We evaluate the solicitation strategies with our own simulator which is based on a simple communication model: two nodes can communicate with a nominal bit-rate if their geometric distance is smaller than a threshold value (modeling the wireless range of the radio devices). The simulation model does not incorporate any link layer issues such as collisions or interference. This simplification is supported by the fact that we only consider sparsely connected networks where collisions and interference between different associations are rare. For the simulations, we further assume that the setup time for these associations is negligible.

The results we present in this paper were done using the random waypoint mobility model [9]. In the random waypoint model, a node chooses a random destination point, and moves towards it on a straight line with constant speed. When it arrives there, it repeats the process. We use the random waypoint mobility model because nodes with this model happen to move relatively large distances before changing direction which is necessary to spread contents when the network is partitioned. We also did experiments with the random walk mobility

model and observed the same performance trends as we report for the random waypoint mobility model, but with generally longer data propagation delays. Our implementation of the random waypoint follows the guidelines of [12] to make sure that the simulations are conducted in steady state [3]. Furthermore, the nodes in our simulations do not pause after reaching a waypoint and are thus constantly moving.

B. Scenario Settings

We assume an environment where human beings carry Bluetooth-enabled devices (e.g., mobile phones). For that purpose we set the moving speed of the nodes to 1 m/s (corresponding to the average speed of humans), the radio range of each device to 10 meters (the wireless range of class 3 Bluetooth devices), and the nominal rate of the radio devices to 1 Mb/s. Entries have a fixed size of 250 KB. This corresponds to a download time per entry of approximately 4 seconds when two associated peers are downloading simultaneously. The entry size was chosen to allow downloads to finish within a single association with a high probability (most association times in our scenario settings are larger than 10 seconds). Our results are not specific to the exact entry size but rather to the assumption that only few entries need several consecutive associations to be downloaded.

We evaluate the performance of the different solicitation strategies with a scenario where mobile users retrieve entries from feed channels that are updated in the Internet. Note that we also envision wireless ad hoc podcasting as a useful service for spreading contents generated by the mobile nodes themselves. However, we only focus on the first scenario as part of this evaluation. The podcast entries are made available to the mobile nodes in the wireless ad hoc domain through an Internet access point (i.e., a node that has one link to the Internet and one wireless link to associate with mobile nodes.). The mobile nodes that are within transmission range of the access point download the latest entries from the access point in the traditional podcasting mode. All other nodes share entries in the wireless ad hoc mode as proposed in this paper. For practical reasons, we limit the mobility range of the mobile nodes to a square. The results we present in this section are based on a square of side length of 300 meters in which the access point is located at the center. Larger squares resulted in longer propagation delays, but the relative order of the presented performance metrics remained the same.

New entries are published and made available in the Internet on a regular basis. For simplicity, the publishing rate of new entries is periodic and the same for all of the feed channels. The mobile nodes in the wireless network are subscribed to different podcast channels. The popularity of individual different channels is different. We model the channel popularity according to Zipf's law [18]. The law implies that the popularity P_n of a channel follows:

$$P_n \sim \frac{1}{n^\alpha}, \tag{1}$$

where n is a channel ranked n^{th}, and α is a constant close to one. It has been shown that many collections such as for example the popularity of web pages, or the frequency of keywords entered in search engines, follow this law, which makes it a natural choice to model the popularity of podcast channels in our analysis.

