Lappeenranta University of Technology
Faculty of Technology Management
Degree Programme in Information Technology

Kimmo Kangas

COST OPTIMISATION OF MOBILE ADVERTISING CLIENT DATA TRANSFER

The topic was approved by the head of the degree programme on 15 August 2008.

Examiners: Professor Heikki Kälviäinen
Ahti Muhonen, M.Sc.

Supervisor: Ahti Muhonen, M.Sc.

TIIVISTELMÄ

Lappeenrannan teknillinen yliopisto
Teknistaloudellinen tiedekunta
Tietotekniikan koulutusohjelma

Kimmo Kangas

Langattoman mainosasiakasohjelman tiedonvälityksen kustannusoptimointi

Diplomityö
2009
74 sivua, 28 kuvaa, 10 taulukkoa ja 7 liitettä
Tarkastajat: Professori Heikki Kälviäinen
FM Ahti Muhonen
Hakusanat: kustannusoptimointi, mainonta, matkapuhelin, XML-optimointi, välimuistioptimointi
Keywords: cost optimisation, advertising, mobile phone, XML optimisation, cache usage optimisation

Langattoman mainosasiakasohjelman aiheuttama tiedonvälitys verkon yli saattaa kuulostaa epämiellyttävältä monen sovelluskehittäjän mielestä, jotka harkitsevat sovelluksen rahoittamista mainosrahalla, koska tiedonvälityksen aiheuttamat kustannukset saattavat pelottaa loppukäyttäjät pois sovelluksen käyttäjäkunnasta. Tässä diplomityössä rakennettiin simulaatioympäristö mallintamaan todellista asiakas-palvelin-ratkaisua, jotta voitiin mitata tiedonvälityksen määrä erilaisten yhteystyyppien yli. Tiedonvälityksen optimointiin kokeiltiin muutamaa XML-pakkaukseen erikoistunutta ja muutamaa yleiskäyttöistä pakkausmenetelmää. Myös protokollaa optimoitiin. Kustannusoptimointia silmälläpitäen välimuistin käyttöä optimoitiin ja mainosten etukäteen latausta paranneltiin käyttämään ilmaisia yhteyksiä tiedon lataamiseen. Välitetyn tiedon rakenne ja eri optimoinnit analysoitiin ja todettiin, että välimuistin käyttöä ja etukäteen latausta tulisi kehittää ja XML-protokollaa pitäisi muuttaa yhdistämään raportteja ja pakata joko käyttämällä WBXML:a tai gzip:iä.

ABSTRACT

Lappeenranta University of Technology
Faculty of Technology Management
Degree Programme in Information Technology

Kimmo Kangas

Cost Optimisation of Mobile Advertising Client Data Transfer

Master's thesis
2009
74 pages, 28 figures, 10 tables and 7 appendices
Examiners: Professor Heikki Kälviäinen
Ahti Muhonen, M.Sc.
Keywords: cost optimisation, advertising, mobile phone, XML optimisation, cache usage optimisation

Data traffic caused by mobile advertising client software when it is communicating with the network server can be a pain point for many application developers who are considering advertising-funded application distribution, since the cost of the data transfer might scare their users away from using the applications. For the thesis project, a simulation environment was built to mimic the real client-server solution for measuring the data transfer over varying types of connections with different usage scenarios.

For optimising data transfer, a few general-purpose compressors and XML-specific compressors were tried for compressing the XML data, and a few protocol optimisations were implemented. For optimising the cost, cache usage was improved and pre-loading was enhanced to use free connections to load the data. The data traffic structure and the various optimisations were analysed, and it was found that the cache usage and pre-loading should be enhanced and that the protocol should be changed, with report aggregation and compression using WBXML or gzip.

ACKNOWLEDGEMENTS

I would like to thank Nokia Oyj for giving me the opportunity to finally finish my studies by finding a subject that was interesting and challenging enough. I wish to offer special thanks to my supervisor, Ahti Muhonen, for giving me enough time and decision power to create the thesis independently under good guidance.

Also, I want to thank supervising professor Heikki Kälviäinen, who pushed me to meet the deadlines by being constantly interested in the work and its progress.

I would like also to apologise to all of my friends and my brother, whom I have dismissed recently while trying to finalise this thesis. Thanks for being patient!

Special thanks go to my girlfriend, who has supported me even when I have had very difficult times with the thesis, and also when I have been very difficult myself. Thanks, and my apologies!

Ja lopuksi haluan kiittää ja muistaa vanhempiani, jotka ovat kannustaneet ja tukeneet minua koko opiskelujeni ajan ala-asteelta tähän päivään asti. Ilman heitä tämä hetki ei olisi mahdollinen, kiitos!

Kimmo Kangas

TABLE OF CONTENTS

ABBREVIATIONS

2G	Second generation of telecommunication hardware standards
3.5G	Beyond third-generation telecommunication hardware standards
3G	Third generation of telecommunication hardware standards
API	Application programming interface
CPC	Cost per click, a business model where advertisers pay when users click on an advertisement
CPM	Cost per mille (cost per thousand impressions), a business model where advertisers pay when users see an advertisement
CTR	Click-through rate, percentage of clicks per shown advertisements
DTD	Document type definition
EXIWG	Efficient XML Interchange Working Group
GPRS	General Packet Radio Service, a packet-oriented mobile data service for 2G cellular systems
HSDPA	High-Speed Downlink Packet Access, a high speed 3.5G mobile data service
HTTP	Hypertext Transfer Protocol
MANET	Mobile ad hoc network, a network made by connecting the mobile devices nearby together
MCC	Mobile country code
MMA	Mobile Marketing Association
MMS	Multimedia Messaging Service, an extension to the SMS standard
MNC	Mobile network code
MSXML	Microsoft XML core services
PPM	Prediction by partial match
ROI	Return on investment
SAX	Simple API for XML
SIM	Subscriber Identity Module
SMS	Short Messaging Service
UI	User interface
USB	Universal Serial Bus

W3C	World Wide Web Consortium
WAP	Wireless Application Protocol
WBXML	WAP Binary XML
WCDMA	Wideband Code Division Access
WLAN	Wireless Local Area Network
XML	Extensible Markup Language
XSD	XML schema definition

1 INTRODUCTION

1.1. Background

The current trend in the Internet world is to provide services free of cost for the end user. However, since 'there is no such thing as a free lunch', the service providers have had to find alternative ways of monetising their business. Advertising has proved to be a functional solution, at least as judged by the number of free services available on the Internet and the revenue figures of Google, which is monetising its 22 billion dollar business almost solely by advertising (97% of Google's 2008 revenue came from advertising) [1].

Monetising one's business through advertising is based on a content or service publisher selling the audience to advertisers who in return hope to make deals with the consumers, or at least have their brand known to the larger public. The value in the advertising business comes through the number of times an advertisement has been shown (also known as number of impressions) or through the number of clicks an advertisement has received. The publisher value can be increased by specifying the publisher's target audience in great detail, so that the advertisers are willing to pay more for the advertising space in hopes of increased return on investment (ROI). [2]

Consumers' expectation of having free services on the Internet is becoming extended also towards services and applications targeted to mobile phones, and there even exists an operator (Blyk [3]) offering advertising-funded mobile subscriptions in some countries. The mobile phone environment is more attractive to advertisers than is the mass marketing on the Internet, because an advertisement on the small screen of a mobile phone has more impact, and mobile phones are more personal, making click-through rates (CTRs) higher since the advertisements can be targeted more accurately to individual users than is possible with Internet display advertising [2], [4].

However, advertising with rich graphics and interactive capabilities in the mobile phone environment is not that straightforward: the mobile phone environment today, even with high-end multimedia phones, is very different from the desktop environment. While execution performance, memory, battery life, storage space, and screen sizes are constantly increasing, the data connection speed and costs are still problematic in targeting of mass markets. [2], [4], [5], [6]

Current pricing models make the advertising data quite costly for the user, because typically data traffic is paid for by amount of transferred data and transferring a hundred kilobytes of advertising data can cost the end user half a euro [7]. Pricing models based on the time the connection is open are used also, but in those cases the need to open and close the connection for every time data will be transferred degrades the user experience. Also, since the charging in this model is based on rates per hour or fraction thereof, transferring small pieces of data every now and then will become expensive.

Even though third-generation mobile network (3G) access and flat-rate data contracts are slowly becoming more commonly available in developed areas' metropolises, they are not yet widely used globally and there are still many regions that only have second-generation (2G) networks [8] in which the data connection latencies are huge and radio bandwidth is limited, with first priority for calls [9]. The next-generation (3.5G) High-Speed Downlink Packet Access (HSDPA) networks are an attempt to overcome the latency problems of General Packet Radio Service (GPRS) by using shorter transmission time intervals and multiplexing the data from several users to the same transmission slots [10], but the capacity of a cell tower is still limited and has to be shared between the packet data and calls, and experiments have shown that sending of small payloads does not reap the full benefit of increased transmission speed [11], [12].

The slowness and cost combined with the fact that most of the time the phone is not even connected, to save on battery life, give advertising in the mobile phone environment a challenging playground. To overcome these problems and to enhance the user experience through shorter view loading times and increased responsiveness, a client-side program has been developed to handle some of the advertising logic and

advertisement caching in the mobile phone, helping to save many roundtrips to the network server.

While local caching of advertisements in mobile phones solves problems with bad user experience, there remains the need to transfer a lot of data between the client and the network server: graphics for the advertisements with screen resolutions constantly increasing, advertising metadata needed for targeting the advertisements, report data related to advertisement impressions and clicks, and profiling and context information for the targeting of advertisements.

1.2. Objectives and restrictions

The assumption is that end users are willing to accept the advertising if the targeting works properly and the user sees the advertising as a service, or receives free services or applications [2], but they do not want to pay the extra data costs caused by the advertising traffic. The purpose of this thesis project was to build a simulation environment for testing different usage scenarios in a client–server environment that behaves like the real-world system and has an unlimited number of advertising campaigns available server-side. The simulator is used for measuring data traffic over different connections and for investigating different ways of minimising the data transfer costs paid by the end user.

The assumptions for the simulated environment are as follows: 1) the data transfer over the mobile phone network costs the end user money, and 2) the user is within range of a free network (Wireless Local Area Network (WLAN), Universal Serial Bus (USB) cable, or Bluetooth) every now and then.

Cost optimisation is considered only from the end user's point of view with purely technical improvements in the pull delivery model over standard data connections. Thus, partnering with operators and other business model enhancements are not addressed in this thesis. Push, broadcast, and mobile ad hoc networks (MANETs) are not considered as a delivery mechanism [5], and neither is the use of the Multimedia

Messaging Service (MMS) or Short Messaging Service (SMS) as a data bearer, because there are serious privacy issues in association of the phone number with the anonymous profile data collected.

In investigation of the protocol data, only application-layer data optimisations [12] are considered. Dynamic advertisement content selection and optimisation based on network qualities are excluded. Because advertisers already provide fine-tuned compressed image data, changing the compression parameters to employ more lossy methods is out of the question.

The chosen methods will be evaluated mainly on the basis of the implementation effort required and the simulated percentage of cost savings, but also execution performance and battery consumption are considered, in case they are significantly compromised.

As a conclusion, suggestions will be made on how the system should be changed in order to optimise the cost, or what areas should be studied further.

1.3. Structure of the thesis

The thesis is structured such that related work and the definition of costs for mobile data transfer are described in Chapter 2 and the design and functionality of the current client–server system are described and analysed in Chapter 3. Chapter 4 covers the cost optimisations that can be made by manipulating the transferred data, and Chapter 5 investigates the optimisation and pre-loading algorithms for improving cache usage.

The simulator details and optimisations implemented are described in Chapter 6, and Chapter 7 presents the use cases and advertisement data for running the simulations. Detailed results for the simulations run with the simulator are found in Chapter 8, and summary conclusions and recommended actions and future study are presented in Chapter 9.

2 OPTIMISATION OF COST

2.1. Related work

There exist many studies related to mobile advertising [2] and different technical solutions for delivering the advertisements to the handset. Many papers [2], [5], [6] mention the cost and the limited connectivity as problem areas, and the common approach for addressing this is to utilise MANETs, broadcast, or push delivery [5], [13] over different bearers (SMS [14], Wireless Application Protocol (WAP) push [15], or Bluetooth push [16]) to transfer the main payload. The systems focusing on pull delivery [6], [17] transfer rich media advertising content and do not pay attention to the cost of the data transfer.

In the context of generic wireless computing, studies [17], [18], [19] have been completed for predicting future need and pre-loading the content to cache, but the focus in these is on allowing the applications to work in offline mode, or on being able to deliver the data to the user in a timely manner. In these systems, there is usually data content loaded that is newer used, which clearly is not a cost-optimised solution.

There are also studies presenting a system for utilising different bearers on the fly to save on battery life [20] or to speed up the communication [21], and much work currently centres on optimising the content delivered by sending only the necessary data [18] or by compressing the messages with both lossy [22] and lossless [23], [24] schema-aware and general-purpose algorithms [25].

2.2. Cost of the advertising data

The cost of the advertisement traffic for the end user is in direct correlation with the amount of data transferred. The data transfer can be charged by byte or by hour, but in any case less data transfer means less cost. In the current system, the only limit to the quantity of data transferred is a pre-defined constant value, which limits the data

transmission per session but does not give any other predictability to the cost. This means that for any given session, the cost of advertising data can be anywhere between zero and the maximum.

2.3. Optimisation alternatives

In addition to optimising the quantity of data transferred, the cost can be optimised by enhancing the cache usage and pre-loading. When an advertisement has been loaded to cache, it always should be used to minimise unnecessary data transfers, and when a fixed-rate data connection is available (WLAN, Bluetooth, or USB), it should be used to fill the cache with advertisements that are predicted to be needed before the next available fixed-rate connection.

3 THE CURRENT IMPLEMENTATION

3.1. System overview

Advertising-funded applications on the mobile phone are integrated with the advertising client to provide access to different advertising services provided by the advertising client. The applications use the advertising application programming interface (API) provided by the advertising client to fetch the advertisements from the network server or from the local cache and to return profiling data to the server. The advertising client is a middleware component that handles all advertising-related communication between mobile phone and network server. The advertising system follows the flexible client–server architecture described by Jing et al. [19], giving the advertising client the possibility of acting as a lightweight advertising server, but also leaving flexibility to forward the advertisement requests directly to the network server.

The advertising client fetches the advertisements from the network server and caches them for later serving to applications. This way, the responsiveness can be increased and the power consumption reduced. The main logical components and their dependencies are illustrated in Figure 1. This thesis focuses on the interactions between the advertising client and the advertising server components, which occurs via the Extensible Markup Language (XML) API.

The main responsibilities of the advertising client component are as follows:

- Serving advertisements to applications.
- Advertisement caching and fetching from the network server.
- Gathering profile information.
- Sending reports to the advertising server (on user actions and impressions).
- Executing actions.
- Performing targeting based on context (keyword, category, publication, placement, and time).

