Wireless Opportunistic Podcasting: Implementation and Design Tradeoffs

Martin May, Clemens Wacha
Computer Engineering and Networks Laboratory
ETH Zurich
8092 Zurich, Switzerland
{may,wacha}@tik.ee.ethz.ch

Vincent Lenders
Department of Electrical Engineering
Princeton University
Princeton, NJ 08544, USA
vlenders@princeton.edu

Gunnar Karlsson
Laboratory for Communication Networks
KTH, Royal Inst. of Tech.
100 44 Stockholm, Sweden
gk@ee.kth.se

ABSTRACT

Podcasting has become a very popular and successful Internet service in a short time. This success illustrates the interest for participatory broadcasting, but podcasting is alas only available with fixed infrastructure support to retrieve publicized episodes. We aim at releasing this limitation and present herein a podcasting system architecture based on opportunistic wireless networking that allows us to extend podcasting to ad hoc domains. The design is explained and presented together with a prototype implementation running on hand-held devices. We analyze different fundamental design tradeoffs based on measurements.

Categories and Subject Descriptors:
C.2.1: Network Architecture and Design

General Terms: Design, Experimentation.

Keywords: ad hoc networking, delay tolerant broadcast, opportunistic communication, wireless podcasting.

1. INTRODUCTION

The rapid rise of podcasting rests on two strong trends in communications: first, the interest in participatory media that allow anybody with a computer and network access to produce and distribute contents, and, second, the rapid growth in use of the Internet for distributing bulk data. These two trends are admittedly not new and emanate from the introduction of the world-wide web and the subsequent popularity for making personal and commercial web pages publicly available. Content distribution has been driven also by various file-sharing protocols. Podcasting is different in its form since the contents are produced by a person or a collective of persons and they are mostly streaming media.

A third trend in communications is the boom of wireless. For cellular communication, the number of mobile phones in the world exceeds a billion and the matching network infrastructure is now in the third generation, with standardization working on its long-term evolution. Cellular communication is also provided for local areas by access points operating in the license-free spectral bands. Many electronic devices such as portable media players, digital cameras, mobile gaming terminals, hand-held computing devices and positioning systems are also capable of communicating wirelessly by means of Bluetooth and WLAN. An interesting observation is that the first two trends regarding participation and bulk data are not greatly reflected in the wireless domain: The main service category in addition to telephony appears to be streaming of professionally produced contents targeted for cellular phones. (This is of course not true when WLAN is used as a cordless extension of the Internet.)

The design presented in this paper is for a wireless podcasting system that offers a means for bringing the participatory media and bulk content distribution into the wireless domain. The basis for the service is an opportunistic distribution of contents among mobile users. Users in our system exchange podcasts when they cross each other in urban areas, when they meet in public transportation and at sport, art or music events. We would like the system to be open to anybody who wants to provide and consume contents. Hence, it is based on unlicensed short-range communication like IEEE 802.11, Bluetooth, infrared and future ultra-wide band communication. By relying on short-range communication, the network will be highly disrupted most of time. The communication is further challenged by relatively short transfer opportunities which might be in the range of a few seconds when for example two nodes cross each other.

Our opportunistic podcast service leverages the mobility of the nodes. Nodes mix and associate randomly with one another as they move, and two nodes that have been in contact may never get in touch again. To cope with these challenging conditions, the application data is structured in our design so that also short contacts may be used to exchange meaningful units of application data, and downloads may be resumed in case of disruptions. Further, our wireless opportunistic podcasting distribution system is entirely receiver-driven. That means that nodes solicit contents of interest and there is hence no information that is pushed as in traditional data routing schemes. The contents are organized into podcast channels (often referred to as *feeds* in the podcasting wording) and may originate both from nodes in the ad hoc domain or from servers in the infrastructure domain. The data is exchanged in short units, which we refer to as *chunks*. Fragmenting contents into chunks allows the application to resume incomplete downloads, either from the same node or from an arbitrary node encountered in the future (which has the same contents). The application must be able to replay chunks delivered in a random order and it must tolerate that not all units of chunks within a channel will be delivered. We believe that the service still has uses: a *poetry channel*

CHANTS'07, September 14, 2007, Montréal, Québec, Canada.

could distribute people's own reading of poems, a *snapshot channel* could provide tourists' views of a city by means of pictures or small videos taken with their mobile phone and communication-capable cameras.

In this paper, we present a full implementation of the podcasting for hand-held devices and provide results from associated performance measures.

2. RELATED WORK

This paper complements our work on ad hoc podcasting we presented in [8]. In the previous work, we evaluated the impact of different content solicitation strategies to improve the podcast distribution speed in mobile scenarios. This paper focuses on the system architecture design and the implementation of the system on mobile devices.

There has been substantial work on peer-to-peer content distribution for the Internet. BitTorrent is a successful instance of such systems where users who retrieve contents act simultaneously as clients and servers. It has post-facto gained interest in the research community; see for instance [5]. Our podcasting system has similarities to BitTorrent, but the mobility assisted delivery means that data are provided in a random order from a random mix of peers whereas peer-to-peer content distribution systems like BitTorrent selects peers based on specific rules.

