Rainer Ronnie Pereira Couto

Compressão Adaptativa de Arquivos HTML em Ambientes de Comunicação Sem Fio

Dissertação apresentada ao Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Minas Gerais, como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Belo Horizonte
19 de dezembro de 2003

Acknowledgements

As the popular saying goes: “Behind a every great man there is always a great woman”. Well, I’m not that great yet, but I already have the love of four great women to inspire me: my mother, my two sisters and my niece. There are no words to describe how important you are in my life. I love you more than you can imagine.

It would be impossible to get here without the help of my friends. They were my strength when I wanted to give everything up, so this work is dedicated to all of them, specially to:

- Ricardo A. Rabelo, it is an honor to have you as my friend and to work with you.
- Linnyer Beatrys Ruiz, my “third” sister. You were always a bright light in my path during all these years. Thank you for your company and peaceful words.
- Fernanda P. Franciscani, my friend of work, laughs and studies. I wish I could have someone as special as you in whatever place God takes me to work.
- Ana Paula Silva, thank you for being so loving and so tender.
- Renan Cunha, Wagner Freitas, Vinícius Rosalen, Fátima and Lidiane Vogel. Thank you for the companionship during those hot afternoons inside SIAM lab.
- Many thanks also to Cláudia, Camila, Márcia and Luciana. Your friendship along these past 10 years means a lot to me.
- Antonio A. Loureiro, my advisor and my “academic father”. Thank you for all the words, patience, time, and knowledge you dedicated me. My only wish is to make you proud.
- Last, but not least, Wesley Dias Maciel. Words would not be good enough to express all your companion meant to me over these 4 years. My best friend, you will always be in my heart.

Abstract

The increasing development of mobile computing technologies has allowed users to access the Internet any time and anywhere. However, as the resources of mobile devices are scarcer than their counterparts in wired network, there is the need to adapt data documents in order to provide efficient access to applications and services in wireless environments. In this work, it is proposed a model which predicts how and when a file should be compressed before its transmission over a wireless channel, aiming to minimize the impact on energy consumption and to improve response time. Experiments in real scenarios and simulations using ns2 simulator with both IEEE 802.11 and Bluetooth technologies prove the efficiency of this model when compared with other simplistic approaches adopted in literature.

Resumo

O atual desenvolvimento das diversas tecnologias da computação móvel tornou possível o acesso à Internet de qualquer lugar e a qualquer instante. Entretanto, para se ter um acesso eficiente em um ambiente de comunicação sem fio, é necessário introduzir algum processo de adaptação do conteúdo da Web, uma vez que os recursos dos dispositivos móveis são inerentemente escassos. Este trabalho propõe um modelo para adaptação de conteúdo HTML que prevê dinamicamente quando um arquivo deve ser comprimido antes de sua transmissão. Esse método de adaptação permite reduzir o tempo de transmissão do arquivo, porém introduz o custo da compressão e descompressão do arquivo. O modelo proposto aqui tenta justamente distinguir os cenários nos quais o tempo economizado na transmissão não ultrapassa o custo com o processamento do arquivo. Os experimentos e simulações com os ambientes Bluetooth e IEEE 802.11 comprovam a eficiência e simplicidade desse modelo quando comparado às técnicas atuais de adaptação por compressão.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The development of mobile computing technologies has increased rapidly during the last decade. Mobile devices and wireless communication are already present in our daily activities such as when using devices like cellular phones, Personal Digital Assistants (PDAs) and notebooks, or interacting with embedded systems and sensors. Nonetheless, the advances in wireless and mobile world have not kept pace with their counterparts in wired networks due to restrictions such as less processing power, limited transmission range, complex management of energy consumption, limited size and low bandwidth availability [1]. Processing power in mobile devices is notably inferior when compared to a common desktop computer. Both transmission range and energy consumption are regulated by communication protocols. For instance, Bluetooth devices are often smaller, have lower transmission range and less energy consumption than devices designed for a IEEE 802.11 [2] environment.

There is a tendency towards making mobile devices and networking capabilities increasingly powerful. However, no matter how powerful mobile devices become, the design of applications for wireless communication still have to take into account the characteristics of this complex environment. Energy is a very important restriction since users demand carrying light-weight devices and not heavy batteries. Compared to wired data communication, wireless communications suffers from more interference and more fluctuations on

bandwidth as it depends on the user's physical location, higher bit error ratio and others features of this environment. Besides, they also have to deal with specific problems such as seamless communication, hidden node problems, disconnection, etc.

These restrictions present real challenges when developing applications for wireless environments. One solution to overcome these problems is *adaptability* [1, 3, 4]. Adaptation consist in altering or adjusting the application's behavior according to its awareness of the environment. Adaptation can be done in several ways depending on the application, the environment, user or systems goals and communicating devices. It is more effective when it occurs dynamically, that is, when the adaptation process "polls" the environment and decides which action should be taken during execution time. The aim is to save or at least minimize consumption of scarce resources, such as energy and bandwidth, or to optimize properties that are more visible to users – like response time and data quality. Examples of adaptability are 1) caching emails or Web pages on the mobile device itself so the user can access their content even if a wireless connection is not available at the moment, 2) adaptive power control that enables devices to transmit data in a lower transmission power when they are closer to base stations or other communication points and 3) adaptive channel allocation, which enables devices to select among several channels the one without interference of other signals [3]. The adaptation process may be located in different places such as in the application, in the system, or in a specific *middleware*. The middleware approach has the advantage of software reuse since already existing applications, systems and protocols need little or no modification at all. Usually, a middleware designed for wireless environment would be divided in two components – one in the device and another one in a point between the device and its peer on the wired network (for instance, the base station). An example of middleware architecture is shown in Figure 1.1.

This type of middleware could be used to alter, remove, cache or serve multimedia content to mobile devices, that is, it is possible to perform different adaptation schemes. For instance, images can be removed from emails sent to users before transmitting them

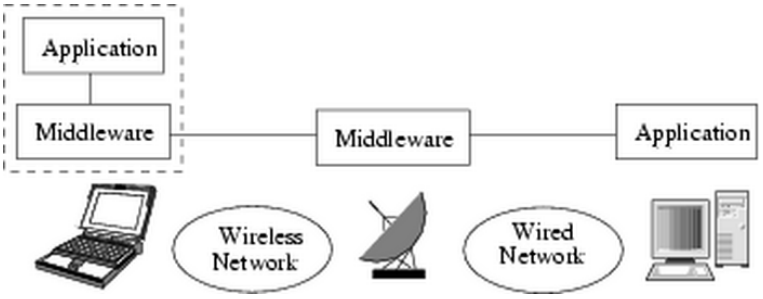


Figure 1.1: Example of a typical middleware architecture used for adaptation in wireless environments.

over the wireless channel, video quality can be altered to fit screen size, frequently accessed data can be cached next to users, location dependent data can be served to users inside a specific area, etc.

Most applications for handheld device are developed for PalmOS and Windows operating systems (e.g., PocketPC, Windows CE). In both cases, these operating systems mimic the traditional interaction concepts of desktop computers such as icons, windows, and menus. As computers and the Internet become more popular, the acceptance of this interface concept also increases. In fact the World Wide Web, probably the most popular Internet application, uses the direct manipulation interface based on this concept. Thus, it is natural to assume that the WWW will become a popular application for mobile environments as well.

In general, the common format of documents in the Web is HTML and its variants (ASP, JSP, PHP, XML, DHTML and others). The term *variant* does not refer to the functionality of each type of document, but to them being represented as text files using verbose and similar languages (in this paper, the term HTML will refer to all these formats). Compressing one document prior to its transmission over a wireless environment seems to be an obvious adaptation solution [5, 6] if the only resource intended to be saved is bandwidth. Compression can also be applied to images and videos, since different formats compensate quality for data size. For instance, if a BMP image is detected it can be converted to a JPEG of GIF image before sending it over the wireless interface. On the other

hand, compressed files need to be compressed and decompressed which may increase total response time and may consume more energy than uncompressed transmissions. Suppose the following scenario: a Web server accessed by different clients through a wireless channel (figure 1.2). As each device has its own features and these features vary highly in this heterogeneous environment, the decision to compress one file is not a simple task and should be done carefully by the server. For some application/device combination compression can be worthwhile, while the same application in a different device might perform better with no compression at all.

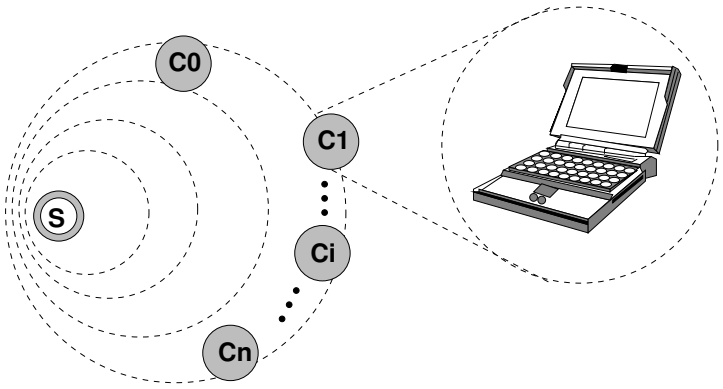


Figure 1.2: S is a Web server and Ci's are the requesting clients.

This work studies the conditions in which compression should happen and how it should be done when dealing with HTML file transmissions. The focus is on HTML files, but the technique developed here could be extended to images and videos as well. So, this work intends to:

1. study which compression method is more adequate to be used in mobile devices for HTML file transmission;

2. propose an adaptive model that predicts when compression of HTML files should happen;

3. evaluate the performance of the proposed model through simulation and experiments using two different communication protocols: IEEE 802.11 and Bluetooth.

The rest of this text is organized as follows: in Chapter 2 related work is briefly described. In Chapter 3 the advantages and disadvantages of the most popular compression methods are analyzed and the most adequate compression algorithm to be used in mobile devices is selected. The proposed adaptive model is presented in Chapter 4. In Chapter 5 both communication protocols used in this work are presented, namely IEEE 802.11 and Bluetooth (Section 5.1), the deployed scenario is explained (Section 5.2) and the results of experiments and simulations are presented (sections 5.3 and 5.4, respectively). Chapter 6 describes the implementation of a proxy which uses the adaptive model. Conclusions and future work are presented in Chapter 7.

Chapter 2

Related work

According to [3], *adaptability* is the system ability to configure itself dynamically in order to take advantage of the environment it is inserted in. It is assumed that the system has means to get information about environmental conditions. When one of these conditions changes, an adaptation process should take place so the better solution/answer is achieved. Although not restricted to the wireless world, adaptation fits better to this environment due to its intrinsic characteristics.

Adapting an application means changing its normal behavior in any of its functional modules, going from higher layers such as user interface to lower layers in the communication procedures. This last module – communication – plays a key role on distributed applications and it is one of the most affected modules when a change in the environmental conditions occurs. In order to react to these changes, adaptation techniques may be applied/inserted in any of the several layers that compose the *network access stack*. Next, some adaptation processes applied to wireless and wired network protocols that attempt to improve Web-like access experience through compression, which is the object-study application of this work, are reviewed.

Fox et al. [7, 8] proposed a proxy architecture to apply data-specific distillation (lossy compression) in a variety of applications. Besides the development of their own Web-

proxy – *TranSend* – and its evaluation, their work has the merit to argue in favor of proxy architecture as being the most effective and extensible technique to handle client and server heterogeneity. The programming model presented is based on *workers*, a set of task-specific elements specialized in operations such as transformation (distillation, filtering, etc.), aggregation (collecting and collating data from various sources), caching (both original and transformed content), and customization (change data to user specific needs and preferences).

Housel, Samaras, and Lindquist [9] presented WebExpress, a split-proxy design transparent to clients and servers, which intercepts HTTP data streams and performs several optimizations, including file-caching, forms differencing, protocol reduction, and elimination of redundant HTTP header transmission.

Steinberg and Pasquale [6] proposed a Web middleware architecture – WSC (Web Stream Customizer) – for mobile devices that allows users to customize their view of the Web for optimal interaction. Like WebExpress, WSC explores the proxies capabilities of HTTP to adapt the point-to-point traffic along the client-server path. It uses a simple adaptive model based solely on network characteristics to compress text data (lossless compression) and image data (lossy compression).

Jeannot, Knutsson and Björkman [10, 11] proposed an adaptation model that intercepts an outgoing TCP/IP traffic and applies compression over small chunks of data. By implementing a queue between TCP/IP sockets and the sender application, the interception process occurs without major changes in the system. The authors showed how their model can adapt dynamically to the varying network conditions and processor resources through experiments, yielding especially good results for large amounts of data transmitted over wired networks. The measurement of the available bandwidth is done by analyzing the behavior of the queue, that is, how fast it shrinks or grows. Although the model can be applied to wireless transmissions, it does not consider special features of this environment, such as energy consumption of client side application. Moreover, their model only yields

good results for transmission of large amounts of data, which is not the case for mobile HTTP browsing.

Mogul et al. [5] used data compression and *delta-encoding* to minimize the amount of data transmitted and response time. Delta-encoding is a technique to update cached documents by sending only the information about portions of the page which have changed. They propose specific extensions to the HTTP protocol to achieve these goals. [12, 13, 14] try to reduce the amount of data transmitted to mobile devices by extracting relevant information of HTML pages. Their goal is to expose the mobile users, whose devices have screen size limitations, to a minimum but relevant set of information each time it performs a new request. The techniques applied by them involve removing HTML and data components such as pop-ups, images and extraneous links, and text summarization, which means using HTML structure to extract the most relevant data.

On the other hand, there are several proposals that focus on lower layer modifications to improve response time or energy consumption of mobile devices.

Jung and Burleson demonstrated in [15] the advantages of using compression through VLSI (*very-large-scale integration*) hardware components to increase effective bandwidth in wireless communication systems. They modelled and simulated a real-time, low-area, and low-power VLSI lossless data compressor based on the first Lempel-Ziv algorithm [16] to improve the performance of wireless local area networks. Simulations were made with IEEE 802.11 protocol in which only the payload of MAC layer packets were compressed before transmission. The results showed that the architecture can achieve an average compression rate of 50 Mbps consuming approximately 70mW in 1.2u CMOS, which is an ideal scenario for WLAN use. This means this compression scheme can improve throughput and delay of a network while minimizing average power consumption.

Lilley et al. [17] described a TCP/IP header compression framework. This model included a simple, platform-independent header description language and a platform-specific code generation tool. Kranshinsky and Balakrishnam [18] investigated the interaction be-

tween the Power-Saving Mode (PSM) of IEEE 802.11 and the performance of TCP transfers under typical Web browsing workloads. It is proposed another PSM that dynamically adapts to network activity and simultaneously reduces energy consumption and response time.

The work presented here differs from those described above in two points: first, it is being proposed a platform-independent adaptive model that predicts when a HTML file should be compressed before its transmission. It is not a static application and it could be used in a proxy-design middleware to achieve better performance, for instance. Second, both transmission time and decompression time are analyzed and taken into consideration. Many proposals have ignored this overhead and, as the results will show this is a crucial factor for determining response time perceived by users and energy consumption in each mobile device. Third, the compression method used in this work was selected based on results evaluated over specific HTML collections. Many works have disregarded this step selecting compression algorithms that are widely known instead of algorithms that yields the best result for HTML file transmission.

Chapter 3

Compression methods

Over the past years, mass storage systems and network transfer rates have increased regularly. At the same time though the demand for these items has grown proportionally or even higher as each day more and more data are created and this huge volume of data needs to be handled efficiently – storing, recovering, and transmitting it. In many cases, the solution to cope with such an amount of information is to use *compression*. The advantage is twofold: it provides both lower transmission times and less storage space, thus satisfying the needs of users and information providers.

Compression methods can be divided in two main categories: lossless (mainly used for text compression) and lossy (used for general data compression). *Lossless* compression involves changing the representation of a file, yet the original copy can be reconstructed exactly from the compressed representation [19]. *Lossless* compression is a specific branch of the more general *lossy* compression methods, which are methods that tolerate the insertion of noise or small changes during the reconstruction of the original data. They are usually used in images and sound compression or any other digital data that is an approximation of an analog waveform.

The theory of data compression was formulated by Claude E. Shannon in 1948 [20] when he established the fundamental limit to lossless data compression. Shannon established

that this limit is determined by the statistical nature of the information source and called it the *entropy rate* - denoted by *H*. He then proved mathematically that it is impossible to compress the information source in a lossless manner with compression rate higher than *H*.

Shannon also developed the theory of lossy data compression, also known as *rate-distortion theory*. In lossy data compression, the decompressed data does not have to be exactly the same as the original data, that is, some amount of distortion, *D*, is tolerated. Shannon showed that, for a given source (with all its statistical properties previously known) and a given distortion measure there is a function *R(D)*, called the *rate-distortion function*, that gives the best possible compression rate for that source. The theory says that if *D* is the tolerable amount of distortion, then *R(D)* is the best possible compression rate. If the allowed distortion D is set to 0, then compression becomes lossless and the best compression rate *R(0)* is equal to the entropy H (for a finite alphabet source). The conclusion is that *lossless* compression ($D = 0$) is a specialization of the more general *lossy* compression ($D \geq 0$).

Lossless data compression theory and rate-distortion theory are known collectively as *source coding theory*. Source coding theory sets fundamental limits on the performance of all data compression algorithms. The theory in itself does not specify exactly how to design and implement these algorithms. It does, however, provide some hints and guidelines on how to achieve optimal performance.

In the next section some of the best-known compression algorithms used today will be briefly reviewed. In section 3.2 results of compression methods applied to different data sets will be shown. In the next chapter these results will be used to create an adaptive model to predict when a file should be compressed before its transmission over a wireless channel.

3.1 Compression Algorithms

Two strategies are used to design text compression algorithms: *statistical* or *symbolwise* methods and *dictionary* methods. *Symbolwise* methods work by estimating the probabilities of symbols (often characters) occurrences, and then coding one symbol at a time using shorter codewords for the most likely symbols. *Dictionary* methods achieve compression by replacing words and other fragments of text with an index to an entry in a dictionary [19].

The key distinction between symbolwise and dictionary methods is that symbolwise methods generally base the coding of a symbol in the context in which it occurs, whereas dictionary methods group symbols together, creating a kind of implicit context. In the first scheme, fixed-length blocks of bits are encoded by different codewords; in the second one, variable-length segments of text are encoded. This second strategy often provides better compression ratios. Hybrid schemes are also possible, in which a group of symbols is coded together and the coding is based on the context in which the groups occurs. This does not necessarily provide better compression but it can improve speed of compression.

