DeltaCast: Efficient File Reconciliation in Wireless Broadcast Systems

Julian Chesterfield
University of Cambridge,
Computer Laboratory,
Cambridge, UK
julian.chesterfield@cl.cam.ac.uk

Pablo Rodriguez
Microsoft Research,
7 JJ Thompson Avenue,
Cambridge, UK
pablo@microsoft.com

Abstract

Recently, there has been an increasing interest in wireless broadcast systems as a means to enable scalable content delivery to large numbers of mobile users. However, gracefully providing efficient reconciliation of different versions of a file over such broadcast channels still remains a challenge. Such systems often lack a feedback channel and consequently updates cannot be easily tailored to a specific user. Moreover, given the potentially large number of possible versions of a file, it is impractical to send a tailored update for each particular user.

In this paper we consider the problem of efficiently updating files in such wireless broadcast channels. To this extent, we present DeltaCast, a system that combines hierarchical hashes and erasure codes to minimise the amount of battery power and the amount of time needed to synchronise each mobile device. Based on our experimental results, we show that DeltaCast is able to efficiently identify the missing portions of a file and quickly updated each client.

1 Introduction

Wireless communication systems are one of the fastest growing areas of communication technology, with new devices and standards constantly emerging, developed to transmit digital signals over both short and long distances. There are two main delivery approaches for wireless data services: *point-to-point* and *point-to-multipoint* or broadcast systems [1]. Point-to-point systems employ a basic client-server model, where the server is responsible for processing a query and returning the result to the user via a dedicated point-to-point channel. In broadcasting or *DataCasting* systems, on the other hand, the server actively pushes data to the users in an open loop fashion. The server determines the data to be transmitted and schedules the broadcast. A user listens to the broadcast channel to retrieve data without any signalling to the server and thus is responsible for his own query processing.

Point-to-point access is particularly suitable for light loaded systems where contention for wireless resources and server processing is not severe. As the number of users increases, however the overall system performance can quickly degrade. Compared with point-to-point access, broadcasting is a very attractive alternative [2][3]. It allows simultaneous access by an arbitrary number of mobile clients without causing any inter-receiver interference and thus allows efficient usage of the scarce wireless bandwidth and server resources.

Wireless data broadcast services have been available as commercial products for many years (e.g., StarBand [4] and Hughes Network [5]). However, recently there has been a particular push for Wireless DataCasting systems both in Europe and in the US. Both 3gpp as well as 3gpp2 are developing plans to provide multicast/broadcast support over UMTS/CDMA networks [6] [7]. Moreover, systems such as Digital Audio Broadcast (DAB) and Digital Video Broadcast (DVB) standards in Europe [8] and ATSC in the US which have been traditionally used for the transmission of digital radio and television, are also capable of providing data services, a feature that is anticipated to experience significant utilisation. One of the key motivators for the rapid growth of digital broadcasting and in particular, data services over DAB and DVB, is that cell phone manufacturers are announcing plans to trial new devices incorporating built-in DAB/DVB receivers over the next year [9]. This will enable a wide range of DataCasting services into mobile devices. Another example of DataCasting services is the smart personal objects technology (SPOT) by Microsoft [10], which further highlights the industrial interest in and feasibility of utilising broadcast for wireless data services. Using such DataCasting services, mobile devices can continuously receive timely information such as maps, software, news, weather, traffic information, etc.

In this paper, we focus on wireless data broadcasting services and mechanisms for efficiently reconciling content on distributed receiving devices to the latest available version. Reconciliation of data files between two computers is a well studied problem with efficient solutions such as rsync [11] utilised on a wide scale in the Internet. Such solutions, however, rely on two-way communication channels to identify the appropriate delta information between two files and are best suited for point-to-point reconciliation. In DataCasting systems, mobile users with intermittent coverage or settings may synchronise at arbitrary times, thus, generating a potentially very large number of different versions. DataCasting systems often lack a feedback channel, which prevents the server from gathering information about the user's file and therefore provide a tailored delta update. Even if the server had knowledge of which versions each client has, it is difficult to tailor the broadcast channel to a specific user without delaying other users with different versions. All these conditions make it hard to construct a system that gracefully provides differential updates to a large number of different users in a multicast/broadcast system.

One possible solution is to make sure that the users update their own data to the most recent version every time by simply transmitting the latest version of the file. It is generally expected, however, that changes to content versions of a file on average are likely to be incrementally small, hence the motivation for widespread adoption of protocols like rsync, providing a good deal of redundancy between current and previous data. By forcing the user to repeatedly download the same portions of the file, inefficient battery consumption and an increase in download latency can be experienced as a result.

Up until recently, fixed power devices have been the most common DataCasting service receivers, e.g. home radios and computers, or vehicular based receivers. In the near future however it is anticipated that there will be a proliferation of small, handheld, mobile receivers such as cellphones and PDAs capable of receiving DataCasting services. Access efficiency and energy conservation therefore are two critical issues for users in a wireless data broadcast system. Access efficiency refers to maximising the speed with which data is downloaded, while energy conservation minimises the mobile client's power consumption when accessing the data of interest.

In this paper we present *DeltaCast*, a new mechanism that provides efficient file reconciliation in DataCasting systems. DeltaCast combines decomposable hierarchical hashing schemes with low-cost random linear codes to enable efficient file synchronisation among an unlimited number of receivers. As we will see later in the paper, DeltaCast has the ability to update different receivers' versions simultaneously through the use of *master* encoded deltas that can be used by any receiver at any point in time.

The rest of the paper is structured as follows. Section 2 discusses the design of *DeltaCast* in response to the specific challenges of the environment. We discuss the use of decomposable hashes and erasure codes to minimise the amount of data downloaded by each receiver. Section 3 we outline the performance considerations of the system, and our approach toward measuring the benefits and trade-offs of different file synchronisation techniques. We present results from our experiments with the *DeltaCast* system, comparing the optimised *DeltaCast* against alternative approaches, and demonstrating that it is a very efficient approach to reconciling data in broadcast systems. In section 4 we discuss related work, and we conclude the paper in section 5.

2 DeltaCast Design

DeltaCast is an efficient file synchronisation protocol for one-way, receiver driven incremental file updating. A typical use of DeltaCast would be in the radio broadcast environment, where one-to-many DataCasting can provide a highly effective means to reach unlimited numbers of receivers without requiring any increase in network capacity to scale the service to larger numbers. *Deltacast* in fact is not just useful within the wireless broadcast environment, we expect it to provide similar benefits in terrestrial point-to-multipoint environments such as IP Multicast, Content Distribution Networks and Peer-to-peer overlay systems.

A limitation of such radio broadcast systems, however is that data can only be delivered asymmetrically. Typically, receiving devices do not have any return channel to the source, and therefore the transmission must be uni-directional. Typical applications in this environment might be providing downloads of local information to mobile devices, maps, software, video, etc. An advantage of DataCasting is that multiple channels can be used to transmit content in parallel, and the scalability of the system is only constrained by the amount of content that can fit in the available spectrum bandwidth. Data content is typically carouseled continuously to enable receivers to 'tune-in' at any stage to the relevant channel and receive content. Since this is a broadcast only environment, low layer errors are handled through strong redundant Forward Error Correction (FEC) applied to the source signal. In this paper we will assume that the receiver's application does not experience errors due to signal interference and that any missing data is due to lack of application level synchronisation.