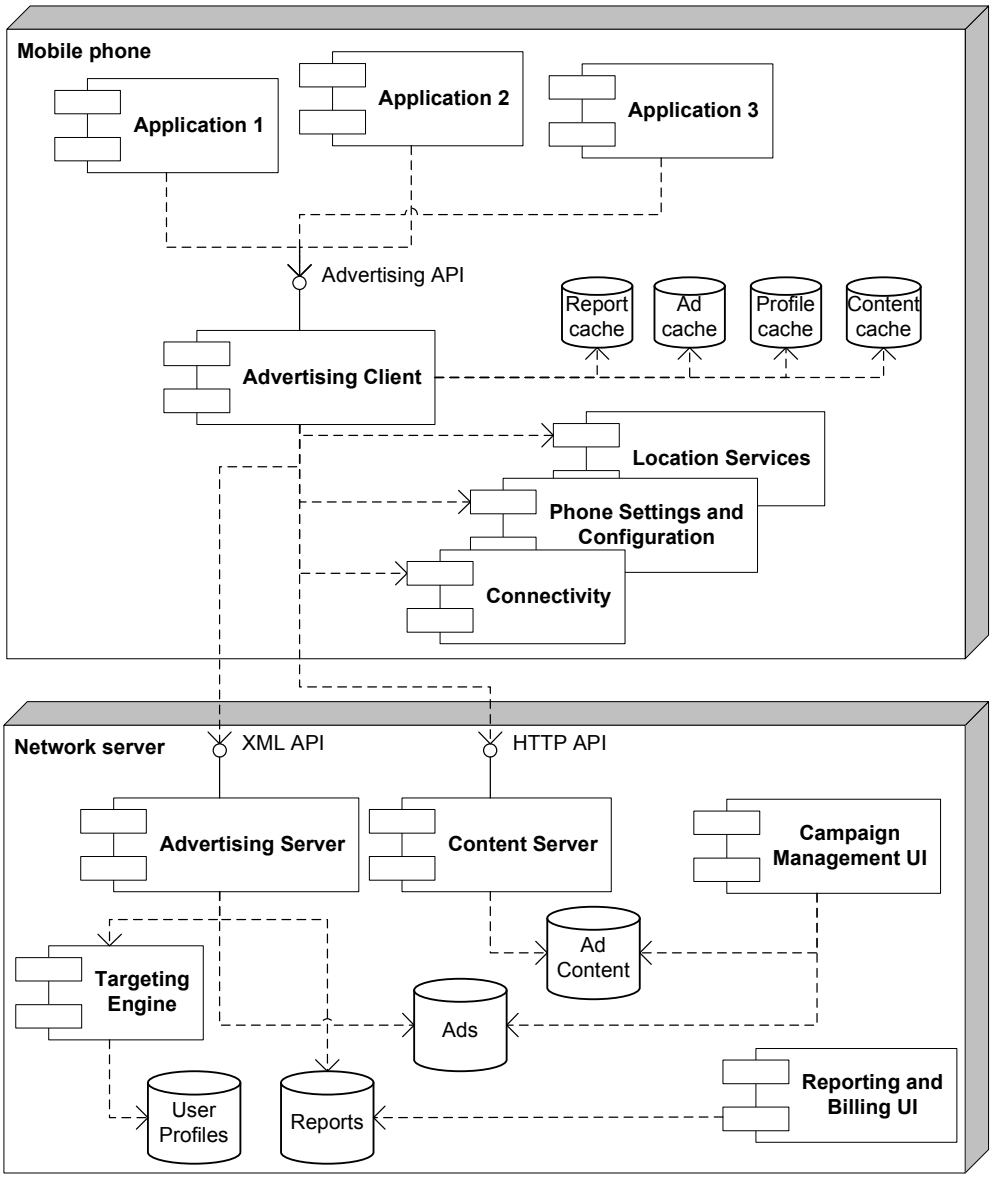

Figure 1. Overview of system components.

The main responsibilities of the network server subsystem are as follows:

- Serving of advertisements to the advertising client.
- Targeting based on the user profile.
- Advertisement campaign management.
- Reporting.
- Billing.

3.2. High-level message flow

When the application is connected to the network for updating its content or checking for updates, the advertising client is informed to pre-load a set of advertisements to cache in order to reduce the latencies in the application's usage later when the user is using the application for browsing the content or various views. When the advertising client is fetching advertisements from the network server, it also sends all the cached reports to the network server. The high-level message flow is presented in Figure 2.

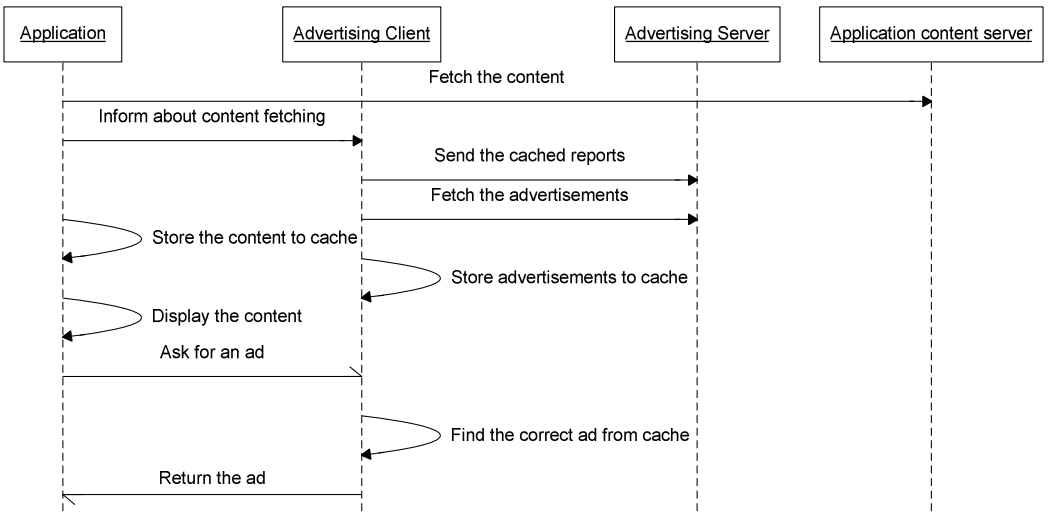

Figure 2. High-level message flow.

3.3. Advertisement targeting

Targeting of the advertisements is a major element in the advertising system; the better the system can target the advertisements to users, the better the return on investment [6]. While usage of the local cache increases responsiveness and improves battery life since data transmission is power-consumptive [20], the caching of advertisements also raises new kinds of problems, such as how to fetch correctly targeted advertisements in advance or how to enable real-time targeting with cached advertisements.

The advertisements can be targeted on the basis of the user profile, which includes information on elements such as the device, the home network, user behaviour on the phone (e.g., application usage, call logs, and browser logs), user data (e.g., contacts, messages, and notes), and demographics. The availability of the information depends on the user preferences, and the data will be analysed and summarised in the handset before being sent to the network server for use in targeting advertisements by user profile.

The context-information-based targeting that is also supported by the advertising client includes taking into account the user's location, availability (phone profile, calendar, presence status, etc.), and the current time and day, as well as advertisement context (where the advertisement will be viewed, also known as the advertisement spot). This should be taken into account in optimisation of the cache pre-loading and usage, by, for example, loading a certain number of generic advertisements instead of only targeted ones.

3.4. Transferred data

The transferred data consist of received advertisement content (e.g., banner images) and advertisement metadata (e.g., targeting, placement, and action information), reports sent (e.g., on session, impression, and action) and profiling information, and protocol overhead.

3.4.1. Advertisement metadata and content download

Downloading the advertisement content is the biggest contributor in the overall data transfer. The current flow of interactions when an application is to show an advertisement is shown in Figure 3. The optimisation point identified in this flow is the decision-making point between fetching the advertisement from server and from cache. The choice depends greatly on the contents of the cache and the incoming advertising request – i.e., does the cache contain an advertisement with suitable targeting parameters, or does one have to be fetched from the network? The optimisation of cache

usage is studied further in Chapter 5. Optimisation of the transferred data is covered before this, in Chapter 4.

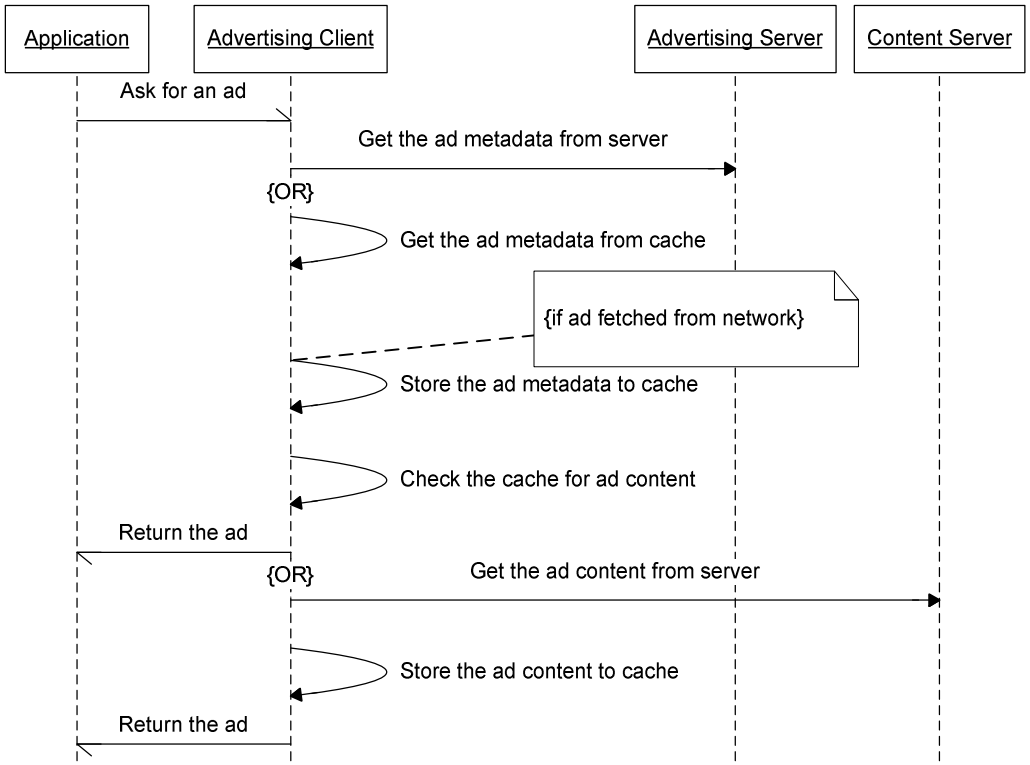

Figure 3. Message flow of advertisement fetching.

3.4.2. Report and profile data upload

The reports are generated on the basis of the advertisement displays (impressions) and user actions (clicks on the advertisement). When the application is started, a session report will be created automatically (see Figure 4), containing the application start time and the duration of the application session. The session information is needed for statistical purposes, since it can be combined with impression and actions reports to provide a complete picture of how well the advertising works for a specific application. This report represents a minor proportion of the total traffic and thus no optimisation points there.

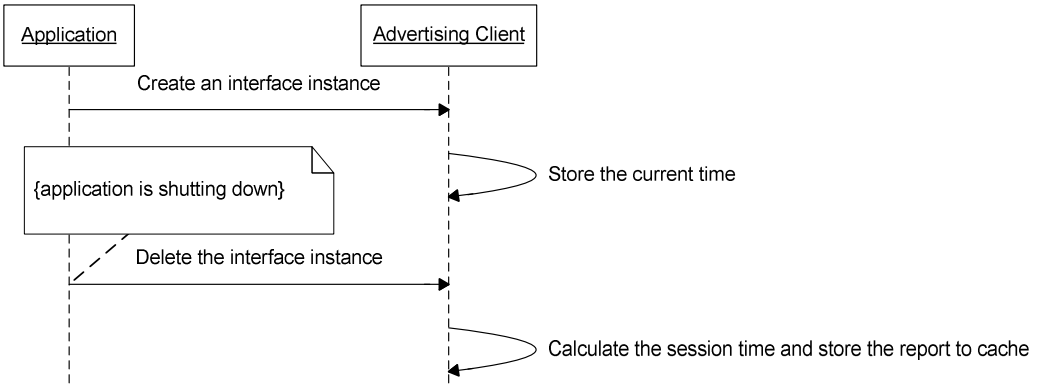

Figure 4. Message flow of session report creation.

When an advertisement is displayed, an impression report will be created (see Figure 5). The impression report contains the start time and the duration of the impression, which are used for generation of statistics only. Billing of advertisers is based on impression counts, which means that one optimisation point could be in aggregation of the reports, which is covered in Chapter 4.

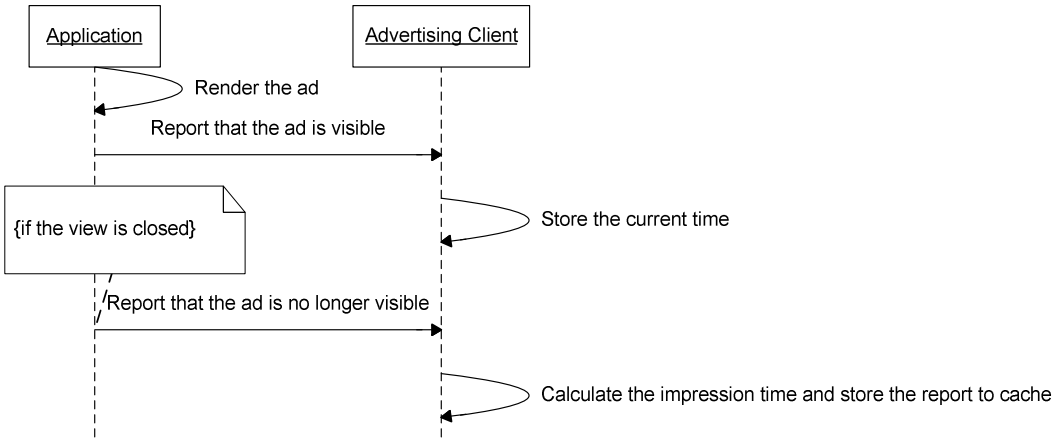

Figure 5. Message flow of impression report creation.

When an advertisement has been clicked, an action report will be created (see Figure 6). The action report contains the start time and duration of the selected action, which are also used only for statistical purposes. These could be optimised by aggregating the data, which is another potential optimisation further addressed in Chapter 4.

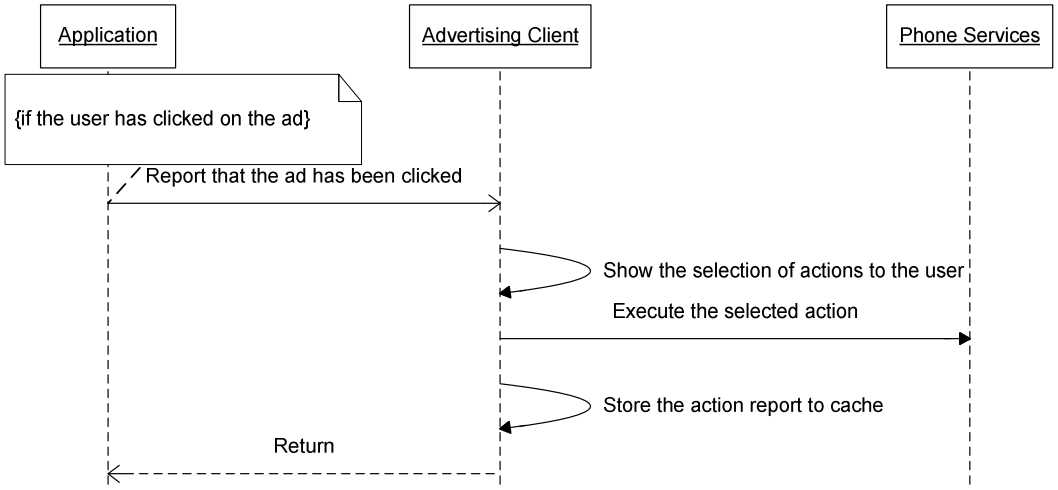

Figure 6. Message flow of action report creation.

The reports will be stored to cache and sent to the advertising server when the next advertisement request is sent (see Figure 7), or when there are old enough reports in the cache. The reports have to be sent to the server in one form or another because the advertising business runs on reports and thus, the only optimisation point here is for the protocol layer.

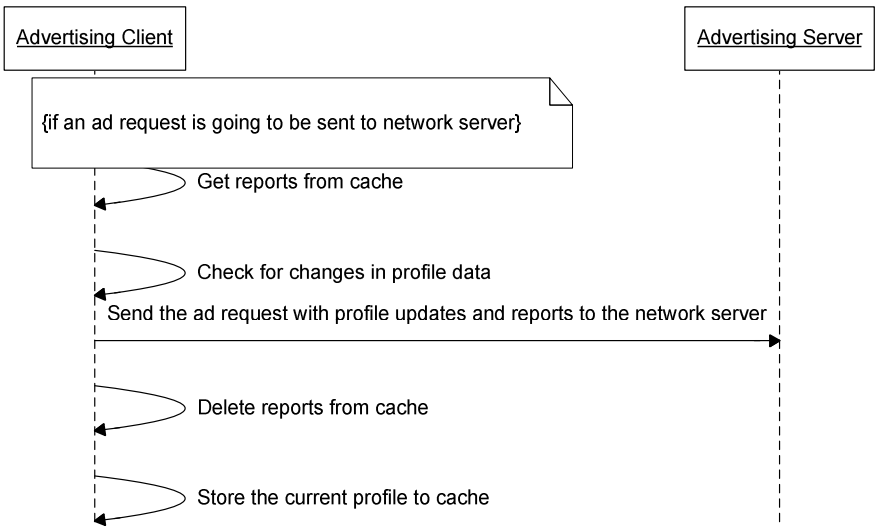

Figure 7. Message flow of report sending.