Many compression methods have been invented and reinvented over the years. One of the earliest and best-known methods of text compression for computer storage and telecommunications is Huffman coding. Huffman coding assigns an output code to each symbol, with the output codes being as short as 1 bit or considerably longer than the input symbols, strictly depending on their probabilities. The optimal number of bits to be used for each symbol is the $log_2(\frac{1}{p})$, where p is the probability of a given character inside the text being compressed. Moreover, Huffman algorithm uses the notion of **prefix code**. A prefix code is a set of words containing no word that is a **prefix** of another word of the set. The advantage of such code is that decoding is immediate. However, there are two problems with Huffman coding: first, it relies on the fact that symbols have to be represented as integral number of bits codes. For example, a character with optimum number of bits to code equal to 1.6 would be coded with 1 or 2 bits. Either choice leads to

a longer compressed data as each code has influence in the other ones. The second problem is that the probability function generally is not known from the beginning, demanding an extra pass through the input file to estimate it.

Despite its problems, Huffman code was regarded as one of the best compression methods from its first publishment (in the early 1950s) until the late 1970s, when *adaptive compression* allowed the development of more efficient compressors. Adaptive compression is a kind of dynamic coding where the input is compressed relative to a model that is constructed from the text that has just been coded. Both the encoder and decoder start with their statistical model in the same state. Each of them process a single character at a time, and update their models after the character is read in. This technique is able to encode an input file in one single pass over it and it is able to compress effectively a wide variety of inputs rather than being fine-tuned for one particular type of data. Ziv-Lempel method and arithmetic coding are examples of adaptive compression.

Ziv-Lempel methods are adaptive compression techniques that have good compression yet are generally very fast and do not require large amounts of memory. *Ziv-Lempel* algorithms compress by building a dictionary of previously seen strings, grouping characters of varying lengths. The original algorithm did not use probabilities - strings were either in the dictionary or not, and to all strings the same probability is given. Some variants of this method, such as *gzip*, use probabilities to achieve better performance. The algorithm can be described as follow: given a specific position in a file, look for a previous position in the file that matches the longest string starting at the current position; output a code indicating the previous position; move the current header position over the string coded and start again. *Ziv-Lempel* algorithms were described in two main papers published in 1977 [16] and 1978 [21] and are often referred to as LZ77 and LZ78. These two version differ in how far back the match can occur: LZ77 uses the idea of a sliding window; LZ78 uses only the dictionary. Some well known variants of LZ77 are *gzip* and *zip*, whereas Unix *compress* is based on LZ78. In 1984, Welch [22] proposed a series of improvements over

LZ78, creating the so called LZW algorithm, one of the most popular compressors today, used in formats such as GIF and TIFF and is also part of PostScript Level 2 [1].

Arithmetic coding is an enabling technology that makes a whole class of adaptive compression schemes feasible, rather than a compression method of its own. This technique has made it possible to improve compression ratios, though compression and decompression processes are slower - several multiplications and divisions for each symbol are needed - and more memory must be allocated during processing. Arithmetic coding completely bypasses the idea of replacing an input symbol with a specific code. Instead, the idea is to represent the input string by one floating-point number n in the range [0..1]. In order to construct the output number, the symbols being encoded have to have a set of probabilities assigned to them. Then, to each symbol is assigned an interval with size proportional to its respective probability. The algorithm works as follow: Starting with the interval [0..1], the current symbol determines which subinterval of the current interval is to be considered. The subinterval from the coded symbol is then taken as the interval for the next symbol. The output is the interval of the last symbol. Implementations write bits of this interval sequence as soon as they are certain. The longer the input string is, the more numbers after the floating point are needed to represent it.

Some of the best compression methods available today are variants of a technique called *prediction by partial matching* (PPM), which was developed in the early 1980s. PPM relies on arithmetic coding to obtain good compression performance. Since then there has been little advance in the amount of compression that can be achieved, other than some fine-tuning of the basic methods. On the other hand, many techniques have been discovered that can improve the speed or memory requirements of compression methods. One exception is Burrows-Wheeler Transform or BWT. BWT method was discovered recently by Burrows and Wheeler in [24] and it is implemented in *bzip*, one of the best text compressor available currently. It can get ratio performance close to PPM, but runs significantly faster. The

[1] Unisys [23] holds the patent of LZW

BWT is an algorithm that takes a block of data and rearranges it using a sorting algorithm. The resulting output block contains exactly the same data elements that it started with differing only in their ordering. The transformation is reversible, meaning the original ordering of the data elements can be restored with no loss of fidelity. At last, the sorted block is passed to a entropy encoder, typically Huffman or arithmetic encoder.

3.2 Performance Evaluation

This section presents the results of different compression algorithms applied to different data sets. The goal here is to analyze which compression methods are suitable to be used on wireless transmissions of Web files. Four different collections are used in these experiments. The Calgary and Canterbury Collections are well known data sets used to analyze performance of compression algorithms. In order to have a more realistic measure, it was formed a new collection by crawling 9513 files from the Web. For the sake of comparison, it was also measured compression performance for XML collections. The first XML collection is the National Library of Medicine public collection [25]. The second one was formed by all XML files hosted in the Sigmod [26] Web site.

Table 3.1 summarizes all different compression algorithms tested.

Table 3.1: Description of the compression algorithms tested.

Algorithm	Description
LZW	The original LZW algorithm
COMP1	Arithmetic coding
COMP2	Improved Arithmetic coding
HUFF	Huffman coding
LZ77	The original LZ77 algorithm
LZARI	Improved LZ77
LZSS	Improved LZ77
PPMC	PPM compressor
GZIP	LZSS variant
BZIP2	Burrows-Wheeler algorithm
XMILL	XML specific compressor

3.2.1 Calgary and Canterbury Corpus Collections

The Calgary Corpus collection was developed in 1987 by Ian Witten, Timothy Bell and John Cleary for their research paper on text compression modeling [27] at University of Calgary, Canada. During the 1990s it became the standard *benchmark* for lossless compression evaluation. The collection is now rather dated but it is still reasonably reliable. Table 3.2 has the details.

Table 3.2: Description of Calgary Corpus files.

Index	File	Category	Size (bytes)
1	bib	Bibliography (refer format)	111261
2	book1	Fiction book	768771
3	book2	Non-fiction book (troff format)	610856
4	geo	Geophysical data	102400
5	news	USENET batch file	377109
6	obj1	Object code for VAX	21504
7	obj2	Object code for Apple Mac	246814
8	paper1	Technical paper	53161
9	paper2	Technical paper	82199
10	paper3	Technical paper	46526
11	paper4	Technical paper	13286
12	paper5	Technical paper	11954
13	paper6	Technical paper	38105
14	pic	Black and white fax picture	513216
15	progc	Source code in "C"	39611
16	progl	Source code in LISP	71646
17	progp	Source code in PASCAL	49379
18	trans	Transcript of terminal session	93695

Nine different types of text are represented in this collection. For better results some types have more than one representative file. Normal English, both fiction and non-fiction, is represented by two books and papers. More unusual styles of English writing are found in a bibliography and a batch of unedited news articles. Three computer programs representing artificial languages and a transcript of a terminal session are included. Some non-ASCII files are also included: two files of executable code, some geophysical data, and a bit-map black and white picture. The geophysical file is particularly difficult to compress because it contains a wide range of data values while the picture file is highly compressible

because of large amounts of white space in the picture represented by long runs of zeros.

Developed in 1997, the Canterbury Corpus is an improved replacement for the Calgary Corpus. Today, this collection is the main benchmark for comparing compression methods. The main files in the collection are listed on table 3.3.

Table 3.3: Description of Canterbury Corpus files.

Collection	Index	File	Abbrev	Category	Size
main	1	alice29.txt	text	English text	152089
	2	asyoulik.txt	play	Shakespeare	125179
	3	cp.html	html	HTML source	24603
	4	fields.c	Csrc	C source	11150
	5	grammar.lsp	list	LISP source	3721
	6	kennedy.xls	Excl	Excel Spreadsheet	1029744
	7	lcet10.txt	tech	Technical writing	426754
	8	plrabn12.txt	poem	Poetry	481861
	9	ptt5	fax	CCITT test set	513216
	10	sum	SPRC	SPARC Executable	38240
	11	xargs.1	man	GNU manual page	4227
artificial	12	a.txt	a	The letter 'a'	1
	13	aaa.txt	aaa	The letter 'a', repeated 100,000 times.	100000
	14	alphabet.txt	alphabet	Enough repetitions of the alphabet to fill 100,000 characters	100000
	15	random.txt	random	100,000 characters, randomly selected from [a-z—A-Z—0-9—!—]	100000
large	16	E.coli	E.coli	Complete genome of the E. Coli bacterium	4638690
	17	bible.txt	bible	The King James version of the bible	4047392
	18	world192.txt	world	The CIA world fact book	2473400
misc	19	pi.txt	pi	The first million digits of pi	1000000

All files were chosen because their results on existing compression algorithms are "typical", therefore results should hold true for new methods. Additionally, four different collections are made available: Calgary Corpus, Artificial Corpus, Large Corpus, and Miscellaneous. The Calgary collection is provided for historic interest. The Artificial collection contains files for which the compression methods may exhibit pathological or worst-case

performance – files containing little or no repetition (e.g. random.txt), files containing large amounts of repetition (e.g. alphabet.txt), or very small files (e.g. a.txt). The Large Corpus is a collection of relatively large files. While most compression methods can be evaluated satisfactorily on smaller files, some require very large amounts of data to get good compression, and some are so fast that the larger size makes speed measurement more reliable. The Miscellaneous Corpus is a collection of "miscellaneous" files that is designed to be added to by researchers and others wishing to publish compression results using their own files.

Figure 3.1 plots the compressed ratios of Calgary Corpus files. Table 3.4 at the end of the chapter shows each point individually. As it can seen, PPM, arithmetic (COMP2) and bzip2 methods had the best performances, followed by gzip and other LZ-like methods. Huffman had the worst performance. This is an expected behavior, as explained in Section 3.1.

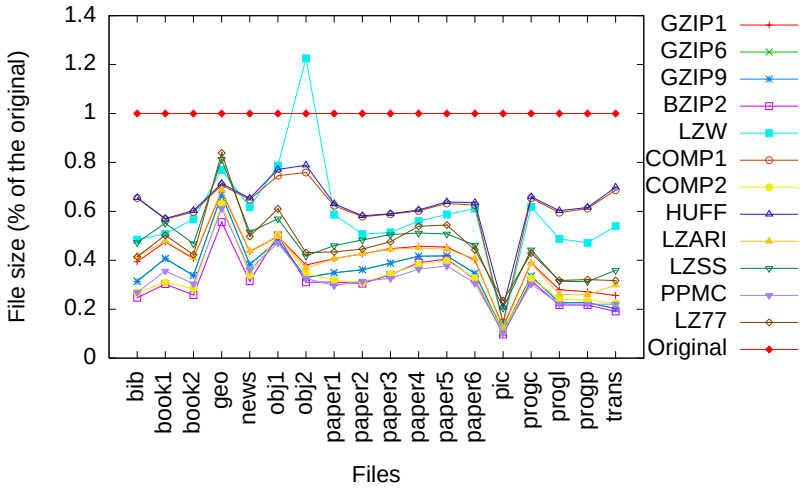
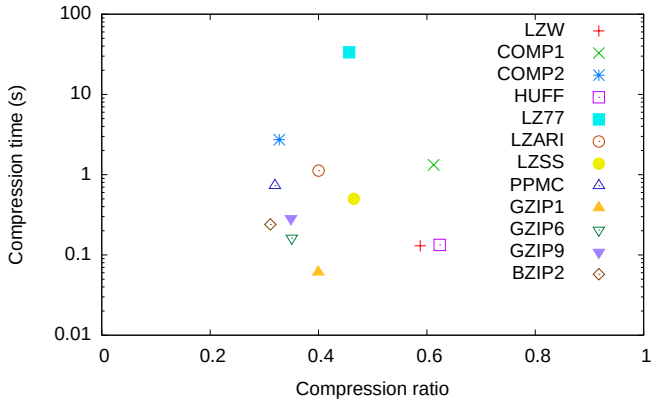


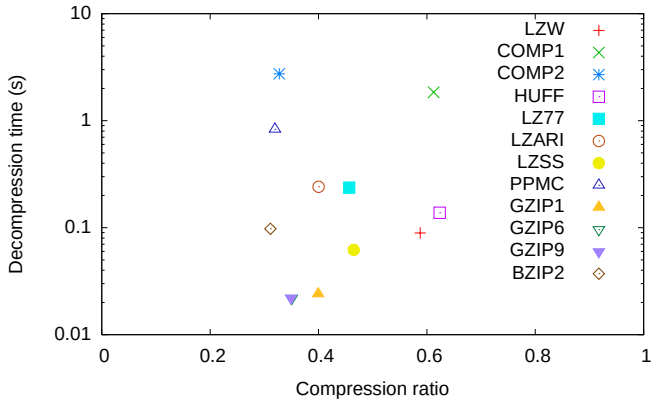
Figure 3.1: Compressed sizes of Calgary Corpus files.

When dealing with compression in wireless environments it is important to assess the impact of compression and decompression processes over the total response time perceived by users, as these time overheads can overcome the optimization achieved by reducing transfer time. In these systems compression algorithms with the best *compression*

time/compression ratio tradeoff should be used. Figure 3.2(a) plots tradeoff between average compression time and average compression ratio for all Calgary Corpus files and for each compression algorithm analyzed. Figure 3.2(b) does the same analysis for average decompression time.



(a) Compression



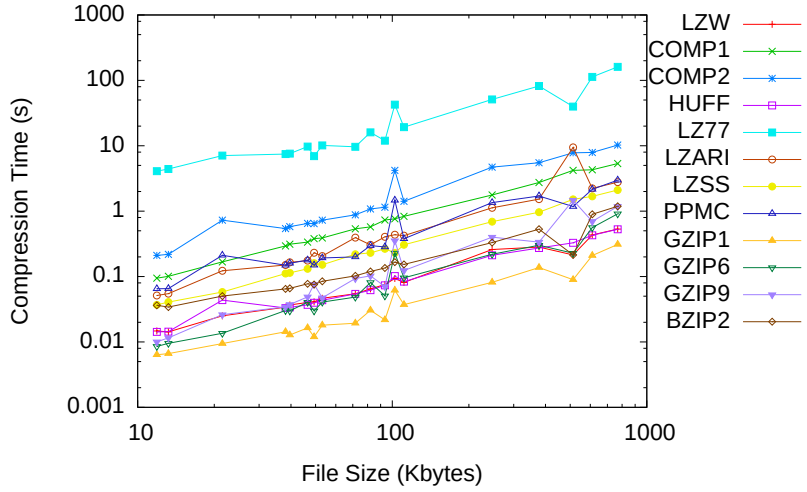
(b) Decompression

Figure 3.2: Compression and decompression times versus Compression Ratio (Calgary Corpus).

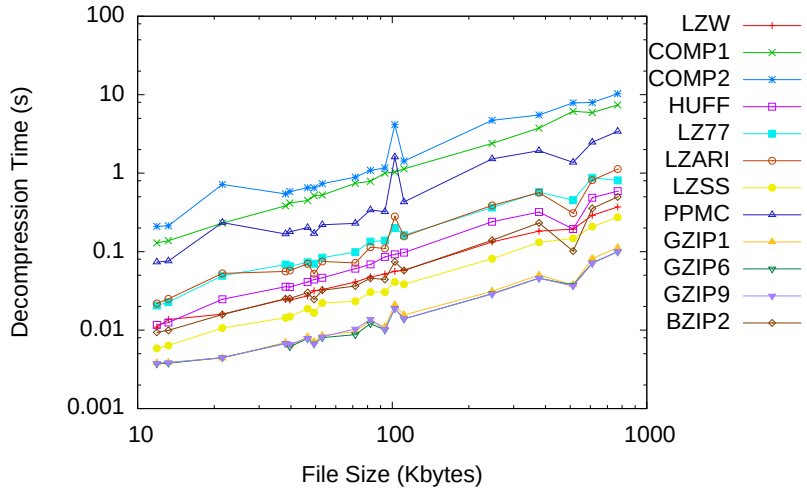
In both figures, gzip method yields the best time-ratio tradeoff. Gzip receives as input a factor which determines how much effort should be made to compress a file. This factor may vary from 1 through 9. In the previous experiments gzip was evaluated with compression factors of 1 (less effort), 6 (default value) and 9 (more effort). Although decompression time is approximately the same for these three versions, compression time and ratios show high variations, from what it can be concluded that demanding less effort on gzip may be the

best choice. Although PPM and BZIP2 yields good compression rates, their compression and decompression times are almost one order of magnitude higher than gzip.

It is important to notice that compression and decompression times grow proportionally to file size (figures 3.3(a) and 3.3(b)). This is an important aspect of compression once it can be used to predict, with certain precision, how long it will take to compress or decompress a file. To make such a prediction, one must perform a linear regression on the values obtained and create a formula that calculates time according to each file size. This subject will be explored again on chapter 4.



(a) Compression time.



(b) Decompression time.

Figure 3.3: Compression and decompression times for the Calgary collection.

Figure 3.4 show compression ratios for Canterbury Corpus files. Again PPM, arithmetic and bzip2 methods had the best global performance. Three files presented an anomalous behavior due to their natural properties: *a*, *aaa* and *alphabet*. They are either too small or have an enormous amount of repeated information. In both cases some algorithms may present unexpected behavior. As this is not the case for most files in Internet, these values can be disregarded. Table 3.5 at the end of the chapter shows all values individually.