The closest research field is the delay tolerant networking. The Delay Tolerant Network Research Group (DTNRG) [1] has proposed an architecture [3] to support communication 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 of any size are transferred in bundles in an atomic fashion that ensures node-to-node reliability. Multicast for delay-tolerant networks has been proposed in [12]. In contrast to multicast, our work assumes open user groups. The infostation concept is akin to our proposal and the paper in Ref. [10] studies means for avoiding exploitation of other nodes. We differ in that we make the nodal exchanges governed by a protocol instead of a social contract between users.

We show in [7] 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). Contact patterns of human mobility have been analyzed in the Haggle project [4]. This project aims at developing an application-independent networking architecture for delay-tolerant networks. In contrast, we implement 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.

BlueTorrent [6] is a cooperative content sharing system for Bluetooth. It differs from our approach in the search mechanisms and the content structuring. We rely on a channel-based content structure with a subscription model whereas BlueTorrent employes flat structuring with traditional query string search. TACO-DTN [11] is a content-based dissemination system for delay tolerant networks. It is implemented as a publish/subscribe system and was mainly designed to distribute temporal events whereas our approach is implemented as a pure receiver-driven system and optimized for dissemination of streaming media.

We use bloom filters in the searching for contents; a survey of the use of bloom filters in networking is given in [2].

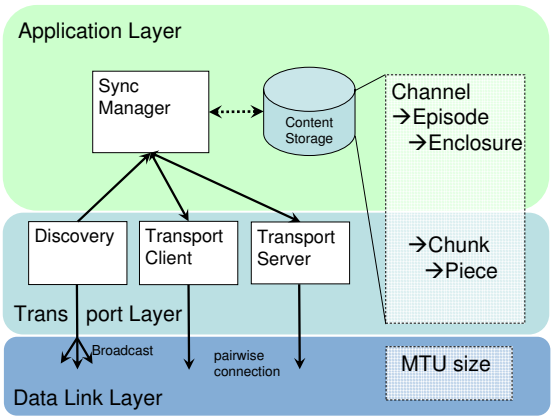


Figure 1: Service architecture for opportunistic podcasting.

3. DESIGN AND IMPLEMENTATION

This section describes the design and implementation of our opportunistic podcasting system. We first give a brief overview, then describe the system architecture, and finally disccuss the protocols involved in a pair-wise association when two devices meet.

3.1 Overview

In our architecture, we organize the content into *channels* to facilitate the search and the download of podcasts. Therefore, every user subscribes to the channels that he or she is interested in, and his or her device will try to retrieve any contents belonging to those channels. Following the approach used in the Internet-based podcasting protocols (RSS and the Atom syndication protocol), we structure channels into *episodes* and *enclosures*. To make efficient use of contacts with a small duration, we further divide enclosures into *chunks*, transport-level data units of a size that can typically be downloaded in an individual node encounter. Each chunk is further divided into *pieces*, the atomic transport unit of the network. The resulting system architecture is illustrated in Figure 1. Note that the architecture in the figure does not have any routing layer. The transport layer acts directly on top of the link-layer. To distribute contents among the podcasting nodes, we do not rely on any explicit multi-hop routing scheme. Instead of explicitly routing the data through specific nodes, we rely on a receiver-driven application-level dissemination model, where contents are routed implicitly as nodes retrieve contents that they request from neighboring nodes.

3.2 Architecture

We distinguish between an application layer, a transport layer, and the data link layer. In a first level of aggregation, the application organizes the contents in channels. Typical *channels* are for example sports channels, news channels, or music channels (of different types), but also more local or private channels like photo albums or audio recordings that user are willing to share with their surrounding.

Each channel is itself subdivided into *episodes*; each episode consists of one or more *enclosures*. Episodes are specific news programs, sports matches, or contents from specific artists. The enclosures are then smaller parts of the program, or the attachments in podcasting wording (e.g., individual files like videos of soccer matches, or music files).

Channel	Episode	Enclosure	Chunk 256 KBytes	Piece MTU size
Fox News	Today at noon	Video 1	Chunk 1	Piece 1
				Piece 2
				Pieces 3-x
			Chunk 2-x	Pieces x-y
		Video 2	Chunk 1	Pieces 1-x
Classic Music	Bach	MP3 1	Chunk 1	Pieces 1-x
			Chunk 2-z	Pieces x-y
	Mozart	MP3 1	Chunk 1	Pieces 1-x
			Chunk 2	Pieces x-y
		MP3 2	Chunk 1	Pieces 1-x
			Chunk 2	Pieces x-y
Rock Music	Queen	1^{st} song	Chunk 1	Pieces 1-x
		2^{nd} song	Chunk 1	Pieces 1-x
Application Layer			Transport Layer	

Table 1: Data structure for example contents.