The user profile updates are also sent along with advertisement requests, and these are already analysed and summarised in the handset, leaving no optimisation points in this area.

3.4.3. Protocol overhead

The XML API offered by the advertising server is very verbose, and great savings could be made there. The advertising content is transferred as-is in compressed binary format (e.g., JPEG, PNG, or GIF), which is already optimised in the creation phase.

Experience has shown that the operator gateways may filter or block the transferred data, therefore the underlying protocols are selected to maximise compatibility with operator gateways and hence, the direct socket-level connections are not considered. The number of separate Hypertext Transfer Protocol (HTTP) request-response pairs is already optimised through combination of several client requests and reports in one HTTP request, and HTTP 1.1 pipelining and persistent connections [12] are utilised.

3.5. Cache usage

Currently the system loads new advertisements to cache every time an application refreshes its content. Because the number of advertisements used is small, the system works well, since the advertisement content is cached separately. However, as the number of advertisements in the system grows, or the density of content updates increases, it becomes apparent that this is not the optimal solution, since advertisements will be loaded to cache and may never even be shown to the user. The cache usage optimisation is covered in Chapter 5.

4 DATA OPTIMISATION

4.1. General-purpose data compression

General-purpose lossless data compressors are typically based on either dictionary usage or arithmetic estimation. Dictionary-based compressors are in common use, and their variants include such formats as ZIP [26], gzip [27], and bzip2 [28]. The compressors based on arithmetic estimation usually have large memory and execution time requirements but a better compression ratio [29]. The variants of these include prediction by partial match (PPM) [30] and the PAQ series [31].

The dictionary compression algorithms have their roots in the LZ77 algorithm [32], which works by finding duplicated strings in the data. Only the first occurrence of the string is stored as it is; the second one is only a pointer to the previous one, in the form of a distance-length pair. The scanning for duplicates is based on a sliding window, which means that for any given position, the algorithm has a record of the previous *n* characters that it can search for duplicates. After finding a duplicate, the algorithm may continue by checking whether a longer duplicate can be found by moving on to the next character, and it might even ignore the previous duplicate to achieve a better compression ratio. The different variants of the algorithm optimise the finding and storing of duplicate information in different ways and may apply some pre-processing to the data before scanning for duplicates, to increase the probability of duplicate strings.

The arithmetic compressors estimate the probability of a symbol by means of either a static or dynamic model. Static models can be based, for example, on historical data, or they can be generated before the compression, but when computing power is not a limitation, dynamic models can be used. In this case, the model is updated as the file is being compressed. Dynamic models are often used to predict the next symbol by assessing previous symbols (i.e., the context); the algorithms creating these models are also referred to as PPM-based methods. [30]

The arithmetic encoding algorithm encodes a stream of input symbols as a single decimal number. For each symbol, the model contains an allocated range of probability distribution, thus giving each symbol a unique range between 0 and 1. When encoding starts, the overall range is allocated to the first symbol's range and then narrowed by the second symbol's range, and so forth. For instance, if the model contains two symbols, 'a' with a probability of 0.9 and 'b' with a probability of 0.1, the ranges allocated would be 0.0–0.9 for 'a' and 0.9–1.0 for 'b'. Then, when encoding the sequence 'aaab', the algorithm would first make the range 0.0–0.9 the current range because of the first symbol being 'a', then allocate the same sub-range within the current range for the second symbol. The steps in the arithmetic encoding process are described in Table 1.

Table 1: Arithmetic encoding process.

Next symbol	Lower limit	Upper limit
	0	1
a	0	0.9
a	0	0.81
a	0	0.729
b	0.6561	0.729

After the encoding process, we have a range from 0.6561 to 0.729 and can pick, for example, the number 0.7 from that range, which represents uniquely the series 'aaab'. [33]

One arithmetic compression and two dictionary-based algorithms were chosen for compressing the XML data. The dictionary algorithms GNU zip and bzip2 were chosen because of their high performance and wide availability on different platforms, and the arithmetic compressor paq8p was chosen for its compression ratio. Summary of the methods used in selected compressors can be found in Appendix I.

Even though the latest arithmetic compressors can yield compression ratios of up to 24% for already compressed image files, all of the dictionary-based compressors deliver only a 0–1% ratio [34]. As the arithmetic compressors require more processing power, recompressing the image data was not considered a practical option.

4.1.1. The GNU zip algorithm

GNU zip is a widely used compression tool that implements the DEFLATE algorithm [35]. It first applies the LZ77 algorithm by scanning the data for duplicate strings and then stores the duplicate pointers in two separate Huffman trees [36], one containing the match lengths and the other containing the distances.

Huffman trees are used for storing the symbols by means of a variable-length code table, which applies the estimated probability of occurrence of each possible value in relation to the source symbol. The idea is to compress the data by using fewer bits for symbols that occur more often and more bits for those that occur infrequently. [36]

Because of relatively simple processing algorithm, the compression and decompression is fast [37].

4.1.2. The bzip2 algorithm

The bzip2 compressor implements the Burrows-Wheeler block-sorting text compression algorithm [38] together with Huffman coding to obtain considerably better results than are achieved with gzip, approaching the compression ratio of arithmetic compressors. [28], [29]

The Burrows-Wheeler block-sorting text compression algorithm applies a reversible transformation to a block of input text. The transformation does not compress the data but reorders similar symbols close to each other to make the content more compressible with simple compressors such as move-to-front coding. [38]

Move-to-front coding [39] takes advantage of similar symbols occurring frequently within short periods to create a variable-delta presentation of the data. Finally, bzip2 applies the Huffman coding for the data.

The Burrows-Wheeler transformation is time consuming, making the algorithm slower than gzip, especially when compressing data [37].

4.1.3. The paq8p algorithm

The PAQ series of compressors are arithmetic compressors with a large number of dynamic models mixed together. These models estimate the next bit by assessing the previous bits and the result of each prediction is arithmetically coded. The predictions are combined by weighted averaging and the weights are dynamically adjusted to favour the most accurate models to reduce future prediction errors (paq6). The difference from the prediction is then recorded for the decompression algorithm. [23]

In recent versions in the PAQ series, such as paq8p, the adaptive model weighting is replaced with neural network mixing of the different models. After combination of each predicted bit, the neural network is trained with the help of the correct bit. [31]

4.2. XML-specific data compression

Widely used in exchange of data between physically distributed or loosely coupled systems, XML uses schemas to standardise data exchange, but, being human-readable, it is too verbose for efficient transfer or processing in a limited-bandwidth network. To address this issue, the World Wide Web Consortium (W3C) formed the Efficient XML Interchange Working Group (EXIWG) to specify an XML binary format [40].

The XML schema can be derived implicitly from the XML document, or explicitly by a Document Type Definition (DTD) or XML Schema Definition (XSD) file. The file specifies the structure, element and attributes types, and the allowed values. With utilisation of external schema information in compression of the file, the element names do not have to be included in the compressed file, thus making the compressed files theoretically smaller. When only the implicit schema information is available, XML-aware compressors should be able to remove unnecessary whitespace and linefeeds and to compress the structure definition better than the generic-purpose compressors do.

The EXIWG work is still ongoing, but in the mobile phone environment there already exists a widely used binary format called WAP Binary XML (WBXML) [41], which was chosen for evaluation in the present project. In addition to the binary representation,

two XML compressors – XMill [42] and XMLPPM [43] – were chosen, for their good compression ratio with large XML files [25]. These compressors separate the XML structure information from the data and apply different general-purpose compressors to the two. Summary of the methods used in selected XML compressors can be also found in Appendix I.

4.2.1. The WAP Binary XML content format

When converting an XML file to WBXML, the algorithm enumerates all of the elements, attributes, and possible values from the XML schema and generates an integer value for each of these. After obtaining a unique number for each of the elements in XML, the algorithm just converts the textual XML tags to their binary equivalents. In addition to pre-defined names, the compressed file contains a string table that can be used to enumerate duplicate string values inside the XML document. To overcome the limitation of having to have control bits and element enumerations in one byte, the format supports different code pages for enumerated values. [41]

If the source XML document contains large element structures and smaller string values, the WBXML should be comparable to the best general-purpose compressors, but it has two qualities that make it worth using in computationally limited devices: it can be encoded and decoded in stream-level processing, and it makes the parsing more efficient since the parser can compare simple numbers instead of strings.

4.2.2. The XMill algorithm

An XML-specific data compression algorithm that separates the XML structure from the data, XMill is based on a grouping technique that groups and compresses values together on the basis of their element types. For example, where there is a sequence of multiple report elements in an XML document, each one containing spot, time, and duration information, the XML document could be rearranged by grouping all spots, times, and durations together. This usually yields better compression ratios, since each of these groups contain data items with great similarities. [42]

After separation of the structure and rearranging of the values, a general-purpose compression method is applied. This can be selected with a runtime parameter, and four

options were considered: no compression (‘-n’), gzip (‘-z’), bzip2 (default), and PPM-based compression (‘-P’). These options affect the execution speed in ascending order from the firstly mentioned to the last.

4.2.3. The XMLPPM algorithm

XMLPPM is an XML compression algorithm that combines the PPM algorithm for text compression and an approach to modelling tree-structured data called multiplexed hierarchical modelling [43].

XMLPPM takes a slightly different approach and speeds up the decoding and parsing of the compressed file by directly encoding the sequence of Simple API for XML (SAX) events from the XML parser when compressing the source document. It then maintains four separate models for the PPM compression algorithm: one for element and attribute names, one for element structure, one for attributes, and one for strings. Each model maintains its own state, but the arithmetic encoding is shared, allowing the encoding and decoding to proceed incrementally. [43]

4.3. Protocol optimisation

In addition to compressing content and converting it back to its original form when decompressing it, another option is to change the protocol to optimise the quantity of data transferred. As the XML sent consists mostly of report data and the XML received is largely description of the targeting rules that will be applied in loading advertisements from cache in offline mode, one option would be to aggregate and accept loss of accuracy in either of these to minimise the transfer costs.

4.3.1. Report data aggregation

Detailed, itemised reports consume a lot of space when transferred to the network, so by losing some accuracy and aggregating the data before sending we could save tremendous amounts in data costs. One report from the current implementation can be seen in Figure 8, where all the data values are highlighted and all the rest is just specifying the structure.

```
<ad-imp id="srv-53108" creative-id="srv-53109"
        country="210" offline="yes">
  <spot>
    <metadata>
      <image max-width="320" max-height="60" />
      <publisher id="nokia" publication="media" />
      <channel name="ringtones" />
      <placement>top</placement>
    </metadata>
  </spot>
  <start-time>20090128T015055+0200</start-time>
  <duration>255</duration>
</ad-imp>
```

Figure 8. Example of report data.

The data of multiple reports could be optimised by grouping the report details within common advertisement spot data (advertisement context) since there usually are many fewer advertisement spots than reports, but, since the detailed information on each impression is not even used at the moment, the optimisation could go even further by aggregating the reports through counting only the number of each type of report. This means that, instead of each report being sent individually, only the number of impressions and actions for a particular advertisement in a particular spot would be sent. The result can be seen in Figure 9.

```
<spot publisher="nokia" publication="media"
      category="ringtones" placement="top">
  <ad id="srv-53108">
    <ad-imp id="srv-53109">6</ad-imp>
    <action-click id="srv-55798">1</action-click>
  </ad>
</spot>
```

Figure 9. Example of aggregated report data.

Within less space than it took to send just one impression report, it is possible to send several impression and action reports, with the disadvantage of losing timestamps and durations of individual reports. However, since the billing is based on reports, the benefit of aggregating the report data depends on the duration of the offline period (the time for which the advertising client is not communicating with the network server) and on the amount of delay that is acceptable in returning the reports to the server.

4.3.2. Reporting on only unused impressions

With loss of more details from the reports and shifting of the paradigm toward a more cost-friendly solution, traffic could be optimised even further by reporting only unused impressions and actions. This would result in a logic that would request an advertisement for a certain spot and then attempt to apply all of the reserved impressions in whatever spot the advertisement may be shown in. If this should fail, the client would report the number of unused impressions to the server so that the server would know to free the impressions for some other client. This would entail the server being unable to respond anymore to the client's impression report by indicating that the advertisement is no longer valid, and the information on the spots in which the advertisements were actually shown would be lost. Also, the publisher value would decrease, since the information about which applications generate the audience would be lost. The resulting impression reports would be reduced to quite simple one-line elements as shown in Figure 10, though the action reports would remain the same.

```
<ad-deleted id="srv-53108" unused="8" />
```

Figure 10. Example of a new impression report.

4.3.3. Removal of offline targeting capabilities

After reduction of the data sent to the minimum, the optimisation could go still further by removing details from the received data. In the current implementation, most of the XML received describes the targeting rules used to determine when and where the advertisement can be shown. If the logic for serving advertisements from the cache in offline mode would be changed to attach the parameters from advertisement fetching to the advertisements received, it would not be necessary to receive these over the network connection. However, this would reduce the versatility of the cached advertisements, because the offline algorithm would not have knowledge of where the advertisements are really allowed to be shown; with this reduced flexibility, the only possibility allowed would be one-to-one mapping between cached advertisements and spots. In view of overall data optimisation considerations, this could not bring very good results, since the same advertisement cannot be shared between different applications, but the positive

side would be that the server has full control over the serving of advertisements and in some cases costs could be saved with the current implementation.

4.3.4. Other methods

There are numerous ways of optimising the XML, such as flattening the structure by increasing the usage of attributes, removing unnecessary containers, and using string formatting instead of XML elements [44], but those are not covered here, since they would require a complete redesign of the protocol and new implementation on both the client and the server.

5 OPTIMISATION OF CACHE USAGE

5.1. Advertisement caching

In the current implementation, the advertisement caching logic is quite simple and the effective sharing of the cached advertisements can be very limited since the campaigns are sold for specific applications at a specific time for a specific number of impressions (cost per mille (CPM) business model). However, in the future, with movement toward user targeting and performance-based selling of advertisements (advertisers paying by the number of clicks, CPC business model), caching will be utilised more and more effectively as advertisements are targeted more to the users instead of for applications and advertisement spots. There are three aspects to consider in improving the cache usage:

1. If an advertisement has been loaded, take the most out of it and use all of the impressions reserved for it every time before removing it (or before it expires).
2. Use free connections to pre-load the advertisements to cache that will be most likely to be needed in future.
3. Advertisements that are loaded to cache should be reusable, not very specifically targeted (location is also relevant later), and not disposable.

The first element is limited by how well the cached advertisement parameters match the request parameters. The second is limited by the data connection parameters for getting the advertisements into the cache, and the third is limited by cache size and the needs of the advertisers. The exact parameters limiting cache usage are described in Figure 11. From these parameters, the advertisement validity, report time and data connection time, duration, data limit and speed, are not considered in this thesis. The advertisement request and advertising client parameters are included.

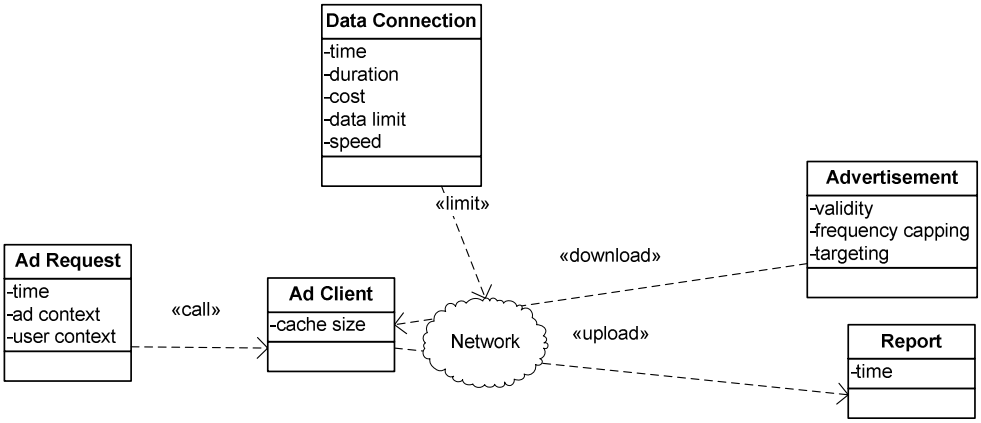

Figure 11. Data actors in the system.