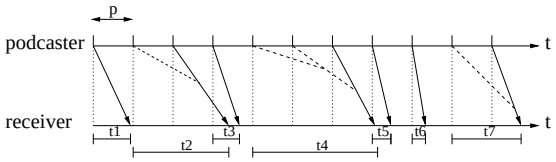


Fig. 5. Freshness metric

C. Performance Metrics

Our generic application of podcasting in the wireless network is typically spreading news-related content. For this type of content, newer entries are of highest interest and they typically obsolete older entries. For example, the weather forecast of yesterday is irrelevant, if we have the forecast of today. For this purpose, we assess the performance of the system by considering the *freshness* of content at the receivers. This metric should reflect the elapsed time from the moment an entry is published until the mobile nodes receive it. However, due to the inherently high delivery delays, some entries might not be delivered because newer entries of the same channel are available before older ones are delivered. Therefore, we define the freshness metric as the period between publication time of a new entry and reception time of this entry or any following entries if delivered before. This definition is visualized in Figure 5. The figure shows a podcaster node publishing new entries at a fixed rate p and a receiving node. The arrows represent the delivery delays of the entries. The first entry is delivered to the receiver and the freshness t_1 is thus the time interval between the publication time and the reception time. The second entry is overtaken by the the third entry. Therefore, the freshness t_2 is the time interval between the publication of the second entry and the reception of the third entry.

	$\alpha = 0.5$	$\alpha = 1$	$\alpha = 1.5$
no caching	5^{th}	5^{th}	5^{th}
most solicited	4^{th}	4^{th}	4^{th}
least solicited	3^{rd}	3^{rd}	3^{rd}
uniform	2^{nd}	2^{nd}	1^{st}
inverse proportional	1^{st}	1^{st}	2^{nd}

TABLE I

FAIRNESS (RANKING).

Our performance criteria are the overall and the per channel average freshness. One the one hand, we want to maximize the overall performance, but on the other hand, we also want to avoid starving individual channels. Ideally, a specific solicitation strategy achieves both objectives simultaneously: achieving a high overall freshness while maintaining a fair distribution among the different channels.

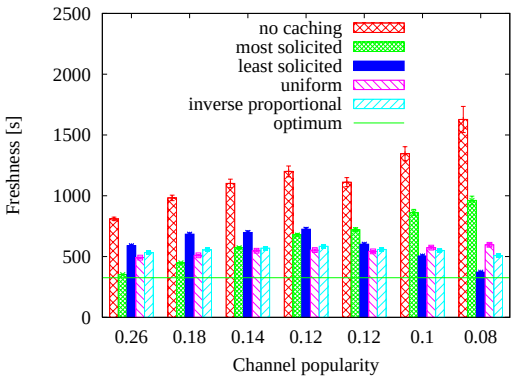
We assess the overall performance by averaging the user performance (the freshness) over all channels. In contrast, the channel performance is assessed with a fairness metric. In particular, we consider the max-min fairness. Max-min fairness is a simple, well-recognized approach to define fairness in data networks [2]; it aims at allocating as many resources as possible to poor users, while not unnecessarily wasting resources. This fairness metric applied to our podcasting problem implies that a strategy is fair if is not possible to increase the performance of an individual channel without decreasing the performance of another channel having a lower performance. In other words, the channel performance is said to be max-min fair when: firstly, the performance of the worst channel is maximized; secondly, the performance of the second worst channel is maximized, etc.

D. Results

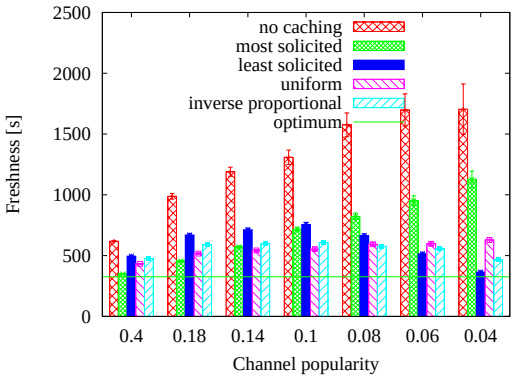
In a first step, we analyze the overall and per channel performance for different channel popularity distributions. The different popularity values are obtained by varying α in the Zipf law. Then, we assess the impact of the entry publishing interval on the overall and per channel performance.