Due to the heterogeneity of the receiver group and the wireless environment, it is likely that a) there will be intermittent connectivity for individual receivers (e.g. loss of signal, power-down overnight etc.) and b) that different receivers start the synchronisation process at different points in time, resulting in a high variation in most recently synchronised versions of data distributed across the whole receiver set. This variability is compounded by the fact that the source has no knowledge of the exact distribution of data versions. In addition to which, many devices utilising such a service are likely to be small, mobile and consequently power constrained. The solution must take all these considerations into account.

2.1 Problem Definition

The setup for the file synchronisation problem is as follows. We have a current file f_{new} with size F, and a set of N outdated file versions $f_{old,j}$, for $j \in 1,..,N$ over some alphabet $\sum$, and two types of machines C (the clients) and S (the server) connected through a broadcast data channel and no feedback link. We assume that machines in C have copies in $f_{old,j}$ and S only has a copy of f_{new}, and the goal is to design a protocol that results in C holding a copy of f_{new}, while minimising the amount of time and the resources consumed at the clients.

2.2 General System Overview

The Deltacast system comprises two distinct phases; the *detection* of changes between old and new content and the *update* of such changes. During the detection phase, receivers determine what portions of the local file match the target updated version at the server and can be re-utilised. Note that the receiver may not match any portion of the local file, in which case it requires to download the full file. Unmatched portions are discarded, while matched portions are kept in a temporary file. During the update phase, each receiver downloads enough data to fill in the gaps left by the un-matched portions.

The server receives no feedback at all from the receiver group and sends out a constant stream of hashes and data. Hashes are used during the detection phase, while data is used during the update phase. During the detection and the update phase, erasure encoding is used. Such erasure codes ensure that any information received by any receiver is useful.

To identify the unmatched portions with a high level of granularity, the server does a multi-level hierarchical division of the target file into blocks of a fixed size. The size of a block at each hierarchy level being half the size of the preceding level, i.e. decreasing powers of 2. For every block of data the server then generates a small, unique hash. When all the hashes at a given level have been generated, the server uses such hashes to create *erasure hashes* for a particular level. There is a potentially very large number of unique erasure hashes that can be

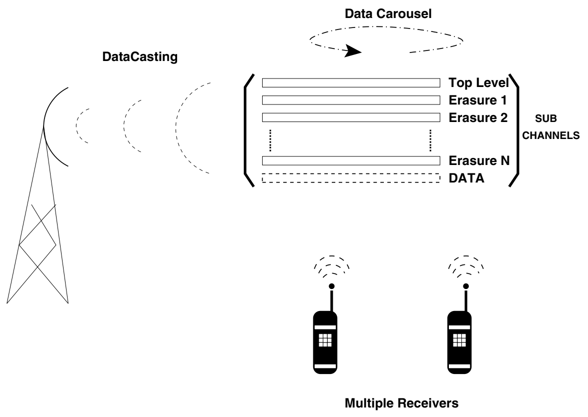

Figure 1: DeltaCast Architecture.

produced, which are published over the broadcast channel (normally erasure hashes for each level are sent over different channels). The same process is then repeated at each level of the hierarchy, producing a constant stream of erasure hashes at each level of the hierarchy. At the bottom level of the hierarchy, the server also produces *encoded data blocks*, which are broadcast in a separate channel and are used by the receivers to fill in the unmatched portions of their local files. Figure 1 shows the overall architecture of the DeltaCast system.

Receivers in the detection phase download enough hashes at each hierarchy level to determine the unmatched portions of the local file. To this extent, receivers first tune in to the top-level hash channel and download a complete set of hashes. Typically the top-level hashes are sent un-encoded since no additional benefit is achieved by encoding them. The receivers know the hashing algorithm used by the server, and thus, can generate a local database of hierarchical hashes corresponding to the local file. By comparing each downloaded hash against the local database, the receiver identifies the high level areas of the file that are matched. For each area of the file where the hash comparison failed, the receiver downloads enough erasure hashes from the subsequent level (normally, the number of erasure hashes required is twice the number of un-matched hashes at the previous level). Such erasure hashes are then combined with the hashes of the matched blocks at that particular level in order to reconstruct the missing hashes for the un-matched blocks. Since the hashes at each subsequent level correspond to a smaller area of the content, the detection of changes becomes more fine-grained each time.

Once the receiver has either reached the lowest level of hashes, or has identified that there is no further benefit to detecting finer-grained changes by reconstructing a further sub-level of hashes, it enters the *update* phase. In the update phase, for every unmatched hash at the lowest level, the receiver downloads a corresponding number of erasure data blocks. Given the erasure blocks, and the set of matched content blocks, the receiver can then reconstruct the latest version of the content as published by the server. The final file is verified with a strong signature provided by the server.

2.3 The Details

DeltaCast enables receivers to efficiently identify changes in their own content and download only as much data as required to synchronise their version with the source. The most common techniques for providing identifying matched portions between a source and receiver involve content hashing over partial portions of a file. Due to the high probability of data similarity in incremental version updates, forwarding block hashes, can potentially save a significant amount of bandwidth and time through identifying the precise portions of the old version which need updating, rather than forwarding the whole data file. However, identifying the most efficient block size for which the source sends hashes to the receiver, is challenging and greatly varies depending upon the nature of the content changes. Utilising small block sizes increases the number of blocks and consequently the amount of hashes transmitted. In contrast, utilising large block sizes decreases the amount of hash data transmitted, but lowers the precision of each hash, consequently reducing the ability to detect fine-grained data changes. In this manner there is a distinct trade-off between the granularity of the hashes and the amount of data transmitted, the optimal setting of which can only be determined retrospectively by comparing the two versions.

One way to solve this problem is to use a *hierarchical hashing* scheme that gradually decreases the block sizes to narrow down the unmatched portions. However, hierarchical hashing schemes often rely on multiple rounds of communication between the server and the client, sending only the appropriate hashes at each hierarchy level. Such multi-round protocols cannot be implemented in open-loop DataCasting systems. Moreover, since the server does not know which versions are currently held by the receivers, it needs to broadcast all hashes for each block at every hierarchy level. This wastes precious bandwidth resources at the broadcast channel but, even more significantly, increases the average time for a user to update its file since many hashes in the carousel will correspond to already matched file portions.

The DeltaCast approach is to utilise a hierarchical hashing scheme combined with highly efficient *erasure coding* and *decomposable hashes* in order to efficiently reconcile different file versions at multiple receivers simultaneously.

The approach is as follows:

1. The source performs a hierarchical division of the file into n different layers $i \in 1, .., n$, with each layer having a different block size $b_{max}, ..., b_{min}$, where b_{max} is the block size at the top layer $i = 1$, and b_{min} is the block size at the lowest layer $i = n$. In this paper we assume that the block size is halved at each level, i.e. $b_{i-1} = b_i * 2$. For each block at a given level, the server calculates a block hash $h(b)$, which is of size H bits. The only constraint on block sizes is that $b_{min} > H$.