5.2. Adjusting pre-loading according to cache content

The current implementation loads new advertisements to the cache every time the application refreshes its content from the network. The number of pre-loaded advertisements is based on estimated user behaviour, which in practice means that one advertisement is loaded for each downloaded item (news story, catalogue page, e-mail message, video, song, etc.) that the user might view.

This works well in the current environment, where the number of active advertising campaigns for any given application is small, because the server cannot return guaranteed different advertisements for each of these requested items; instead, it might return just one generic advertisement, which is then used from cache in all of the views. However, when the number of advertisements in the system grows, different advertisements could be targeted for each of the items separately, thus creating the possibility of loaded advertisements not being shown even once if the user does not view the item.

This is not an optimal solution for the future, when the advertisements will be more targeted and there are plenty of them in the system. It can lead to situations wherein the cache already contains proper advertisements for the application's needs. By changing the pre-loading logic to first scan through the cache contents to calculate how many

applicable advertisements exist already and then fill in the blanks from the network server for the estimated required number of advertisements, it should be possible to achieve considerable cost savings. Usually advertisements that are loaded can be shown to the user more than once, so also the client's advertisement serving algorithm should be changed to use all of the cached impressions before connecting to the network for more advertisements.

5.3. Use of free connections

A mobile data connection over cellular networks is not the only way to get the advertisement data to phones. Many newer phone models can access Wireless LAN networks, and many users are also connecting their phones to desktop computers via Bluetooth, USB, or infrared connection to transfer data. When these connections are used, it can be assumed that moving the advertisement data does not add to the cost. It is possible that in some rare cases the WLAN connection is charged for by the byte, but those cases are not considered in this work.

When the free connection is available, it can be detected and used for downloading more advertisements to cache from the network server. The decision on which advertisements to pre-load could be made intelligently by predicting the future need according to historical data – for example, via some of the PPM methods [30] – but, since free connections are not that common in targeting to mass markets (especially in developing countries), those methods were not studied any further.

The simplest way of utilising the free connection is to change the advertisement serving algorithm to obtain fresh advertisements from the network whenever such a connection is detected. This way, the cache can be filled with newer advertisements to increase the probability of being able to use these when only other connections are available.

To further improve the algorithm, a parameter can be added for multiplying the number of advertisements fetched from the server. Since the server always returns a certain percentage of generic, less targeted advertisements, the number of these returned is

greater when the number of advertisements loaded grows; the server also returns more generic advertisements in addition to the most targeted ones. This should decrease the amount of data that must be transferred when free connections are not available, but the cache size might limit the benefit, since the algorithm might end up overwriting existing generic advertisements from the cache with ones targeted for the relevant application to keep the cache size under the limit. Changing the client-side algorithm to keep a certain level of generic advertisements in the cache was not studied. The effect of parameter value on algorithm functionality and cache state is described in Table 2.

Table 2: The functionality of the cache algorithm optimisation parameter.

Parameter value	Algorithm functionality	Cache state after pre-loading (when a free connection is available)
0	Use cache when possible	Cache contains advertisements for estimated need.
1	Use network when free connection is available	Cache contains more advertisements for the application than the estimated number needed. A certain proportion of the advertisements returned are general and can served also to other applications.
2	Use network when free connection is available to load twice the estimated number of advertisements needed	Cache contains more general advertisements since the server always returns a certain percentage of these.
3	Use network when free connection is available to load three times the estimated number of advertisements needed	Cache contains even more general advertisements to be shared between all applications.

5.4. Cache size

The available cache space is the biggest limitation in pre-loading of advertisements. The basic rule for the cache is that the advertisements will remain there until they expire or their impressions run out. However, the cache size has to be limited, since the phone environment usually has very limited storage capability and no single piece of software can use all the available space for its own purposes. This leads to situations in which the advertisements that are shown the most have to be deleted before expiry in favour of pre-loading of advertisements for other applications. The probability of this happening increases when free connections are used to pre-load the advertisements to cache.

6 SIMULATOR DESIGN

6.1. Overview

Because the current system requires a complex set-up of many components on the server side and on the client side, a simulator was built in order to measure the transferred data quantities with ease in the specified usage scenarios. The simulator was built first to mimic the real environment for studying the implementation of the existing client–server advertising solution, and then enhanced for testing different methods of optimising the costs in the simulated usage scenarios.

The easy-to-use dynamic object-oriented programming language Python [45] was chosen to speed up the development and to allow quick testing of different optimisations. The building of the simulator also allowed running of the same use cases over and over again in an environment that has an unlimited number of advertisements available and where the advertisement content and the use cases can be adjusted precisely.

The components in the system are illustrated in Figure 12. ‘Test App’ contains all of the application logic and runs the simulations on the basis of the ‘Use Case Data’, ‘Ad Client’ simulates the client-side software, ‘Ad Server’ is a server simulator, ‘Ad Engine’ contains the advertisement storage and searching logic, the ‘Data Models’ component contains definitions for different data structures used by all other components, and ‘Ad Data’ contains all of the advertisements used in testing.

The comtypes Python library [46] is used for accessing Microsoft XML Core Services (MSXML) [47], which is needed for validating the generated XML against the protocol schema. This makes the code Windows-dependent, but this can be easily replaced with some other validation code (e.g., using libxml2 [48]) if support for other operating systems should become necessary.

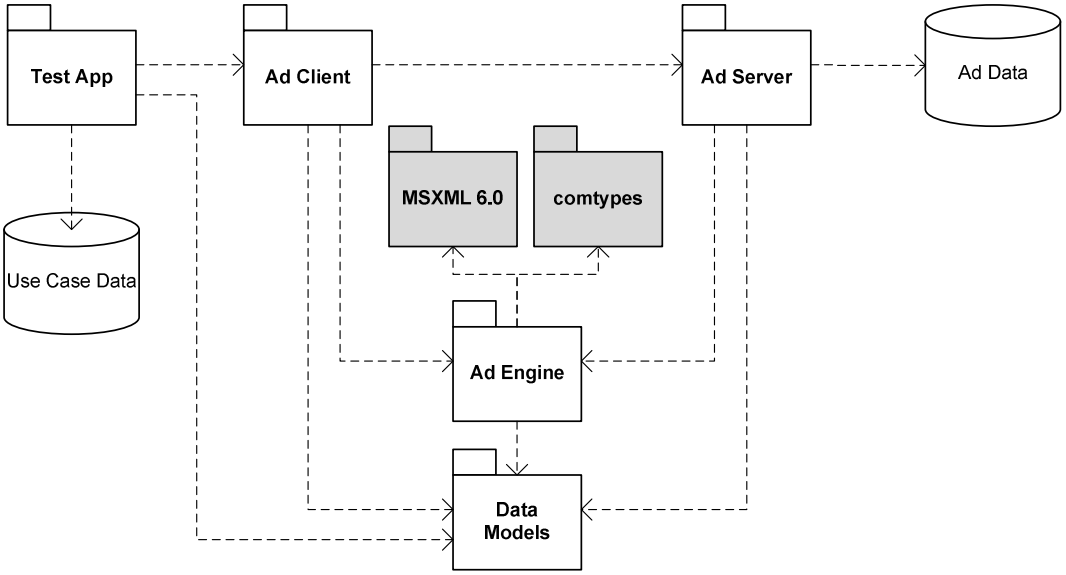

Figure 12. Simulator components.

6.2. Class structure

Class dependencies and interface functions are shown in Figure 13. The application is modelled by means of the Model-View-Controller [49] design pattern, where the Test class acts as the controller driving the simulated advertisement requests toward the model, which is the client class. The user interface (UI), acting as the view, handles all viewing and formatting of the results.

On high abstraction level, the functionality of the simulator is directly analogous to the client-server environment. The most notable differences are the sharing of single advertising engine component and absence of all the server side logic beyond the advertisement loading and serving.

All the classes use common data structure class definitions from ‘Data Models’ component, providing efficient and clean implementation for the functionality related to processing the data elements.

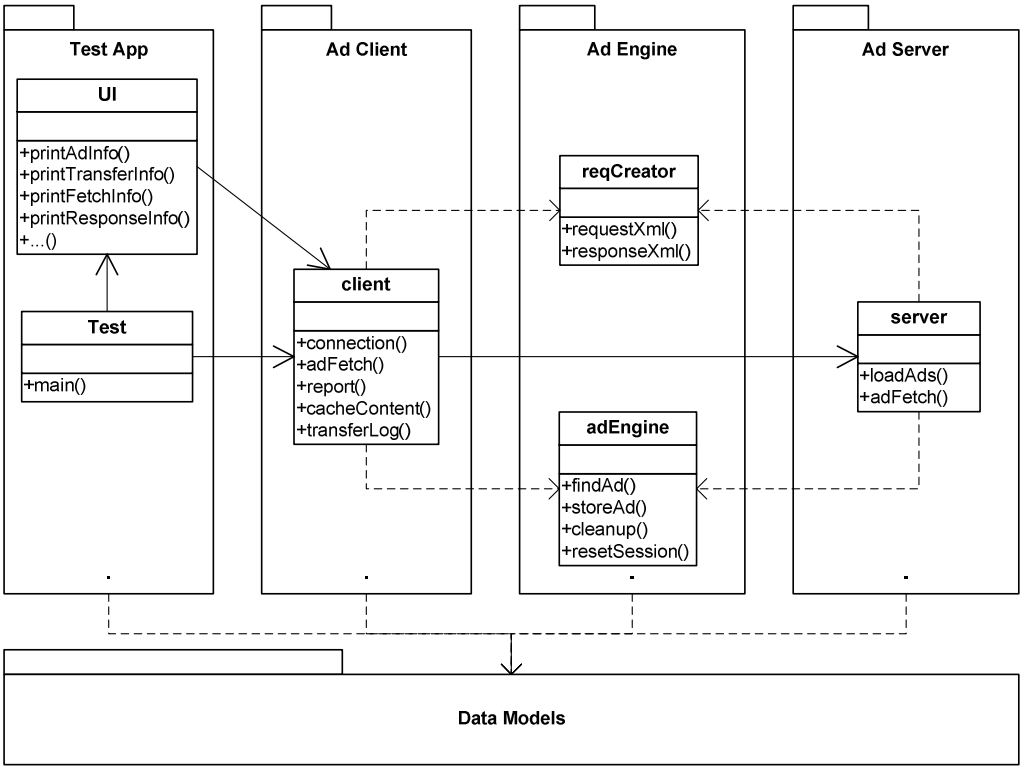

Figure 13. Simulator class structure.

6.3. Data models

Data model classes were created for wrapping parameters that specify the user's and advertisement spot's context information, advertisement and report data, and connection type and speed. All parameters supported by the simulator are described in Figure 14.

The user context is specified by device information, current network and Subscriber Identity Module (SIM) card parameters, such as the Mobile Country Code (MCC), Mobile Network Code (MNC) and cell tower identification, and demographics. The advertisement context is specified by spot parameters and the advertisement data contains targeting parameters and information for click-to action. Each report has type and time information and the connection is specified by type and speed.

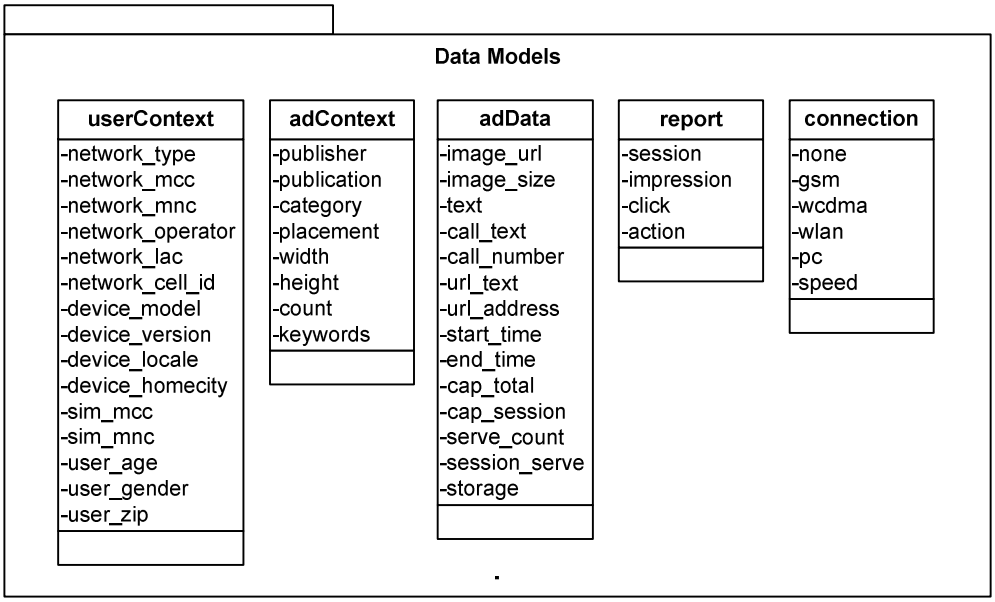

Figure 14. Simulator data models.

6.4. Measurements

For measuring the simulation results, considerable statistical information gathering logic was built into the simulator. Advertisement serving counts are monitored on the server and at cache level; detailed information on data transfer over different connection types, broken down by data category, is collected; cache usage statistics are updated; advertisement requests, reports, and actions are recorded in the online and offline cases; and all of this is broken down further by usage category. Also, for data compression, all of the various compression results are recorded, so each simulation run results in a lot of numbers and many request/response files that can be analysed in detail for assessment of the optimisation results.

6.5. Cost optimisations

In implementation of the different optimisations for the simulator, performance was not considered, and all of the optimisations were controlled by function parameters, making it easy to run the same simulations with different optimisation combinations enabled.

6.5.1. Data compression

Before application of the various compression algorithms, all whitespace was removed after generation of the XML data. Python aided in testing of the compression algorithms, by providing built-in implementation for gzip and bzip2, so applying these for the XML data was straightforward.

For the rest of the compression algorithms, an Open Source project was taken (XMLPPM [50], XMill [51] and libwbxml [52]) and the tool was compiled from the source code. This executable was then called from Python, resulting in a sub-optimal sequence: generate raw XML, remove whitespace, write result to file, validate file against protocol schema, and call external compression algorithm to compress the file.

To obtain the best results with WBXML, a list of XML tags, attributes, attribute values, and commonly used strings had to be extracted from the custom protocol schema. For easier extraction, the schema was converted to DTD with the free XML editor XMLPad [53]. After reading of the DTD and parsing of all elements, attributes, and attribute values, a few known strings were added manually to the table in order to make the WBXML more efficient. These tables were then included in the source code for the WBXML encoder and decoder, and the custom versions were compiled.

6.5.2. Protocol optimisation

All of the protocol optimisations were implemented directly in the XML generation phase, and a new version of the protocol schema was created for verifying that the generated requests and responses match the optimisation idea, and that all the required data would be transferred. In addition to automatic verification, the generated traffic was also inspected manually to verify the logic.

Aggregation of report data was done only while the device was not connected, in order to maintain the business logic. Adjusting the length of the aggregation period could bring great savings, but that was not tested here. The removal of targeting data coming in with the response was handled by just commenting out the function that writes that bit of XML data in the server response. Also, the client side had to be changed to

associate the incoming advertisements with the outgoing parameters, so that the cache algorithm could target the advertisements to the correct spots.

6.5.3. Cache usage optimisation

The cache usage optimisations were implemented directly in the offline advertisement serving and pre-loading algorithms. The optimisation level and cache size were controlled by method parameters, thus providing the possibility of creating nested loops for getting a matrix of the results. Using the cache optimisations causes longer offline periods, so the report aggregation should produce better results, although the business impact of this was not considered.

6.6. Simulation sequence

The specified usage scenarios are loaded from a database run by the Test class. The implementation initialises the random seed to one to guarantee similar simulation runs. Random numbers are used for generating the identifiers in the XML and for creating variety for the advertisement requests specified by the use cases. Also, different instances of the random generator were used for XML generation and for running the simulations, in order to get exactly the same runs each time.

For processing a use case, current connection type must be first defined and set to the client. Next, the advertisement fetch must be executed with user and advertisement context, and time, defined by the use case. After the advertising client has completed the processing of the advertisement request (i.e. searched local cache, requested advertisements from server simulator, checked targeting parameters, stored new advertisements to cache, updated serve counts and displayed statistical data), the Test class creates the reports from simulated impressions and user actions with request timestamps and random durations. These reports will be stored within the local cache for sending to the server along with the next advertisement fetch. Finally, statistical information is fed to the UI. The high level execution sequence for an advertisement session is illustrated in Figure 15.