1) Varying α: In the following, we present the results for a representative set of experiments consisting of 50 mobile nodes and 7 different channels. Each node is interested in one specific channel. The channel popularity was chosen according to values of $\alpha = 0.5$, $\alpha = 1.0$, and $\alpha = 1.5$ in the Zipf distribution. The publishing rate is identical for all channels.

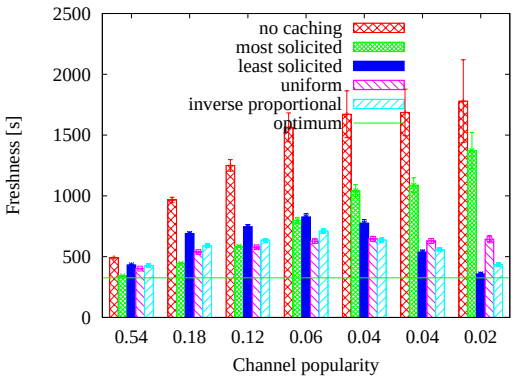
The per channel freshness is plotted in Figure 6 with a 90 percent confidence interval. On the horizontal axis, we plot the channel popularity. For example, a value of 0.4 means that 40 percent of the nodes are interested in this channel. On the y-axis, we plot the freshness. The lower the value, the better is the performance. To compare with the best performance we might obtain, we plot a benchmark value (marked with optimum) which is the value one would get if all the nodes would be interested in the same channel. Note that when all



(a) $\alpha = 0.5$



(b) $\alpha = 1$



(c) $\alpha = 1.5$

Fig. 6. Channel performance for various values of α (publishing interval: 120 seconds).

	$\alpha = 0.5$	$\alpha = 1$	$\alpha = 1.5$
no caching	1090	1027	861
most solicited	588	545	498
least solicited	614	594	559
uniform	524	507	486
inverse proportional	551	539	514

TABLE II

OVERALL PERFORMANCE OF FRESHNESS IN SECONDS (THE SMALLER THE VALUE THE BETTER).

nodes are interested in the same podcast channel, the solicitation strategy has no impact on the performance and the freshness is only influenced by the node mobility and capacity of the wireless links.

Independently of the value of α, we see the clear benefit of soliciting and caching entries from unsubscribed feeds in the public cache. Specially, the performance of the less popular channels suffer when the nodes are not caching for others. With the *most solicited* strategy, all channels have a considerably better performance than *no caching*. However, the fairness among the channels is poor as less popular channels are discriminated. The *least solicited* strategy does a better job for the unpopular channels, but it tends to privilege the unpopular channels too much since the least popular channels now outperform the most popular channels. Also, channels with medium popularity are somewhat discriminated with this strategy. The *uniform* and the *inverse proportional* strategy both achieve the best fairness. The best of these two strategies in terms of max-min fairness depends on the popularity distribution as we can see in Table I. For $\alpha = 0.5$ and $\alpha = 1$, the *inverse proportional* strategy is fairer. For $\alpha = 1.5$, which corresponds in a less even channel popularity distribution, the *uniform* strategy is fairer. There is a simple explanation for this behavior. Large values of α produce uneven popularity distributions compared to smaller values. Since the *inverse proportional* strategy chooses channels to solicit with a probability that is inversely proportional to the popularity, a large value of α increases the probability that unpopular channels are solicited. Therefore, this strategy starts to behave in a similar way as *least solicited* which is worse than *uniform* for all values of α.

We consider now the overall system performance regardless of individual channels. The resulting freshness values for the same values of α as used before are given in Table II. The *uniform* strategy achieves the best overall performance for any α. Depending on the value of α, the *inverse proportional* strategy (for $\alpha = 0.5$ and $\alpha = 1$), or the *most solicited* strategy (for $\alpha = 1.5$) has the second best performance. The *least solicited* strategy is always worse than these three strategies. The reason is mainly that this strategy solicits and caches too extensively for feeds that are rarely requested.