2. For the first level of the hierarchy, the server transmits the top-level hashes continuously. For each subsequent level of the hierarchy, the server calculates a large number of *erasure hashes* based on the block hashes at that level. For a given level, erasure hashes are calculated as a random linear combination of the hashes at that particular level and have the property that they can be used in place of any hash. The total number of possible different erasure hashes is very large. Erasure hashes for each level are continuously transmitted in a separate channel.

3. One extra channel is included to forward the data content. Data content is not sent in its original form but as erasure blocks of the original data.

The reconstruction at the receiver proceeds as follows:

1. Let F be the size of the file at the server. The receiver first downloads F/b_{max} top level hashes and coarsely identifies the matching blocks. To do this, it checks the values across the whole file using a rolling block window and marks any unchanged data blocks. The set of blocks where the hashes do not coincide are marked as unmatched. The size of that set at level i is represented by $u(i)$.

2. For each level $i > 1$ of hash values, the receiver downloads $u(i - 1) * 2$ erasure hashes in order to reconstruct the correct set of hashes at level i. As we will see later in this section, we can halve the amount of erasure hashes downloaded at level i using decomposable homomorphic hashes. Such hashing functions hold the property that $h(a) = h(b) + h(c)$ where a is the top level block, and b and c are the sub-level blocks corresponding to the top-level block a. This property enables us to reconstruct all hashes at level i by downloading only $u(i - 1)$ erasure hashes.

3. The same process is repeated at each level of the hierarchy. After decoding the hashes at level n and identifying the number of failing blocks $u(n)$, the receiver downloads $u(n)$ encoded data blocks from the data content channel. After proper decoding, the receiver is able to reconstruct the file and thus synchronise it's local version with the source file.

Figure 2 illustrates the encoding approach at the source, highlighting the hash relationship between levels, and the decreasing size of the blocks.

2.4 Hash Accuracy

One important parameter in the DeltaCast system is determining the size of the hash H. Provided that the probability distribution of hash values is perfectly random, the statistical accuracy of a hash of H bits can be expressed as a function of the comparison file at the receiver, f_{old}, of length m and the block size b on which the hash is based. Suppose that S initially sends a set of H-bit hashes to C, and that C uses these hashes to check for matches in f_{old}. As the comparison block window at the receiver is rolled across every byte position in the file, there are $m - b + 1$ possible match positions in f_{old} that need to be checked, and for each position there is a certain probability of a false match. Other hashes based on fingerprints [26] could also be used to reduce the complexity of matching blocks over the file, however since these hashes use variable size blocks, trying to maintain a clean hierarchical structure becomes more complex and additional signalling would be required from the source. Since the probability of a certain match occurring at any single position in the file is expressed as $1/2^H$, the probability of any particular match *not* occurring (i.e. no collision) anywhere in the file is $(1 - 1/2^H)^{m-b+1}$. Thus we can say that the chance of a false match is $1 - (1 - 1/2^H)^{m-b+1}$. For an even chance of not getting a false match, the required hash size can be approximated by $lg(m) + log(m/b)$, while $lg(m) + lg(m/b) + d$ bits are needed to get a probability less than $1/2^d$ of having a false match between files.

For every file downloaded, the server also sends a 16-byte MD5 hash of the entire content that allows the client to check the correctness of the final version. Thus, in the unlikely event that collisions occur due to weak block-level hashes, the client can then download the content again.

2.5 Erasure Hashes

To be able to quickly identify the missing data portions, receivers need to download the correct set of block hashes as fast as possible. Different receivers may be missing different data blocks, however and therefore need a different set of hashes. Thus, a common approach is for a server to continuously send the full set of hashes at each level to ensure that all receivers can synchronise their file regardless of what version they may have. If the server knew which receivers where missing what portions, it could devise an more efficient protocol where certain hashes were to be transmitted more frequently than others. However, without any feedback channel, the server has no choice but to send all hashes. As a result, clients must wait for the specific hash to arrive in the carousel, potentially wasting a lot of time.

DeltaCast solves this problem by encoding the original hashes to produce erasure hashes that can be used by *any* receiver in place of *any* missing hash at a given level (a similar thing happens for the data content channel). To this extent, we have implemented a rateless erasure encoder/decoder that generates a set of random linear erasure blocks. Each erasure block is produced by selecting a unique set of co-efficients, applying the co-efficients across the original source blocks and then calculating the sum of all the blocks over a finite field(e.g. GF 2^{16}). As long as the co-efficients are generated randomly over a sufficiently large field, the probability of generating a non-innovative code becomes insignificantly small.

Such codes hold the property that for a system of n source blocks, the receiver only needs to obtain any combination of n erasure or original data blocks, where the blocks are linearly independent, in order to reconstruct the original source data. For the receiver to be able to decode the erasure blocks, the encoded data is also tagged with the set of random co-efficients that were used to generate the linear combination. The co-efficients can be explicitly broadcast with the encoded block, although a more efficient approach would be to use a seeded pseudo-random generator, in which case only the seed value and the index would be required. To improve the decoding time, one can also use well-known sparse encoding techniques which require downloading some extra data but significantly reduce the overall decoding time [36].

2.6 Decomposable Hashes

Decomposable hashes permit receivers to download half the number of erasure hashes at each level. The idea of using decomposable hashes is that if we have already transmitted a hash value for a parent block at a given hierarchy level, then for certain types of hash function we could compute a child hash at the next hierarchy level from the top-level hash and one extra child hash. In practise though, designing appropriate hash functions to implement this is nontrivial.

Assume that a top-level block is created out of bytes $[l, r]$, and the corresponding children blocks are $[l, m]$ and $[m, r]$. We say that a hash function is decompos-

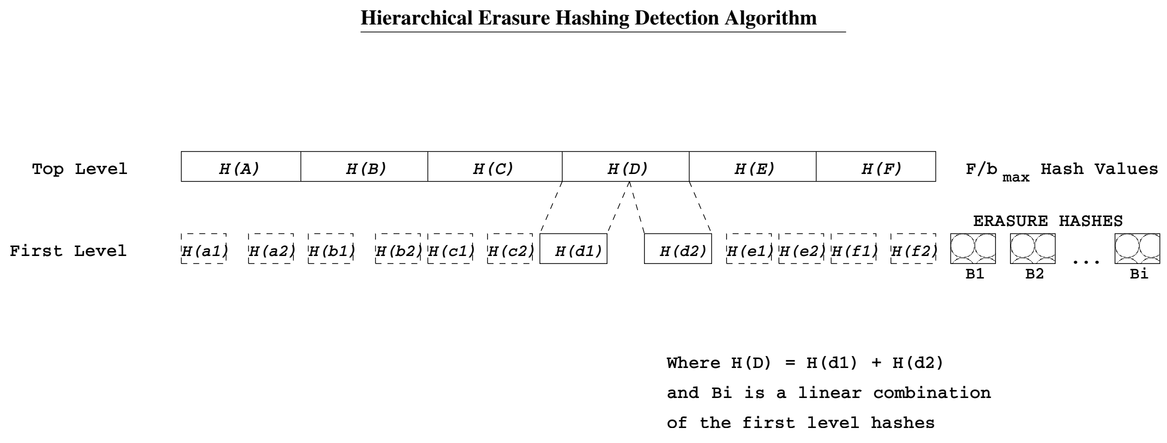

Figure 2: DeltaCast hashing scheme. Two levels.

able if we can efficiently compute $h(f[m+1,r])$ from the values $h(f[l,r])$, $h(f[l,m])$, $r-l$, and $r-m-1$, and also $h(f[l,m])$ from $h(f[l,r])$, $h(f[m+1,r])$, $r-l$, and $m-l$.