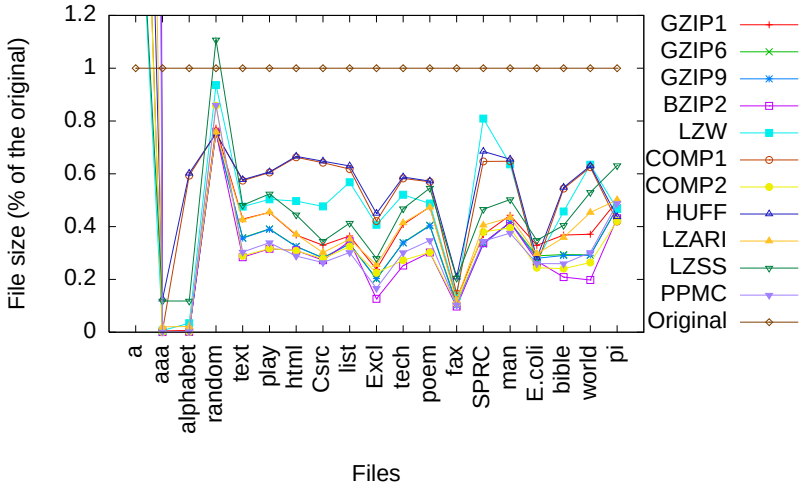
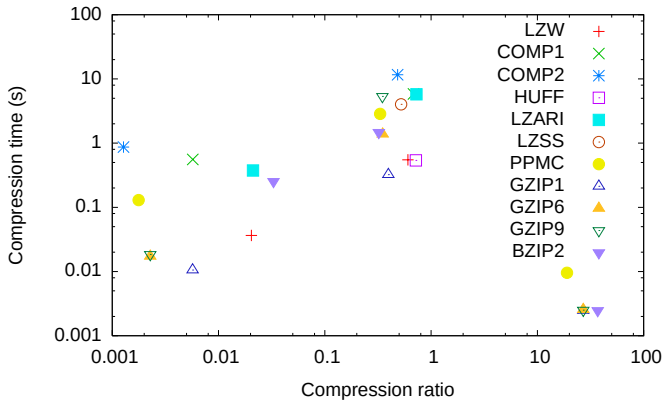


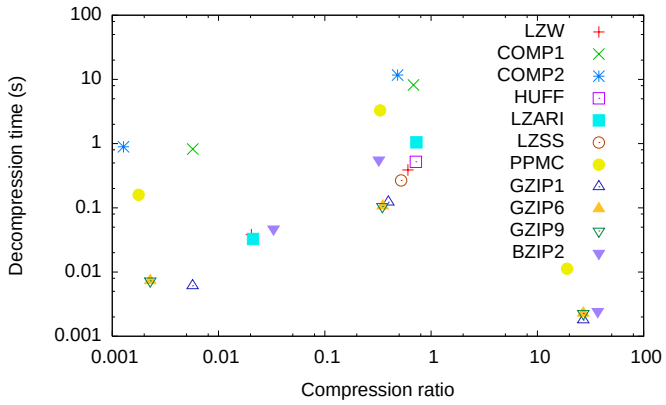
Figure 3.4: Compressed sizes of Canterbury Corpus files.

Figures 3.5(a) and 3.5(b) compare compression and decompression times with compression ratios and again gzip had the best tradeoff. As said before, files in the Artificial subset do not have a regular structure when compared to the other files in the collection. When compressing these files some compression algorithms present anomalous behavior, creating undesirable outliers. Files that are either too small or present an enormous amount of repeated information can lead to such a result. For this reason three different sets of points were plotted on those figures: first group (to the left) represent all algorithms that yields compression ratios smaller than 0.1 for files named *aaa* and *alphabet*; second group (to the right) represents all algorithms that yields compression ratios higher than 10 for the file named *a*; the third group (center) represents all algorithms that yields compression ratios between 0.1 and 10 for all files. Each point represents the average of all measures, so it is

reasonable to assume that this is the mean compressing ratio of each algorithm.



(a) Compression



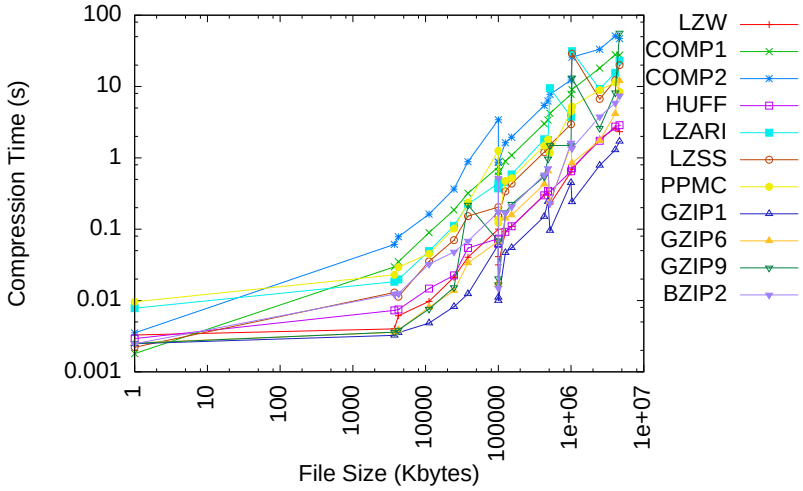
(b) Decompression

Figure 3.5: Compression and Decompression times versus Compression Ratio (Canterbury Corpus).

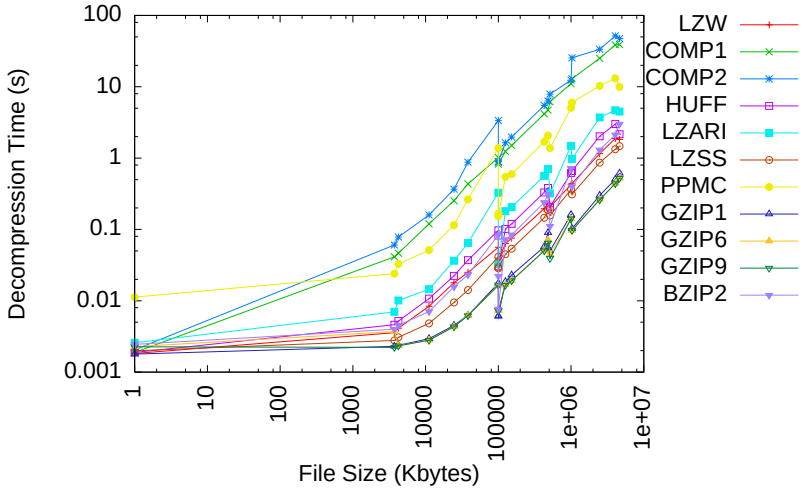
Figures 3.6(a) and 3.6(b) plot compression and decompression times versus file size. As Canterbury Corpus has irregular files, the proportion of both compression and decompression times versus file size are affected by them. However, it is still possible to determine such proportion by just ignoring or amortizing the effects of these outliers.

3.2.2 HTML Files

The HTML collection was formed by crawling 9513 files from the Web. Although the files have different types (HTML, DHTML, ASP, JSP, PHP) they all can be described as text files written in some mark-up language and may contain both natural language text



(a) Compression time.



(b) Decompression time.

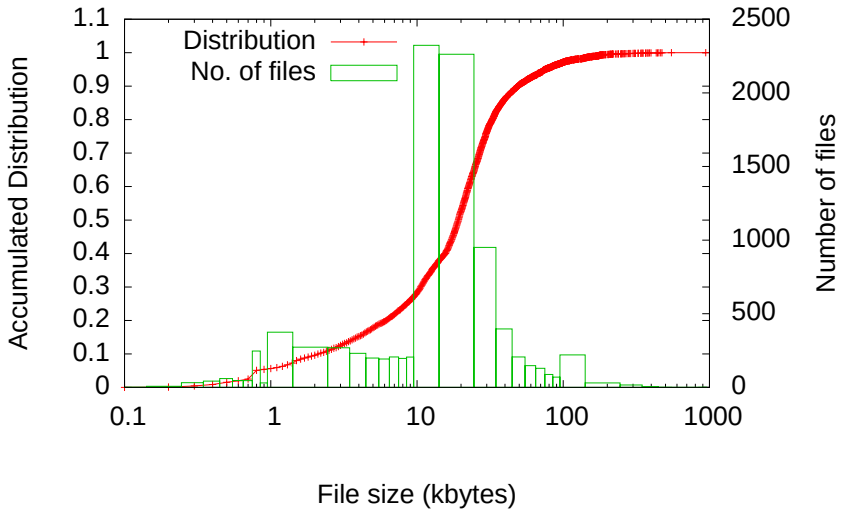
Figure 3.6: Compression and decompression times for Canterbury Corpus collection.

and programming language code. From now on, all these files will be referred as *HTML files*. HTML files account for more than 97% of all documents found in the Internet that are considered as textual information [28]. HTML files are exclusively textual, since their contents consist of formatting tags and the text itself. This kind of language compensates easiness of human reading with extremely verbose and large documents.

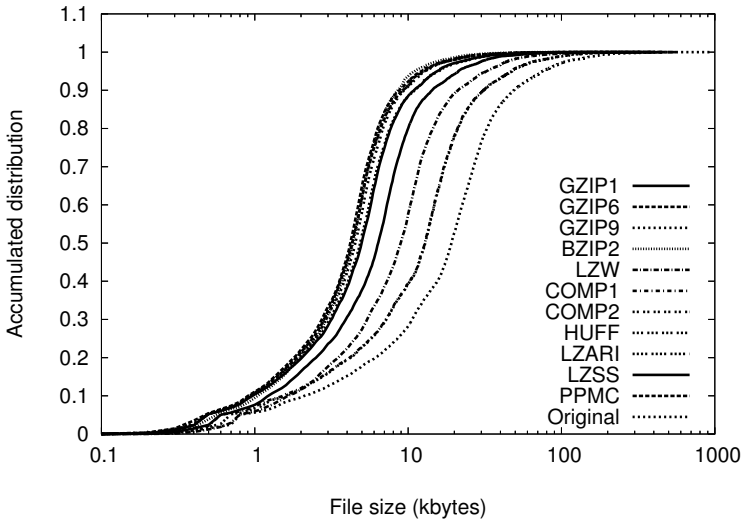
Performance of 11 algorithms was evaluated for this collection. The goal of this experiment is to measure how much a typical HTML can be compressed and to have an idea of how fast compression and decompression processes can be performed. The experiments in

the previous section gave us an idea of how these metrics are for text collections. As it will be seen now, HTML have a similar behavior.

Figure 3.7(a) gives the accumulated size distribution of all files in the collection and the amount of files in each range. As it can be seen, there is a concentration of files in the range of 10-30Kbytes. Figure 3.7(b) represents accumulated compressed size distribution for all algorithms tested.



(a) Accumulated distribution of the original HTML files.



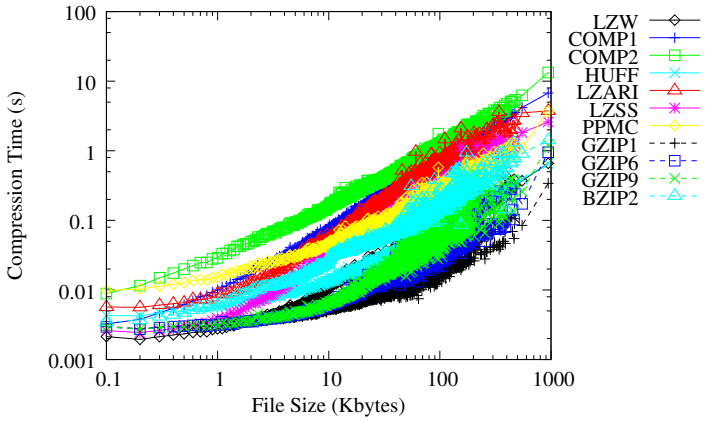
(b) Accumulated distribution of compressed HTML files.

Figure 3.7: Accumulated distribution for original and compressed files in the HTML collection.

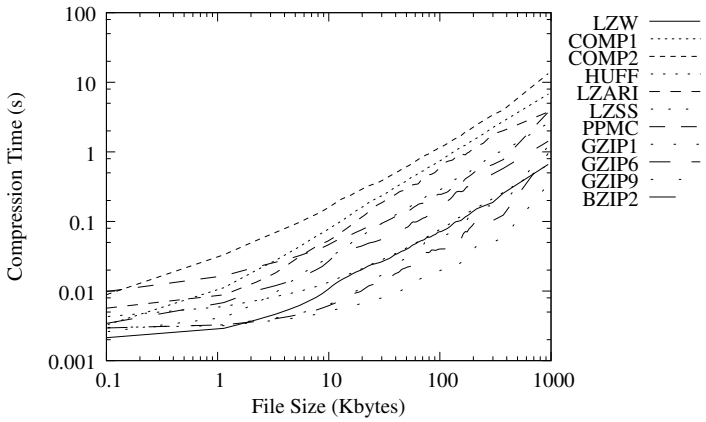
By doing the same experiments with this collection, the following results can be observed. Regarding small files (smaller than 5kbytes), PPM and arithmetic methods perform better than others. PPM, bzip2, arithmetic and gzip-9 methods achieve almost the same compression limit for larger files. Finally, although gzip-1 and gzip-6 can not compress as much as gzip-9, they compress and decompress faster than others, yielding a good time/ratio tradeoff. Figures 3.8(a) and 3.9(a) show compression and decompression times. The best way to visualize these results is plotting them using some kind of linear regression of curve fitting. Although Bezier curves are not used to create a function definition for the data, this curve fitting method serves the purpose of giving an approximation of the function line as it would be obtained through linear regression. Figures 3.8(b) and 3.9(b) shows the curve fitting for all algorithms tested and make it easy to observe how the performance of each compression technique vary according to file size.

Figures 3.10(a) and 3.10(b) illustrate compression and decompression times versus compression ratio, respectively. Instead of a point representing the average of all measures taken, it was plotted lines representing time versus compression ratio for the several ratios obtained with each algorithm. This way it is possible to see how compression ratio and compression/decompression time are related. An algorithm with good time/ratio tradeoff would generate a line with points concentrated in the lower left corner of the graph.

The time/size proportion for compressing and decompressing a HTML file is not the same for all sizes, which differs from the results obtained for Corpuses collections. For example, for gzip-6 algorithm (figure 3.11), it is possible to distinguish two different sets of points: one for files smaller than 10kbytes and another one for files larger than 10kbytes. In both cases, it is possible to estimate time/size relation by curve fitting or linear regression on those two set of points. In this case, two straight lines could be used to represent the function.



(a) Compression time (all points).



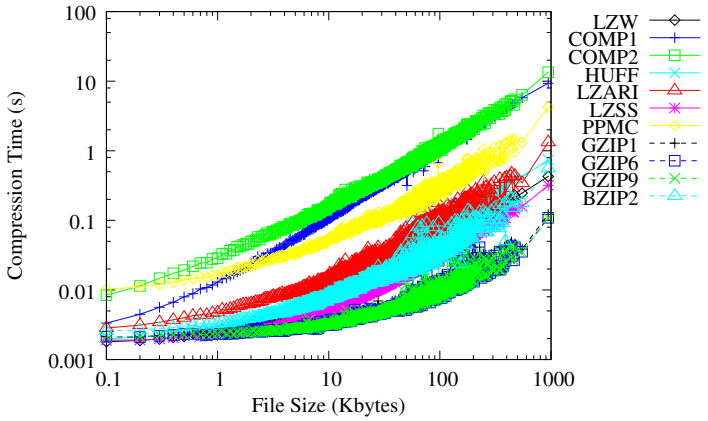
(b) Compression time (using a smooth bezier curve to plot all points).

Figure 3.8: Compression time versus file size (HTML Collection).

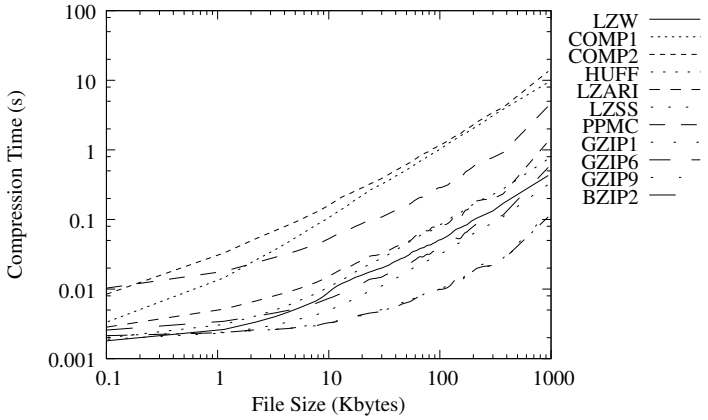
3.2.3 XML Files

The XML specification defines a standard way to add *markup* points to documents containing structured information [29]. Structured information contains both content (words, pictures, etc.) and some indication of what role that the content plays. XML specifies neither semantics nor a tag set and it is not a markup language *per se*. XML is really a meta-language for describing markup languages. In other words, XML provides a facility to define tags and the structural relationships between them.

XML was created so that richly structured documents could be used over the web, what could not be achieved efficiently by other markup language like HTML and SGML. Although very similar in many concepts, XML differs from these languages in many ways



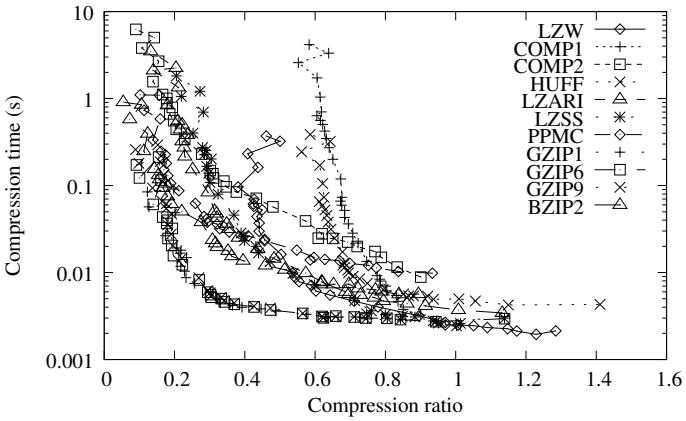
(a) Decompression time (all points).



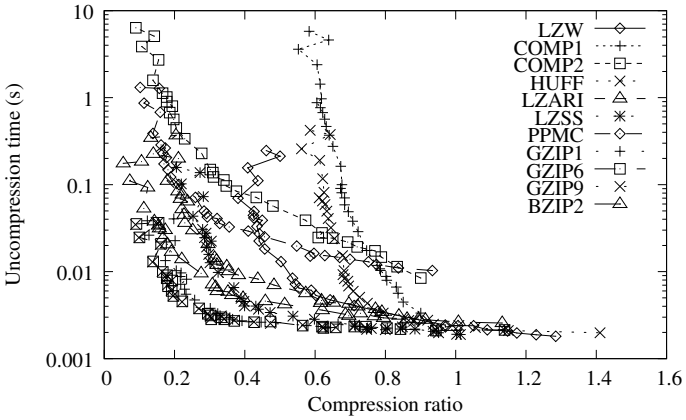
(b) Decompression time (using a smooth bezier curve to plot all points).

Figure 3.9: Decompression time versus file size (HTML Collection).