Below the application layer, the transport layer organizes the data into *chunks*. Chunks are smaller data units that should be able to download over short contacts. The use of smaller file blocks is also supported by the idea of integrating forward error correction, for instance the use of fountain codes, to speed up and secure the data transfer, specifically when chunks are received unordered. The chunks themselves are then again cut into smaller parts to optimize the interaction with the data link layer; i.e., the size is set to the MTU size of the datalink layer. Table 1 illustrates an example mix of contents organized and structured as proposed in our architecture.

Our proposed system is designed to work on any MAC architecture, however, to be effective even in the presence of short contact durations, short setup times and high data rates are important for achieving high application throughput.

Our design is further characterized by two fundamental choices. First, we allow only pairwise associations even when the MAC layer supports multi-point communication. Second, we never push contents in the network and rely instead on receiver-driven dissemination. The arguments for these two choices are simplicity and optimal usage of short contact durations. Problems and limitations of broadcast or multicast distribution have been heavily discussed in reliable multicast research for many years. But, besides being much simpler to implement, the pairwise connection also optimizes the resources available for maximal throughput over short contacts, allowing a node to complete one download instead of ending up with several downloads interrupted by mobility. Furthermore, the transport layer is able to optimize the flow control between the two nodes and is not constrained by the slowest receiver in range.

Since we do not perform multi-hop routing explicitly, the system performance is mainly determined by the selection of nodes we are synchronizing with and the order by which we download contents from the peers. This task is performed by the *synchronization manager*, depicted in Figure 1. The synchronization manager is responsible for orchestrating and maintaining state about past encounters and current neighboring devices.

At the beginning, the devices have to determine the channels for which they wish to receive contents. This might impose a bootstrap problem for new devices that do not know of any channels. In the Internet, users typically lookup podcasts of interest with the help of search engines. However, in the ad hoc domain such search engines will not be accessible. We solve this problem by introducing a discovery channel that is known to all devices participating in the service. This channel includes IDs and meta-information of all channels. Contents from this channel are propagated in the same manner as other channels and as a result, new channels are discovered as peers associate and exchange contents.

3.3 Association phases

We next describe the process when two nodes associate. We differentiate two phases: a *discovery phase* in which both nodes detect that they are in range and a *download phase* in which the nodes negotiate and perform episode downloads.

3.3.1 Discovery phase

We assume that every device that participates in the opportunistic podcasting belong to the same network, e.g., with IEEE 802.11 every device is configured in ad hoc mode and uses the same SSID.

While the detection of new devices in the network is handled by the MAC layer, the application in the nodes has to take care of the discovery of devices that participate in the wireless podcasting service. Therefore, each device sends out periodic *hello* messages to the broadcast address of the wireless network. Note that if broadcast is not available at the MAC layer as in Bluetooth networks, the same idea could be implemented by letting the nodes periodically scan for devices in the neighborhood (e.g., using inquiry scans in Bluetooth). Hello messages contain the following information:

- Device ID: a globally unique twenty-byte ID.
- Status (*ready* or *choked*): A device advertises itself as *chocked* when it is already associated with another peer. Otherwise, the status is *ready* and it accepts new associations.
- Service port: the transport port on which the synchronization manager listens for incoming associations.
- New content date: includes the date at which a device last downloaded new contents. This field helps reducing the re-synchronization attempts in case two devices are within range for longer periods.
- Sequence number: a number incremented by the sender for each hello message. This number is used by the receiving nodes to determine if intermediate hello packets were lost.

The synchronization manager keeps track of the discovery messages received from each peer and maintains a history list.

An important aspect of the discovery process is to identify peers with a good connection. Since we designed our system for ad hoc scenarios, we have to consider the specifics of wireless communication. If a peer is far away or if the connection between the peers is hindered by walls or obstacles, we might be able to receive individual hello packets but most of the packets between those two peers will get lost. In order to discriminate such associations, we differentiate the peers based on a stability measure that accounts for the number of hello messages received in the past.

The stability measure works as follows. We count the number of hello packets that arrive at each node. The first packet received marks the peer as *fading in*. If we have received a certain number of consecutive hello packets (currently 3 in our implementation), the peer is marked as *stable* and we might associate with this node. If we do not receive hello packets for more than a timeout value (six seconds in our implementation), the peer times out and its entry is deleted (i.e., it moves to the state *away*). A peer which is not yet registered in the peer table always starts in the *away* state as indicated in Figure 2.

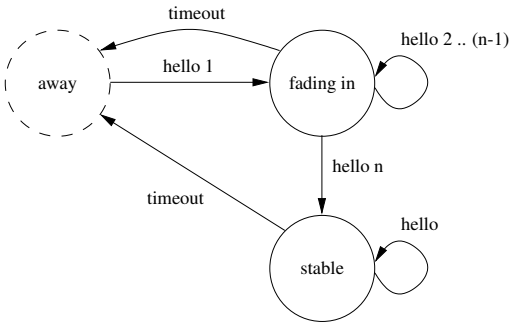


Figure 2: State diagram of discovery phase.

