Data Transfer Using Controlled Compression

by

Ada Ying Dee Cheung

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 1996

I hereby declare that I am the sole author of this thesis.

I authorize the University of Waterloo to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize the University of Waterloo to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

The University of Waterloo requires the signatures of all persons using or photocopying this thesis. Please sign below, and give address and date.

Abstract

In a scenario in which a large amount of data is to be transferred from one site to another, it is desirable to optimize the throughput of data transfer. There are two transmission options. One option is to compress the data before the transfer, and to decompress them at the receiving site. The other option is to send the data without doing any compression. Ideally, we would like to choose the option that accomplishes the transfer as quickly as possible. This choice is complicated by many factors such as the load of the machines, and the degree of traffic congestion of the network. These factors affect the transmission rate differently depending on whether or not we use compression. This thesis proposes and evaluates two algorithms for automatically determining whether compression should be used. One algorithm is based on sampling past performance with and without compression. The other directly exploits feedback from the network. These algorithms have been implemented and evaluated under different conditions. The evaluation shows that both algorithms can pick the correct mode of data transfer to use in different environments. The algorithm based on network feedback is more complicated and requires tuning to perform well.

Acknowledgements

I wish to thank my supervisor, Dr. Kenneth Salem, who provided expert and insightful advice to me throughout the work. I would also like to thank the readers of this thesis, Dr. David Taylor and Dr. Gordon Cormack. Special thanks also go Dr. Joanne Atlee and Mr. Jim Duff for their permission to use their machines for running experiments in this thesis. Finally, thanks go to my family for their love and support without whom this work could never be done.

Financial support for this thesis is provided by Natural Science and Engineering Research Council of Canada, University of Waterloo Faculty of Mathematics scholarship, and ITRC fellowship.

Trademarks

Solaris and SunOS are trademarks of Sun Microsystems, Inc. Unix is a trademark of UNIX System Laboratories, Inc.