A decomposable hash function allows us to significantly save on the cost of delivering hashes for identifying matching data. Since receivers already have a hash for the parent block, they only need to receive one additional hash per pair of sibling blocks, the hash for the other sibling can then be computed from these two. Utilising decomposable hashes is particularly efficient in the deltacast system since erasure hashes can be produced in the same way out of decomposable hashes. In fact, decomposable hashes can be used as input to the erasure decoding algorithm at the next level of the hierarchy where, given the parent hash and an additional erasure block, both of the next level hashes can be reconstructed. Decomposable parent hashes can be interpreted as simple erasure codes created as a linear combination of the two child block hashes.

Decomposable hashes can be implemented using Homomorphic hashing functions such as the ones described in [12][13][14] and [15] where $h(a+b) = h(a) + h(b)$ or $h(a*b) = h(a)*h(b)$, a and b being two different blocks. To make use of decomposable hashes in DeltaCast we ensure that $h(f[l,r])$ for a block at a given level is equal to $h(f[l,m]) + h(f[m,r])$, where $h(f[l,m])$ and $h(f[m,r])$ are the hashes for the corresponding sibling blocks at the next hierarchy level. To this extent, we define $h(b_1, ...b_n) = \sum_{v=1}^{v=n} g^v \cdot b_n$, where b_i is an individual block, $(b_1, ..., b_n)$ is the parent block made of the concatenation of blocks $1, .., n$, and g is a generator number of a given Galois field. A more detailed study of such functions is described in [15].

3 Experimental Results

Next we consider the performance of the algorithm. We evaluate the improvement in data download requirements through utilising hierarchical hashing versus single-layer hashing, and additionally consider the performance from a temporal perspective, comparing the time to synchronise content using Deltacast against a full content download as well as the non-encoded case.

3.1 Performance Considerations

Within a fixed wire environment such as the Internet, the key optimisation metric for data download performance is minimising the overall download latency from initialisation to completion of the data transfer from source to receiver. Factors which typically contribute to this latency are protocol communication overheads, e.g. for multi-round conversations between senders and receivers, the delay for which can vary considerably depending on the link. In the broadcast radio environment, latency is typically tightly controlled and bounded by the physical propagation speed of the signal and the channel capacity. Since the environment can only support one-way broadcast communication, we do not consider any interactive protocol communication overhead. Instead, we must consider the amount of data that is transmitted in order to synchronise the local receiver data version with that of the server. We consider the average time for which a receiver may have to wait to receive sufficient data, and in addition we must consider the overheads of erasure decoding introduced by the Deltacast system. The latter consideration is particularly important in drawing a fair comparison since although the decoding may be handled offline, it is still power intensive, and whilst we do not measure precise energy consumption values, we include it as a latency factor in our experiments.

3.1.1 Power Efficiency

Minimising receiver power consumption is a responsibility for both the receiver device and the protocol designer. The process of tuning a radio to a channel and demodulating data being transmitted consumes energy. Attempting to minimise the frequency with which a receiver must demodulate data in order to synchronise itself with the transmission from a source is therefore crucial. By utilising erasure coding, Deltacast ensures that *any* data the receiver may demodulate will be useful. DeltaCast leaves a certain amount of intelligent interpretation up to the receiver by providing it with multiple encoded channels from which to select and download data as required. There is no passive listening required while certain data segments rotate through the carousel since any erasure block can be used as data. There is a trade-off in utilising erasure coding in terms of the decoding requirements and consequently additional power is consumed by the device in the process of decoding erasure blocks, however we maintain that there are many optimisations that can be applied to minimise the impact of the decoding complexity such as using sparse matrices, or dividing the source data into smaller files. The benefits of utilising erasure codes therefore greatly outweigh the processing disadvantages, and we demonstrate that the latency overhead of decoding is much lower than the download latency of unencoded data transmission.

3.1.2 Broadcast Channel Capacity

The Datacasting environment is bandwidth constrained due to the finite limitation of regulated wireless spectrum compared to the diversity of content that users might like to see. However, there is provision for data transmission, and there a large variety of applications that are anticipated for use within the environment. Channels are typically assigned in multiples of 64Kbit/s due to the tight correlation with traditional audio and video transmission. Wide-band popular applications might receive more than one channel, enabling faster or more frequent data updates, however typically it is assumed that each channel would be utilised by one data carousel. In our experiments therefore we consider the usable throughput of the channel to approximate close to 50kbit/s, leaving an additional 14Kbit/s to account for protocol headers and any additional signalling information that may be required by the source. Within the channel we further subdivide the data into sub-channels. We do not draw a distinction between physical division of the spectrum or logical, e.g. IP level multicast addressing, since this is highly technology dependent, however we assume that channels can be arbitrarily divided into smaller units.

3.1.3 Latency

Minimising data transmission bandwidth through efficient encoding is highly desirable, however it is also important to consider the delays imposed by the system on a user device. Certain types of information have more real-time constraints than others, e.g. stock quotes, or news sites, and we therefore also address the performance of the system in terms of data synchronisation latency. DeltaCast performs extremely favourably in this respect since the utilisation of erasure codes does not impose any download delays from the data carousel to the receiver, and the actual data that is required is in fact minimal in comparison to any other schemes we have considered. In the following section therefore we also address the latency performance of the system, and present some quantifiable results to outline the trade-off in decoding latency versus channel download latency.

3.2 Experimental Setup

For analysing the performance of Deltacast, we utilised a random subset of a collection of ten thousand web pages available at [16]. The collection comprises multiple text/html pages which were crawled repeatedly every night for several weeks during Fall 2001. Pages were selected at random from two massive web crawls of hundreds of millions of pages and are thus a fairly reasonable random sample of the web. We consider such a data set to be representative of a content subscription based channel such as a news, sports or weather information stream which would likely be composed of textual information, images and meta-data such as HTML formatting tags.

For our experiments we divided collections of pages into *content channels*, each one comprising one hundred sites. In a commercial DataCasting environment, such channels might be tailored to specific types of information, e.g. sports, news, etc. We assumed that distributed across the user group is a wide range of versions of the file, some recent, and some non-existent. Whenever a user wants to sync its version to the source, it updates all the pages within a *content channel*. For each *content channel* we have four different instances in time: a base set and three updated versions crawled 2, 20, and 72 days later, respectively. Our experiments always considered the cost of updating an earlier version to one of the more recent versions.

We also measured the impact of different types of content such as source code distributions and binary files. We consider that these content types are representative of alternative applications such as geographical map information and software updates for mobile devices. We ran experiments for multiple content channels and averaged the performance over all content channels, the results of which are presented in this section.

To evaluate the performance of DeltaCast we will first study what is the impact of the number of hash levels on the overall performance of the system for different types of file. Too few hash levels may not provide enough granularity while too many hash levels may waste precious bandwidth to download unnecessarily detailed hashes. Next, we compare the performance of an optimised DeltaCast system with that of a flat hash system that only transmits one level of hashes. We then study the impact of using DeltaCast to concurrently update users with different file versions. Finally, we analyse the latency benefits of using DeltaCast to update users that join the data carousel at random times and determine the overhead of the decoding times imposed by using encoded data.