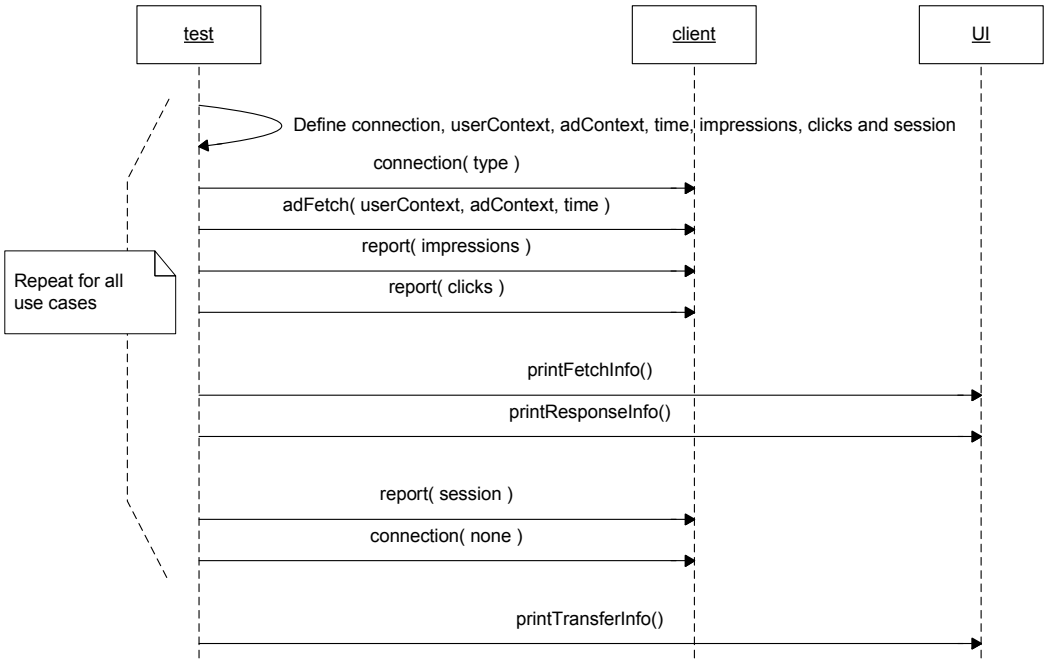

Figure 15. Use case call flow.

6.6.1. Fetching advertisements from cache

When the client does not have a connection, or there are valid advertisements in the cache, the call sequence is straightforward, as seen in Figure 16.

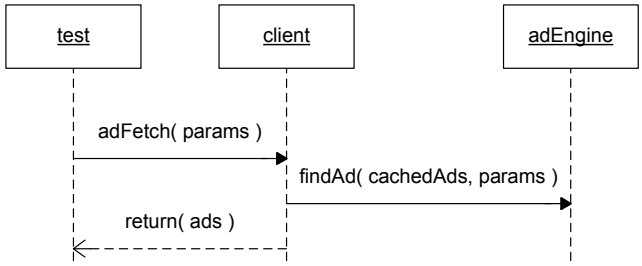

Figure 16. Fetching of advertisements from cache.

6.6.2. Fetching advertisements from the server

When an advertisement needs to be fetched from the server, a few more things must be done in order to mimic the real environment and to get the correct request and response XML created. This sequence is depicted in Figure 17. In addition to the cache search case, the server component must be involved and the advertisement must be stored into cache after it has been received from the server.

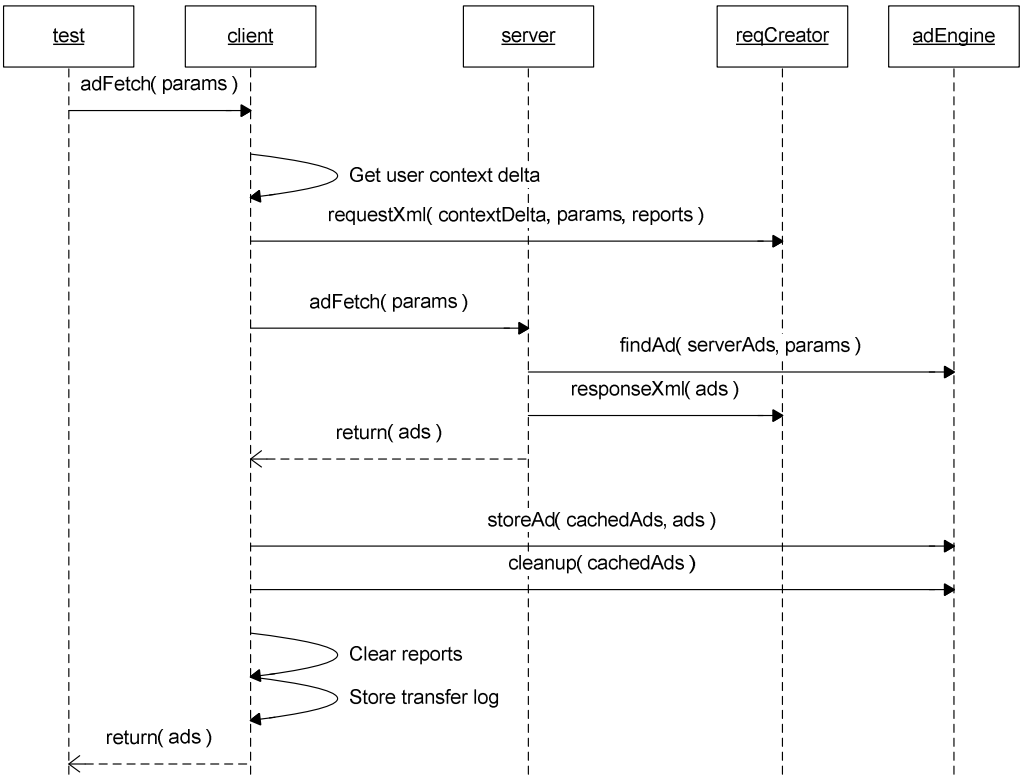

Figure 17. Fetching advertisements from server.

6.6.3. Verification of the functionality

For validating the correct functionality of the simulator, two approaches were used: 1) manually checking the input and output parameters of an advertisement fetch and the request and response data generated and 2) validating the generated XML against the protocol XML schema definitions. Figure 18 shows the call flow for displaying the statistics in the UI.

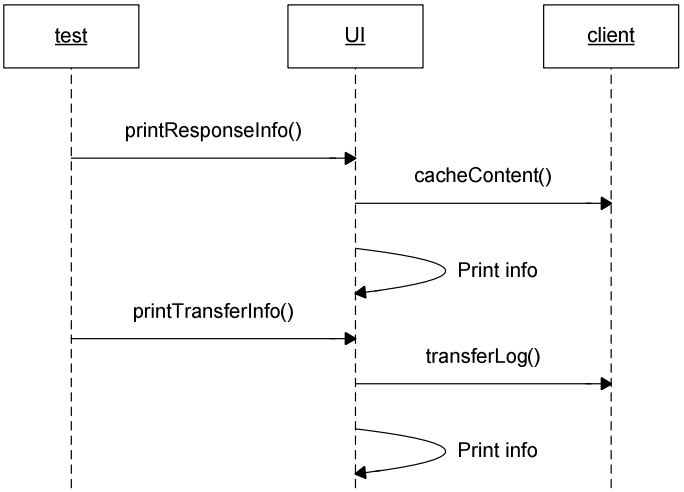

Figure 18. UI call flow.

The XML validation was done by calling the MSXML Windows COM API through the comtypes Python library, and the XML content was passed via the file system. The call flow is described in Figure 19. In validation of the correctness of the data compression methods' functionality, all files were extracted back to their original form and the result was compared against the original XML.

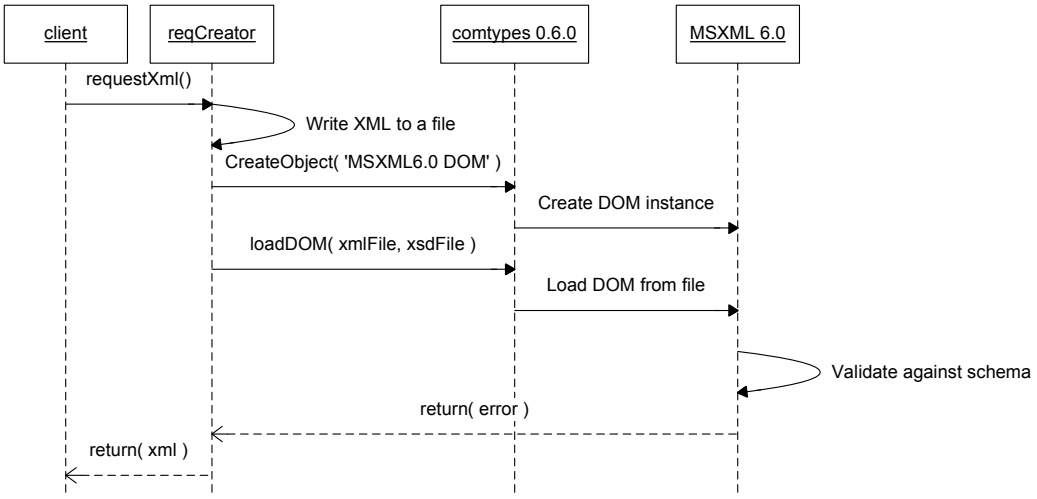

Figure 19. XML validation.

7 USE CASES

7.1. Use case definitions

The use cases for running the simulation were created by describing the weekly application usage and advertisement exposure and running these for a one-month period (four cycles). Each week, some randomness was added to obtain more variety and unpredictability to the requests:

- User context was randomised.
- The number of times a day was randomly decreased or increased by one or zero.
- The time of a request was randomised so that the order of requests was different each day.
- The keywords attached to a request were selected randomly from a selection of 10 keywords.
- The number of impressions randomly decreased or increased by one or zero.
- Time and duration of impressions were randomised.
- Actions were executed randomly with a 10% click-through-rate.

The random seed was initialised to one before each simulation run, to keep the runs the same. Different application usage patterns were distributed over three user groups, with all of them accessing the same advertisement data on the server-side.

7.2. Applications

Applications running on mobile devices are beginning to offer the same functionality as desktop applications. The Mobile Marketing Association (MMA) categorises mobile applications into six types: communications, games, multimedia, productivity, travel, and utilities [54]. This categorisation was abstracted into three top-level categories – media, utilities, and communication – and one application was taken to represent each category.

From the simulation point of view, the application's connectivity and advertisement request types play the most important role; thus, one never-connected and two occasionally connected or always-connected applications were chosen. To gain variety in the advertisement requests and to truly test the caching functionality, other parameters were changed a great deal during the simulation runs. This way, the actual application type does not matter, since it is possible to simulate the requests from multiple applications with the three chosen ones by using different categories and placements. Table 3 describes the applications chosen, their categories, and placements. Also, a set of random keywords was used for each application's advertisement requests.

Table 3: Advertisement request parameters.

Application	Categories	Placements	Keywords
E-mail	Communications	Bottom	Food
		List	Pizza
Idle screen	Utilities	Bottom	Sports
			Action
Media	News	Top	News
	Music	Playlist	Local
	Video	In-game	Football
	Games	Pre-roll	Entertainment
	Graphics		Rock
	Applications		Global
	Ringtones		

The advertisement placement in the e-mail application was set up in the simulation such that when the user checks for new e-mail messages or reads one, new advertisements are loaded from the server and shown if the user has actually received new e-mail and opens the list view or message view.

For the idle screen, the advertisement placement was set up in such a way that no new advertisements are fetched from the server and the user sees an advertisement when starting to use the phone, if suitable advertisement can be found in the cache. This obviously leads to several cache misses, but it is interesting to compare how different pre-loading algorithms affect the behaviour of idle screen advertising.

The media application downloads a new advertisement when updating the media content from the server. When the media are consumed, advertisements will be shown.

7.3. User groups

For testing of the different optimisations, three user cases were chosen, representing active, casual, and inactive users. The user configurations were done to represent quite different usage behaviours, for showing the optimisations' effects in practice for different users. The active user receives a lot of e-mail and consumes a lot of media content over WLAN and Wideband Code Division Access (WCDMA) connections. The casual user uses e-mail and consumes some media over WCDMA, and the inactive user just uses media once a week, on weekends. Table 4 summarises the daily application usage, where the media download figures indicate the number of media items (news items, videos, music items, etc.) downloaded.

Table 4: Application usage for selected user categories.

Activity per day	Active user	Casual user	Inactive user
E-mail refreshed (weekdays)	10	4	-
E-mail refreshed (weekends)	8	4	-
E-mail received (weekdays)	10	2	-
E-mail received (weekends)	8	2	-
Idle screen activated (weekdays)	60	30	10
Idle screen activated (weekends)	50	15	20
Media downloads over WLAN (weekdays)	14	-	-
Media downloads over WLAN (weekends)	21	-	-
Media downloads over WCDMA (weekdays)	7	7	-
Media downloads over WCDMA (weekends)	-	14	3.5

The selected user categories use the phone's idle screen with different frequencies. The categories are based on a target market segment that uses other features of the phone than call and SMS; thus, the 'inactive' user also consumes media content on the mobile device.

The user's context information was set to change randomly from one advertisement request to the next, for variability in the data sent, but it was not used as a targeting parameter, for simplicity. In the current model, it is assumed that user-context-based targeting can be done totally server-side because even location-targeted advertisements are usually relevant even after the user has moved from his or her previous location. However, this would be a whole new topic of study and is covered in references [55] and [56].

7.4. Advertisements

The advertisement data set-up was such that each advertisement request from the server component would return a new set of advertisements, revealing the limitations of the current system, in the case where separate image caching does not help at all. In practice, when one is doing display advertising, this is not a likely scenario, but it is optimal for testing the data transfer optimisations.

The advertisement data set on the server was set up to include 28% generic advertisements without any targeting; 56% advertisements targeted for specific categories, keywords, placements, or applications; and 16% following very specific definitions of the allowed appearance. This distribution was selected to allow testing of cache behaviour with the assumed set-up. Adjustment of the distribution was not studied. The detailed distribution is shown in Figure 20.

The average image size for an advertisement was 5.1 kilobytes, which is slightly biased towards the MMA X-Large banner size [57], because advertisers generally prefer great-looking graphics in mobile advertisements. Although the campaign start and end dates were included in the data, they were not used. All advertisements included one or two click-to actions, and the advertisements were limited to 10 impressions per user for targeted advertisements and to 15 for generic advertisements.

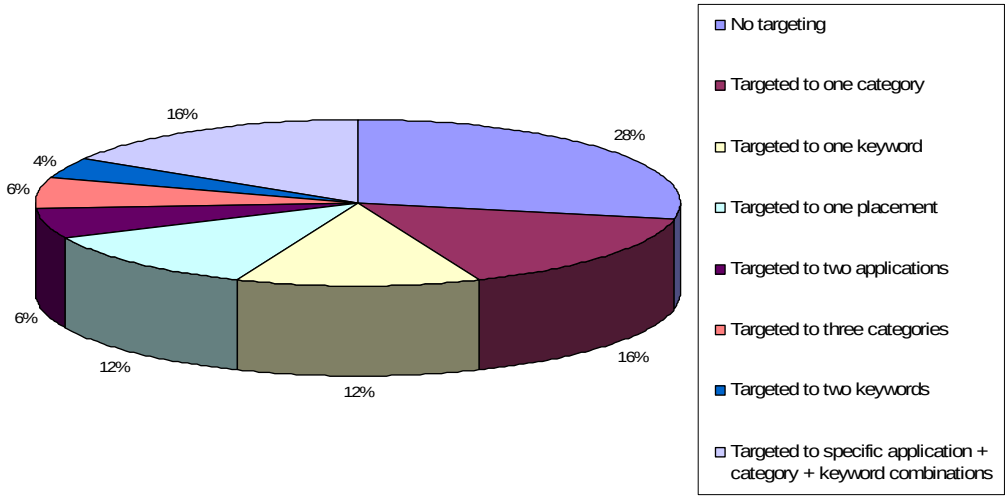

Figure 20. Advertisement data targeting distribution.

8 RESULTS

8.1. Original set-up

After the simulations were run for the selected use cases, major differences could be seen in the online usage of different user groups. In offline usage, there is a smaller difference, but the number of cases when no advertisement can be shown (total misses in Table 5) is about 14% for the 'inactive' user.