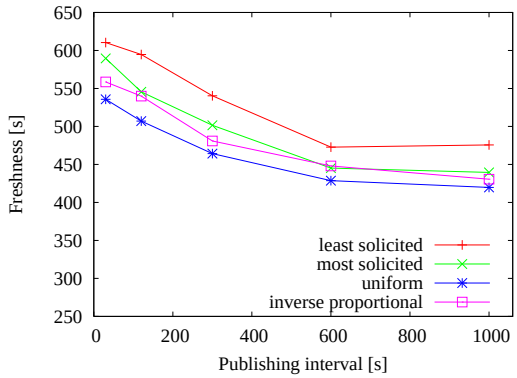


Fig. 7. Overall performance ($\alpha = 1$)

2) Varying Publishing Rate: In the next simulations, we analyze the performance of the different strategies depending on the rate at which new entries are published for the podcast channels. For this, we use the same settings as before and set $\alpha = 1$.

We first look at the overall system performance to see if the publishing rate impacts the trends we observed before (with a publishing interval of 120 seconds). The freshness averaged over all nodes is plotted in Figure 7 (note that *no caching* is not included in the plot, as the resulting freshness is much larger than the other four strategies). We can see that the trends identified previously remain valid. The *uniform* strategy has the best, and the *least solicited* has the worst (without considering *no caching*) overall performance independent of the publishing rate. The performance of the *inverse proportional* is only slightly better than the *most solicited* strategy.

Notice further that the average freshness for all strategies decreases as the publishing interval increases. The reason is that when the publishing interval increases, the probability that a node finds a new entry, it does not have, decreases. This is caused by the fact that with large publishing intervals, nodes tend to cache more entries for others since most of them have the latest entries of their subscribed channels. In contrast, for low publishing intervals, most nodes tend to download entries for themselves as new entries appear more rapidly.

The per channel performance is shown in Figure 8 for $\alpha = 1$, and a publishing interval of 30 seconds as well as 1000 seconds. The result for a publishing interval of

Publishing interval [s]	30	120	1000
no caching	5^{th}	5^{th}	5^{th}
most solicited	4^{th}	4^{th}	4^{th}
least solicited	3^{rd}	3^{rd}	3^{rd}
uniform	2^{nd}	2^{nd}	2^{st}
inverse proportional	1^{st}	1^{st}	1^{nd}

TABLE III
FAIRNESS (RANKING).

120 seconds is shown in Figure 6(c). We observe that varying the publishing rate does not change the fairness relations of the different strategies. As we can see in Table III. The best fairness is achieved by the *inverse proportional* strategy, followed by the *uniform*, the *least solicited*, the *most solicited*, and no caching at all.

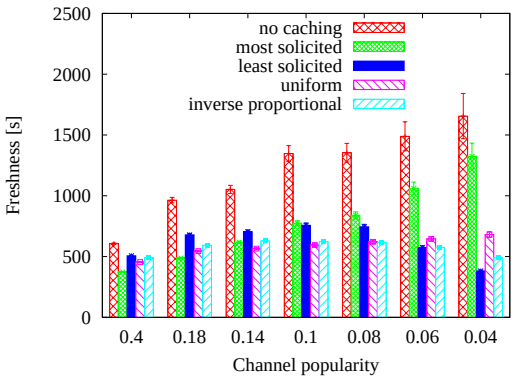
E. Discussion and first conclusions

We have analyzed the impact of the public caching strategies on the overall and the per channel performance of wireless ad hoc podcasting. We have first shown that soliciting and caching of unsubscribed feed channels considerably improves both the overall and the per-channel performance. This result is not trivial because in our ad hoc podcasting scheme, downloads for subscribed channels are prioritized over downloads that serve to fill the public cache of the nodes.