3.3 Number of Hierarchy Levels

We next evaluate the performance trade-offs of utilising multiple hash layers in DeltaCast. To this extent we compared the performance of DeltaCast for multiple hash levels when reconciling a content channel of 1.7 MBytes of source data incorporating 100 Web pages. We also measured the performance on a binary file of size 1.9 MBytes. The motivation being that as we will see next, the optimal number of hashing hierarchies depends upon the nature of the content.

We consider a DeltaCast system that uses a 6 Byte hash size, and a hash-hierarchy that uses a top-level block size of 1024 Bytes, and a bottom-level block size of 8 Bytes. Using such hash sizes we guarantee a probability of collision smaller than 2^{20}. (we did not observe any collisions on our data set due to our use of 7-byte hashes). Each level of the hierarchy has a block size that is half the size of the previous level. For each hash-level we use decomposable hashes. We note that although utilising blocks as small as 8 bytes on html content does provide improvements in bandwidth efficiency, the decoding overhead for such block sizes is very large making the use of such small block sizes infeasible. We therefore focus our attention more on 16 Byte blocks and higher.

Figure 3 illustrates the impact of the number of hierarchy levels on the amount of data required to reconcile a set of Web pages. For each combination of hierarchy levels we have illustrated the division of hash data and content data out of the total data downloaded. We can clearly see that in utilising a smaller number of hierarchy levels, the amount of data required is quite high since hashes are not able to efficiently identify file

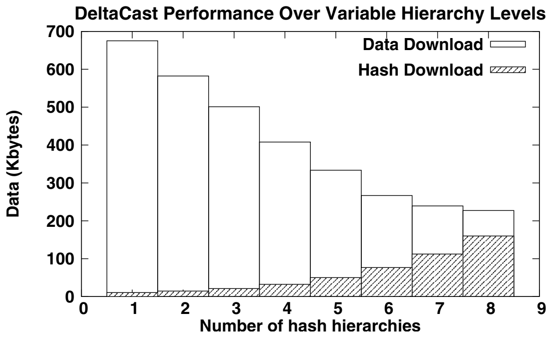

Figure 3: DeltaCast for different Hash Hierarchy Levels - Web content. Both the new and the outdated content approximates to 1.7 MBytes worth of data each. This figure demonstrates that utilising finer-grained hashing (i.e. a greater number of hierarchy levels) is particularly efficient in this context.

changes. As the number of hierarchy levels increases, the amount of hashes downloaded increases, but the overall total data required decreases significantly. The reason for this is that deeper hashes provide a finer level of granularity providing greater clarification of the exact file changes enabling a higher degree of redundancy detection between versions and consequently requiring a smaller amount of actual data downloaded.

From Figure 3 we can see that DeltaCast receivers can use up to eight hierarchy levels and still enjoy a reduction in the total data download. The reason for this is that changes in Web pages are typically on the order of several KBytes, thus, one can use smaller lower block sizes and still find duplicate data portions. Obviously, the number of hierarchy levels cannot go below the point where the block size equals the hash size since the overhead of downloading hashes would be prohibitive. In this example, for an eight-level hierarchy, DeltaCast receivers only need to download about 240 KBytes of data and hashes, which roughly corresponds to 11.4% of the file.

For a file that statistically incorporates changes in larger block sizes such as appending data to the start of a file, or adding a binary module to a software distribution, utilising smaller hash hierarchies will not provide any finer granularity in change detection thereby not reducing the amount of final data blocks downloaded. Applying unnecessary levels of hierarchy in this manner actually increases the bandwidth overhead. To better understand the variable impact of the number of hierarchies based on the content type, figure 4 illustrates the reconciliation of two different versions of a binary file, where the difference in the file lengths was on the order of 135 KBytes. From this Figure we can see that using too many hierarchy levels wastes precious bandwidth since hashes

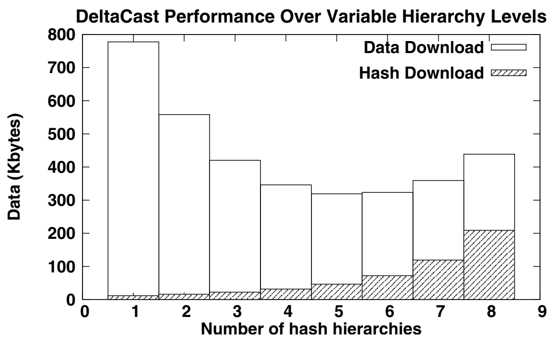

Figure 4: DeltaCast for different Hash Hierarchy Levels - Binary content. Both the new and the outdated content approximates to 2 MBytes worth of data each. This figure demonstrates that in this context (binary content with larger block size changes than HTML data) utilising finer -grained hashing (i.e. a greater number of hierarchy levels) does not provide any additional benefit below an original data block length of 64 Bytes.

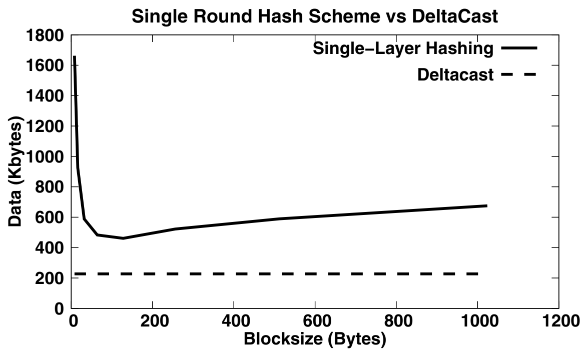

Figure 5: The Single-Layer hashing approach with Different Block Sizes.The figure illustrates the trade-off in hash accuracy versus amount of data downloaded. For comparison, we include the performance of deltacast, however since the algorithm uses hierarchical blocksize hashing in each round we illustrate a constant value for data downloaded.

are unnecessarily downloaded, providing no additional help in identifying missing portions. The optimal level of the hierarchy in this example is around five levels, which is quite different from the case of Web content. Ideally our system should attempt to make some intelligent decision about the optimal number of hashing hierarchies to apply.

To ensure that nodes do not download an unnecessary amount of hashes, DeltaCast servers could use a simple rule that limits the number of hierarchy levels depending on the content type, thus, allowing more levels for content that would be expected to incorporate finer changes, and less levels for content with less fine grained changes and lower intra-file redundancy such as with binary data. Another way to tackle this problem is to allow the DeltaCast receivers to *dynamically* choose the number of hierarchy levels that they want to use and halt the hierarchical detection process when needed. Receivers constantly monitor the new amount of changes detected with each new hash hierarchy. When this value goes below a certain threshold, receivers stop downloading further hash levels and begin downloading the missing data blocks. Such a scheme adapts well to different types of content but also to different types of users with multiple file versions. Each user with a different version will dynamically pick the right hierarchy level that best matches its level of changes. This receiver-driven adaptive approach also provides an efficient way to limit the overhead of deltacast when no redundancy exists between the local and the published content. For instance, in the worst case where no redundancy exists, the receiver would only download 2 levels of hashes in order to identify the lack of usable content and before switching to the simple file download case, which in the example in figure 3 would result in an overhead of just 0.8%.