Another issue arises in scenarios when nodes are in range for longer durations and new contents become available. The surrounding nodes might not be aware of newly available contents, unless they constantly synchronize with all neighbors and check for updates. In our implementation we hence decided to add information about new content availability in each discovery packet (the *new content* field as mentioned previously). This *new content* field helps peers to determine if a re-synchronization might be beneficial. After an association gets terminated, the device checks if it has received new contents. If this is the case, the application updates its internal *new content* date. As this date is advertised with every discovery packet, other devices automatically know when they should trigger a re-synchronization with this device in order to download the new data. One might also think of including channel information in this field to inform which channels have been updated. However, since these hello messages are sent on a periodic basis, we decided to only include the date to keep the hello messages as small as possible.

3.3.2 Download phase

Once a podcasting device has been detected, the synchronization manager switches to the *transfer* mode which handles the synchronization. The *transfer* mode is based on a client-server model. The communication protocol is a request-response system. It is not a strict system though; some requests do not generate a response whereas other requests can generate several responses. The goal is to send as few packets as possible to reduce the communication overhead. Figure 3 illustrates the state diagram of the *transfer* mode. Due to space limitations, we do not discuss all states in details but focus on the three most important stages of a synchronization between two devices: the negotiation, query, and download stage.

In the NEGOTIATION stage, both devices determine if some of the subscribed channels are available on the other device. Instead of querying every single channel, the devices exchange a bloom filter that contains all *channel IDs* a device offers. The *bloom filter* on every device is initialized with the *channel IDs* of every *channel* on the device. Both devices then start to test their subscribed channels against the bloom filter of the other device and create a list of matching channels. This is a local process and does not involve the exchange of messages. False negatives are not possible, which means that a *channel* not found in the *bloom filter* is certain not to exist on the other device. False positives however are possible. A *channel* found in the *bloom filter* might still not exist on the other device. Although we still might query *channels* that do not exist on the other device, the number of queries is reduced and thus the filter speeds up the synchronization as we will see in the next section. Ideally, we would like to have a constant false positive probability

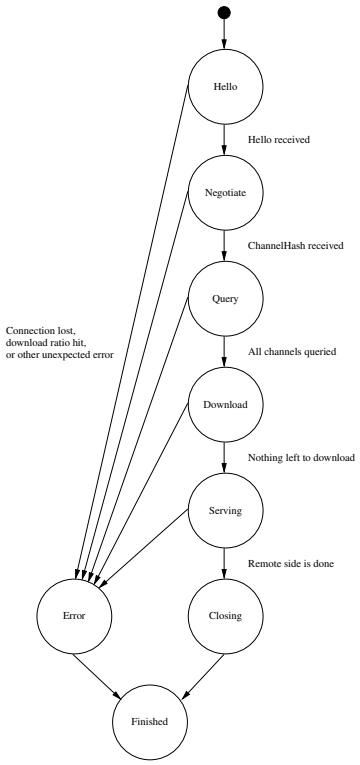


Figure 3: Transfer mode state diagram.

regardless of the number of *channel IDs* in the filter. Because we always know how many *channel IDs* have to be maintained in our bloom filter, we automatically adjust the filter size according to the number of channels. Every time the number of channels changes, we re-calculate the optimal size and re-create the bloom filter. As this happens relatively rarely, the overhead associated with this process is not critical. The mechanism described here is applied to the lists of channel contents only, but it could easily be applied to the synchronization of episodes and enclosures within a channel.

In the QUERY stage, a device confirms the channels selected in the previous stage and then retrieves a list of episodes offered by the remote device within those channels. Our implementation supports three different types of episode retrievals:

1. The peer requests any random episodes within a channel that the remote peer offers. This is particularly useful if the episodes are independent of one another (like a selection of poems).

2. A peer requests any episodes which are newer than a given date starting with the newest episode. This is particularly useful for news feeds in which the latest news feed has the most value.

3. The peer requests any episodes that are newer than a given date starting with the oldest episode. This type of retrieval is useful for episodes which depend on each other like a series.

The actual download of enclosures is handled in the DOWNLOAD stage which usually takes up most of the connection time. The device starts to process the list of enclosures that was created in the previous stage. The download itself works analogous to the download process of BitTorrent. Missing chunks which are available at the remote peer are randomly selected and downloaded. Remark that a chunk is divided into several pieces which are requested

one-by-one in data-link frames. The concept of chunks allows us to start downloading an enclosure even if the enclosure is only partially available at the remote peer. It does not matter if the remote peer only has part of an enclosure or if the connection gets terminated. It is always possible to resume an interrupted download from any other peer in a later synchronization attempt which helps to minimize the data to be retransmitted. Note that we share the wireless link capacity in a fair manner between the two associated nodes. That is, both nodes get half of the download throughput. However, we do not enforce any strict guarantees at the application level but rely on the MAC layer multiplexing for this.