Contents

1 Introduction **1**

1.1 Motivation 1

1.2 Thesis Overview 2

2 Data Compression **4**

2.1 Introduction 4

2.2 The Original Lempel-Ziv 77 Algorithm 6

2.3 The Enhanced Lempel-Ziv 77 Algorithm 8

3 The Controlled-Compression Algorithms **11**

3.1 Algorithm 1 11

3.2 Algorithm 2 13

4 The Experiments : Setup and Environment **24**

4.1 The Setup and Environment 25

5 Evaluation of the Algorithms **27**

5.1 Experiments 27

5.1.1 Constant Load on the Sender Machine 27

5.1.2 Variable Load on the Sender Machine 28

5.2 Results and Interpretation 29

5.2.1 Constant Load on the Sender Machine 29

5.2.2 Variable Load on the Sender Machine 31

5.2.3 Drawbacks of the Algorithms 36

6 Related Work **37**

7 Conclusions and Future Work **40**

7.1 Summary and Conclusion 40

7.2 Future Work 41

Bibliography **42**

List of Figures

1.1 The sender/receiver model 1

2.1 Compression/decompression process of a file X 5

2.2 An example of the original Lempel-Ziv 77 compression 7

2.3 Search window and maximum length of match 8

2.4 Modified coding scheme based on Lempel-Ziv 77 algorithm 9

2.5 Hash table update condition for transparent mode 10

3.1 Original model with modifications for Algorithm 2 13

3.2 An analogy of the classic feedback approach 14

3.3 A graph with throughput plotted against the fraction of blocks that uses regular mode 20

3.4 A graph with rate plotted against the fraction of blocks that uses regular mode 22

5.1 Results of data transfer from S to R_H with no additional load on S 29

5.2 Results of data transfer from S to R_L with no additional load on S 30

5.3 Results of data transfer from S to R_H with 4 units of additional load on S 31

5.4 Results of data transfer from S to R_L with 4 units of additional load on S 32

5.5 Effect of a variable additional load on the performance of Algorithm 1: S to R_H 33

5.6 Percentage of the blocks using the regular mode 33

5.7 Effect of a variable additional load on the performance of Algorithm 2: S to R_H 34

5.8 Percentage of the blocks using the regular mode 34

Chapter 1

Introduction

1.1 Motivation

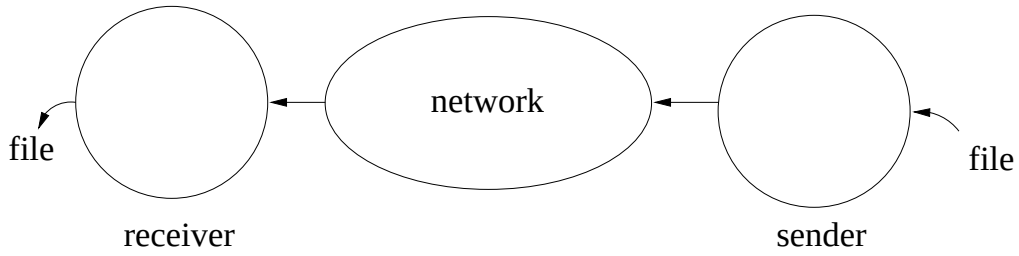

Figure 1.1: The sender/receiver model

Figure 1.1 shows a scenario in which a large amount of data is to be transferred from one site to another. There are two transmission options. One option is to compress the data before the transfer, and to decompress them at the receiving site. The other option is to send the data across without doing any compression. Ideally, we would like to choose the option that accomplishes the transfer as quickly as possible.

The choice of compressing or not compressing data is complicated by many factors affecting data-transmission rate. These factors affect the transmission rate differently depending on whether or not we use compression. The load on the sending and the receiving machines, and the degree of traffic congestion of the network are two of the factors. For example, if the load on the sending machine is heavy, compression may not work very well since the load will reduce the compression rate. Other factors include the speed of the sending and receiving processors, and the type of data being sent. Ideally, compression increases transmission rate since the size of the file to be transferred is reduced. However, compression may not increase the rate because of factors like the load on the sending machine.

1.2 Thesis Overview

The purpose of this thesis is to propose and develop algorithms for automatically determining whether compression should be used. One algorithm is based on sampling past performance with and without compression. The other directly exploits feedback from the network. To evaluate the effectiveness of the algorithms, experiments have been designed and run. These experiments were performed under a variety of conditions. For example, connections with different bandwidths and varied loads on sending machines were used.

Chapter 2 gives an introduction to data compression and how data compression can be divided into different categories. The Lempel-Ziv 77 algorithm, on which the compression algorithm used in this thesis is based, is described. Moreover, an enhanced compression algorithm used for sending a mixed stream of compressed and uncompressed data is described. In Chapter 3, the algorithms for automatically determining whether compression will or will not be used are described in detail.

In Chapter 4, experiments designed to evaluate the algorithms in Chapter 3 are described in detail. The conditions under which the programs are run are also described. In Chapter 5, the results of each of the experiments in Chapter 4 are analyzed and described. As well, the two algorithms in Chapter 3 are compared. Chapter 6 consists of a discussion on research work done in related areas. Finally, Chapter 7 consists of a summary of results and a discussion of the algorithms in Chapter 3. Some directions towards future work are also suggested.

Chapter 2

Data Compression

In this chapter, an introduction to data compression and the categories into which data compression is divided are presented. Moreover, the compression algorithm that this research work is based on is described. An enhanced version of this algorithm, modified so that a mixed stream of compressed and uncompressed data can be sent, is also described.

2.1 Introduction

Data-compression techniques attempt to transform data into a more compact representation. For many years, data compression has been used in file systems to reduce the amount of space needed to store data [BJLM92]. Data compression can also reduce data transmission costs [Wel84].

Compression techniques can be either lossy or lossless. Lossy compression means that part of the data may be lost after compression whereas lossless compression means that no data is lost after compression. To further illustrate the distinction

between lossy and lossless compression, Figure 2.1 shows the process of compression and decompression.