3.4 Flat Hash System

In this section we compare DeltaCast with a naive hashing scheme that sends only one-level of hashes. For this flat hash system, we vary the block size to determine how an ideal system that could pre-determine the most efficient data block size for all receivers would perform compared to DeltaCast (note that this is an unrealistic scenario but useful for the purposes of a best-case comparison).

Figure 5 shows the total data downloaded to synchronise a collection of Web pages for both a single-layer hashing scheme and DeltaCast. For the single-layer scheme we vary the block size from 8 Bytes to 1024 Bytes. For DeltaCast we use an 7 level hierarchy with top-level blocks equal to 1024 Bytes and bottom-level blocks equal to 16 Bytes. For both schemes we use a hash of size 6 Bytes. From this Figure we can see that the single-layer scheme suffers from large overheads for very small block sizes since the amount of hashes downloaded is very high. Similarly, when the block size is very large, the number of hashes is significantly reduced but missing portions are only coarsely matched, thus resulting in very large portions of redundant data being downloaded. For this specific set of data, the optimal working point for a single-layer scheme is around 128 Bytes.

When comparing the optimal working point of a single-layer scheme with DeltaCast, we see that DeltaCast is much more efficient. The DeltaCast hierarchical system does not need to download deep-level hashes for already matched blocks and is still able to efficiently lo-

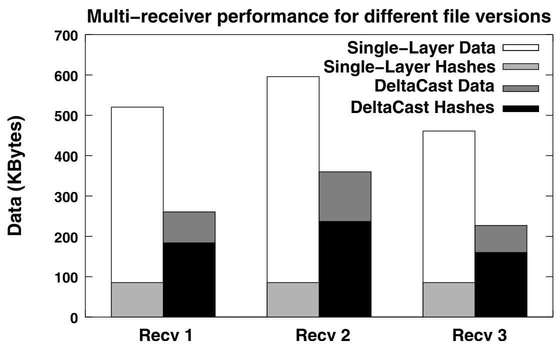

Figure 6: Performance Comparison for Multiple Receivers with Different Content

cate the high-level unmatched data portions. From Figure 5 we see that the performance improvement of DeltaCast compared to the optimal single-layer protocol is greater than a factor of 2.

3.5 Reconciling Different Versions

We now consider the case of three receiver groups with different versions of the same content, all of which are older than the current version being released at the source. We measure the performance of the schemes based on 4 versions of 100 web sites as outlined in section 3.2. Figure 6 illustrates the performance improvement provided by DeltaCast using seven layers compared to a single-round hashing scheme using an optimal block size of 128 Bytes.

All file versions are very similar, within 100 KBytes from largest to smallest. Thus, the overhead of sending hashes in the single-round protocol is very similar across file versions. The actual data downloaded for each single-round protocol experiment, however, ranged from 581KB to 450KB. In contrast, however the DeltaCast experiments on average required approximately half as much data, ranging from 351KB to 221KB. In general we would expect more recent files to have less differences and therefore require less data to be transmitted overall. This is clearly the case for version 3, but not so much for versions 1 and 2.

3.6 Latency Considerations

We next consider the benefits of DeltaCasting in terms of the amount of time that a user needs to reconcile its content. When hashes and data are sent unencoded, on average a user may need to wait for long *idle* periods before he can download a specific hash from the carousel. In the worse case, a user may need to wait for the full carousel to be repeated, thus, significantly increasing the average latency to update a file. With DeltaCast, on the other hand, *any* erasure hash can be used by *any* receiver to decode all the hashes on a given level. The same is true for the lowest level of the hierarchy, which carries the data payload packets. With DeltaCast, increasing the number of hash levels to provide better resolution does not significantly affect the overall latency since receivers do not have to wait long idle periods of time for the right hashes to arrive at each hierarchy level. This is not the case with a hierarchical hashing system that does not use erasure hashes. The higher the number of hierarchy levels, the longer a receiver will have to remain idle since idle times are almost certain to occur at each hierarchy level.

Assume a user is receiving data from a carousel that does not use encoded hashes. If the user needs to download m data units from a given level (a data unit can be either a hash, or a data block), and there are a total of N data units at that particular level, we can calculate the probability that a user receives the required m units by round i or before as $F(i) = \frac{\binom{N-m}{i-m}}{\binom{N}{i}}$. Based on this probability distribution, one can easily calculate the average latency before content is updated.

Next we quantify more precisely the latency benefits of using DeltaCast vs using a hierarchical hashing scheme with unencoded hashes and data. To this extent, we implemented an emulator where each hash hierarchy level is sent in a different carousel with the lowest carousel carrying the data payload. We consider four hierarchical levels. We assume that the total rate of the datacasting channel is 50 Kbit/s (which corresponds to the typical throughput of a DAB channel) and that this channel is evenly partitioned between the data payload and the hashing data. Moreover, the rate corresponding to the hashing data is evenly partitioned among all hash levels. We assume that clients join the carousel at random times.

In Figure 7 we show the average time that it takes for a user to update it's local version of Web content to reconcile with that of the server. We consider four different schemes (a) simple full file download with no hashes, (b) single-layer hashing, (c) hierarchical hashing scheme with no encoded data and (d) DeltaCast. The total latency represented in this Figure includes (i) the time to download the hashes or the erasure hashes, (ii) the time to download the missing data, (iii) the *idle* time waiting in the different carousels for the specific hashes and data to arrive, and (iv) the time to decode the encoded hashes and data. Note that not all latency factors are part of every scheme. For instance, the total latency in scheme (a) is determined by the latency factor (ii). The latency in

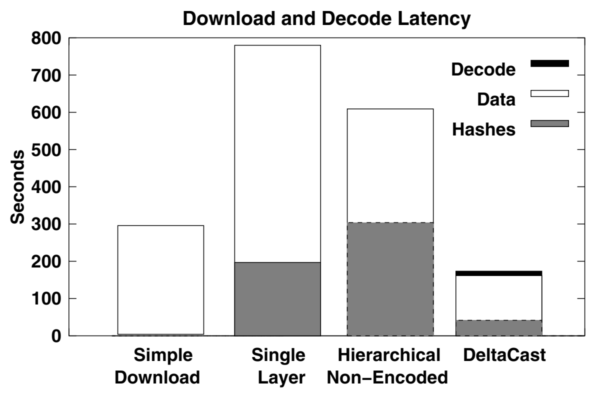

Figure 7: Average Latency for (a) simple full file download (uses all available channels), (b) optimal single-layer hashing algorithm (available bandwidth sub-divided into 2 channels between hash and data), (c) non-encoded hierarchical-hashing system and (d) DeltaCast (which also includes the time to decode the erasure data). The 2 hierarchical hashing schemes (c and d) in this case utilise 4 levels of hashing, and 1 data level.

schemes (b) and (c), however, is determined by factors (i), (ii), and (iii), while the latency in scheme (d) is determined by factors (i), (ii), and (iv).

From this Figure we can see that a hierarchical hashing scheme with no encoded data and a single-layer hashing scheme have an average latency that is 2-2.6 times higher than doing a full file download and more than 4 times higher than DeltaCast. The reason for this is that such schemes rely on finding the exact set of hashes and data to be downloaded from the carousel, which can result in very large idle times waiting for the right data to arrive. Observe, however, that while the latency is quite high, these schemes still provide significant bandwidth and power savings compared to a full file download since only a fraction of the time is used in downloading data.