If a device cannot find any more data to download it waits for the remote peer and terminates the connection as soon as the remote peer is done as well. Additionally, we impose an artificial limit on the associate duration. This allows a more fair content distribution in scenarios where multiple nodes are co-located, and one node has contents that the remaining nodes would like to have.

When two devices associate but have no channels or episodes to exchange, the devices could stop the synchronization without going into the download stage. A better alternative is to download contents that are not of direct interest to the user of the device, but which might be of interest to other users in future encounters. We have investigated different caching strategies for this cooperative approach in [8]; such an implementation is however not included in this paper.

4. EVALUATION

This section evaluates important design choices of the proposed architecture as well as the performance we might expect from a wireless opportunistic podcasting service. Our evaluation relies on our prototype implementation and focuses on typical deployment scenarios in which (i) a bunch of nodes are co-located (e.g., people in a bus) or (ii) when users are mobile and cross each other (e.g., two persons crossing each other in a street).

4.1 Measurement setup

The devices we performed our measurements on are 19 HP Iraq hx2400 devices (Marvell PXA270 processor - 520 MHz) with 128 MB RAM and running Windows Mobile 2003 Second Edition as well as five IBM ThinkPad T60 laptops (Intel Centrino Core Duo T5600 - 1,83 GHz) with 1GB of RAM and running Windows XP. We were mostly interested in the performance on limited devices like the iPAQs, since we foresee that our application will run on portable devices such as mobile media players. We measured the performance on laptops in order to better understand where the bottleneck of the system lies. All devices communicated over their integrated WLAN interface (802.11b) turned into ad hoc mode. Our podcasting application is written in C++. Since we were unable to access RAW sockets in Windows Mobile Edition, we implemented our application using traditional UDP broadcast sockets (for the discovery phase) and TCP sockets (for the synchronization and download phase). Note however that our podcasting architecture does not rely on the TCP/UDP/IP protocol suite and the performance of our system is expected to be the best when directly implemented on top of the MAC layer.

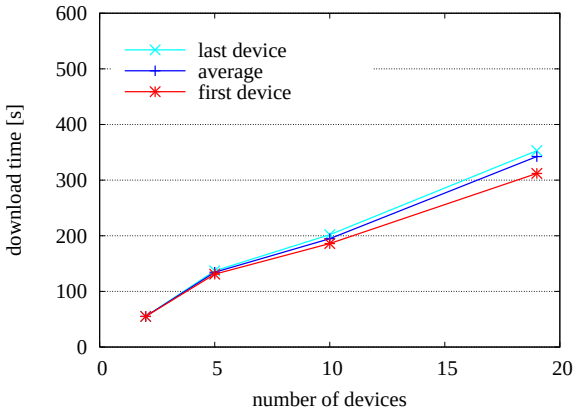
The devices that participate in the podcasting system are configured in ad hoc mode using the same SSID. We were not interested in any IP-related performance issues like auto-address configuration, indeed we intend to implement our service directly on top of the MAC layer, we configure the IP addresses of all devices manually in a private IP range. To facilitate the offline log file analysis, we synchronized the local clock of the devices before every measurement, because the mobile devices had a considerable drift.

4.2 Discovery phase

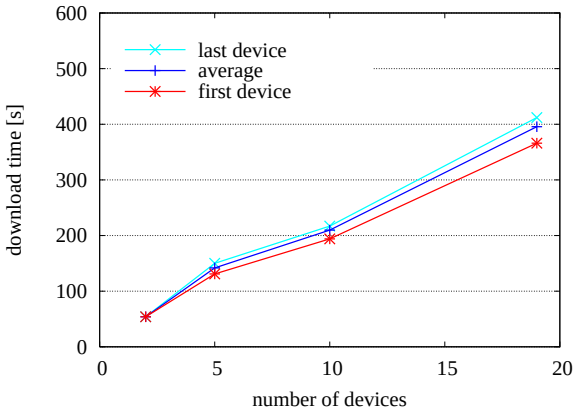
An important aspect of opportunistic wireless podcasting is the discovery time. The discovery time is the time between the moment two nodes move into transmission range until they discover each other at the application layer and start the synchronization phase. This time is directly impacted by the interval at which the hello messages are sent. The more frequent these messages are sent, the quicker they will discover each other and be able to transfer episodes. However, these hello messages should not be sent too frequently mainly because of two reasons. First, too many such messages will cause the battery of the devices to deplete faster. Second, we observed that the processing of incoming hello messages has a significant impact on the application performance. When many devices are in direct transmission range, the multitude of incoming hello messages limits the download throughput of the actual episode downloads. For example with 19 iPAQs in range, the average download time of episodes was around 15 percent lower with a hello interval of 1 second compared to 2 seconds. This performance degradation is not because the hello messages are consuming much bandwidth, but because the processing of those messages at the receiver is consuming CPU resources. The measurements that follow in this paper all have a hello sending interval of 2 seconds, which we view as a reasonable tradeoff between discovery time and download performance.