Figure 2.1: Compression/decompression process of a file X

The input file, X, is compressed, and the output of the compression process is Y. Y is then decompressed, and the output of the decompression is Z. If Z is guaranteed to be equal to X (for all inputs X), then the compression is lossless, otherwise it is lossy. Only lossless compression techniques will be involved in this research.

There are many lossless compression techniques available. The two main categories are static and adaptive techniques. Static techniques make use of a known data dictionary, which is some statistical distribution of the input data, to do compression. An initial pass over the data may be used to construct the data dictionary before the data are actually encoded in the second pass. Adaptive techniques, on the other hand, construct the data dictionary on-the-fly. Huffman encoding is an example of the former, whereas Lempel-Ziv 77 is an example of the latter. In this research, the Lempel-Ziv 77 algorithm is used.

Compression techniques also vary in their ability to compress data, in their compression speeds, in their decompression speeds, and in the amount of memory they use. The Lempel-Ziv 77 algorithm used in this research is fairly fast, and compresses data fairly well compared to other algorithms in the same category

according to the comparison test presented in [WMB94]. This test was done using the Calgary corpus as input. The amount of memory used is also moderate. The Lempel-Ziv 77 algorithm is also a simple algorithm to implement [WMB94].

2.2 The Original Lempel-Ziv 77 Algorithm

The Lempel-Ziv 77 compression algorithm was developed by two Israeli researchers, Abraham Lempel and Jacob Ziv in the late 1970s. The Lempel- Ziv method of compression is described in [WMB94] as follows:

> [The Lempel-Ziv 77 algorithm] makes use of adaptive compression — a kind of dynamic coding where the input is compressed relative to a model that is constructed from the data that has just been coded. By basing the model on what has been seen so far, the algorithm is able not only to encode in a single pass through the input file, but is also able to compress a wide variety of inputs effectively rather than being fine-tuned for one particular type of data such as English text.

Figure 2.2 shows an example of Lempel-Ziv 77 compression.

Compression is done symbolwise. The output of the encoding is a *length* and an *offset* for a sequence of one or more symbols from the input. Perhaps it is easier to explain the encoding using an example.

Suppose our input is the sequence "abaababc". To encode the first symbol, 'a', it is checked against the previous processed input (Previous input is to the left of the symbol to be encoded) to see if there is a match. There is no match since there is no previous processed input in this case. So the first symbol is encoded