The number of levels in the unencoded hierarchical scheme was set equal to four in this experiment (similar to DeltaCast). We have also tested the performance of such a hierarchical hashing scheme with a higher number of levels and note that the overall latency increases. This is due to the fact that the number of blocks increases as the hierarchy level expands, and thus, it becomes more difficult to find a specific set hashes.

When we compare the performance of DeltaCast to the other schemes we can see that DeltaCast provides significantly lower latencies. DeltaCast requires the same amount of data to be downloaded as a hierarchical scheme that uses non-encoded data. However, with DeltaCast, idle times practically do not exist since any hash or data block will be useful for any receiver as long as it is linearly independent with all the other blocks. A non-linearly independent packet can be generated and received, however since the encoded data field is selected to be very large, the probability of such an event occurring is extremely small.

One drawback of DeltaCast is that erasure hashes and erasure data blocks need to be decoded before they can be used. Decoding time is proportional to the amount of blocks in a given carousel and also proportional to the amount of blocks that need to be decoded. We assume that encoding times are negligible since data can be pre-encoded at the DataCasting head-end. Decoding times are more important since they increase the overall latency and consume resources at the receiver's device. We show the results of an implementation of a fast, software-based, sparse erasure decoder that we have developed. The tests were conducted on a Pentium IV workstation with 256 MBytes RAM which could provide slightly higher performance than a typical small mobile device, however we believe that the results indicate the benefits of utilising erasure coding far outweigh the arguments against. Other more popular Reed-Solomon decoders can also be used for the same purpose.

From Figure 7 we can see that the total latency time for DeltaCast is roughly 2 times lower than the latency of the best scheme. Even when we account for the total decoding time of hashes and data, DeltaCast still offers significant benefits since less data is downloaded and there are no idle waiting times. Decoding times increase as more levels are used since the total number of blocks at lower levels of the hierarchy is significantly higher. However, even for multiple levels of hierarchies, the total decoding time is only few seconds, thus, providing an almost negligible decoding overhead. DeltaCast, trades off latency and downloaded data for decoding time, however, as we have seen in this section the overall penalty is quite low.

3.7 Device Utilisation

Following the experiments from the previous section, we now identify the amount of time that the user's device is utilised, either to download hashes and data, or to decode the file. This time relates to the amount of battery power consumed at the receiver's device. We assume that during the idle times, no power is consumed. However, this assumption provides a lower estimate on the overall device utilisation for those schemes that have to wait long idle times (e.g. hierarchical hashing with no encoded data, and single-layer schemes). During such idle waiting times devices perform periodic attempts to check whether the right hash/data is being carouselled, which could consume non-negligible amounts of battery power. With DeltaCast, on the other hand, all required hashes and data are downloaded consecutively one after the other without random time gaps in between, thus, eliminating the need for periodic data checks.

In table 1 we show the device's activity time for all four schemes described before. The active time total and the average latency values for DeltaCast also include the decoder time. From this table we can see that DeltaCast utilises much fewer resources compared to a full download or a single-layer hashing scheme (approximately 1.7-2.1 times less), which translates into significant battery savings. Compared to a hierarchical hashing scheme with no encoded data, DeltaCast has a 7% higher device utilization due to the extra time to decode data (total of 12 seconds). However, from the same table we can see that by allowing data to be encoded, DeltaCast provides an overall latency that is 4.6 times lower, which clearly outweighs the decoding overhead.

Table 1 also summarises all the results presented up to now, averaged over a large number of Web content channels. Such results include the average latency to update the content, the amount of time that the user's device is utilised, the amount of data downloaded, and the decoding time for four different schemes. From this table we can clearly see that DeltaCast can very quickly reconcile files in a broadcast system while providing significant battery savings.

4 Related Work

Next, we give a brief summary of related work. The rsync protocol [11] is the basis of the very widely used rsync tool. In rsync, receivers use two sets of hashes (one weaker than the other) computed over a fixed block size throughout the file. Such hashes are sent to the source, which then does two passes to identify possible matches. It first compares the fast hash incrementally over the whole file (fixed blocks, utilising a sliding window). If the fast hash matches, then, it tests the more accurate hash. If the accurate hash also matches, then the source only needs to send a hash index to the receiver. The receiver rebuilds the file based on either the hash index or a new data segment sent by the source.

There are a number of theoretical studies of the file synchronisation problem [18], [19], [21], [20]. Within this framework, [22] discusses a relationship between Error Correcting Codes and file synchronisation. Orlitsky and Viswanathan [22] showed a relationship between Error Correcting Codes for noisy channels and file synchronisation that may be on some level related to the DeltaCast erasure-hash approach. A number of authors have studied problems related to identifying disk pages, files, or data records that have been changed or deleted [23], [25], [24].

Hash-based techniques similar to rsync have been explored by the OS and Systems community for purposes such as compression of network traffic [28], distributed file systems [29], distributed backup [30], and web caching [27]. These techniques use string fingerprinting techniques [26] to partition a data stream into blocks. However, several of these techniques usually require variable-size blocks which would complicate the design of a DeltaCast system based on erasure and decomposable hashes.

In [28] efficient erasure encoded data is used to accelerate the delivery of content in multicast-based systems that are prone to errors. We assume that data is reliably delivered to the receivers and focus instead on how to perform efficient file reconciliation. In this regard, [17] provides a framework to reconcile encoded files in peer-to-peer environments.

There has also been some related work that tries to minimise the overhead caused by periodically tuning in to the broadcast channel to find the appropriate data. As such, there are a number of indexing techniques that can be used to improve the performance. [34] [31][32][33]

The work that is closest to ours is [35]. In [35] an erasure-based algorithm is proposed to replace the need for multi-round communication protocols in client-server reconciliation protocols. By estimating the number of required edit operations on the file, and sending sufficient erasure hashes, the receiver is able to regenerate the correct set of hashes at each level. The server must send at least $2k$ erasures, where k is an upper bound on the edit distance. The server continues to send hashes for smaller block sizes until the hash length is equal to the block size, i.e. the original data. In reality this algorithm is not practical since it requires the server to know k accurately before hand. An alternative solution is to have the receiver send sufficient erasures to the source, which then detects how much of the file has changed and sends deltas of the file back to the client. However, both these approaches are better suited for client-server environments with a symmetric communication channel, which does not exist in most DataCasting systems.

5 Summary

We have considered the problem of efficiently reconciling two versions of a file in a broadcast system. Broadcast data systems are becoming very popular as a means for distributing information to a large number of receivers (e.g. news, maps, software, etc.). In a mobile environment where receivers connect at arbitrary times and have power constrained devices, designing a fast reconciliation protocol that minimises battery consumption is quite challenging. This is even more the case in broadcast systems where servers have no information about what type of content or what version resides at each receiver's device and often have to make blind decisions about what content to broadcast.

	DeltaCast			Hierarchical			Single Layer			Simple Download		
Avg. Latency	41.6	120.1	173.4	204.9	583.03	797.93	303.79	305.21	609	0	291.6	291.6
Active Time	41.6	120.1	173.4	41.6	120.1	161.7	303.79	62.9	366.69	0	291.6	291.6
Data Download	31.7	366.8	398.5	31.7	366.8	398.5	83.4	366.8	450.2	0	1780	1780
Decoding Time	0.4	11.6	12	0	0	0	0	0	0	0	0	0

Table 1: Performance Comparison. Time in seconds, Data in KBytes (hashes/data/total).