[29]. In HTML, both the tag semantics and the tag set are fixed, that is, no matter in which context they are, they will always have the same meaning. Moreover, HTML is not flexible enough to allow users to extend documents in their own way. SGML is the *Standard Generalized Markup Language* defined by ISO 8879. SGML has been the standard, vendor-independent way to maintain repositories of structured documentation. Although SGML provides arbitrary structure, it is not well suited to serve documents over the web since it is too hard to implement it on a (supposedly) simple and light web browser. The most obvious of these hard-to-implement features are the ones that were put into the standard years ago to minimize keystrokes in manual entry: omitted start- and end-tags, omitted quotes on attribute values, comments within other markup declarations, multiple



(a) Compression



(b) Decompression

Figure 3.10: Compression and Decompression times versus Compression Ratio (HTML Collection).

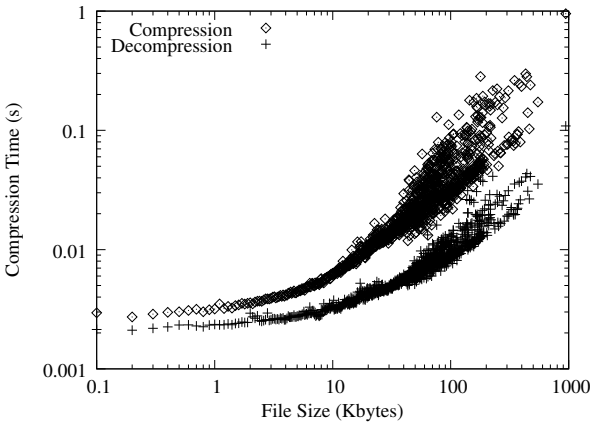


Figure 3.11: Compression and decompression times for HTML files and gzip-6 algorithm.

comments in a single comment declaration, and public identifiers. Full SGML systems should be used to solve large and complex problems that justify their expense.

Any file that follows the general markup rules can be a valid XML file. Most XML applications follow a more rigorous set of rules having their structure defined by a *Document Type Definition (DTD)* or by a *schema*, which provides even more information about the content than a DTD.

As in HTML, XML substantially increases the size of data files over the size when the same data is represented in raw format. By using compression, the impact of this inherent inflation can be minimized. Compression of XML files can be greatly increased when the schema is available. The schema allows the XML tags to be encoded with high efficiency. In addition to providing high compression of tag data, knowledge of the schema allows the element data to be encoded efficiently. Because schemas supply the data type information, compression routines optimized for specific data types can be used, providing extremely high compression ratios. If a file does not conform to the expected schema, the data is safely encoded using high-performance general-purpose coders.

For these experiments two different public available XML collections ([25] and [26]) were used. The first one is a medical collection, with articles describing medical procedures, diagnosis and general research. The later one is formed by the pages hosted in the ACM Sigmod Web site, all written in XML.

File size distribution of the first collection is represented in figure 3.12, along with size distribution of the same files compressed by different methods: PPM, gzip-1, gzip-9 and XMill. XMill is a XML data compressor that can take advantage of XML structure to provide higher compression ratios. This means XMills uses several compression algorithms and based on the data type represented in each XML field decides which algorithms should be used to compress the data. Regarding compression ratios, next experiments prove that XMill performs as good as PPM and outperforms both gzip versions.

Figures 3.13(a) and 3.13(b) shows that compression time of XMill is slightly better than gzip with level 9 compression effort. As a matter of fact, XMill uses gzip with compression level 6 to compress text data type. Besides, XMill takes an extra time to decide which

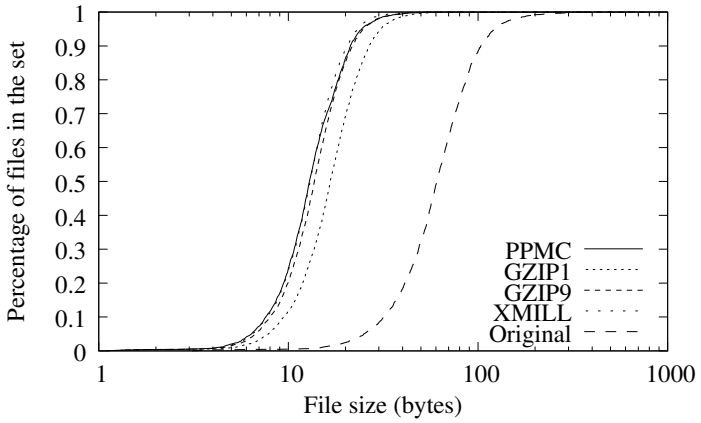


Figure 3.12: File size distribution of XML files.

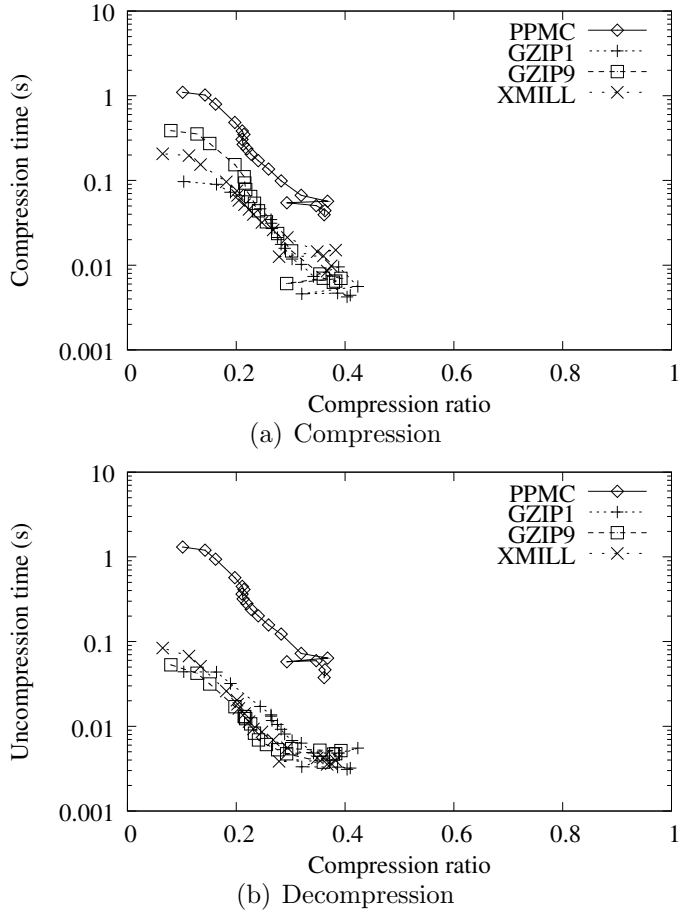
algorithms should be used to compress data. Although this is a minimum increase, it collaborates to make XMill a little slower than gzip-1. PPM expends much more time in both processes. It achieves the better compression ratios but consumes much more time to compress and decompress files.

The same measures for the second collection are presented next. As it can be seen on figure 3.14, most of the files in this collection have size in the 1Kb-10Kbytes range, while the the majority of files in the previous collection were in the 10Kb-100Kbytes range.

Compression time and compression ratio tradeoff are described in figure 3.15(a). Figure 3.15(b) relates decompression time and compression ratios.

Again gzip had the best compression time performance, although gzip and XMill are almost equivalent when decompressing those files. It is possible to observe the same behavior described for HTML files relating compression and decompression times with file sizes in these collection, although compressing and decompressing XML files presents a more regular proportion between file size and time than HTML files. As it can be verified in figures 3.16(a) and 3.16(b) for the medical collection and in figures 3.17(a) and 3.17(b) for the Sigmod collection, it is possible to predict, with a certain precision, how long it will take to compress or decompress an XML file on these collection by making a linear regression on those data points. This property is valid for all algorithms analyzed.

From all these experiments two conclusions can be drawn: first, the compression method



(a) Compression

(b) Decompression

Figure 3.13: Compression and Decompression times versus Compression Ratio (XML Collection).

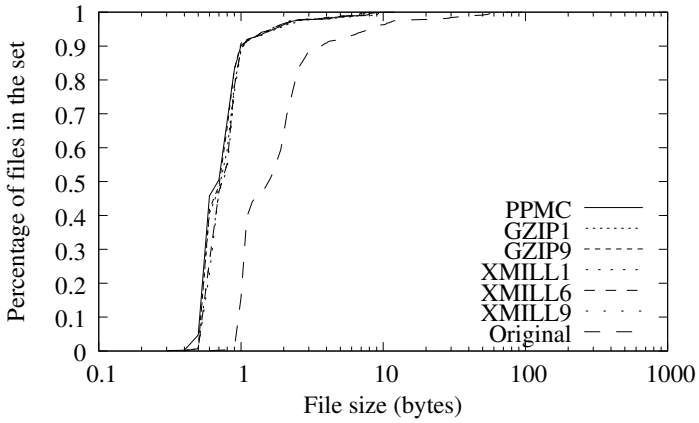
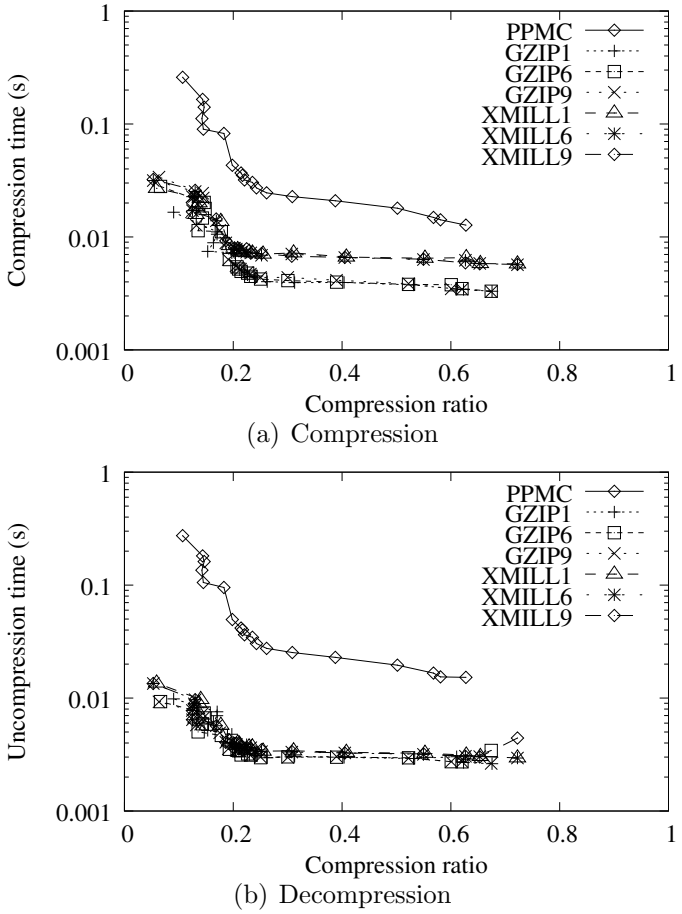


Figure 3.14: File size distribution of XML files.

that yields the best compression ratio may not be the most adequate one to be used for wireless transmission of compressed files. More important than the amount of bytes com-

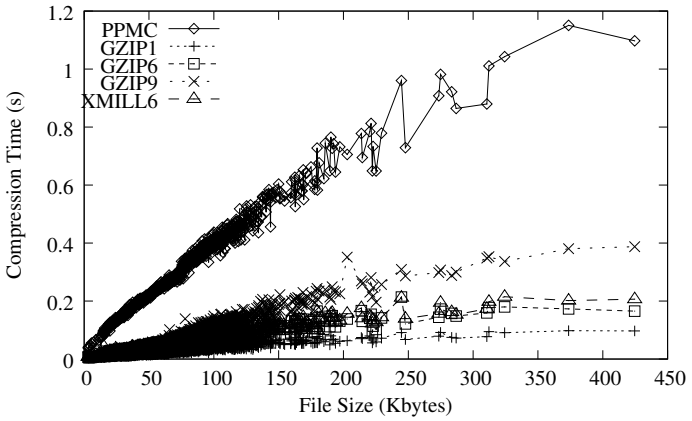


(a) Compression

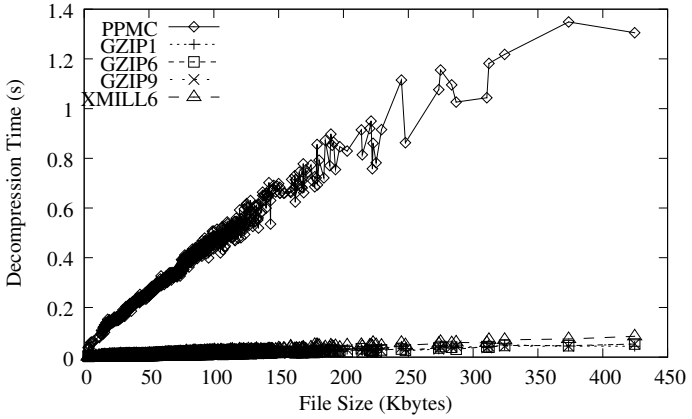
(b) Decompression

Figure 3.15: Compression and Decompression times versus Compression Ratio (XML Collection).

pressed is time spent to compress and decompress the file itself. Decompression may highly impact total transmission time and power consumption of a portable device. The second conclusion is that compression and decompression times can be estimated by evaluating the specific algorithm over a representative set of files no matter which algorithm was used. As it will be seen in the next chapter, this property plays an important role in the strategy proposed to transmit HTML files over wireless channels. Another important observation is that gzip and other LZ variant methods yield good performance in all cases studied: text, HTML and XML. Consequently it is not necessary to create or adapt the compression method each time the content type changes. Of course performance will not always be the best possible, but the results achieved will not be much worse than the best result possible.

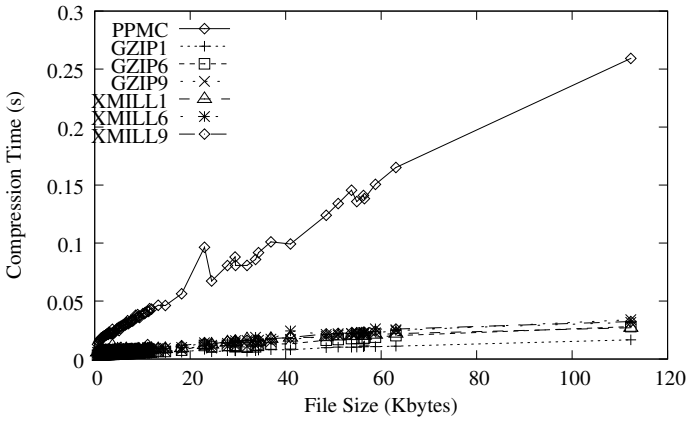


(a) Compression time.

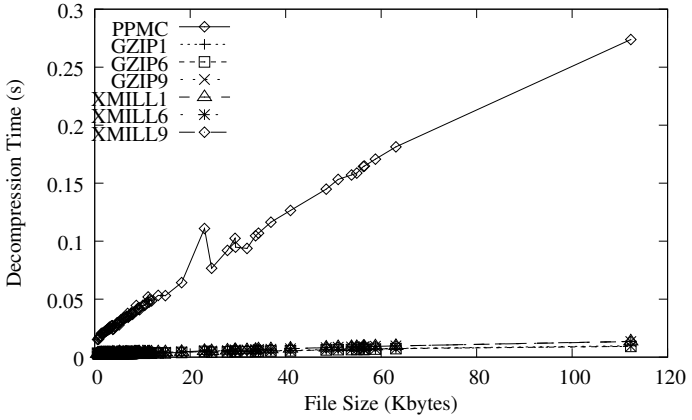


(b) Decompression time.

Figure 3.16: Compression and decompression times for XML medical Collection.



(a) Compression time.



(b) Decompression time.

Figure 3.17: Compression and decompression times for XML Sigmod Collection.

Table 3.4: Compressed size of Calgary Corpus files.

ARQ	ORIG	BZIP2	COMP2	PPMC	GZIP9	GZIP6	GZIP1	LZARI	LZ77	LZW	LZSS	COMP1	HUFF
paper5	11954	4837	4763	4507	4995	4995	5424	5321	6507	7017	6069	7559	7633
paper4	13286	5188	5081	4835	5534	5536	6073	5979	7160	7440	6798	7998	8043
obj1	21504	10787	10819	10135	10320	10323	10707	10717	13126	16925	12247	16038	16584
paper6	38105	12292	12455	11683	13213	13232	15282	15520	16865	23301	17609	23837	24230
progc	39611	12544	12896	12018	13261	13275	15455	15333	17007	24464	17531	25922	26111
paper3	46526	15837	16028	15154	18074	18097	20819	20693	22130	23916	23485	27391	27469
progp	49379	10710	11668	10903	11186	11246	13382	12770	15858	23285	15445	30207	30416
paper1	53161	16558	16895	15766	18543	18577	21612	21616	23076	31175	24467	33130	33550
progl	71646	15579	17354	15861	16164	16273	20038	18692	22753	34914	22521	42617	43181
paper2	82199	25041	25453	25871	29667	29753	35078	35098	36653	41664	39703	47540	47830
trans	93695	17899	21023	20668	18862	18985	23966	27979	29710	50543	33641	64340	65434
geo	102400	56921	64590	62673	68414	68493	69810	70385	85922	78753	83183	72405	73084
bib	111261	27467	29840	29915	34900	35063	43871	46222	46094	53844	52591	72792	72941
obj2	246814	76441	85465	80309	81087	81631	93906	90674	106484	302540	103002	187306	194635
news	377109	118600	128047	136068	144400	144840	164199	165143	187823	232808	194435	244496	246606
pic	513216	49759	55461	53022	52381	56442	65540	61301	120385	70215	105311	75066	106949
book2	610856	157443	174085	185883	206158	206687	248846	251451	258339	346529	285942	364788	368521
book1	768771	232598	237380	273629	312281	313376	365005	367576	386475	390807	424147	436926	438577

Table 3.5: Compressed size of Canterbury Corpus files.