Another important aspect regarding discovery is related to the time it takes to discover nodes having new contents of interest, after being previously associated to them. We are interested in the time it takes at the application layer to detect this new piece of contents. For this purpose, we evaluate the benefits of having the *new content* date information in the hello packets by comparing the performance we would obtain through periodic checking (involving re-associating and re-synchronizing) for new contents at the neighboring devices. We study these two strategies in a scenario comprising of iPAQs that are placed close together on a table and have been mutually synchronized (this would for example correspond in a scenario where people are in the same bus, thus within transmission range). Then, we introduce a new node having an episode of 5MB that is of interest to all other nodes (for example, a person entering the bus with new content). We measure the time it takes until the other nodes have this new episode assuming all of them are subscribed to the channel of the newly arrived episode. This time for various number of devices averaged over three different measurement runs is plotted in Figure 4. For these sets of measurements, we set the maximum association time to 10 seconds, i.e., an association will stop after 10 seconds of download time, irrespectively of the completeness of the total download (recall that this mechanism is in place to allow a fairer content distribution among the nodes). In Figure 4(a), we see the performance having the *new content* field in the hello messages. Figure 4(b) shows the result when the devices re-associate every 30 seconds to check whether a neighboring device has new contents and Figure 4(c) illustrates the performance when devices re-associate every 60 minutes. We differentiate three time metrics: (i) the time until the first device has finished the complete download of the new episode, (ii) an average over all devices, (iii) and the time until the last device has finished the download.

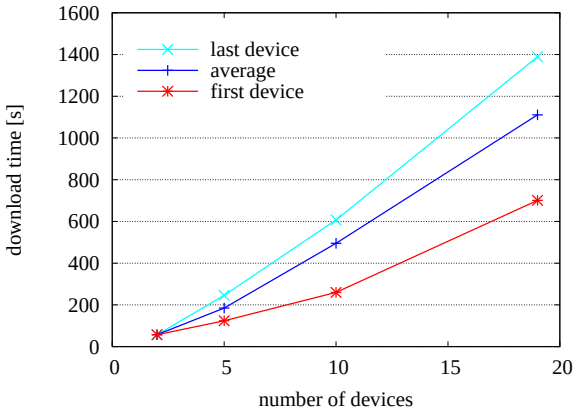
The download times for a re-synchronization timeout of 60 minutes are considerably larger. This is because all nodes try to download the episode from the device that originally brought it. However, after some time, some nodes have already successfully downloaded some chunks from this episode and could potentially share them. When the re-synchronization timeout is large, the nodes do not make optimal usage of the ability to download chunks from



(a) Hellos messages with *new content* field.



(b) Periodic re-association (every 30 seconds).



(c) Periodic re-association (every 60 minutes).

Figure 4: Time for a new episode of 5MB to be downloaded.

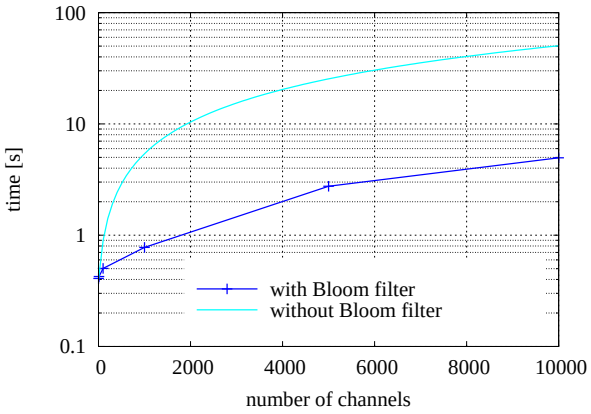


Figure 5: Synchronization time with and without bloom filter for different number of channels in a node's cache.

these other nodes. For a re-synchronization timeout of 30 seconds, the download times improve dramatically which means that nodes now start to better spread the chunks among themselves. The best performance is when having the *new content* field in the hello messages. Then, the neighboring nodes learn almost instantaneously when new chunks are available at the neighboring devices.

The difference between the *new content* hello messages and re-synchronizing every 30 seconds is not large in this setting. However, the measurements where completed with one individual channel which makes the synchronization time rather small. When the number of channel becomes large, the difference between these two strategies is more pronounced as the synchronization time increases. Furthermore, as continuously re-synchronizing requires more communication and thus more battery power at the mobile devices.

4.3 Synchronization

We next look at the synchronization time. The synchronization time is the time between the moment two devices have discovered each other and associate until they start downloading episodes of interest. Figure 5 shows the performance of our implementation for different number of channels using bloom filters compared to the time it would take without bloom filters (by sending the raw meta information of all channels). The shown results are from measurements with two iPAQs lying next to each other on a table (less than one meter). The channel meta information for those measurements is on average 500 bytes, corresponding to the average size we found in Internet podcast feeds. We vary the size of the bloom filter in order to keep the rate of false positives below one percent. As a result, the bloom-filter size of for example 1000 channels is approximately 1200 bytes.