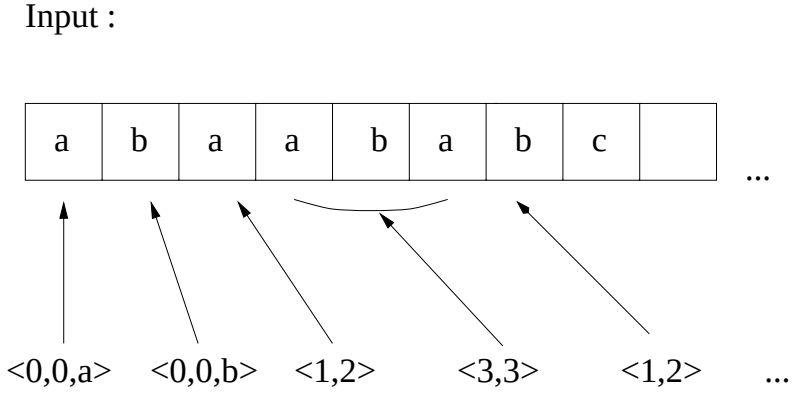

Figure 2.2: An example of the original Lempel-Ziv 77 compression

with *length* being 0 and *offset* being 0, and the symbol 'a' is sent. To encode the second symbol, 'b', this symbol is checked against the previous input (symbols to the left) to see if there is a match. There is no match in this case, and so 'b' is encoded with *length* being 0 and *offset* being 0, and the symbol 'b' is sent. As for the third symbol, 'a', it is checked against the previous input to see if there is a match. It turns out that there is a match with the first input symbol — 2 symbols to the left of the symbol currently being encoded. Whenever there is a match, the encoding algorithm tries to find the longest match possible by continuing to compare the next symbol after a match is found. In this case, the fourth symbol, 'a', is compared against the second symbol. This time there is no match, so the third symbol is encoded with *length* being 1 and *offset* being 2. Now the fourth symbol, 'a', becomes the current symbol to be encoded. In this case, there is a match with the third symbol as well as a match with the first symbol. However, the longest match occurs when the substring 'aba' starting from the fourth symbol matches with the same substring starting with the first symbol. So the encoded output for the fourth, the fifth, and the sixth symbol together is *length* being 3, and *offset* being 3. The rest of the input sequence is encoded similarly until all the

input has been encoded.

In general, the output of the encoding process is a *length* and an *offset* representing one or more symbols. The symbol to be coded is searched for in a search window of size S bytes in the previously processed input. There is also a limit M on the maximum length of match. Figure 2.3 illustrates the search window and the maximum length of match. In the figure, S is the search window and M is the maximum length of match.

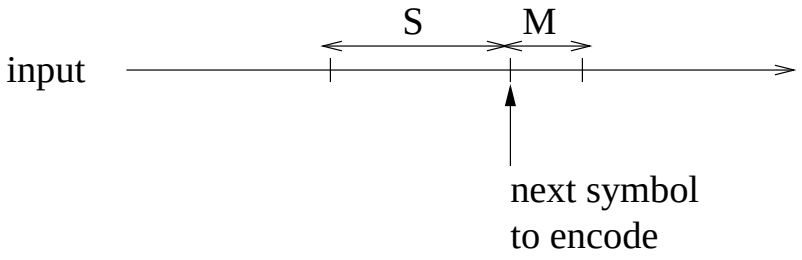

Figure 2.3: Search window and maximum length of match

2.3 The Enhanced Lempel-Ziv 77 Algorithm

In order to be able to transfer large volumes of data efficiently from the sender to the receiver, input data are partitioned into blocks, and for each block of data being sent, a decision is made as to whether compression is used. A coding scheme that can be used to send a mixed stream of compressed and uncompressed data is needed. The new scheme is based on the Lempel-Ziv 77 algorithm. It has two modes: the regular mode in which blocks of data are compressed, and a transparent mode in which data blocks are not compressed. In the regular mode, the output is coded

using the simple Lempel-Ziv coding scheme described previously. In the transparent mode, a block of length N bytes is encoded by prepending the triple $< 0, 257, N >$ to the block itself. The first two elements of the triple are an invalid $< length, offset >$ pair used to tell the decoder that unencoded data will follow. Figure 2.4 summarizes the enhanced coding scheme used in this thesis.