ARQ	ORIG	BZIP2	COMP2	PPMC	GZIP9	GZIP6	LZARI	GZIP1	LZSS	LZW	COMP1	HUFF
a.txt	1	37	3	19	27	27	6	27	2	2	3	4
grammar.lsp	3721	1283	1209	1123	1246	1246	1311	1356	1537	2114	2299	2344
xargs.1	4227	1762	1670	1588	1756	1756	1837	1872	2124	2688	2737	2770
fields.c	11150	3039	3106	2942	3136	3143	3360	3674	3841	5313	7159	7224
cp.html	24603	7624	7594	7097	7981	7999	9124	9054	10941	12225	16299	16391
sum	38240	12909	14533	13187	12772	12924	15494	14134	17803	30938	24735	26196
aaa.txt	100000	47	104	130	141	141	2076	481	11808	671	572	12504
alphabet.txt	100000	131	151	224	315	315	2143	660	11834	3402	59307	60160
random.txt	100000	75684	85830	86011	75689	75689	75773	77301	110713	93615	75468	75315
asyoulik.txt	125179	39569	39657	42440	48829	48951	56916	56813	65551	62973	75733	75963
alice29.txt	152089	43202	43964	46220	54191	54435	64625	65144	73117	72318	87358	87864
lcet10.txt	426754	107706	116295	128762	144429	144885	176616	174132	199722	222206	248666	250759
plrabn12.txt	481861	145577	145725	167486	194277	195208	227933	228778	262943	234930	274478	275787
ptt5	513216	49759	55461	53022	52382	56443	61301	65541	105311	70215	75066	106949
pi.txt	1000000	431671	420891	487441	470439	470439	501087	497288	630986	466553	419257	437352
kennedy.xls	1029744	130280	231750	171160	209733	206779	263168	245037	288123	419235	433743	463129
world192.txt	2473400	489583	653089	746881	721413	724606	1121137	917896	1309270	1569210	1545481	1558814
bible.txt	4047392	845623	974496	1050937	1176645	1191071	1450583	1490038	1638297	1851170	2202081	2218604
E.coli	4638690	1251004	1132538	1204968	1299066	1341250	1378814	1526476	1613103	1254888	1176940	1302286

Chapter 4

Adaptive model

The first step to create the adaptive model was to define in which scenarios compression should be used. It is easy to conceive at least two ones: when compression yields a reduction in response time or in power consumption. The second step was selecting the parameters these properties depends on: bandwidth, transmitted file size, compression ratio, type of device where decompression is done, power consumption for transmission and processing, packet error ratio among others. Third, all these parameters were combined together to achieve the previously described goals – reduce time or power costs.

In order to predict when compression can generate some improvement either in response time or power consumption, the model shown in figure 4.1 is proposed. Modules T (time) and E (energy) have as inputs the many parameters (as described above) and each one uses its own analytical model to return one value, V_t and V_e respectively, indicating whether compression is worthy in each case. These values are passed to module C, which is responsible for making the final decision D_c. A "weight factor" – W_t and W_e – is associated to each module representing its reliability.

The values returned by each module are binary: 1 if there is some time/energy improvement and 0 if there isn't. The reliability factor values range from 0 to 1 (1 indicating

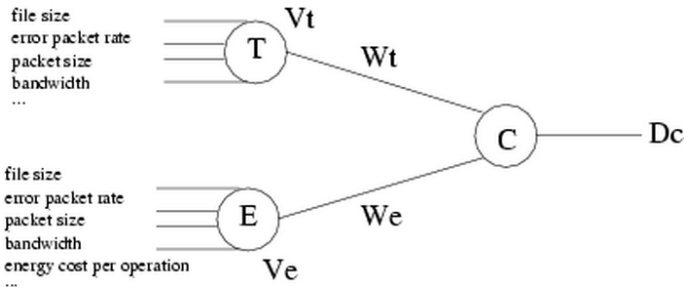


Figure 4.1: Predictive model.

the most reliable value). Then,

$$W_t, W_e \in [0, 1]$$

The final decision, D_c, is made through the use of simple mean formula.

$$D_c = \begin{cases} 0 & \text{, if } W_t = W_e = 0 \\ \frac{W_t \cdot V_t + W_e \cdot V_e}{2.0} & \text{, otherwise} \end{cases}$$

A file is transmitted in its original form when $D_c < 0.5$ and it it compressed if $D_c \geq 0.5$. This means compression will only be made if both decision modules have high reliability values and agree about compression or when one reliability value is higher enough to compensate the other one, in the case of a disagreement.

The next step is feedback. After transmission, data about total transmission time, decompression time if it is the case, the total energy spent by the receiver device, the amount of transferred bits, the amount of bits with errors and the amount of grouped error bits are all collected. All these parameters are used to evaluate the new values of available bandwidth, packet error ratio and reliability factors.

Available bandwidth is calculated based on past transmissions and its formula is

$$bw_{i+1} = \alpha \cdot bw_i + (1 - \alpha) \cdot \frac{Tam_i}{tt_i}$$

where Tam_i is the amount of transferred bits and tt_i is the total transmission time of

the last transmission. After running some tests, the value of α was defined as $\frac{7}{8}$ in the experiments.

The packet error ratio is calculated as follows. First, the number of packets between two consecutive error bursts is evaluated with the formula:

$$Pac = \frac{[10^{-(Tx)} \cdot \frac{B_e}{8}]}{T_m}$$

where T_x is an exponent representing the error bit rate seen on past transmissions (usual values are around -5, indicating 1 bit with error in 10^5 transferred bits), B_e is the amount of grouped bits with errors (usual values are between 3 and 10), T_m is the packet size measured in bytes (that´s why it is divided B_e by 8) and Pac is the number of packets between two consecutive error bursts. This last value is used in the formula

$$P_{err} = \frac{1 + D_i}{Pac}$$

where D_i is the probability of two consecutive error packets due to the same error burst and P_{err} is the packet error ratio.

The new W_t and W_e values are recalculated based on real and predicted values of a transmission. If a gain is observed, either in time or energy, the respective factor is increased; otherwise the factor is reduced, as described next.

$$w \leftarrow \begin{cases} w + \frac{(1-w)}{2} & \text{, if there is a gain} \\ w - \frac{w}{2} & \text{, otherwise} \end{cases}$$

Determining an expected gain is done as follow. Before transmitting a file, each module evaluates two expected values for this operation – one using compression and another one not using it (nc and c indices will refer to not compressed and compressed values respectively). This way, there are two expected transmission times (TE_{nc} and TE_c) and two expected energy consumptions (EE_{nc} and EE_c). The returned value, V_e and V_t, of

each module is based on their differences. For instance, if $TE_c > TE_{nc}$, meaning that transmission time with no compression is higher than the transmission with compression, then V_t is 1; otherwise V_t is 0. The same idea is applied to energy. After transmitting the file, whether using compression or not, total time – TT – and total energy – ET – are compared against the expected time/energy values. For instance, if a file is transmitted with compression TT would be compared with TE_{nc} and if an improvement was really achieved, that is, $TT < TE_{nc}$, then W_t is increased and it means the module is able to make good predictions. Otherwise W_t is reduced, meaning the module is doing poor predictions. After a while, depending on how stable the environment is, the reliability factors will stabilize at some value.

Next it will be presented the description of how the values are evaluated inside each module. First, response time is analyzed. The total expected time to transmit a file is the sum of the expected time spent to send all packets, the expected time spent to resend the ones with errors and the expected time needed to receive all acknowledgments. If compression is used decompression time has to be added. Compression time is negligible since servers have much more power than mobile devices and one compressed file copy could be stored together with the original one, in order to avoid compression at each request. Thus:

$$\begin{aligned} TE_{nc} &= Tsend_{nc} + Tresend_{nc} + Tack_{nc} \\ &= \frac{tam_{nc}}{bw} + P_{err} \cdot \frac{tam_{nc}}{bw} + (1 + P_{err}) \cdot \frac{tam_{nc}}{tam_{pkt}} \cdot \frac{tam_{ack}}{bw} \end{aligned}$$

$$\begin{aligned} TE_c &= Tsend_c + Tresend_c + Tack_c + Tdec \\ &= \frac{tam_c}{bw} + P_{err} \cdot \frac{tam_c}{bw} + (1 + P_{err}) \cdot \frac{tam_c}{tam_{pkt}} \cdot \frac{tam_{ack}}{bw} + Tdec \end{aligned}$$

$$\Delta TE = (1 + P_{err}) \cdot \frac{\Delta tam}{bw} \cdot (1 + \frac{tam_{ack}}{tam_{pkt}}) - Tdec$$

where tam is the file size, tam_{pkt} is the packet size, tam_{ack} is the acknowledgment size, and k is a constant that depends on three factors: decompression method used, receiver device and file size.

For energy prediction, only the energy spent by the receiver device for receiving all packets, sending all acknowledgments and decompressing the file should be taken into consideration. Sending packets and compressing files is done by the server but these values can be disregarded as is it supposed that the server is connected to a fixed and infinite power supply.

$$\begin{aligned} EE_{nc} &= Esend_{nc} + (Ereceive_{nc} + Eack_{nc}) \\ &= 0 + [P_r \cdot (1 + P_{err}) \cdot \frac{tam_{nc}}{bw} + P_s \cdot (1 + P_{err}) \cdot \frac{tam_{ack}}{bw} \cdot \frac{tam_{nc}}{tam_{pkt}}] \end{aligned}$$

$$\begin{aligned} EE_c &= Esend_c + (Ereceive_c + Eack_c + Edec) \\ &= 0 + [P_r \cdot (1 + P_{err}) \cdot \frac{tam_c}{bw} + P_s \cdot (1 + P_{err}) \cdot \frac{tam_{ack}}{bw} \cdot \frac{tam_c}{tam_{pkt}} + \\ &\quad P_d \cdot Tdec] \end{aligned}$$

$$\begin{aligned} \Delta EE &= P_r \cdot (1 + P_{err}) \cdot \frac{\Delta tam}{bw} + P_s \cdot (1 + P_{err}) \cdot \frac{tam_{ack}}{bw} \cdot \frac{\Delta tam}{tam_{pkt}} \\ &\quad + P_d \cdot Tdec \end{aligned}$$

P_s and P_r are the power values related to sending and receiving packets by the mobile device. P_d is the processor power spent to decompress a file in the mobile device.

It should be noticed that in this analytical model none of the used metrics is bounded to specific methods of measurement or technological aspects. This means that some metrics could be replaced or simply ignored without loosing the general idea. The bandwidth, for

instance, could be measured by constantly sending ping messages or observing the rate at which a TCP buffer is flushed. The packet error ratio could be evaluated in a different way or it could be ignored if it is the case. Energy and time could be evaluated more accurately by changing their formulas or by doing measurements in the hardware itself.

As discussed in Section 3, tests were performed over a collection of text and Web documents in order to determine the constants to be used in the model, as the above formulas assume the use of a predicted compressed file size and a predicted decompression time. To make predictions for these values, Table 4.1 was built. This table gives an approximate compression ratio as a function of compression method and original file size for HTML files.

Table 4.1: Approximate compression ratio as a function of compression method and original size for HTML files.

	K_{mt}			
Size (KB)	LZSS	LZW	GZIP	PPM
0–1	0.68	0.72	0.60	0.55
1–2	0.58	0.65	0.56	0.46
2–3	0.52	0.60	0.51	0.41
3–4	0.47	0.56	0.46	0.37
4–5	0.45	0.64	0.41	0.35
5–7	0.42	0.50	0.34	0.31
7–10	0.40	0.48	0.32	0.29
10–15	0.36	0.43	0.29	0.25
15–25	0.33	0.38	0.27	0.22
≥25	0.32	0.33	0.25	0.20

Hence, compressed file size is a function, F_c, that can be calculated using the original file size, Tam_{nc}, and the compression constant, K_{mt}, for the respective $Method$.

$$Tam_c = F_c(Tam_{nc}, Method) = K_{mt} \cdot Tam_{nc}$$

In Chapter 3, LZSS was shown to give good results for compression size and decompression time resulting in a good tradeoff between these two metrics. Therefore, LZSS is a good candidate to be used in the future experiments. If there is the need to change the

compression method it suffices to modify the K_mt constant to the new proper value.

The time taken to decompress a file is a function, F_d, of three parameters: compressed file size – Tam_c, compression method – $Method$, and mobile device to be used – $Machine$. Thus:

$$\begin{aligned} Tdec &= F_d(Tam_c, Machine, Method) = K \cdot Tam_c \\ &= K \cdot K_{mt} \cdot Tam_{nc} \\ &= K_t \cdot K_{cl} \cdot K_{mt} \cdot Tam_{nc} \end{aligned}$$

where K_{cl} is a constant related to the mobile device and K_t is related to decompression method and is obtained through linear regression on the results of compression on the HTML (more details on Section 5.2). For instance, Table 4.2 represents K_t values for a notebook with Pentium processor of 133 MHz and 24 Mbytes of memory. K_t is calculated through linear regression of all measures taken for decompression time versus file size for the HTML collection (refer to figures 3.9(b) and 3.11) and taking the inclination of the line obtained. For this device, K_{cl} could be defined as 1. If for instance processor speed doubles, K_{cl} could be defined as 0.5 without the need to reevaluate all metrics again.

Table 4.2: Decompression time constant as a function of compression method.

Method	K_t
LZSS	4×10^{-6}
LZW	6×10^{-6}
PPM	2×10^{-4}
GZIP	4×10^{-6}

It is worth to emphasize the how flexible this model is. Regarding the model's parameters, it is easy to add new ones by just changing and/or creating new prediction formulas (e.g., if server compression time, server energy consumption or bandwidth utilization should be considerer). It is also possible to specify parameter priorities by just changing maximum reliability factor values (e.g., if a user cares more about the energy

consumption rather than response time). It is also possible to modify formulas when there is no information about some metric (e.g., bits with error) or when its relevance is negligible. Obviously, in this case, the predicted result would be less accurate. Also, the model´s idea and its implementation are completely independent which is always a good asset in designing communication protocols and policies.

Chapter 5

Experiments

5.1 Communication protocols

In order to validate the predictive model it is necessary to measure its performance over different conditions. A truly adaptable model should present good performance no matter what the environmental conditions are and should not make performance worse than it would be in a traditional system, that is, with no modifications. In the world of wireless communications, *Medium Access Control* – MAC – layer plays an important role on defining how and when the wireless carrier is used to transmit data. The protocol used in this layer, in fact, is what distinguish one form of communication from another. For this work, two different types of MAC communication protocols were used: IEEE 802.11 and Bluetooth. A brief description of each one is given next.

5.1.1 IEEE 802.11

The IEEE 802.11 standard was created in 1999 to support communication in Wireless Local Networks (WLANs). The specification defines one MAC layer and several possible physical layers, making possible to access the medium in three different ways: FHSS (Frequency Hopping Spread Spectrum), DSSS (Direct Sequence Spread Spectrum) and infrared.

In the 802.11 protocol, Basic Service Sets (BSS) are the architectural units. A BSS is defined as a group of communicating devices under control of a unique coordination function (*Distributed Coordination Function – DCF*), which is responsible for determining when a device can transmit/receive data. Devices can communicate directly (point-to-point) or with the help of a predefined structure. Networks which communicate like the former case are known as *ad-hoc* networks, and like the later case are known as *infra-structured*. Infra-structured networks use *base points* to interconnect devices and to provide mobility across different areas.

There are two basic communicating rates, 1 Mbps and 2 Mbps. Standards 802.11a and 802.11b altered 802.11 physical layers in order to provide higher rates, such as 5.5 Mbps and 11 Mbps (802.11b). The standard 802.11a can achieve 54 Mbps using its own multiplexing technique, which makes communication impossible between 802.11b and 802.11a devices.

5.1.2 Bluetooth

Bluetooth's main goal is to provide a mechanism to interconnect low-power devices using radio communication. Bluetooth networks allow fast exchange of data and voice among devices such as PDAs, pagers, modems, cell phones and mobile computers.

Bluetooth operates in the well known ISM 2.4 GHz spectrum and it was specified based on robustness and low cost, aiming to form networks known as Wireless Personal Networks (WPANs). Its transmission range can vary from 10 cm to 10 m, but its specification allows up to 100 m range if transmission power increases. Bluetooth networks are formed when a device assumes the *master* condition and starts managing the *slaves* nodes. Any device can assume the master and slave role along the time. A difficult challenge of Bluetooth networks is the configuration and reconfigurations steps, that is, what happens when a devices leaves or enters the communication area.

5.2 Scenarios

For 802.11 experiments, the following scenario was built: an embedded system communicating through a wireless channel with a file server and retrieving files using HTTP protocol. This is similar to a mobile user with a PDA browsing an Intranet of a mall or office. The embedded system used was DIMM-PC/486-I model of Jumptec [30] which is a 486 processor of 66 MHz speed and 16 Mbytes of memory, and Lucent WaveLan 802.11 [31] as wireless card. The receiving, transmitting, and processing costs are specified in Table 5.1.

Table 5.1: Energy costs for 802.11 experiments.

Operation	Current (mA)	Voltage (V)	Power (W)
Transmission	330	5	1.65
Reception	280	5	1.40
Decompression	510	5	2.55 (66 Mhz)
	267	5	1.33 (33 Mhz)

In order to apply the model, all values for the parameters had to be evaluated again for this system, as described in Chapter 4. As the only changes happened in the hardware (and not in the compression algorithms), ratio compression values remained constant. Figure 5.1 shows the linear variation of decompression time versus file size for the HTML collection for files smaller than 15 Kbytes. The new Kt value, which can be obtained by linear regression, was defined as $1,875 \times 10^{-5}$ and K_{cl} was set to 1 on this new system.

The scenario of Bluetooth experiments is almost the same as the one of 802.11. The main difference is hardware configuration. Client and server applications run in a 400 Mhz Pentium II computer, with 256 Mbytes of RAM and running OS Suse Linux 8.1 (kernel 2.4.18). Client's communicating device was developed by Ericsson [32]; server's communicating device was developed by Widcomm [33]. The Affix [34] protocol stack was used in the application development. Half of the test collection was used in the tests and Kt

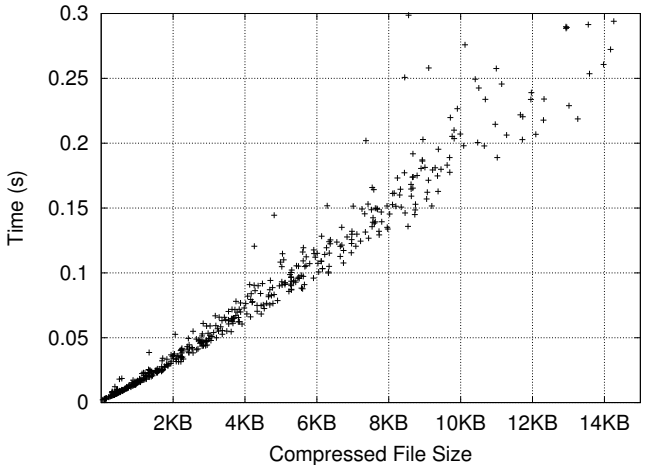


Figure 5.1: File size versus decompression time for files smaller than 15 Kbytes.

parameter had to be reevaluated for these machine.