Table 5: Advertisement request statistics in simulation runs.

Ad requests in online mode	Active user	Casual user	Inactive user
Number of requests	1137	417	22
Ads requested, total	4796	1181	88
Ads received, total	4796	1181	88
Received from cache	0	0	0
Received from server	4796	1181	88
Total misses	0	0	0
Impressions	5693	1064	41
Actions	585	113	2

Ad requests in offline mode	Active user	Casual user	Inactive user
Number of requests	1604	718	360
Ads requested, total	1604	718	360
Ads received, total	1604	718	309
Received from cache	1604	718	309
Received from server	0	0	0
Total misses	0	0	51
Impressions	1571	720	295
Actions	170	67	32

Cache statistics	Active user	Casual user	Inactive user
Ads fully used	111	15	0
Ads removed otherwise	4585	1063	0
Impressions removed	52470	12045	0

Data transfer for different use cases is presented in Table 6. Of note here is that the XML accounts for about 26% of the total transferred data when a lot of content is loaded over a WLAN connection, but when less content is loaded at once the XML is up to 39% of the data. This confirms the assumption made about the data structure.

Table 6: Data transferred in simulation runs.

Data transfer	Active user WCDMA		Active user WLAN	
Total data	11473790	100.0%	23348337	100.0%
Image data	7271726	63.4%	17327346	74.2%
XML data	4202064	36.6%	6020991	25.8%
Request data	2750475	24.0%	2806935	12.0%
Reports	2303882	20.1%	2323201	10.0%
Profile	87086	0.8%	94402	0.4%
Other	359507	3.1%	389332	1.7%
Response data	1451589	12.7%	3214056	13.8%

Data transfer	Casual user WCDMA		Inactive user WCDMA	
Total data	8808514	100.0%	717431	100.0%
Image data	6121579	69.5%	440285	61.4%
XML data	2686935	30.5%	277146	38.6%
Request data	1474671	16.7%	190529	26.6%
Reports	1133351	12.9%	172653	24.1%
Profile	66195	0.8%	3880	0.5%
Other	275125	3.1%	13996	2.0%
Response data	1212264	13.8%	86617	12.1%

8.2. XML compression

After removal of all unnecessary indention and linefeeds, the XML data size decreased, on average, 27.1 per cent.

Table 7: XML data after removal of whitespace and linefeeds.

Use case	Original XML data	XML data after	Drop	
Active user (WCDMA)	4202064	3091089	1110975	26.4%
Active user (WLAN)	6020991	4359329	1661662	27.6%
Casual user	2686935	1966165	720770	26.8%
Inactive user	277146	200816	76330	27.5%
Average	3296784	2404350	892434	27.1%

When applying other compression methods to the XML data, we can obtain a compression ratio of 60% (WBXML) to 78% (paq8p), with XMill-N being a little behind with 39%. WBXML takes a bit more space since every report contains the timestamp and spot details, which are not compressed at all, but the gzip performs surprisingly well. The effect of compression methods can be seen in Figure 21 and a complete list of the compression figures can be found in Appendix II.

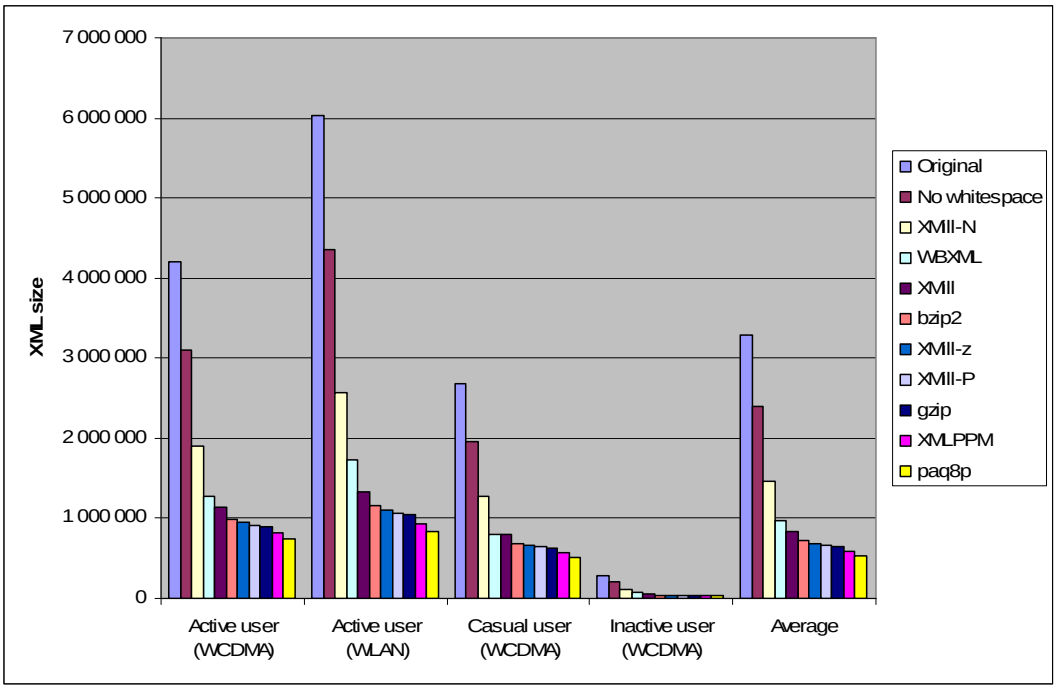

Figure 21. Size of XML data with different compression methods applied.

Also, the performance hit of running a large number of statistical models, combined with neural networks, was noticeable: paq8p took seconds on a decent laptop, whereas the others took considerably less than a second. This confirms the assumption that the arithmetic compressors are not yet usable in mobile phones.

8.3. Protocol optimisation

In aggregation of the reports, the total data size decrease from the original with whitespace removal is in average 40% and when applying the data compression, WBXML is almost on par with the compression algorithms, and the average drop is 63%, while paq8p yields 72%. This is because actual data amount stored within the XML structure is less, in comparison to previous runs. Also the differences between other compression mechanisms are getting smaller, yet the gzip is keeping the position. The XML data size after report aggregation and compression can be seen in Figure 22 and a complete list can be found in Appendix III.

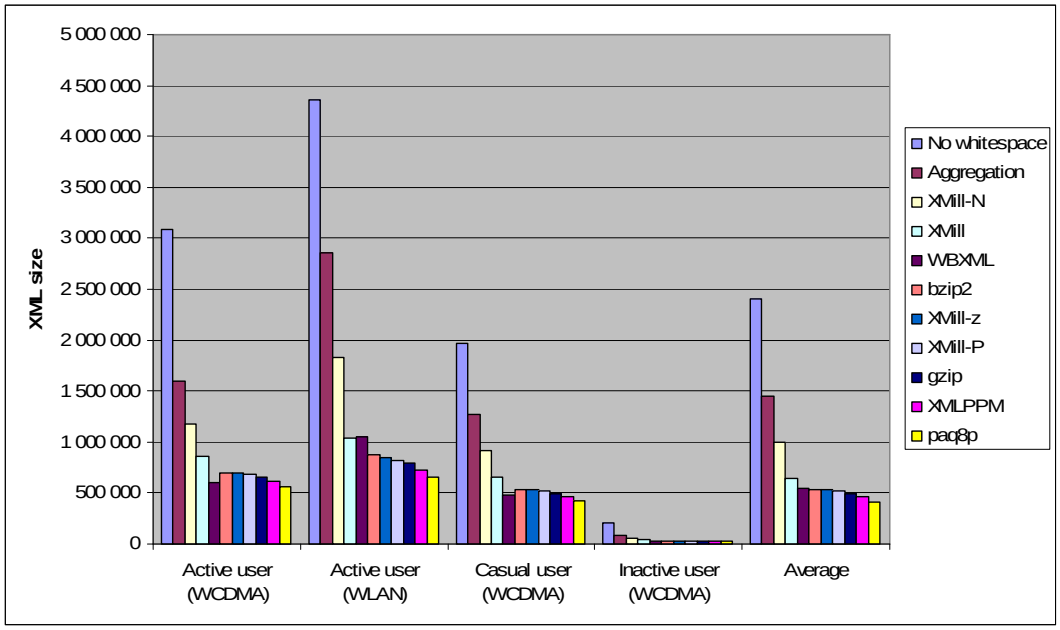

Figure 22. Size of XML data after aggregating reports.

With more detail lost from the reports, the benefit is not that great. Click reports and ad deletion reports still must be sent, and this affects any revenue-sharing between publishers. The effect of report ignoring can be seen in Figure 23 and the numbers can be found from Appendix IV.

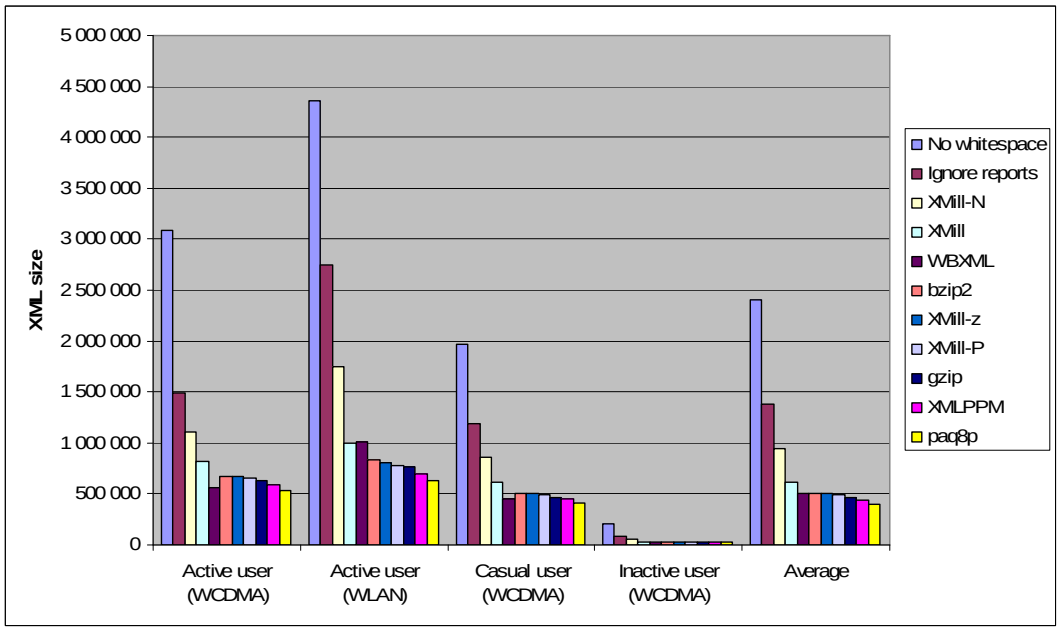

Figure 23. Size of XML data after ignoring reports.

With the targeting data removed from the XML, the quantity of data transferred falls and WBXML gets better with reduction of the actual data in the XML, but still paq8p is better on average, due to the active user's WLAN traffic, where click reports still generate some data in the XML. The results can be seen in Figure 24 and details in Appendix V. This optimisation works only for the current implementation because advertisements are not used from the cache, but instead they are loaded from the network server every time the cache is refreshed. When the cache usage improvements are applied, this destroys all the benefits of caching the advertisements.

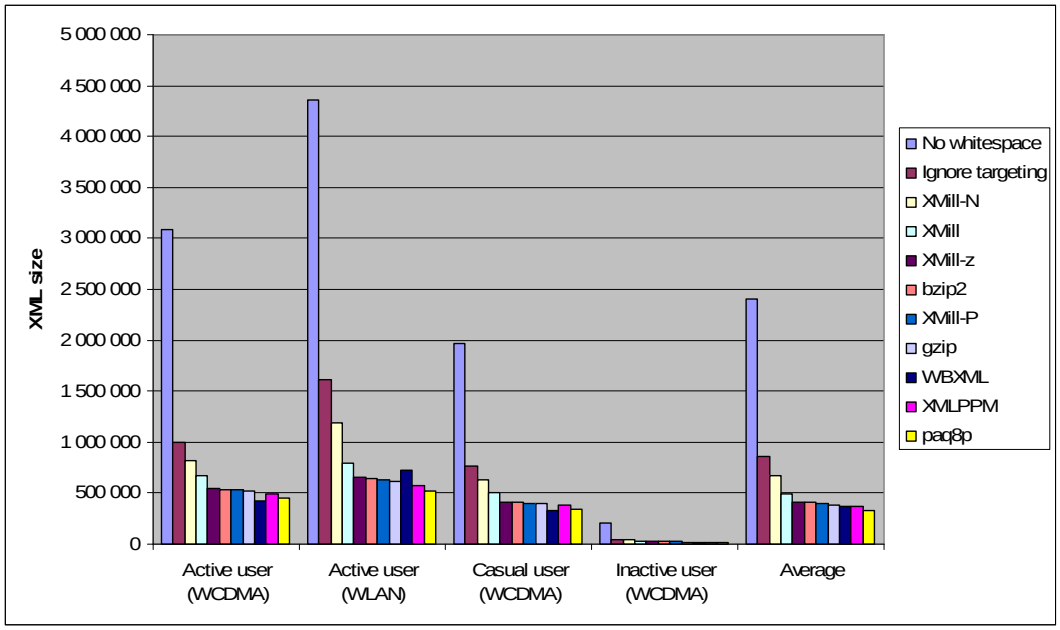

Figure 24. XML quantities after ignoring of reports and targeting information.

8.4. Cache usage improvements

By changing the caching algorithm to serve all impressions from cache before going to the network to fetch more advertisements, substantial improvement can be seen in the overall data usage. The increased role of reports starts to be seen when advertisements are served from cache, and now the XML actually takes more space than the binary data, so compressing only the XML data gives us good results for the overall data. The data transfer breakdown can be seen in Table 8.

Table 8: Data transfer after optimisation of cache usage.

	Active user			
Data transfer	WCDMA		WLAN	
	Bytes	Drop	Bytes	Drop
Total data	737805	92.9%	5361650	75.3%
Image data	279830	96.2%	2363926	86.4%
XML data	457975	85.2%	2997724	31.2%
Request data	417436	79.8%	2664438	-26.5%
Reports	395936	76.9%	2532337	-46.3%
Profile	3960	94.2%	25182	65.7%
Other	17540	93.7%	106919	64.7%
Response data	40539	96.1%	333286	85.2%

	Casual user		Inactive user	
Data transfer	WCDMA		WCDMA	
	Bytes	Drop	Bytes	Drop
Total data	1604443	80.2%	184802	71.2%
Image data	762223	87.5%	107701	75.5%
XML data	842220	57.2%	77101	61.6%
Request data	728537	34.4%	59479	57.5%
Reports	669444	20.7%	51746	58.9%
Profile	11169	78.3%	1896	36.8%
Other	47924	77.6%	5837	46.6%
Response data	113683	86.7%	17622	71.0%

The result was somewhat expectable since the advertisements can be shown around 10 times more than previously and it seems that the changed cache algorithm was able to take full advantage of this. The cache usage statistics and number of requests served from cache can be seen in Table 9. It can be seen that all the advertisements that were downloaded were used fully, although the number of cache misses in idle screen increased. This is because the cache contains less variety in advertisements, thus giving smaller probability for finding an advertisement from cache that can be also shown in idle screen.

This should work well with report aggregation and reporting only unused impressions as the algorithm tries to use all impressions from cache first. However, in comparison of aggregation and reporting only unused impressions when cache usage optimisation is on, the difference should not be that great, since little content is lost (basically just advertisement spot information). The savings may be only theoretical, though, since the advertisement deletion reports would also consume space and the clicks would still be reported.

Table 9: Request and cache usage details with optimised cache.

Ad requests in online mode	Active user	Casual user	Inactive user
Number of requests	1137	417	22
Ads requested, total	4796	1181	88
Ads received, total	4796	1181	88
Received from cache	4279	1033	65
Received from server	517	148	23
Total misses	0	0	0
Impressions	5693	1064	41
Actions	585	113	2

Ad requests in offline mode	Active user	Casual user	Inactive user
Number of requests	1604	718	360
Ads requested, total	1604	718	360
Ads received, total	1540	602	102
Received from cache	1540	602	102
Received from server	0	0	0
Total misses	64	116	258
Impressions	1508	600	100
Actions	165	59	7

Cache statistics	Active user	Casual user	Inactive user
Ads fully used	481	132	12
Ads removed otherwise	0	0	0
Impressions removed	0	0	0

The effect of compression methods can be seen in Figure 25. When comparing the XML compression against overall data, the differences can not be almost seen at all between the best compression ratios. The complete list of numbers can be found in Appendix VI.