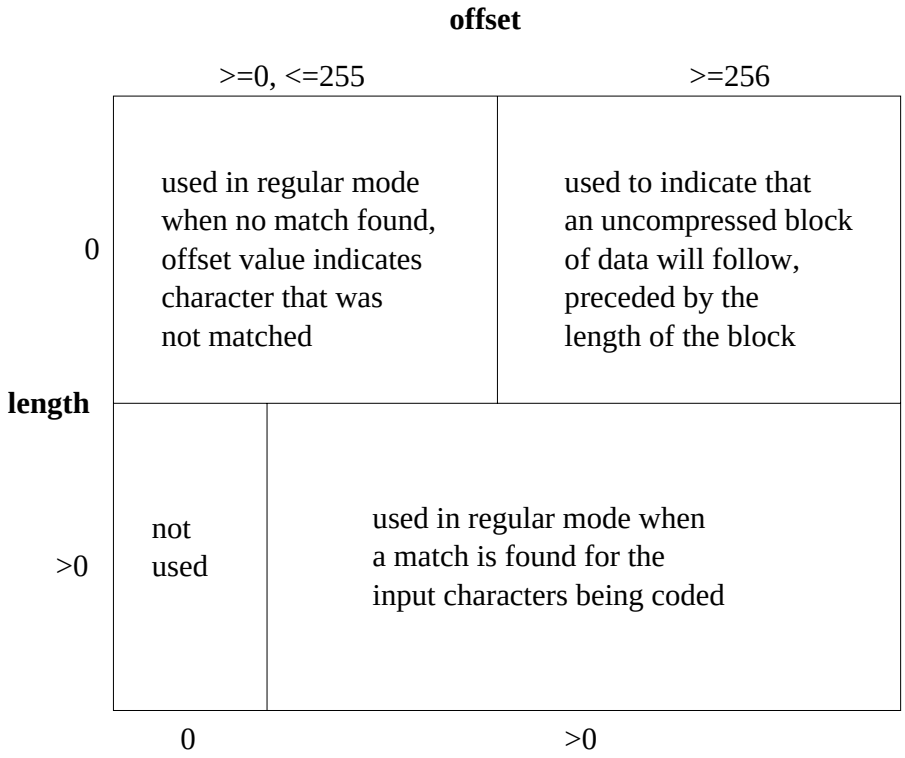

Figure 2.4: Modified coding scheme based on Lempel-Ziv 77 algorithm

For efficiency purposes, other improvements to the basic Lempel-Ziv 77 algorithm have also been made. A hash table is used to speed up the search for matching strings described in Section 2.2 . In regular mode, the hash table is updated once for every character in the input. If a block of data using the regular mode is preceded by one using the transparent mode, the hash table may be out-of-date if

it is not updated in the transparent mode. So, to ensure that the hash table is up-to-date when it is needed, the hash table is updated when data to be processed in the transparent mode is within the search window S. Figure 2.5 illustrates the above condition.

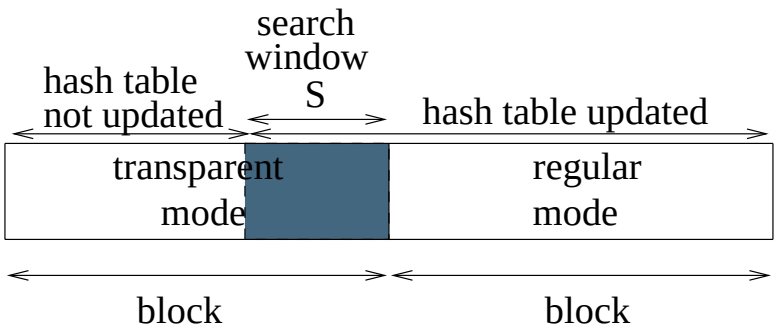

Figure 2.5: Hash table update condition for transparent mode

Chapter 3

The Controlled-Compression Algorithms

In this chapter, two algorithms for controlling the use of compression are described. Algorithm 1 is based on sampling past performance whereas Algorithm 2 is based on the congestion condition of the communication channel through which data are transferred. In both cases, the decision of whether to use the regular mode (i.e.,compression) or the transparent mode (i.e., data are sent without being compressed) is made for each block of data to be transferred.

3.1 Algorithm 1

Assume that the rate with which data can be transferred using the regular or transparent modes changes only slowly over time. Specifically, assume that the expected time required to transmit a block in a particular mode is given by the average transmission time of blocks recently transmitted in that mode. For block i,

block $i-W$ through $i-1$ are considered to be the recently transmitted blocks. The window size, W, is a parameter of the algorithm.

For the assumption to be reasonable, the value of W should not be too large. If the assumption holds, then it is reasonable to use the performance of the past W blocks to predict whether it would be better to use the regular mode or the transparent mode for the next block.