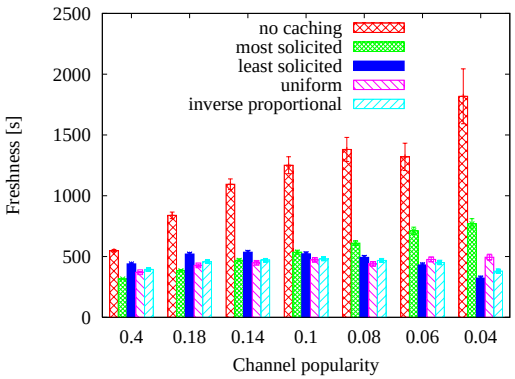
The best overall performance is achieved with the *uniform* strategy. We did not expect this result, as intuitively the best overall performance should be achieved by spreading the most popular content and letting the unpopular feed channels starve. We explain this trend with our two-phase solicitation scheme which consists of first soliciting and downloading entries of subscribed feeds, and in the second phase soliciting content for other peers. With this scheme, even when the nodes are caching content from unpopular channels, the content from the popular channels are spread via a large amount of users which are actually interested in that content.

Regarding the fairness among the different channels, the *inverse proportional* strategy achieves the best result for relatively even feed popularity distributions. For uneven popularity distributions, the *uniform* solicitation strategy performs best. *Least solicited* tends to starve channels with medium popularity. The *most solicited* strategy is discriminating unpopular channels.

Motivated by its performance, we propose to use the *uniform* solicitation strategy in our system design. Furthermore, the *uniform* strategy is by far the simplest strategy to implement. Soliciting nodes only have to remember which channels they have seen in the past and do not have to establish a history of the solicitation popularity.



(a) Publishing interval: 30 seconds



(b) Publishing interval: 1000 seconds

Fig. 8. Channel performance for various publishing rates ($\alpha = 1$).

VI. RELATED WORK

We show in [10] that delay-tolerant broadcasting between mobile nodes results in sufficiently high application level throughput even for streaming. This is the case in urban pedestrian areas with reasonably high densities of users, as well as in public transportation and in places where people gather occasionally (e.g., sport fields, shopping malls, recreational areas).

Podcasting [15] is a fairly new content distribution technology and is gaining attention also from the research community. To the best of our knowledge, this is the first publication that proposes an extension of the podcasting technology to an wireless ad hoc networking environment. Our podcasting system has similarities with Bittorrent, where users who retrieve contents act simultaneously as clients and servers [8]. Note that the ad hoc podcast provides the entries in a random order and nodes that have exchanged data might never meet again.

The closest research field is the one introduced in delay tolerant networking. The Delay Tolerant Network Research Group (DTNRG) [1] has proposed an architecture [5] to support messaging that may be used by delay tolerant applications. The architecture consists mainly of the addition of an overlay, called the bundle layer, above a network transport layer. Messages are transferred in bundles in an atomic fashion between nodes using a transport protocol that ensures node-to-node reliability. These messages can be of any size. Nodes are assumed to have buffers in which they store the bundles.

Multicast routing in DTN has been addressed in [17]. While the goal of our system is also to deliver data to a group of people, our approach is decoupled from any multicast semantics (such as group memberships, et cetera). The infostation concept is akin to our proposal and the paper studies means for avoiding exploitation of other nodes [16]. We differ in that we make the nodal exchanges governed by a protocol instead of a social contract between users.

Contact patterns of human mobility have been analyzed in the Haggle project [6]. This project aims at developing an application-independent networking architecture for delay-tolerant networks. In contrast, we implement in our approach the podcasting service directly on top of the link layer to exploit application-specific policies (like channel interests) in the way information spreads across the mobile users.

VII. Conclusion

We have in this paper described how the common podcast may be extended with an wireless ad hoc mode in order to provide contents when nodes are mobile. This mode will allow the nodes to catch contents when they are within radio transmission range from one another. The mode allows the time between updates of contents to be reduced compared to the current docking mode for content refill. We study various strategies for soliciting of contents and find that the uniform strategy performs well and has little bias for and against channels based on their popularity; it is also easy to implement. The protocols for the ad hoc mode have been specified in the paper. They are simple and suitable for implementation in nodes with limited computational power.