Comparing against the total data, WBXML gives 33% compression ratio, gzip 46% and paq8p 48% when used only for compressing the XML part of the data and leaving the advertisement content as it is. This implies unexpectedly that the gzip is performing almost on par with one of the best arithmetic compressors available, when compressing XML data in this particular domain. The cache usage improvements achieve cost savings of 81% and 90% with the top seven compressors.

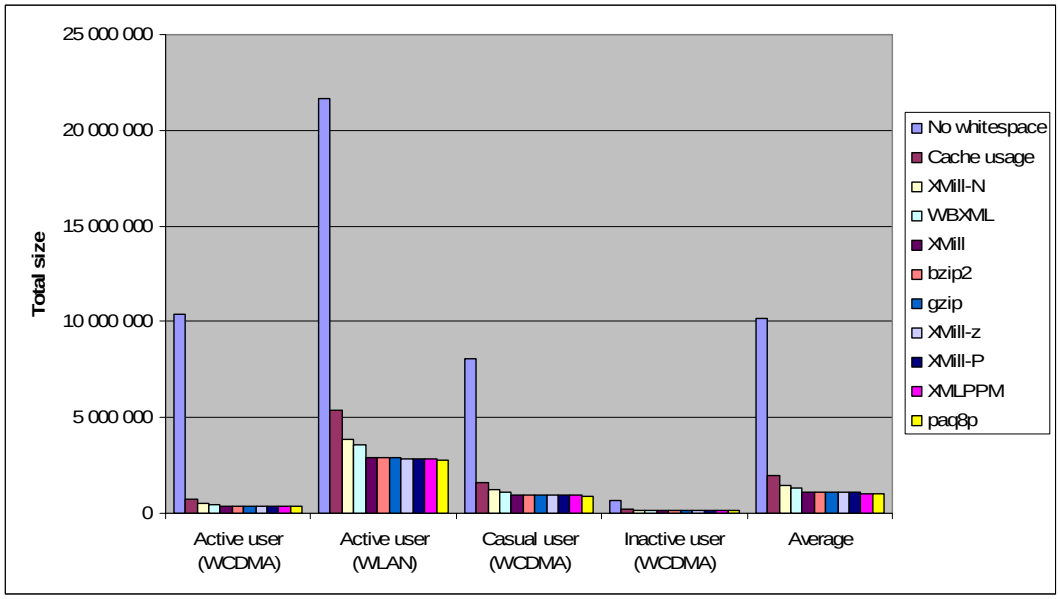

Figure 25. Data transfer with different compression techniques and cache usage.

With aggregation of the reports, the effect on raw XML is good, but as the amount of XML content gets smaller in comparison to binary data, the benefit of compressing the XML diminishes (shown in Figure 26 and in Appendix VII). The top three compressors yield around 14% compression ratios, thus giving total cost savings of around 91% (without compression 89%).

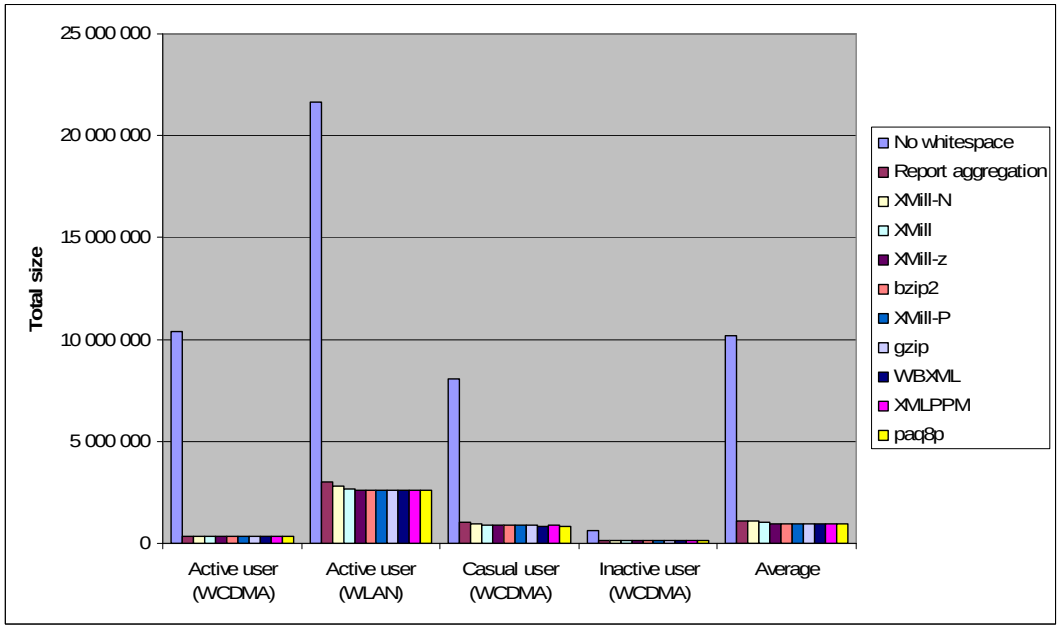

Figure 26. Data transfer with cache optimisation and report aggregation.

For the active user, the cost can be optimised even further by utilising free WLAN connection. However, when the WLAN optimisation is applied, surprisingly also the data quantity transferred over WCDMA increases. This can be explained by looking at the cache statistics in Table 10; when loading advertisements for the WLAN case, we are actually deleting advertisements from the cache without utilising them fully as the cache gets full.

Table 10: Cache statistics when WLAN optimisations are used.

Cache statistics	Active user
Ads fully used	181
Ads removed otherwise	3234
Impressions removed	37055

When the cache size is increased, the WLAN optimisation benefits start to be visible in data transfer over a WCDMA connection. This is presented in Figure 27, where the effect can be seen to ultimately cut the cost away from active user.

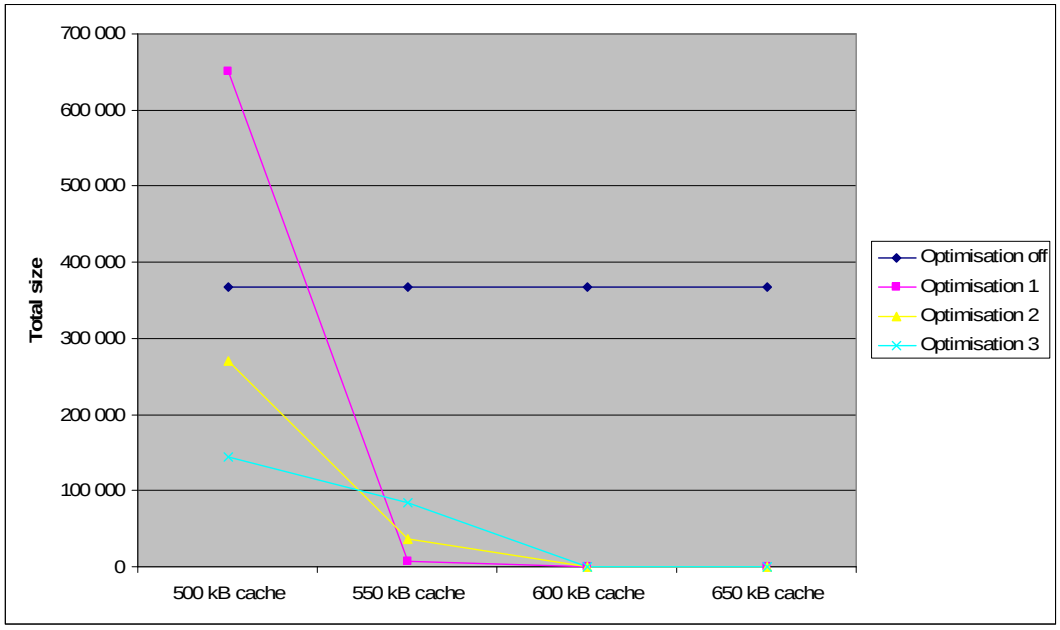

Figure 27. Data transfer over WCDMA (active user) with different optimisations.

No differences in numbers appeared when cache size was increased for the casual and average user. The reason is that the impressions reserved for the cache are already few enough that the size increase does not work to the benefit of advertisement lifetime.

The data transferred over WLAN in active user case can be seen in Figure 28. It increases linearly when the optimisation level increases since the algorithm fetches more data using the WLAN connection.

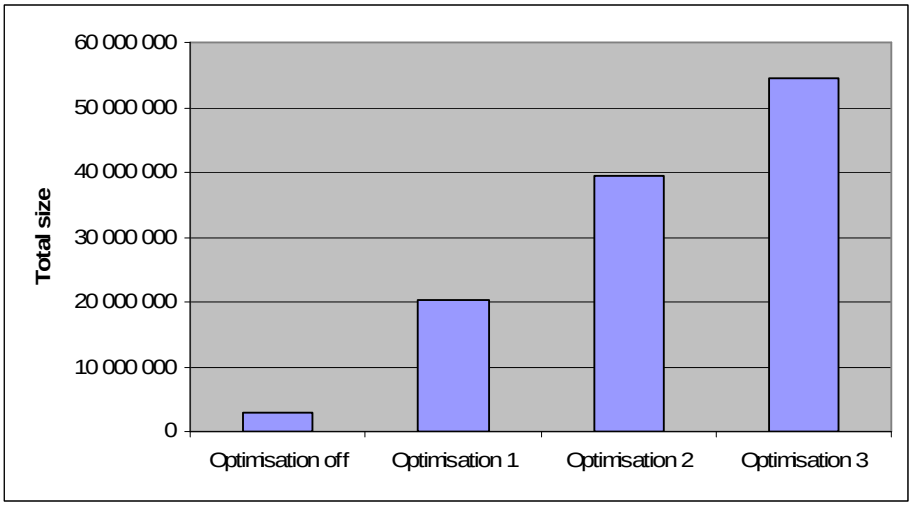

Figure 28. Data transfer over WLAN (active user) with different optimisations.

9 CONCLUSIONS

9.1. Findings

Implementation of the simulation environment was straightforward using the Python language, but the client-server system conceives a lot of functionality around targeting and request-response handling that required considerable amount of implementation and validation effort. Validating the generated protocol messages against XML schema did not provide good enough quality and most of the logic had to be verified manually.

Defined use cases yielded enough variety in the advertisement requests and the idle screen advertisement serving was surprisingly successful with only 14% cache misses in the inactive user group and 0% with others.

It was discovered that the XML protocol creates more overhead than was thought, up to 39%, the majority of that because of the detailed, verbose reports. When the cache is used more optimally, the proportion of XML data in relation to all data grew even greater, exceeding 60%. This can be decreased 27% by removing the whitespace and linefeeds, but the best solution to decrease the quantity of XML data was to change the protocol to discard details from the reports and aggregate the data before sending to the network server. This resulted in additional 40% savings.

All of the selected compression methods worked well with the XML data, and, when compared to overall data, they displayed only minor differences from each other. This flattened out even more when report aggregation was used. Surprising result was that with the data transferred in this environment, the general-purpose compressors were better than XML-specific ones. Assessing only the XML data, WBXML resulted in 60%, paq8p in 78% and gzip in 73% compression ratio, but the paq8p was discovered to be too slow to be used in real-time communication on resource-constrained devices. The gzip was noted of better compression performance than bzip2 in this context, and the top three compressors were paq8p, XMLPPM and gzip.

For overall cost savings, the greatest benefit could be found by optimising the pre-loading algorithm and utilising free connections. By optimising pre-loading algorithm and cache searching, all the impressions for advertisements that were loaded to the local cache could be used completely. The cache usage improvements achieved cost savings of 81%, and 90% with the top seven compressors. Cache misses for idle screen increased slightly, which was expected since the probability of having advertisements with loose targeting decreased. Including quantity of data transferred with optimised cache usage, the WBXML was able to achieve 33% compression ratio, whereas gzip managed 46% and paq8p 48%.

Combining the cache optimisations with report aggregation, the overall compression ratio decreased to 14% with WBXML, 13% with gzip, and 14% with paq8p, thus giving total cost savings of around 91%, whereas it was 89% without compression.

Combining the 20% increase in cache capacity with WLAN connection utilisation in the active user group, all the required advertising data could be moved without generating any cost to the end user. It was discovered that this evident result can be achieved even with simple pre-loading enhancements without needing to predict the need for exact advertisements as long as the system has advertisements that are suitable for any use. It is enough to load about one third of generic advertisements, if the amount of pre-loaded advertisements is high enough.

9.2. Recommended actions

Since the reporting currently is very detailed and verbose, it should be aggregated somehow or at least grouped differently in the XML to reduce redundancy. Also, WBXML should be brought into use, given that implementations already exist and it even speeds up the parsing. Another good option would be to use gzip, which, though requiring one extra step of decompressing and compressing, should be quite fast.

The pre-loading algorithm optimisations should be given priority, since these have the biggest impact on the overall data transfer.

9.3. Recommendations for future study

This work has been technology-focused, and the simulations should be run with real use cases, real-world application set-up, and actual advertising campaigns. Also, the PPM-based intelligent pre-loading algorithms would be worth testing with the simulator when more targeting parameters are brought into use.

Further studies could be done also in the area of adjusting the targeting accuracy in combination with pre-loading algorithm to determine the optimal balance between different levels of targeting and cache misses.

REFERENCES

[1] Google. Financial tables for 2005–2008. http://investor.google.com/fin_data.html, accessed in May 2009.

[2] Vatanparast, R., "Piercing the Fog of Mobile Advertising", *International Conference on the Management of Mobile Business, 2007, ICMB 2007*, pp. 19-19, 9-11 July 2007.

[3] Blyk, About Blyk. http://about.blyk.com/, accessed in May 2009.

[4] Mohamed Yunos, H.; Zeyu Gao, J.; Shim, S., "Wireless advertising's challenges and opportunities", *Computer*, vol. 36, no. 5, pp. 30-37, May 2003.

[5] Bulander, R.; Decker, M.; Schiefer, G.; Kolmel, B., "Comparison of Different Approaches for Mobile Advertising", *The Second IEEE International Workshop on Mobile Commerce and Services, 2005, WMCS '05*, pp. 174-182, 19-19 July 2005.

[6] Zeyu Gao J.; Ji, A., "SmartMobile-AD: An Intelligent Mobile Advertising System", *The 3rd International Conference on Grid and Pervasive Computing Workshops, 2008, GPC Workshops '08*, pp. 164-171, 25-28 May 2008.

[7] Elisa Oyj. Price list for 'Elisa Perusdata' data subscription. http://www.elisa.fi/matkaviestinta/index.cfm?o=199.60, accessed in May 2009.

[8] GSM Association, GSM World Coverage 2008. http://www.gsmworld.com/roaming/GSM_WorldPoster2008A.pdf, accessed in May 2009.

[9] Chakravorty, R.; Clark, A.; Pratt, I., "Optimizing Web delivery over wireless links: design, implementation, and experiences", *IEEE Journal on Selected Areas in Communications*, vol. 23, no. 2, pp. 402-416, Feb. 2005.

[10] Tapia, P.; Wellington, D.; Jun Liu; Karimli, Y., "Practical Considerations of HSDPA Performance", *Vehicular Technology Conference, 2007, VTC-2007 Fall, 2007 IEEE 66th*, pp. 111-115, Sept. 30 2007-Oct. 3 2007.

[11] Jurvansuu, M.; Prokkola, J.; Hanski, M.; Perala, P., "HSDPA Performance in Live Networks", *IEEE International Conference on Communications, 2007, ICC '07*, pp. 467-471, 24-28 June 2007.

[12] Chakravorty, R.; Banerjee, S.; Rodriguez, P.; Chesterfield, J.; Pratt, I., "Performance optimizations for wireless wide-area networks: comparative study and experimental evaluation", *Proceedings of the 10th Annual international Conference on Mobile Computing and Networking* (Philadelphia, PA, USA, September 26 - October 01, 2004), MobiCom '04, ACM, New York, NY, 159-173.

[13] Hristova, N.; O'Hare, G.M.P., "Ad-me: wireless advertising adapted to the user location, device and emotions", *Proceedings of the 37th Annual Hawaii International Conference on System Sciences, 2004*, pp. 10, 5-8 Jan. 2004.