Algorithm 1 determines which of the two modes has performed best recently, and sends subsequent blocks using that mode. Let

- W_R be the set of blocks recently transmitted in regular mode
- W_T be the set of blocks recently transmitted in transparent mode

Note : $\mid W_R \mid + \mid W_T \mid = W$ and $W_R \cap W_T = \emptyset$

Each time a new block is to be transmitted, the algorithm uses the following to determine which mode to use for that block.

```
if W_R = ∅ then use regular mode
else if W_T = ∅ then use transparent mode
else
        t_R = mean time to transmit blocks in W_R
        t_T = mean time to transmit blocks in W_T
        if (t_R < t_T) then use regular mode
        else use transparent mode
```

In the above algorithm, the transparent mode is used at least once for every W blocks transmitted. The same is true for the regular mode. So W not only

represents how quickly conditions can change, it also affects the overhead that this algorithm has to pay since for every W blocks at least one block uses the mode with poorer performance.

Algorithm 1 requires that block transmission time be measured. Our implementation measures transmission time as the difference between the return values given by the Unix system call, "time", called before and after a data block is processed and sent to the network. This includes the time to process the data block, and the time required for the Unix system call, "send", to pass the data to the operating system. The "send" call will block if there is insufficient room in the operating system's buffer for the data. Such blocking delays are also included in the measureed time.

3.2 Algorithm 2

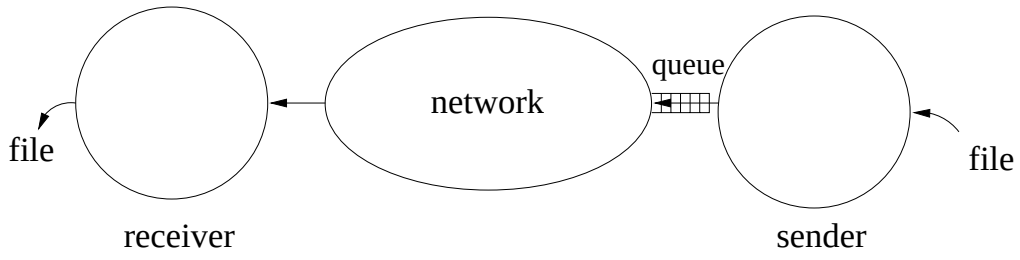

Figure 3.1: Original model with modifications for Algorithm 2

Figure 3.1 shows our model again with some additions. Assume that there is a network queue and that data are sent into the network through this queue. If the rate at which the sender sends data into the queue is greater than the rate at which the data leave the queue and enter the network, the queue eventually becomes full. Whenever the queue is full, the transmission channel (that is, the socket in this

case) through which data is sent to the receiver blocks. On the contrary, if the rate at which the sender sends data into the queue is less than the rate at which the data leave the queue and enter the network, the queue eventually becomes empty. Based on the condition of the queue, we know the relative rates of data entering and leaving the queue. By regulating the relative rates, data can be transferred efficiently from the sender to the receiver.