In this paper we have presented DeltaCast, a practical reconciliation system that combines hierarchical hashing with erasure codes and decomposable hashes. DeltaCast is able to simultaneously update an unlimited number of receivers holding a wide range of file versions. We have evaluated the performance of DeltaCast using a variety of content types, including Web content, and binary files and compared it with several other standard reconciliation techniques. Our results show that DeltaCast is very effective in quickly updating outdated content while conserving scarce power resources in mobile devices. We note also that the Deltacast system is not only useful within the radio broadcast environment, but the technique could also be applied to more general point-to-multipoint systems such as IP Multicast, Overlay networks, Peer-to-peer and Content Distribution Networks within the wired Internet environment. As future work, we plan on producing a stable version of this practical algorithm and use it in a real DataCasting environment to test its performance over a large scale deployment.

References

[1] Q. L. Hu, D. L. Lee, and W.-C. Lee. Performance evaluation of a wireless hierarchical data dissemination system., *In Proceedings of the 5th Annual ACM International Conference on Mobile Computing and Networking (MobiCom99)*, Seattle, WA, August 1999.

[2] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik. Broadcast disks: Data management for asymmetric communications environments., *In Proceedings of ACM SIGMOD Conference on Management of Data*, San Jose, CA, May 1995.

[3] T. Imielinski, S. Viswanathan, and B. R. Badrinath. Data on air Organization and access., *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, May/June 1997.

[4] StarBand., *http://www.starband.com/*

[5] Hughes Network Systems., *http://www.direcway.com/*

[6] Multimedia Broadcast/Multicast System (MBMS), *http://www.3gpp.org/ftp/Specs/html-info/29846.htm*

[7] Broadcast and Multicast Service in cdma2000 Wireless IP Network., *http://www.3gpp2.org/*, October 2003

[8] Digital Audio and Video Broadcasting systems, *http://www.etsi.org/*

[9] First UK user trial of multi-channel TV to mobile phones, *Nokia Press Release*, IBC Amsterdam, 2004.

[10] DirectBand Network. Microsoft Smart Personal Objects Technology (SPOT). *http://www.microsoft.com/resources/spot/.*

[11] A. Tridgell and P. MacKerras, The rsync algorithm, *Technical Report TR-CS-96-05*, Australian National University, June, 1996.

[12] N. Baric and B. Pfitzmann, Collision-free accumulators and fail stop signature schemes without trees, *Advances in Cryptology* EUROCRYPT 97, 1997.

[13] M. Bellare and D. Micciancio, A new paradigm for collision-free hashing: Incrementality at reduced cost, *Advances in Cryptology* EUROCRYPT 97, 1997.

[14] R. Johnson, D. Molnar, D. Song, and D. Wagner, Homomorphic signature schemes, Progress in Cryptology CT-RSA 2002, 2002.

[15] T. Schwarz, R. Bowdidge, and W. Burkhard, Low cost comparison of file copies, *Proc. of the 10th Int. Conf. on Distribute Computing Systems*, 1990, pp. 196202.

[16] Zdelta Home Page, *http://cis.poly.edu/zdelta/*

[17] J. Byers and J. Considine, Informed Content Delivery Acros Adaptive Overlay Networks, *Proc. of ACM SIGCOMM, Augus 2002.*

[18] G. Cormode, M. Paterson, S. Sahinalp, and U. Vishkin, Communication complexity of document exchange, *Proc. of the ACM SIAM Symp. on Discrete Algorithms, Jan. 2000.*

[19] G. Cormode, Sequence Distance Embeddings, *Ph.D. thesis*, University of Warwick, January 2003.

[20] A. Orlitsky, Interactive communication of balanced distribution and of correlated files, *SIAM Journal of Discrete Math,* vol. 6, nc 4, pp. 548564, 1993.

[21] A. Orlitsky, Worst-case interactive communication II: Two messages are not optimal, *IEEE Transactions on Information Theory* vol. 37, no. 4, pp. 9951005, July 1991.

[22] A. Orlitsky and K. Viswanathan, One-way communication anc error-correcting codes, *Proc. of the 2002 IEEE Int. Symp. on Information Theory*, June 2002, p. 394.

[23] Y. Minsky, A. Trachtenberg, and R. Zippel, Set reconciliatio with almost optimal communication complexity, *Technical Report TR2000-1813*, Cornell University, 2000.

[24] D. Starobinski, A. Trachtenberg, and S. Agarwal, Efficient PD synchronization, *IEEE Trans. on Mobile Computing*, 2003.

[25] S. Agarwal, D. Starobinski, and A. Trachtenberg, On the scalability of data synchronization protocols for PDAs and mobil devices, *IEEE Network Magazine*, special issue on Scalability i Communication Networks, July 2002.

[26] R. Karp and M. Rabin, Efficient randomized pattern-matchin algorithms, *IBM Journal of Research and Development*, vol. 31 no. 2, pp. 249260, 1987.

[27] S. Rhea, K. Liang, and E. Brewer, Value-based web caching *Proc. of the 12th Int. World Wide Web Conference*, May 2003.

[28] N. Spring and D. Wetherall, A protocol independent techniqu for eliminating redundant network traffic, *ACM SIGCOMM Conference*, 2000.

[29] A. Muthitacharoen, B. Chen, and D. Mazi‘eres, A low bandwidt network file system, *in Proc. of the 18th ACM Symp. on Operating Systems Principles*, October 2001, pp. 174187.

[30] L. Cox, C. Murray, and B. Noble, Pastiche: Making backup chea and easy, *in Proc. of the 5th Symp. on Operating System Desig and Implementation*, December 2002.

[31] M.-S. Chen, K.-L. Wu, and P. S. Yu. ”Optimizing index allocation for sequential data broadcasting in wireless mobile computing”. *IEEE Transactions on Knowledge and Data Engineering* (TKDE), 15(1):161173, January/February 2003.

[32] T. Imielinski, S. Viswanathan, and B. R. Badrinath. ”Data on air Organization and access”. *IEEE Transactions on Knowledge and Data Engineering* (TKDE), 9(3):353372, May/June 1997.

[33] N. Shivakumar and S. Venkatasubramanian. ”Energy-efficient indexing for information dissemination in wireless systems”. *ACM/Baltzer Journal of Mobile Networks and Applications* (MONET), 1(4):433446, December 1996.

[34] J. Xu and W. Lee and X. Tang, ”Exponential index: A parameterized distributed indexing scheme for data on air”, *In Proceedings of the 2nd ACM/USENIX International Conference on Mobile Systems, Applications, and Services (MobiSys'04)*, Boston, MA, June 2004.

[35] U. Irmak, S. Mihaylov, and Torsten Suel, ”Improved Single-Round Protocols for Remote File Synchronization”, *In Proceedings of IEEE INFOCOM 2005*, Miami, March, 2005

[36] John Byers, Michael Luby, Michael Mitzenmacher, and Ashutosh Rege, A digital fountain approach to reliable distribution of bulk data, *SIGCOMM*, 1998.