Our measurements show that the use of a bloom filter dramatically reduces the synchronization time for a large number of channels. For example, the synchronization time for 2000 channels is around one second, ten times smaller than without the bloom filter. In the worst case, the synchronization time with lots of channels and without the bloom filter would be so large that many device encounters of a small duration could not be used to transfer any episode or even any chunk.

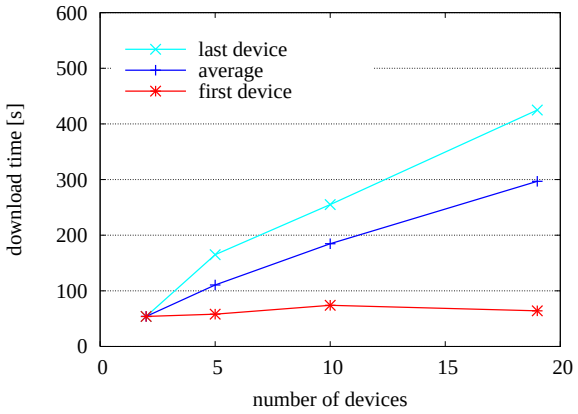


Figure 6: Download time of a 5MB episode without limiting the maximal association time.

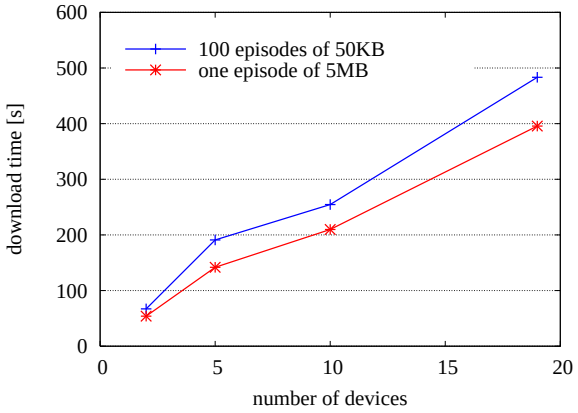


Figure 7: Download time of 100 episodes of size 50 kB versus one episode of size 5MB.

4.4 Download

We artificially limit the association time to ensure that the system remains fair when multiple nodes are interested in the contents provided by just a few nodes. Without this artificial limit, the first node that associates with the node having new contents would be able to download for an indefinite amount of time, letting the other nodes starve for contents. To observe the behavior of the system without this artificial limit, we look in Figure 6 at the download time that we would get without limiting the association time when a new episode of 5MB gets available within a channel to which all surrounding iPAQs are subscribed. As we can see, the time for the first node to finish its download is almost independent of the number of neighboring devices subscribed to the channel. However, the time until the last node gets the new episode is almost 400 seconds larger for 19 devices. This is because some nodes have to wait for the complete download of other nodes before they can start downloading themselves. Comparing with a limit of 10 seconds association time in Figure 4(a), we see that all nodes get the episode at almost the same time. Regarding the average download time though, this limit adds a small penalty compared to the unlimited download approach. This increase in the average value mainly comes from the additional synchronization overhead caused by more synchronization attempts in total.

We also studied how the performance with a large number of small episodes would be different from having fewer large episodes. Figure 7 compares the average download time varying the number of neighboring iPAQs for one episode of 5MB versus 100 episodes of 50 kB. Each of the 100 small episodes belongs to a different channel. The curves show that it is slightly more expensive to retrieve small episodes from different channels than one large episode from an individual channel. This comes from the higher synchronization overhead.

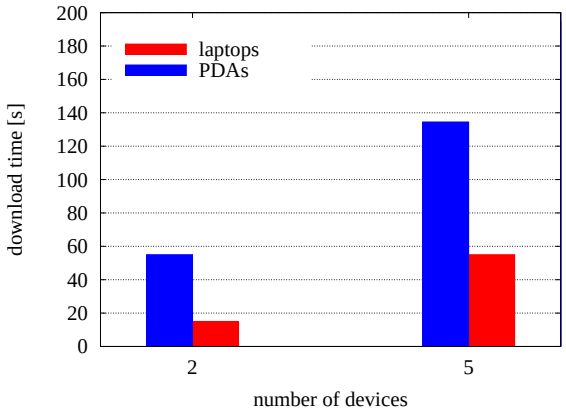


Figure 8: Average download time of a 5MB episode with the PDAs compared to laptops.