[14] Tripathi, A.K.; Nair, S.K., "Mobile Advertising in Capacitated Wireless Networks", *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 9, pp. 1284-1296, Sept. 2006.

[15] Aalto, L.; Göthlin, N.; Korhonen, J.; Ojala, T., "Bluetooth and WAP push based location-aware mobile advertising system", *Proceedings of the 2nd international Conference on Mobile Systems, Applications, and Services* (Boston, MA, USA, June 06 - 09, 2004), MobiSys '04, ACM, New York, NY, 49-58.

[16] Sánchez, J.; Cano, J.; Calafate, C. T.; Manzoni, P., "BlueMall: a bluetooth-based advertisement system for commercial areas", *Proceedings of the 3rd ACM Workshop on Performance Monitoring and Measurement of Heterogeneous Wireless and Wired Networks* (Vancouver, British Columbia, Canada, October 31 - 31, 2008), PM2HW2N '08, ACM, New York, NY, 17-22.

[17] Thawani, A.; Gopalan, S.; Sridhar, V.; Ramamritham, K., "Context-aware timely information delivery in mobile environments", *Computer. J.* 50, 4 (Jul. 2007), 460-472.

[18] Natchetoi, Y.; Kaufman, V.; Shapiro, A., "Service-oriented architecture for mobile applications", *Proceedings of the 1st international Workshop on Software Architectures and Mobility* (Leipzig, Germany, May 10 - 10, 2008), SAM '08, ACM, New York, NY, 27-32.

[19] Jing, J.; Helal, A. S.; Elmagarmid, A., "Client-server computing in mobile environments", *ACM Computing. Surveys* 31, 2 (Jun. 1999), 117-157.

[20] Rahmati, A. ; Zhong, L., "Context-for-wireless: context-sensitive energy-efficient wireless data transfer", *Proceedings of the 5th international Conference on Mobile Systems, Applications and Services* (San Juan, Puerto Rico, June 11 - 13, 2007), MobiSys '07, ACM, New York, NY, 165-178.

[21] Natchetoi, Y.; Huaigu Wu; Yi Zheng, "service-oriented mobile applications for ad-hoc networks", *IEEE International Conference on Services Computing, 2008, SCC '08,* vol. 2, pp. 405-412, 7-11 July 2008.

[22] Apte, N.; Deutsch, K.; Jain, R., "Wireless SOAP: optimizations for mobile wireless web services", *Special interest Tracks and Posters of the 14th international Conference on World Wide Web* (Chiba, Japan, May 10 - 14, 2005). WWW '05. ACM, New York, NY, 1178-1179.

[23] Mahoney, M., "Adaptive weighing of context models for lossless data compression", *Florida Technology Technical Report* CS-2005-16, 2005.

[24] Kattan, A.; Poli, R., "Evolutionary lossless compression with GP-ZIP", *IEEE Congress on Evolutionary Computation, 2008, CEC 2008,* (IEEE World Congress on Computational Intelligence), pp. 2468-2472, 1-6 June 2008.

[25] Augeri, C. J.; Bulutoglu, D. A.; Mullins, B. E.; Baldwin, R. O.; Baird, L. C., "An analysis of XML compression efficiency", *Proceedings of the 2007 Workshop on*

Experimental Computer Science (San Diego, California, June 13 - 14, 2007), ExpCS '07, ACM, New York, NY, 7.

[26] PKWARE, Inc. ZIP file format specification, http://www.pkware.com/documents/casestudies/APPNOTE.TXT, accessed in May 2009.

[27] Gailly, J.–l.; Adler, M. The gzip home page. http://www.gzip.org/, accessed in May 2009.

[28] Seward, J. The bzip2 home page. http://www.bzip.org/, accessed in May 2009.

[29] MaximumCompression. Single-file data compression benchmarks. http://www.maximumcompression.com/data/summary_sf.php, accessed in May 2009.

[30] Moffat, A., "Implementing the PPM data compression scheme", *IEEE Transactions on Communications*, vol. 38, no. 11, pp. 1917-1921, Nov 1990.

[31] Mahoney, M. The PAQ data compression programs. http://www.cs.fit.edu/~mmahoney/compression/paq.html, accessed in May 2009.

[32] Ziv, J.; Lempel, A., "A universal algorithm for sequential data compression", *IEEE Transactions on Information Theory*, vol. 23, no. 3, pp. 337-343, May 1977.

[33] Witten, I. H.; Neal, R. M.; Cleary, J. G., "Arithmetic coding for data compression", *Communications*, ACM 30, vol. 6 (Jun. 1987), 520-540.

[34] MaximumCompression. JPG/JPEG lossless image compression test. http://www.maximumcompression.com/data/jpg.php, accessed in May 2009.

[35] Deutsch, L. P. "DEFLATE compressed data format specification". http://tools.ietf.org/html/rfc1951, accessed in May 2009.

[36] Huffman, D.A., "A method for the construction of minimum-redundancy codes", *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098-1101, Sept. 1952.

[37] MaximumCompression. Multiple file data compression benchmark. http://www.maximumcompression.com/data/summary_mf3.php#data, accessed in May 2009.

[38] Burrows, M.; Wheeler, D. J., "A block-sorting lossless data compression algorithm", *Technical Report 124*, 1994.

[39] Bentley J. L.; Sleator D.D.; Tarjan R.E.; Wei V.K., "A locally adaptive data compression scheme", *Communications of the ACM*-Vol. 29, No. 4, 1986.

[40] W3C. Efficient XML Interchange Working Group. http://www.w3.org/XML/EXI/, accessed in May 2009.

[41] W3C. WAP Binary XML Content Format. http://www.w3.org/TR/wbxml/, accessed in May 2009.

[42] Liefke, H.; Suciu, D., "XMill: an efficient compressor for XML data", *Proceedings of the 2000 ACM SIGMOD international Conference on Management of Data* (Dallas, Texas, United States, May 15 - 18, 2000), SIGMOD '00, ACM, New York, NY, 153-164.

[43] Cheney, J., "Compressing XML with Multiplexed Hierarchical PPM Models", *Proceedings of the Data Compression Conference* (March 27 - 29, 2001), DCC, IEEE Computer Society, Washington, DC, 163.

[44] Rodriguez, S. XML optimization. http://www.arstdesign.com/articles/xmloptimization.html, accessed in May 2009.

[45] Python Software Foundation. Python Programming Language – Official Website. http://python.org/, accessed in May 2009.

[46] Python Software Foundation. Package Index: comtypes 0.6.0. http://pypi.python.org/pypi/comtypes, accessed in May 2009.

[47] Microsoft Corporation. MSXML. http://msdn.microsoft.com/en-us/library/ms763742.aspx, accessed in May 2009.

[48] Veillard, D. The XML C parser and toolkit of Gnome. http://xmlsoft.org/index.html, accessed in May 2009.

[49] Reenskaug, T. MVC—XEROX PARC 1978-79. http://heim.ifi.uio.no/~trygver/2007/MVC_Originals.pdf, accessed in May 2009.

[50] Cheney, J. XMLPPM: XML-Conscious PPM Compression. http://xmlppm.sourceforge.net/, accessed in May 2009.

[51] SourceForge, Inc. XMill project. xmill: http://sourceforge.net/projects/xmill/, accessed in May 2009.

[52] Aymerick. The WBXML library. http://libwbxml.aymerick.com/, accessed in May 2009.

[53] WMHelp.com. XMLPad 3. http://www.wmhelp.com/xmlpad3.htm, accessed in May 2009.

[54] Mobile Marketing Association. Mobile Applications. http://mmaglobal.com/mobileapplications.pdf, accessed in May 2009.

[55] Burbey, I.; Martin, T. L., "Predicting future locations using prediction-by-partial-match", *Proceedings of the First ACM international Workshop on Mobile Entity Localization and Tracking in GPS-Less Environments* (San Francisco, California, USA, September 19 - 19, 2008), MELT '08, ACM, New York, NY, 1-6.

[56] Chen, C.; Lee, C.; Wang, C.; Chung, Y, "Prefetching LDD: a benefit-oriented approach", *Proceedings of the 2006 international Conference on Wireless Communications and Mobile Computing* (Vancouver, British Columbia, Canada, July 03 - 06, 2006), IWCMC '06, ACM, New York, NY, 1103-1108.

[57] Mobile Marketing Association. Mobile Advertising Guidelines. http://mmaglobal.com/mobileadvertising.pdf, accessed in May 2009.

APPENDIX I. DATA COMPRESSORS

	GNU zip	**bzip2**	**paq8p**
Type	Dictionary	Dictionary	Arithmetic
Pre-processor	None	Burrows-Wheeler transformation	None
Duplicate removal	LZ77	Move-to-front	None
Prediction	None	None	Huge amount of dynamic models
Storing	Huffman tree	Huffman tree	Arithmetic coding using dynamically trained neural network
Open Source project	gzip	bzip2	paq8p

	WBXML	**XMill**	**XMLPPM**
Type	Binary XML	Compressed XML	Arithmetically encoded stream of SAX events
Algorithm	Change element names to enumerated binary numbers	Separate structure from content, arrange similar content together, apply general-purpose compression	Parse XML with SAX parser and encode events with four models: one for element and attribute names, one for element structure, one for attributes, and one for strings
Open Source project	libwbxml	XMill	XMLPPM

APPENDIX II. RESULTS OF DATA COMPRESSION

	Original	No whitespace	XMill-N	WBXML
Active user (WCDMA)	4202064	3091089	1909107	1271800
Active user (WLAN)	6020991	4359329	2562807	1725853
Casual user (WCDMA)	2686935	1966165	1266220	792298
Inactive user (WCDMA)	277146	200816	109997	79321
Average	3296784	2404350	1462033	967318
Average drop		27.1 %	39.2 %	59.8 %

	XMill	bzip2	XMill-z	XMill-P
Active user (WCDMA)	1134845	986781	944585	917429
Active user (WLAN)	1326879	1166022	1095398	1055776
Casual user (WCDMA)	802913	688052	661527	643526
Inactive user (WCDMA)	49625	44482	42059	40239
Average	828566	721334	685892	664243
Average drop	65.5 %	70.0 %	71.5 %	72.4 %

	gzip	XMLPPM	paq8p
Active user (WCDMA)	891511	819962	735362
Active user (WLAN)	1040185	938318	835879
Casual user (WCDMA)	618416	574053	518166
Inactive user (WCDMA)	41375	36594	31773
Average	647872	592232	530295
Average drop	73.1 %	75.4 %	77.9 %

APPENDIX III. RESULTS OF REPORT AGGREGATION

	No whitespace	Aggregation	XMill-N	XMill
Active user (WCDMA)	3091089	1593491	1180379	857861
Active user (WLAN)	4359329	2858373	1830059	1039868
Casual user (WCDMA)	1966165	1268375	921990	652684
Inactive user (WCDMA)	200816	87361	58754	36726
Average	2404350	1451900	997796	646785
Average drop		39.6 %	31.3 %	55.5 %

	WBXML	bzip2	XMill-z	XMill-P
Active user (WCDMA)	602347	698463	702104	684911
Active user (WLAN)	1054355	869204	846160	816447
Casual user (WCDMA)	479699	534360	532666	519387
Inactive user (WCDMA)	33218	30543	29779	28881
Average	542405	533143	527677	512407
Average drop	62.6 %	63.3 %	63.7 %	64.7 %

	gzip	XMLPPM	paq8p
Active user (WCDMA)	650853	614291	556436
Active user (WLAN)	795016	727283	650966
Casual user (WCDMA)	495258	466687	421945
Inactive user (WCDMA)	28030	25843	23048
Average	492289	458526	413099
Average drop	66.1 %	68.4 %	71.5 %

APPENDIX IV. RESULTS OF REPORT IGNORING

	No whitespace	Ignore reports	XMill-N	XMill
Active user (WCDMA)	3091089	1492212	1110489	817470
Active user (WLAN)	4359329	2750360	1754675	995012
Casual user (WCDMA)	1966165	1185432	865098	618635
Inactive user (WCDMA)	200816	77384	52686	33980
Average	2404350	1376347	945737	616274
Average drop		42.8 %	31.3 %	55.2 %

	WBXML	bzip2	XMill-z	XMill-P
Active user (WCDMA)	561570	666556	667670	651758
Active user (WLAN)	1009211	833947	808183	779896
Casual user (WCDMA)	445440	507780	504181	491814
Inactive user (WCDMA)	28594	28097	27463	26664
Average	511204	509095	501874	487533
Average drop	62.9 %	63.0 %	63.5 %	64.6 %

	gzip	XMLPPM	paq8p
Active user (WCDMA)	621909	590141	534246
Active user (WLAN)	762952	700143	626164
Casual user (WCDMA)	470900	446489	403613
Inactive user (WCDMA)	25770	24132	21611
Average	470383	440226	396409
Average drop	65.8 %	68.0 %	71.2 %

APPENDIX V. RESULTS WITHOUT TARGETING DATA

	No whitespace	Ignore targeting	XMill-N	XMill
Active user (WCDMA)	3091089	997804	822143	665306
Active user (WLAN)	4359329	1618333	1187430	787159
Casual user (WCDMA)	1966165	768645	631458	503216
Inactive user (WCDMA)	200816	44631	35249	26307
Average	2404350	857353	669070	495497
Average drop		64.3 %	22.0 %	42.2 %

	XMill-z	bzip2	XMill-P	gzip
Active user (WCDMA)	546753	536937	527851	514432
Active user (WLAN)	649143	648399	622032	615986
Casual user (WCDMA)	413122	406103	398376	389479
Inactive user (WCDMA)	21572	21315	20635	20365
Average	407648	403189	392224	385066
Average drop	52.5 %	53.0 %	54.3 %	55.1 %

	WBXML	XMLPPM	paq8p
Active user (WCDMA)	419781	496688	450359
Active user (WLAN)	721007	579125	520119
Casual user (WCDMA)	329183	375792	340770
Inactive user (WCDMA)	19523	19536	17561
Average	372374	367785	332202
Average drop	56.6 %	57.1 %	61.3 %

APPENDIX VI. RESULTS WITH CACHE OPTIMISATION

	No whitespace	Cache usage	XMill-N	WBXML
Active user (WCDMA)	10362815	737805	513081	466600
Active user (WLAN)	21686675	5361650	3883271	3580849
Casual user (WCDMA)	8087744	1604443	1216226	1105462
Inactive user (WCDMA)	641101	184802	152445	138558
Average	10194584	1972175	1441256	1322867
Average drop		80.7 %	26.9 %	32.9 %

	XMill	bzip2	gzip	XMill-z
Active user (WCDMA)	368673	361668	359614	358736
Active user (WLAN)	2932967	2890828	2874801	2868873
Casual user (WCDMA)	972001	949046	938393	942801
Inactive user (WCDMA)	132108	128786	127124	128100
Average	1101437	1082582	1074983	1074628
Average drop	44.2 %	45.1 %	45.5 %	45.5 %

	XMill-P	XMLPPM	paq8p
Active user (WCDMA)	354257	348824	338317
Active user (WLAN)	2839468	2804439	2736892
Casual user (WCDMA)	934407	919023	898658
Inactive user (WCDMA)	127417	125466	123418
Average	1063887	1049438	1024321
Average drop	46.1 %	46.8 %	48.1 %

APPENDIX VII. RESULTS WITH CACHE AND REPORT OPTIMISATIONS

	No whitespace	Report aggregation	XMill-N	XMill
Active user (WCDMA)	10362815	366735	345826	328677
Active user (WLAN)	21686675	2994313	2818840	2679578
Casual user (WCDMA)	8087744	997765	942426	896830
Inactive user (WCDMA)	641101	138040	131265	125624
Average	10194584	1124213	1059589	1007677
Average drop		89.0 %	5.7 %	10.4 %

	XMill-z	bzip2	XMill-P	gzip
Active user (WCDMA)	320189	319570	319073	317257
Active user (WLAN)	2624401	2623758	2616399	2606190
Casual user (WCDMA)	873190	871577	870022	865016
Inactive user (WCDMA)	122415	122174	122050	121370
Average	985049	984270	981886	977458
Average drop	12.4 %	12.4 %	12.7 %	13.1 %

	WBXML	XMLPPM	paq8p
Active user (WCDMA)	313064	315148	311782
Active user (WLAN)	2604090	2590342	2568478
Casual user (WCDMA)	852502	859534	850400
Inactive user (WCDMA)	119336	120649	119418
Average	972248	971418	962520
Average drop	13.5 %	13.6 %	14.4 %