The actual implementation of this model required the insertion of new fields on the HTTP header [35], containing information about the last transfer (figure 5.2). These pieces of information are *piggybacked* to the server when a new request is sent. Before answering the request, the server will pass through the feedback step of the adaptive model with new data. Many servers already retain information about client interaction through the use of sessions, hence inserting new data to these structures is trivial.

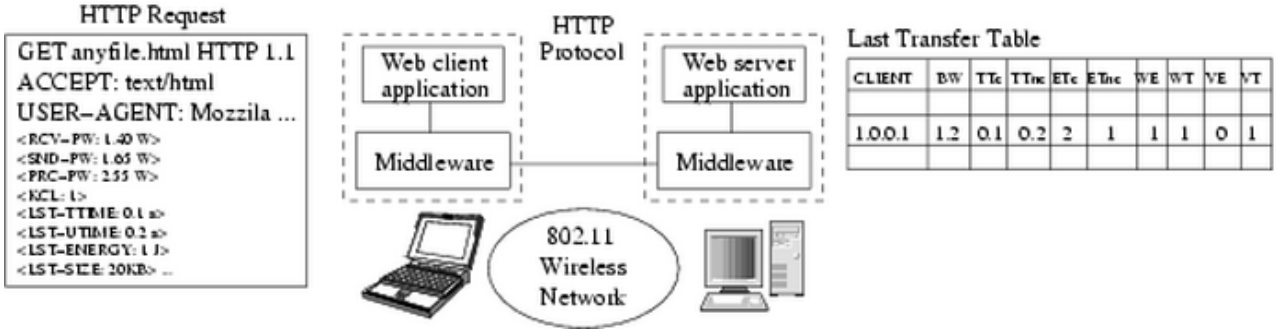


Figure 5.2: Scenario of 802.11 experiments.

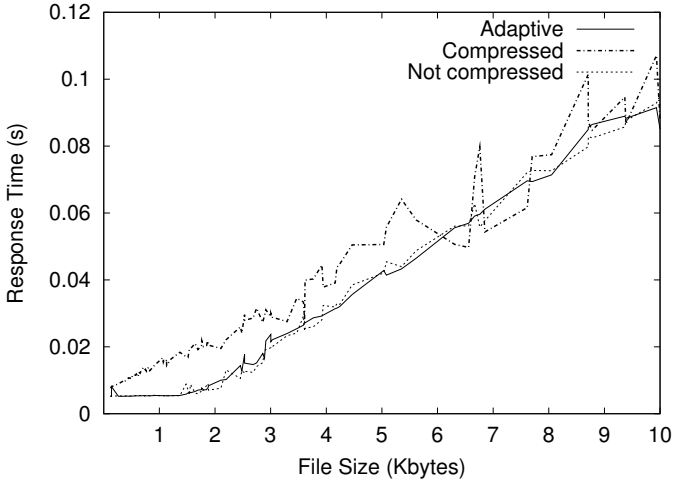
Piggyback model architecture works well in such server and *middleware* systems since they demand a small effort from clients (or client *middleware* side) as they only have

to store the information related to the last transfer until the next request. This can be accomplished by maintaining a set of few variables in the memory while the application is active. Another option for implementation is putting all intelligence on the clients. This requires some modifications in the model, since the client cannot guess the requested file size. A possible solution is to let a client evaluate the range of file sizes that could be compressed and send this information to the server. This range would be analyzed by the server and the requested file would be sent with or without compression depending on it. This would imply more memory/energy costs in the clients with the model's calculations but it frees the server of the burden to store information about each client.

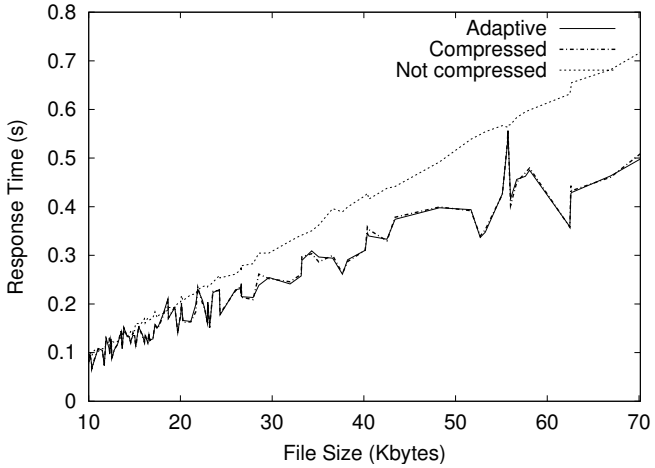
5.3 Results

For the 802.11 experiments, 190 Web files were randomly selected from the HTTP collection for testing. Figures 5.3(a) and 5.3(b) plots response time versus file size for bandwidth of 1 Mbps for each of the three models: no compression at all, always compressing the file and the adaptive model. Figure 5.3(a) compares the three models for files with size smaller than 10kbytes and figure 5.3(a) does the same for size larger than 10 kbytes. It is possible to observe that the adaptive model fits into the best case in both situations, that is, for files smaller than 10 kbytes adaptive model decides that files should not be compressed; for the other files, the best case occurs when files are compressed before transmission. Around 10 kbytes both times (transmission with and without compression) are approximately the same. According to [36], most of the replies are less than 3 Kbytes for online wireless clients – the ones which demand HTTP pages one by one when they browsing on the Internet – and less than 6 Kbytes for offline clients – the ones which demands all pages before browsing through them. Thus, compressing all responses would not be a good policy, even for a low bandwidth link.

Figure 5.4 plots response time performance of the adaptive model over compressed and



(a) File size smaller than 10 Kbytes.



(b) File size larger than 10 Kbytes.

Figure 5.3: Response time for adaptive, compressed and not compressed models.

not compressed models for all tested files. For the small files, gains are about 60 to 70% for sizes ranging from 1 to 2 Kbytes, decreasing out of these limits. For large files, there is a continuous increasingly gain as the size gets bigger. For a size of approximately 50 Kbytes, gains are about 20 to 30%.

Figure 5.5 plots the accumulated energy consumption calculated as described on the formulas presented on chapter 4. The three lines represents accumulated energy consumption for each model (always compressing, no compressing at all and adaptive) for all files in requesting order. As it is implemented, the model applies the same weight to both

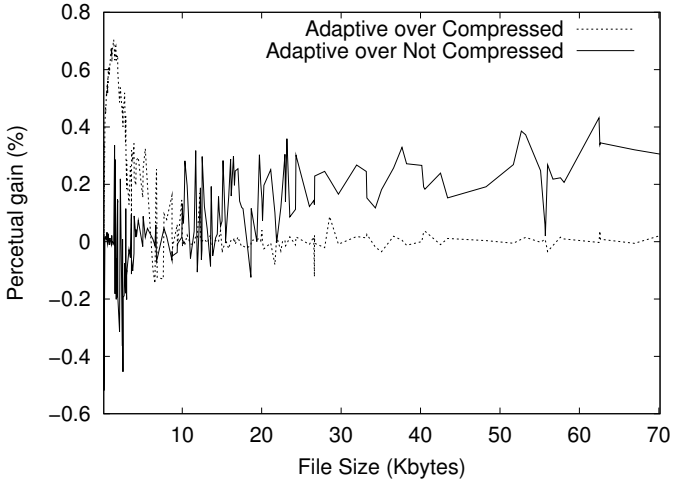


Figure 5.4: Percentual gain of adaptive model over compressed and not compressed models.

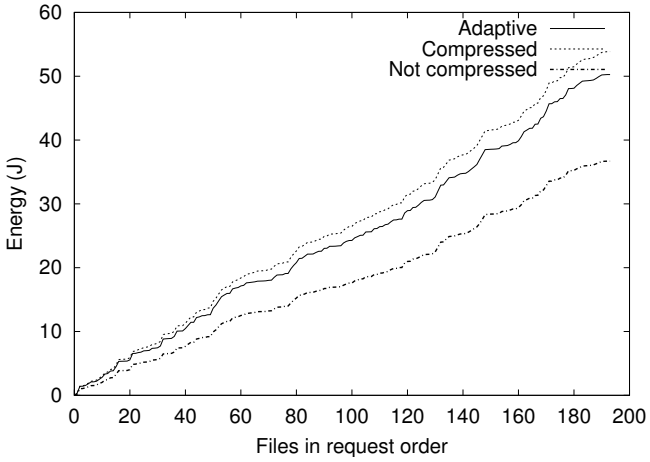


Figure 5.5: Accumulated consumed energy of adaptive, compressed and not compressed models

parameters – time and energy – and compression can happen when either one yields some gain. For the system deployed in these experiments, the process of decompressing a file consumes more energy than transmitting it, which explains the behavior of the figure: adaptive is closer to the worst case. As said before, if a client wishes to give different parameter priorities he just needs to change the specific weights on the formulas of the model. Giving more priority to energy means that the adaptive model line would fit the best case (in this case, not compressing ever); on the other hand, response time probably

would not have the behavior as seen before.

Next figures show how compression depends on the device as much as in the file size and on bandwidth. In Figure 5.6, bandwidth was modified to 2 Mbps and all gains due to compression disappeared, since then decompressing the file taken much longer than the time saved in its transmission. The adaptive model was able to adapt to the best case.

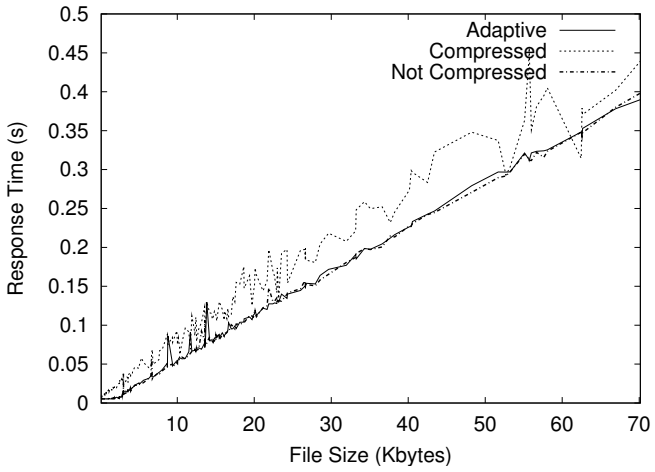


Figure 5.6: Response time for adaptive, compression and not compression models using bandwidth of 2 Mbps and processor speed of 66 MHz.

Next (figure 5.7), processor speed was changed to 33 MHz and the same experiments were executed as before using 1 Mbps as bandwidth. As the same device was used – the only difference was clock speed – it was enough to modify K_{cl} to 2 and to use the respective energy consumption values (Table 5.1). Again, time saved with compressed transmission is not compensated by the time taken to decompress a file in such a slow system.

The conclusion is that even for the same bandwidth channel and for the same device compression decision should be done individually and a static solution would fail to give the best performance always.

Bluetooth protocol minimizes channel interference using its own model of frequency hopping, which puts a limit of 723 kbps as the maximum achievable bandwidth. With such a limited bandwidth and powerful devices, Bluetooth offers the perfect scenario under

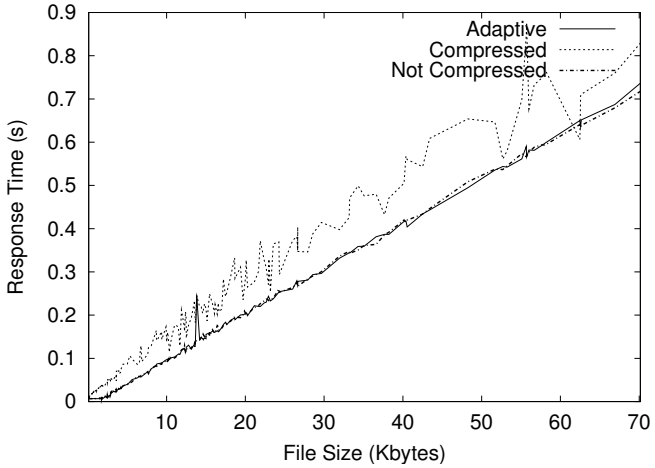


Figure 5.7: Response time for adaptive, compression and not compression models using bandwidth of 1 Mbps and processor speed of 33 MHz.

which compression could be used. The graph in Figure 5.8 shows the large difference in response time for adaptive and not compression models.

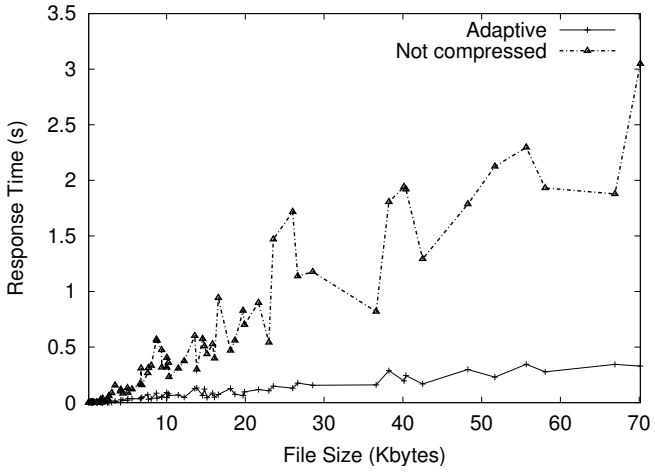
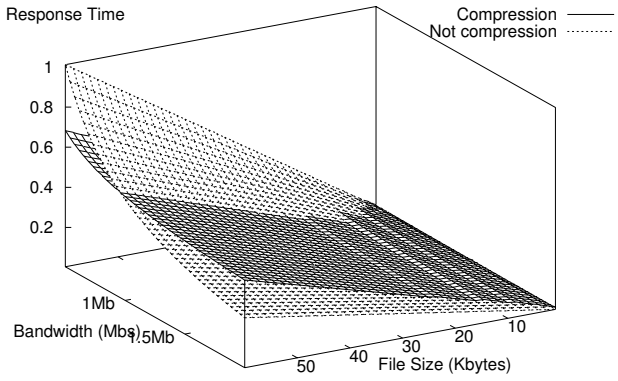


Figure 5.8: Response time for the Bluetooth protocol.

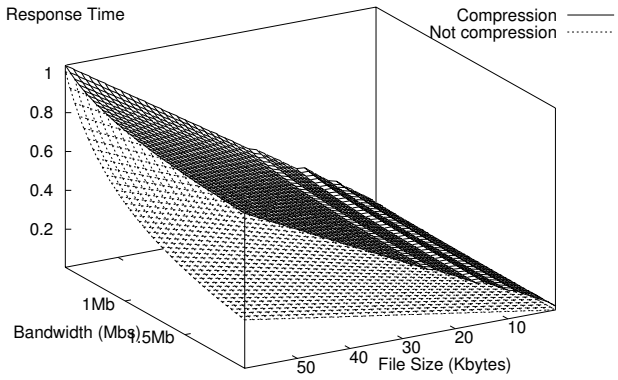
As Bluetooth's main goal is to form personal networks by interconnecting embedded systems, it is expected that devices for this scenario are less powerful than the devices used in the previous experiment. In that case, compressing all files might not be a good choice as decompression could take too long.

5.3.1 An analytical view

It is possible to plot response time as a function of file size, bandwidth, and processor speed using the time formulas of the analytical model (Chapter 4). By making some assumptions like null packet error rate, graphs like the ones shown in Figures 5.9(a) and 5.9(b) can be obtained. These graphs offer an analytical view of the model, which is summarized in Figure 5.10.



(a) Bandwidth 1 Mbps; Processor speed 66 MHz.



(b) Bandwidth 1 Mbps; Processor speed 33 MHz.

Figure 5.9: Response time as a function of file size and bandwidth.

In Figure 5.10, the area limited by the model´s parameters defines where compression should be used. The area is shaped mostly by the compression method used and the media

type being compressed, that is, the Kmt parameter. The offset is given mainly by mobile device characteristics – $Kt \times Kcl$. Packet error rate, packet and acknowledgment sizes may alter both offset and shape in a smaller scale. Power consumption values have a similar behavior.

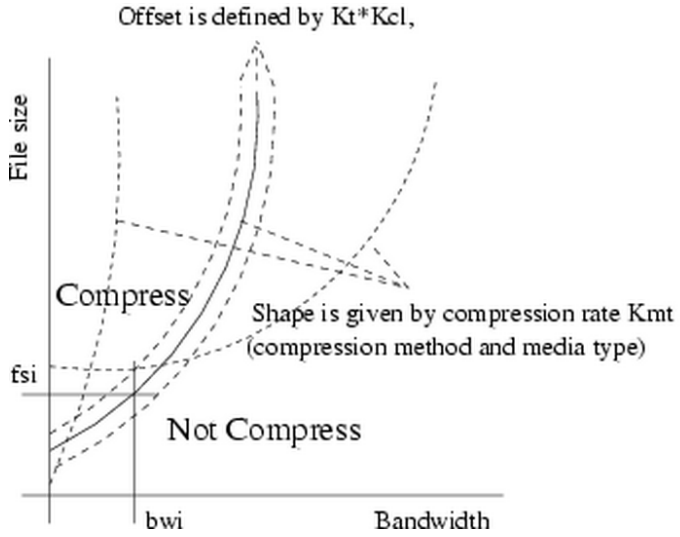


Figure 5.10: Graphical view of compression of the analytical model.

This representation could be used to create a simplified model that runs in the device itself, as mentioned before. Suppose an embedded system for instance. If several bandwidth values bw_i are stipulated, different file size points fs_i would be obtained beyond which compression should be performed. In this manner, a table indexed by bw_i whose entries would be fs_i could be used to decide when compression pays off. Thus, another way to get an adaptive compression method is when clients monitor bandwidth and transmit the specific limit to servers.

5.4 Simulations

In the previous section, response time and energy consumption for the proposed adaptive model were analyzed for one device only. This section presents results for several devices communicating to a server. This is an important analysis since real scenarios may envolve

many clients and the available bandwidth may vary a lot during the course of application execution.

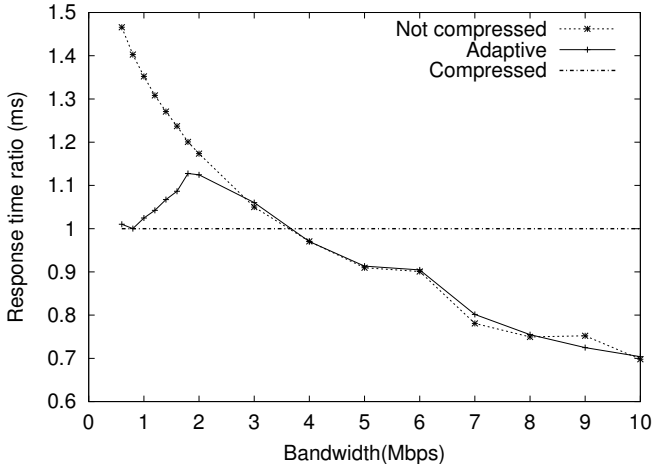
ns-2 [37] was used as the simulation tool and the following parameters were analyzed: communication protocol (Bluetooth and 802.11), number of clients, bandwidth and transmission mode (compressed, not compressed and adaptive). In the simulated scenario, static clients (C_i) made continuous requests to server S (Figure 1.2) through a wireless channel. Both request intervals and file sizes followed exponential distributions with means of 30 s and 3 Kbytes, respectively [36].