4.5 Hardware limitations

The maximal download throughput we measured at the application layer between two iPAQs was around 100-120 kB/s even under ideal conditions (when two devices were very close to each other, not moving, without interference from other transmissions). This is far below what one would expect from an 802.11b connection. We asked ourselves about this surprisingly low value and performed additional measurements to understand the bottleneck. First, we measured the maximum throughput when sending data over TCP between two iPAQs but without processing the data at the receiver (which we have to do in our application to interpret and store the data). We measured then a maximum throughput of around 350 KB/s. Therefore, the throughput degradation in our application must come from the processing and storing of the incoming data to the storage Flash cards. An additional measurement revealed that we can write to the storage card at a rate of around 2.5 MB/s; hence, the storage media itself is not the bottleneck. We profiled the application and realized that the slowdown comes mainly from the *memcpy* call to write data packets that we receive from the network to the card. Unfortunately, it is not possible to get rid of this *memcpy* call which would be necessary to transmit at the maximal rate of the wireless interface.

To show how our implementation performs on more capable devices like laptops, Figure 8 shows a comparison of the download time with the iPAQs and the laptops. The results are from average download measurements of 5 MB episodes showing the download time when two or five devices are placed close to each other on a table. The download time with two laptops is around 3.5 times faster than with two iPAQs. This shows that the laptops are able to transmit at the maximum transfer rate of the wireless interface (approx. 350 KB/s). For the measurements with 5 laptops and 5 iPAQs, we observe an average download time with the laptops which is 2.5 times faster than with iPAQs. This relative improvement is smaller than with two devices because two associated laptops cannot trans-

Scenario	Download Time[s]
mobile	39.8
0.1 m	38.4
20 m	39.0
60 m	45.4

Table 2: Download time of a 3.5MB episode between two nodes.

mit at the maximum transmission rate when two other laptops are associated and transmitting simultaneously as they must share the wireless channel capacity. .

4.6 The effect of mobility

To see how the download time is affected by node mobility, we have performed an additional set of experiments that indicates the performance we might obtain when two people cross-by on a street or in a building's hallway. For this purpose, we placed in the hallway of our laboratory a fixed iPAQ. Another iPAQ carried by a test user is subscribed to a channel for which the fixed iPAQ has episodes stored. The mobile test user walks from the very end of the hallway (the length of the hallway is 60m) towards the fixed iPAQ (at approximately 1 m/s), crosses it and continues in the same direction until it comes out of range. We measure the time it takes for the mobile node to download an episode of 3.5MB from the moment on when it moves inside communication range of the fixed device. This is the time between the moment the two devices see each other's MAC-layer beacons until the complete episode has been downloaded. The average of this time over 6 runs is given in Table 2 (mobile). For comparison, we also give the measured download time when the two iPAQs are fixed with a relative distance of 0.1m, 20m, and 60m. As we can see, the average download time with the mobile node is larger than when the two nodes are at a distance of 20 m but smaller than when the nodes are 60 m apart. This shows that the mobility of the node has not a significant impact on the discovery, synchronization, or download phase but the distance between the two nodes is determining the download time. Also, it shows that an episode of the size of a typical audio file can be transmitted over a single contact in an indoor environment when nodes move at pedestrian speeds.

5. CONCLUSION

We present in this paper an implementation study of wireless opportunistic podcasting for mobile devices. Our prototype implementation is targeted at hand-held devices that communicate over IEEE 802.11 (or other types of short-range radios). We evaluate different tradeoffs in the discovery, synchronization, and download phases. Based on measurements, we provide a comprehensive comparison of the tradeoffs and bottlenecks with real hardware devices. Overall, the resulting performance is promising and shows the feasibility of wireless opportunistic podcasting. With the proposed opportunistic podcasting system, we further hope to broaden the concept of traditional Internet podcasting and that new participatory wireless broadcasting applications using the proposed concepts will emerge in the near future.

6. REFERENCES

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

[2] A. Broder and M. Mitzenmacher. Network applications of bloom filters: A survey. *Proceedings of the 40th Annual Allerton Conference*, 2002.

[3] 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.

[4] Augustin Chaintreau, Pan Hui, Jon Crowcroft, Christophe Diot, Richard Gass, and James Scott. Impact of Human Mobility on the Design of Opportunistic Forwarding Algorithms. In *Proceedings of IEEE INFOCOM*, Barcelona, Spain, April 2006.

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

[6] Sewook Jung, Uichin Lee, Alexander Chang, Dae-Ki Cho, and Mario Gerla. BlueTorrent: Cooperative Content Sharing for Bluetooth Users,. In *Proceedings of PerCom*, White Plains, NY, USA, March 2007.

[7] Gunnar Karlsson, Vincent Lenders, and Martin May. Delay-Tolerant Broadcasting. In *Proceedings of the ACM SIGCOMM CHANTS Workshop*, Pisa, Italy, September 2006.

[8] Vincent Lenders, Martin May, and Gunnar Karlsson. Wireless Ad Hoc Podcasting. In *Proceedings of IEEE SECON*, San Diego, CA, June 2007.

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

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

[11] G. Sollazzo, M. Musolesi, and C. Mascolo, "TACO-DTN: A Time-Aware COntent-based dissemination system for Delay Tolerant Networks," in *Proceedings of the First International Workshop on Mobile Opportunistic Networking (Mobiopp)*, Puerto Rico, June 2007.

[12] 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.