The research will now continue in making the podcasting more secure. The problem is that a node asking for contents for a specific feed channel might be given any entry. As of now, a node has to trust that the provided content actually corresponds to that identified by the entry data structure, and it must trust that the entry belongs to the correct feed channel. We plan to identify and evaluate different schemes to address this problem.

We believe that the wireless ad hoc podcasting is a worthy extension to the present podcast that will allow broadcasting of multimedia contents in densely populated areas.

References

[1] Delay Tolerant Network Research Group (DTNRG). http://www.dtnrg.org.

[2] D. Bertsekas and R. Gallager. *Data Networks*. Prentice-Hall, Englewood Cliffs, New Jersey, USA, 1992.

[3] C. Bettstetter, G. Resta, and P. Santi. The Node Distribution of the Random Waypoint Mobility Model for Wireless Ad Hoc Networks. *IEEE Transactions on Mobile Computing*, 2(3):257–269, 2003.

[4] B. Bloom. Space/time tradeoffs in in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[5] S. Burleigh, A. Hooke, L. Torgerson, K. Fall, V. Cerf, B. Durst, and K. Scott. Delay-tolerant Networking: An Approach to Interplanetary Internet. *IEEE Communications Magazine*, 41(6):128–136, 2003.

[6] A. Chaintreau, P. Hui, J. Crowcroft, C. Diot, R. Gass, and J. Scott. Impact of Human Mobility on the Design of Opportunistic Forwarding Algorithms. In *Proceedings of IEEE INFOCOM*, Barcelona, Spain, April 2006.

[7] M. Duerst and M. Suignard. Internationalized Resource Identifiers (IRIs). IETF RFC 3987, January 2005.

[8] M. Izal, G. Urvoy-Keller, E. Biersack, P. Felber, A. A. Hamra, and L. Garcés-Erice. Dissecting BitTorrent: Five Months in a Torrent's Lifetime. In *Proceedings of Passive and Acrive Measurements Conference*, April 2004.

[9] D. Johnson and D. Maltz. Dynamic Source Routing in Ad Hoc Wireless Networks. In T. Imelinsky and H. Korth, editors, *Mobile Computing*, volume 353, pages 153–181. Kluwer Academic Publishers, 1996.

[10] G. Karlsson, V. Lenders, and M. May. Delay-Tolerant Broadcasting. In *Proceedings of the ACM SIGCOMM CHANTS Workshop*, Pisa, Italy, September 2006.

[11] P. Leach, M. Mealling, and R. Salz. A Universally Unique IDentifier (UUID) URN Namespace. IETF RFC 4122, July 2005.

[12] W. Navid and T. Camp. Stationary Distributions for the Random Waypoint Model. *IEEE Transactions on Mobile Computing*, 3(1):99–108, 2004.

[13] M. Nottingham and R. S. (Eds.). The Atom Syndication Protocol. IETF RFC 4287, December 2005.

[14] A. Vahdat and D. Becker. Epidemic Routing for Partially Connected Ad Hoc Networks. Technical Report CS-200006, Duke University, NC, USA, April 2000.

[15] Wikipedia. Podcasting. http://en.wikipedia.org/wiki/Podcast.

[16] W. H. Yuen, R. D. Yates, and S. C. Sung. Noncooperative content distribution in mobile infostation networks. In *Proceedings of the IEEE WCNC*, 2003.

[17] W. Zhao, M. Ammar, and E. Zegura. Multicasting in delay tolerant networks: Semantic models and routing algorithms. In *Proceedings of the Sigcomm Workshop on Delay Tolerant Networking*, August 2005.

[18] G. K. Zipf. *Human Behavior and the Principle of Least Effort*. Addison-Wesley, Cambridge, Massachusetts, USA, 1949.