The *Compression* module was implemented based on experimental results (Section 5.3). A packet is transmitted only after the arrival of the acknowledgment of the previous packet (window size of 1). Besides flow control, this module uses the proposed adaptive model to decide when a file should be compressed before transmission.

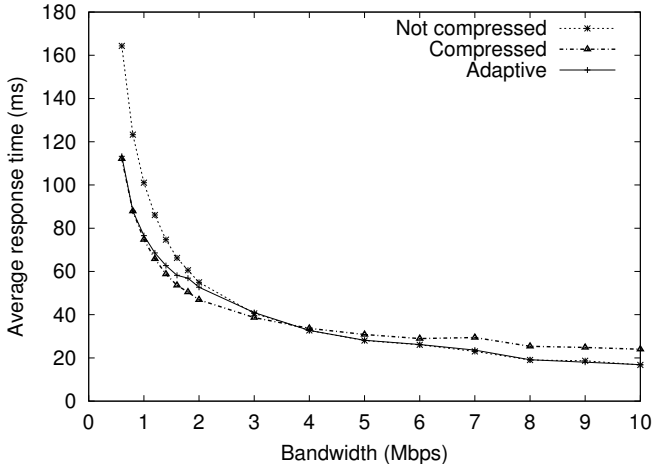
First, results for the IEEE 802.11 protocol are presented. For this scenario it was stipulated a 0.5% channel error rate, 3600 s of simulation time, bandwidth varied from 600 kbps up to 10 Mbps and the number of clients simulated were 1, 5, 10 and 50. Each simulation was run 33 times.

Figures 5.11(a) and 5.11(b) show performance of adaptive model over other models both as relative performance and real time measured in a scenario with only one client and different bandwidths. In figure 5.11(a) line 1 represents the transmission of files with no compression at all. Predictive model adapts well in very low and very high bandwidths environments, a result similar to the one obtained in the last section.

Figure 5.12 shows a different analysis of the model. The prediction formula was modified so it would consider the remaining energy in the device to make its decision. Remaining energy was divided into five classes (80–100%, 60–80%, 40–60%, 20–40%, 0–20%) and for each one an increasing weight was given to the energy parameter, while proportionally decreasing the relevance of time parameter. All nodes (in a 50-node scenario) started with 100% of energy. Figure 5.12 shows that as the remaining energy in the devices become



(a) Performance comparison.



(b) Time comparison.

Figure 5.11: Comparison of adaptive, compression and not compression models (1 client).

scarce, compression turns into a rare event and overall response time performance decays. For this scenario, devices consumes more energy decompressing files than transmitting them, so it is expected that when energy plays a major role compressing files is no longer a desirable choice and clearly the response time is affected by this choice.

Next, Bluetooth performance is analyzed. Bandwidth was defined as 750 kbps, which is a typical value for class 2 Bluetooth devices (10 m range), and the number of nodes inside the piconet varied from 1 to 7. The file server was hosted in the master node and all slaves acted as requesting clients. Under these conditions, compression is performed for all

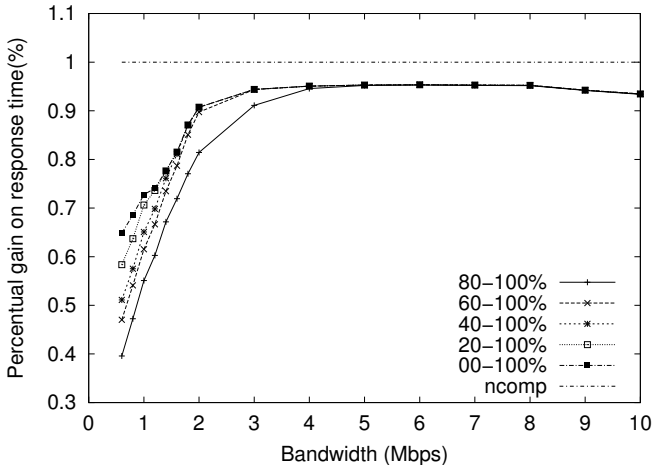


Figure 5.12: Performance of adaptive over not compression model with varying energy weight.

file sizes as transmitting data on Bluetooth takes too long even for small files. Figure 5.13 shows response times for adaptive and not compression model.

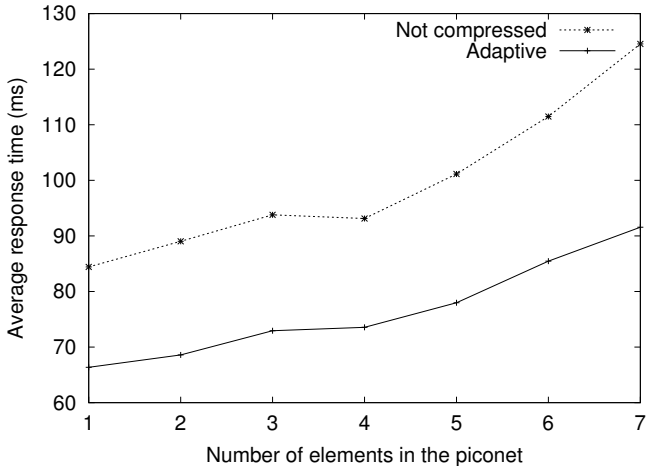


Figure 5.13: Comparison of the adaptive and not compression models in Bluetooth environment.

Chapter 6

Proxy experiments

This chapter presents the results for the proxy system implemented using the adaptive compression model. Figure 6.1 illustrates how the system works. RabbIT2 Web Proxy Cache [38] was modified in order to support compression with adaptive model. This server-proxy-client perfectly suits the proposed model: first, this proxy (which could be replaced by a middleware) doesn't need to be restrained only to compressing files prior to their transmissions; it could be used for several other adaptation techniques developed for wireless networks; second, leaving the intelligence on the server could possibly overload it and there would be a demand to implement the adaptive model on any possible server that could be accessible by mobile clients. Using the proxy alternative, companies and owners of wireless centers could offer this alternative solution to their users and maintain a local control of what happens on their networks. On the client side, there is the need to modify how the HTTP request is formed adding new headers to the actual request. Decompressing files is already a feature present in most browsers today, the only change would be implementing the proper decompressing algorithm chosen for the experiment.

In its original form, RabbIT2 intercepts requests from Web clients and performs the proper HTTP operation in their behalf. It also works as a cache, so new requests can get the copy directly from the proxy instead of going through the Internet. Before transmitting

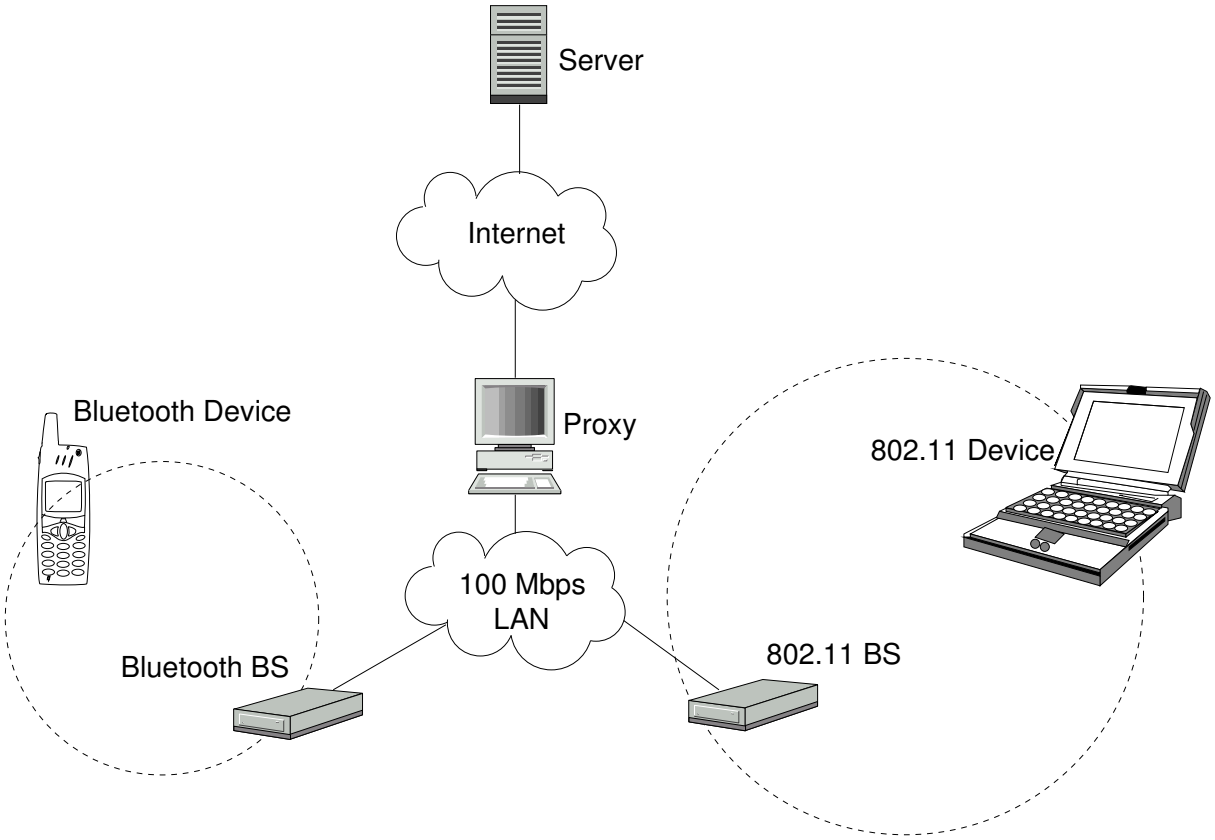


Figure 6.1: Proxy system.

the actual response to clients, the modified version of RabbIT2 uses the proposed adaptive model to check whether the file should be compressed. The proxy was modified so it can store both original and compressed copy of the file. This optimization improves response time, since compression needs to be executed only once. The many parameters analyzed in these experiments includes bandwidth, file size and client device power, as explained in chapter 4.

Next it is explained how the data set was formed (section 6.1), how the proxy has been modified (section 6.2), and how good is the performance of the model in this environment (section 6.3).

6.1 HTTP Trace

From December 1st to December 5th, all HTTP requests and responses transmitted over the fixed network installed on SIAM laboratory at DCC/UFMG were monitored. The staff on this lab is composed mainly by students and technicians and the trace can be considered to be close enough to the traffic that would be obtained in a wireless indoor network covering the same space. In fact, if we only change cables and maintain all machines in the same place, the behavior of all users would be the same. A difference would appear when mobility is inserted in this environment because some users would probably move during certain periods of time causing movements out and into the wireless cell. As this would change the behavior of the trace, mobility was disregarded for these tests. From this trace it is possible to extract the following parameters to be used in the experiments:

- **Content type distribution:** Content type distribution of HTTP responses follows distribution given on figure 6.2. This plot also gives an idea of how many of these files are cacheable (cacheable and non-cacheable files are distinguished by a tag of HTTP responses). GIF and JPEG images add up more than 58% of all traffic. As these documents are not modified constantly during their lifetime (almost all files are cacheable), caching them close to the clients are a good solution to improve their response time. Moreover, lossy compression can be applied on them to reduce even more their size and impact on transmission time and bandwidth consumption. HTML type files has a higher percentage of non-cacheable files than other types. Moreover, most of the static HTML pages (cacheable) have lifetime of only a few seconds, so just caching files doesn't solve all problems in this case (lifetime of pages is also described by a tag on HTML responses). When caching is not possible, compression turns out to be the best solution to minimize response time.

- **Session times:** Session time is important in the model to indicate when parameters should be reset to initial conditions, that is, when a user finishes her interaction with

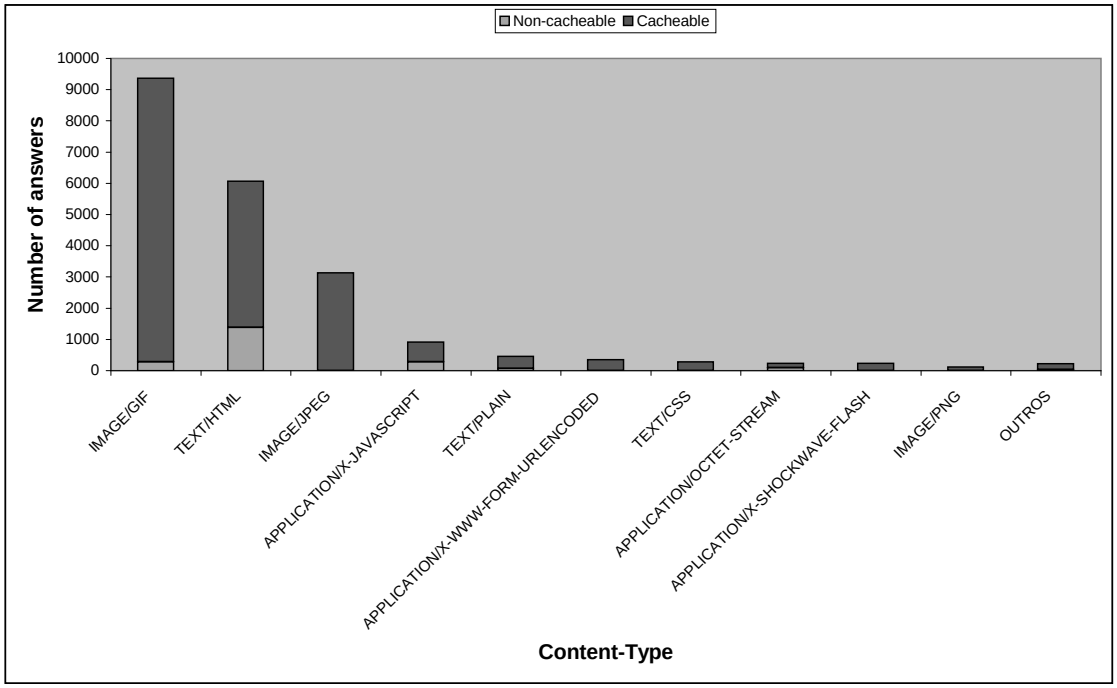


Figure 6.2: Distribution of Content Type.

the proxy. Figure 6.3 shows all sessions seen during the period. As in the laboratory each user has its own machine and each machine has a fixed address, each session can be identified by its IP number. In many real cases, there are no guarantee that this number is always associated with the same user (for instance, when using DHCP), that's why it is important to establish an upper limit indicating when a user session is over or not. A user session is defined as a sequence of request done by the same user during a period of time. The duration of user session time was not used in the experiments since this wouldn't cause a big impact on the results, although resetting the model's value frequently to the initial state may turn the model obsolete, which is why it is so important to define a session time long enough so the user can take advantage of the model but no longer than the period of time the same user is

browsing the Internet.

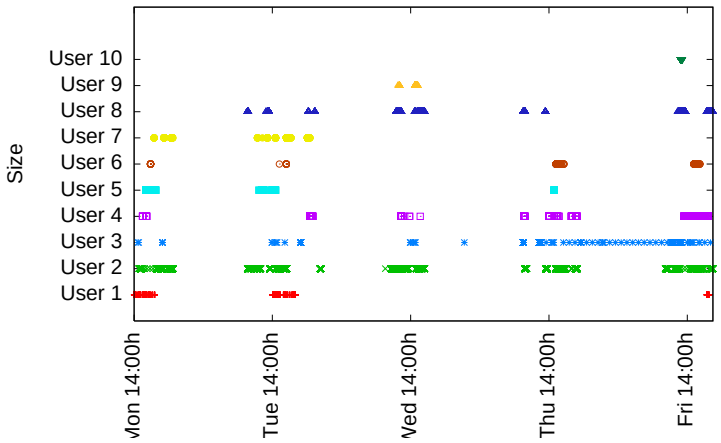


Figure 6.3: Session view of the trace.

Time interval between consecutive requests are also important in order to create a real traffic demand on the proxy. If the intention is to simulate a real scenario, requests would have to be done following a certain time interval distribution. This statistical distribution can be obtained from the trace analyzing user's requests intervals and obtaining the inverse function of the accumulated distribution. Despite the relevance of this parameter to assess performance of proxies and servers systems, it was not used in the experiments as it doesn't impact the adaptive decision model.

Due to hardware limitations, there was only one device with wireless interface available for testing, so we selected User1 trace as testbench. Figure 6.4 represents User1 trace, plotting time and file size of each request.

- **Requests distribution:** It is known that requests for resources on the Web follows a specific distribution, namely the Zipf distribution [39]: usually, a small set of files corresponds to almost all accesses while a large group of files are requested no more than once. A similar behavior can be observed in the trace (figure 6.5), although the distribution it not exactly Zipf. This indicates that caching is effective for pages accessed frequently and that caching both original and compressed versions of a file

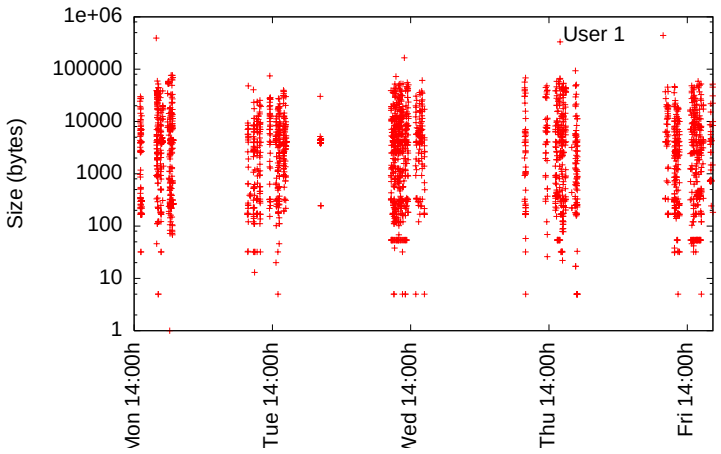


Figure 6.4: User1 Trace.

may highly reduce response time for wireless communicating devices. Since the effect of caching is not the focus of this work this feature was ignored, which means each request necessarily goes to the server. By doing this, measures of response time when compression is applied is more accurate. Caching plus compression is part of future works. As User1 trace is going to be used as the source of data, it is interesting to observe if the same Zipf behavior holds for a single user. Figure 6.6 shows this is true.

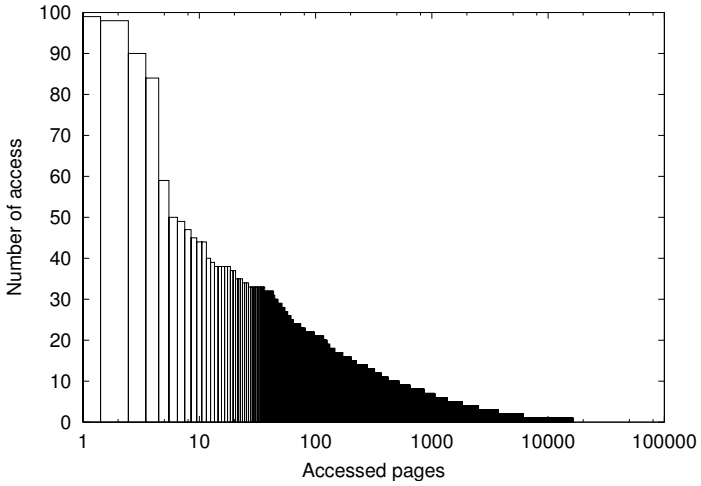


Figure 6.5: Page rank by access (all users).

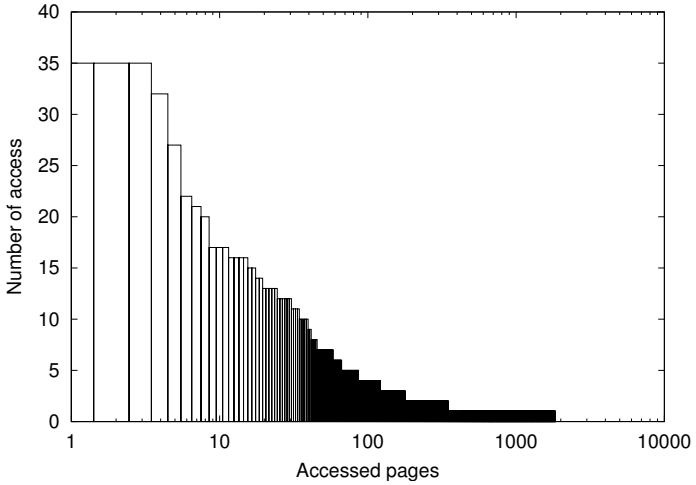


Figure 6.6: Page rank by access (User1).

- **File size distribution:** All packets in the trace were collected and grouped in order to form the original HTTP responses. The resulting files were then stored on a server in the laboratory network. For the experiments, all requests are forwarded to the server which answers accordingly reproducing the original sequence of requests. Having a real file size distribution is important so a realistic measure of the model can be obtained. As it was observed in chapter 5, files of different sizes demand different treatments. Figure 6.7 shows file size distribution for the whole trace. Figure 6.8 shows the same distribution for User1.

6.2 RabbIT 2 Implementation

RabbIT [38] is a proxy for HTTP and its main goal is to speed up surfing over slow links by removing unnecessary parts, like background images, while still showing the page mostly like it is. Since filtering the pages is a heavy process, RabbIT caches the pages it filters but still tries to respect cache control headers of the old HTTP style, which would be a header like *pragma: no-cache*. The whole package of RabbIT is written in *java*.

RabbIT2 works with handlers to execute services over each of its resources. A resource

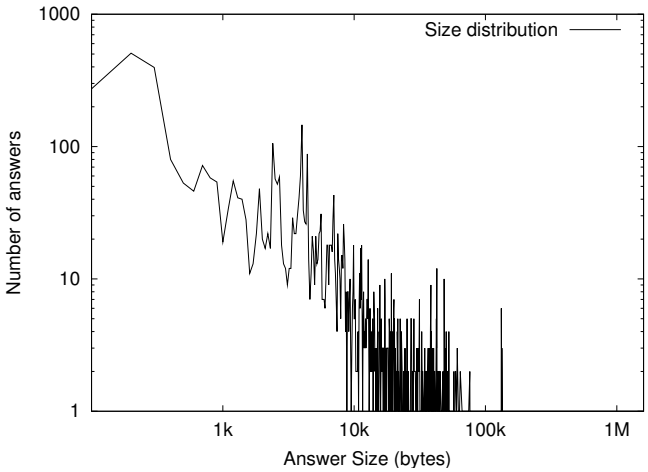


Figure 6.7: File size distribution (all users).

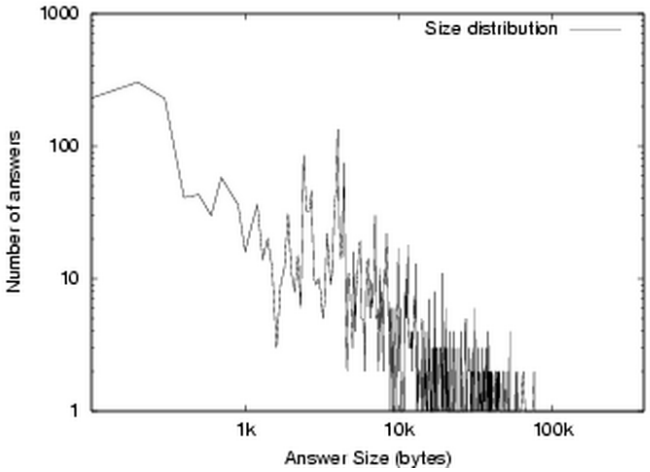


Figure 6.8: File size distribution (User1).

can be any Web object in its cache repository or any object being downloaded from the Web in behalf of a client. Handlers are implemented as java classes offering specific services for their input streams. Examples of handlers are *FilterHandler* and *ImageHandler*, which handle HTML page filtering and image conversion, respectively. For the modified version, we have changed *GZIPHandler*, which is responsible for GZIP compression over HTML files, and *BaseHandler*, the base class for all handlers. BaseHandler has been modified in order to accept insertion and searching of compressed copies of HTML pages in the cache.

For GZIPHandler, the only change was the condition over which compression is made. In the older version compression would be made for all HTML files if a user configured parameter had been set. Now, besides looking over this parameter, RabbIT uses adaptive model to take its decision.

Adaptive model was implemented using several classes. *AdaptModel* class stores all active sessions in the proxy and has specific methods for creation, removing and searching for a session. All sessions are stored in a hash table. An active session is implemented by a *ClientSession* class, which has all parameters and methods needed for adaptive model implementation except the parameters used to predict compression ratio and decompression time, which are present in *ZipModel* class. In these experiments, instead of calculating bandwidth using data sent by clients, bandwidth monitor (class *BwMonitor*) was implemented. The monitor periodically sends SNMP packets to each monitored base station, retrieving the amount of data transmitted over the air interface. This information is used to calculate bandwidth consumption during each monitoring time interval.

6.3 Experimental Results

The setup of the experiment resembles the one shown in figure 6.1: the modified version of RabbIT2 was placed on a machine (Pentium II, 400Mhz, 256MB, Suse Linux 8.0) in the local area network and a Web server was placed in another machine (Pentium II, 400Mhz, 256MB, Windows 2000) in the same network. An 802.11b wireless network was used together with a notebook (Compaq Armada, Pentium II, 365Mhz, 128MB) equipped with a wireless card.

The server used in this experiment was Jigsaw [40]. This server hosted all files present in the User1 trace. The program Wget [41] was used as client application and it was modified so it could transmit all information needed by the model and decompress files when necessary. Wget is a well known program to fetch pages from the Internet.

In this experiment the energy cost part of the model was not evaluated, so the result below considered only time as a parameter to decide wether or not to compress the file. For a 1Mb of nominal bandwidth, the results for adaptive and traditional (no compression) methods of files for User1 trace are shown in figure 6.9. This figure plots accumulated response time for all files and shows that adaptive method can greatly improve response time for this scenario (low bandwidth, high processing power).

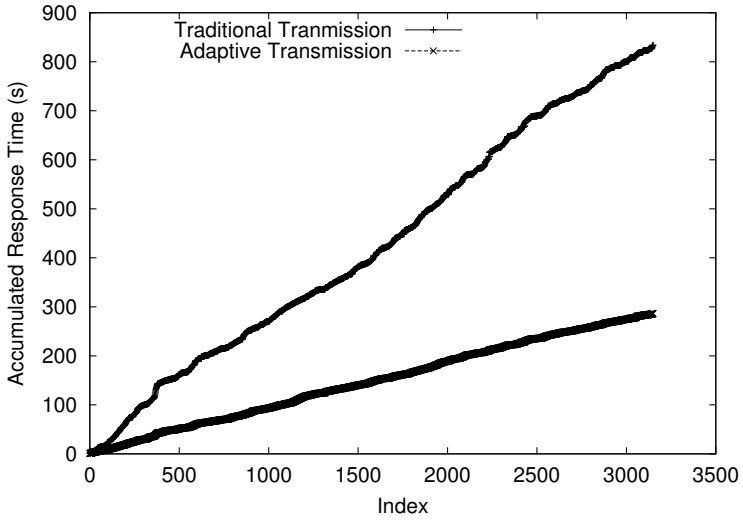


Figure 6.9: Accumulated response time for adaptive and traditional transmissions.

Response time may present a significant improvement when compared to traditional ways of transmitting files over the Internet. It is important to notice that improvements can be even increased with the use of caching mechanisms.

Chapter 7

Conclusion

In this work a new model to adapt HTML documents in wireless environments was presented as long with the associated methodology to apply it. This model dynamically predicts when a requested file should be compressed before its transmission over a wireless channel using parameters such as file size, bandwidth available, energy consumption, compression method and device´s properties. Through experiments and simulations using the IEEE 802.11 and Bluetooth protocols, the method proved to efficient and it was demonstrated how it could be incorporated into existing applications.

Although some present works on adaptive models use compression for data transmission, they do not take into consideration all features of wireless environments, which intrinsically dynamic and heterogeneous. This work is an attempt to develop a model that covers all major features and besides is flexible enough to be implemented and extended.

A preliminary version of this work was presented in [42].

7.1 Future Works

The use of an adaptive model in proxies comes with a cost: degrading system performance and consequently reducing the number of requests a system can handle per second. The time needed to evaluate all parameters and the space needed to store all information about

the model deserve deeper studies. Future work includes to assess the real impact of this adaptive model in the proxy using *httperf*, a tool for measuring Web server performance.

In order do validate this model there is the need to asses performance with many users. Although this was done through simulation in this work, an ideal scenario would be to put the adaptive model in a real proxy system and encourage real users to use it. This is a difficult task to accomplish without strong support, so the first step will be to collect information about a real wireless indoor network (the one used by students and professors of the Computer Science Department of UFMG) and then measure (through simulation) the performance and improvements achieved by using the adaptive model. The process of collecting information is already being carried out. If a high improvement is observed, an adaptive model could be implemented in the Department.

For the experimentes presented here a trace from a fixed network was used. This is an acceptable assumption for wireless indoor environments but does not hold for larger systems such as cellular networks. Web traffic in these networks are especially adapted to smaller devices, so experiments with this type of traffic should be carried out also. Other future experiments include to measure model's performance when facing a sudden variations of available bandwidth or CPU load, which can slow down decompression process.

The mathematical model and analytical part also need more work. After identifying exactly the impacts of each parameter in the model, a more concise formula could be proposed to enhance prediction capacity.

Bibliography

[1] George H. Forman and John Zahorjan. The Challenges of Mobile Computing. *IEEE Computer*, 27(4):38–47, Abril 1994.

[2] Ieee 802.11. http://grouper.ieee.org/groups/802/11/.

[3] R. H. Katz. Adaptation and Mobility in Wireless Information Systems. *IEEE Personal Communications*, 1:6–17, 1994.

[4] Malcolm McIlhagga, Ann Light, and Ian Wakeman. Towards a Design Methodology for Adaptative Applications. In *Fourth Annual International Conference on Mobile Computing and Networking MOBICOM'98*, Dallas, Texas, USA, 1998.

[5] Jeffrey C. Mogul, Fred Douglis, Anja Feldmann, and Balachander Krishnamurthy. Potential Benefits of Delta Encoding and Data Compression for Http. In *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication ACM SIGCOMM '97*, Cannes, France, 1997. ACM Press.

[6] Jesse Steinberg and Joseph Pasquale. A Web Middleware Architecture for Dynamic Customization of Content for Wireless. In *WWW2002*, Honolulu, Hawaii, USA, 2002.

[7] A. Fox, S. Gribble, Y. Chawathe, and E. Brewer. Adapting to Network and Client Variation Using Active Proxies: Lessons and Perspectives. *Special Issue of IEEE Personal Communications on Adapation*, 1998.

[8] Armando Fox, Steven D. Gribble, Eric A. Brewer, and Elan Amir. Adapting to Network and Client Variability Via On-Demand Dynamic Distillation. In *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 160–170. ACM Press, 1996.

[9] Barron C. Housel, George Samaras, and David B. Lindquist. WebExpress: a Client/Intercept Based System for Optimizing Web Browsing in a Wireless Environment. *Mobile Networks and Applications*, 3(4):419–431, 1999.

[10] Emmanuel Jeannot and Björn Knutsson. Adaptive Online Data Compression. In *11th IEEE International Symposium on High Performance Distributed Computing*, pages 379–388, Edinburgh, Scotland, July 24–26 2002.

[11] Björn Knutsson and Mats Björkman. Adaptive end-to-end compression for variable-bandwidth communication. *Computer Networks*, 31(7):767–779, 1999.

[12] Suhit Gupta, Gail Kaiser, David Neistadt, and Peter Grimm. DOM-Based Content Extraction of HTML Documents. In *Proceedings of the Twelfth International Conference on World Wide Web*, pages 207–214. ACM Press, 2003.

[13] Christopher C. Yang and Fu Lee Wang. Fractal Summarization for Mobile Devices to Access Large Documents on the Web. In *Proceedings of the Twelfth International Conference on World Wide Web*, pages 215–224. ACM Press, 2003.

[14] Yu Chen, Wei-Ying Ma, and Hong-Jiang Zhang. Detecting Web Page Structure for Adaptive Viewing on Small Form Factor Devices. In *Proceedings of the Twelfth International Conference on World Wide Web*, pages 225–233. ACM Press, 2003.

[15] Bongjin Jung and Wayne P. Burleson. PerformanceOptimization of Wireless Local Area Networks Through VLSI Data Compression. *Wireless Networks*, 4(1):27–39, 1998.

[16] J. Ziv and A. Lempel. A Universal Algorithm for Sequential data Compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977.

[17] Jeremy Lilley, Jason Yang, Hari Balakrishnam, and Srinivasan Seshan. A Unified Header Compression Framework for Low-Bandwidth Links. In *Sixth Annual International Conference on Mobile Computing and Networking*, Boston, Massachusetts, USA, 2000. ACM Press.

[18] Ronny Kranshinsky and Hari Balakrishnam. Minimizing Energy for Wireless Web Access with Bounded Slowdown. In *Eighth Annual International Conference on Mobile Computing and Networking MOBICOM'02*, Atlanta, Georgia, USA, Setembro 23-28 2002. ACM Press.

[19] Ian H. Witten, Alistair Moffat, and Timothy C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann Publisers, Inc., San Francisco, California, 2nd edition, 1999.

[20] Claude E. Shannon. A Mathematical Theory of Communication. *Bell System Technical Journal*, 27:379–423,623–656, 1948.

[21] J. Ziv and A. Lempel. Compression of Individual Sequences via Variable-rate Coding. *IEEE Transactions on Information Theory*, 24(5):530–536, 1978.

[22] Terry A. Welch. A Technique for High-Performance Data Compression. *IEEE Computers*, 17(6):8–19, June 1984.

[23] Unisys. http://www.unisys.com.

[24] M. Burrows and D. J. Wheeler. A Block-Sorting Lossless Data Compression Algorithm. Technical Report 124, Digital Systems Research Center, Palo Alto, California, May 1994.

[25] NLM National Library of Medicine. Medical subject headings. http://www.nlm.nih.gov/mesh/filelist.html.

[26] ACM Association for Computer Machinery. Acm sigmod online. http://www.acm.org/sigmod/record/xml/.

[27] Timothy Bell, Ian H. Witten, and John G. Cleary. Modeling for Text Compression. *ACM Computing Surveys (CSUR)*, 21(4):557–591, 1989.

[28] E. Alonso, E. S. de Moura, P. Golgher, A. Silva, R. Barra, A. Laender, B. Ribeiro-Neto, and N. Ziviani. Um Retrato da Web Brasileira. In *SEMISH'2000*, Curitiba, Paraná, Brazil, July 2000.

[29] XML. http://www.xml.com.

[30] Jumptec. http://www.jumptec.de/.

[31] Lucent technologies. http://www.lucent.com/.

[32] Ericsson. Bluetooth development kit. http://www.ericsson.com/bluetooth/.

[33] Widcomm. http://www.widcom.com/.

[34] Nokia Research Center. Affix homepage. http://affix.sourceforge.net/.

[35] Jeffrey C. Mogul. Clarifying the Fundamentals of Http. In *WWW2002*, Honolulu, Hawaii, USA, 2002.

[36] Atul Adya, Paramvir Bahl, and Lili Qiu. Analyzing the Browse Patterns of Mobile Clients. In *Proceedings of the First ACM SIGCOMM Workshop on Internet Measurement Workshop*, San Francisco, California, USA, 2001.

[37] Network Simulator. http://www.isi.edu/nsnam/ns.

[38] Robert Olofsson and Fredrik Widlert. Rabbit2 homepage. http://www.khelekore.org/rabbit/.

[39] L. Breslau, Li Fan Pei Cao, G. Phillips, and S. Shenker. Web Caching and Zipf-like Distributions: Evidence and Implications. In *Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM´99)*, New York, NY, USA, 1999.

[40] W3C. Jigsaw - w3c's server. http://www.w3.org/Jigsaw/.

[41] GNU. Wget homepage. http://www.gnu.org/software/wget/wget.html.

[42] R. Couto, R. Rabelo, and A. Loureiro. Compressão Adaptativa de Arquivos HTML em Ambiente de Comunicação Sem Fio. In *Anais do 21o Simposio Brasileiro de Redes de Computadores - SBRC'2003*, Natal, Rio Grande do Norte, Brasil, Maio 2003 (to appear).