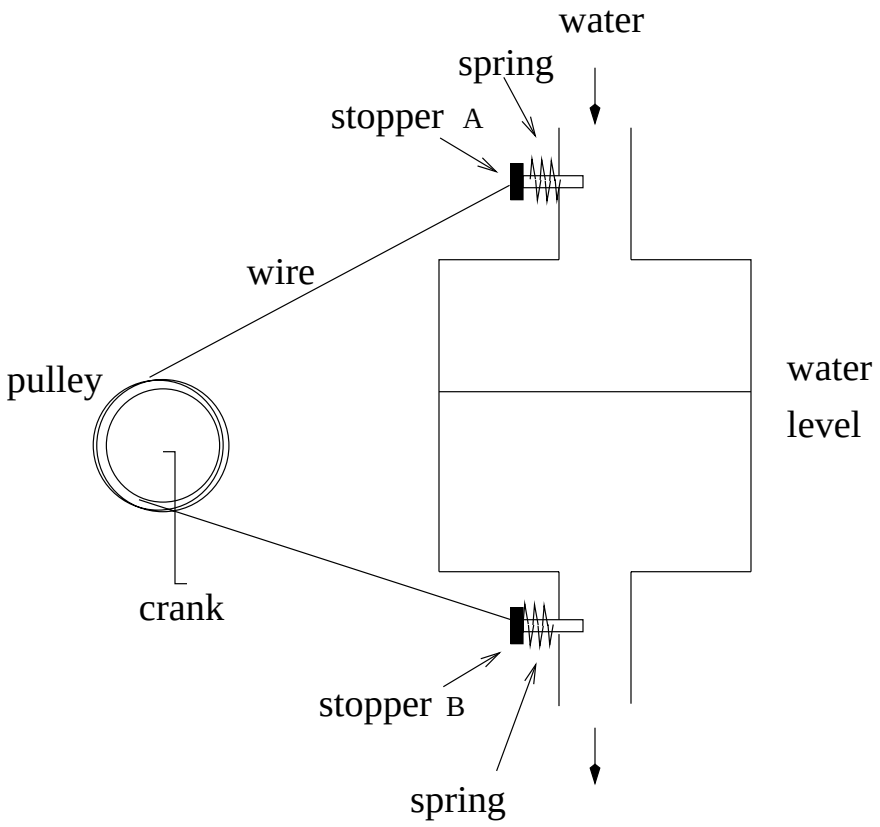

Figure 3.2: An analogy of the classic feedback approach

Algorithm 2 regulates these rates using a feedback approach. Figure 3.2 shows a mechanical system analogous to the feedback mechanism. In the figure, water is running into a funnel through the upper neck and out of the funnel through the lower neck. The upper neck is guarded by stopper A, an adjustable stopper

attached to a spring to control the flow of water into the funnel, linked to one end of a wire wound several times around a pulley with a crank. The other end of the wire is connected to stopper B, also an adjustable stopper attached to a spring used to control the flow of water out of the funnel through the lower neck of the funnel. When the crank is turned in a clockwise direction, the tension on the part of the wire closer to stopper A decreases, and stopper A is pulled further in by its spring. At the same time, the tension on the part of the wire closer to stopper B increases, and stopper B is pulled further out. As a result, the rate of water flow through the upper neck is decreased and that through the lower neck is increased. On the contrary, when the crank is turned in a counter-clockwise direction, the tension on the part of the wire closer to stopper A increases, and stopper A is pulled further out. At the same time, the tension on the part of the wire closer to stopper B decreases, and stopper B is pulled further in by its spring. As a result, the rate of water flow through the upper neck is increased and that through the lower neck is decreased.

The water level in the funnel is an indicator of the relative rates of water flowing into and out of the funnel. Maximum flow is achieved when the two stoppers allow the same amount of water to pass through. The water level in the tank can be monitored to determine when this condition has been achieved. If the rate of water flowing in is greater than the rate of water flowing out, the water level will rise. If the rate of water flowing in is less than the rate of water flowing out, the water level will fall. If the rates are the same, the water level stays constant. Thus, the goal should be to adjust the crank so that the water level remains constant.

In the case of this research, the queue from the sender to the network is analogous to the water funnel. Data leaving the queue is like water dripping out of the funnel through the lower neck. Data entering the queue is like water running into

the funnel through the upper neck. Switching to the regular mode of data transfer is like turning the crank in the clockwise direction. In the regular mode, the rate at which data are sent into the queue is decreased since time is spent in compression at the sender. At the same time, since data are compressed into fewer bytes, it appears to flow out of the queue and through the network more quickly. (This assumes that the network, not the receiver, is the bottleneck after the queue.) On the contrary, switching to the transparent mode is like turning the crank in the counter-clockwise direction. In the transparent mode, since the data are not compressed, the rate at which data enter the queue is greater than that in regular mode, and the rate at which data leave the network is decreased.

Since the throughput of the entire system depends on the minimum of the queue input and output rates, the best throughput is achieved when these rates are equal. This is analogous to keeping the water level constant in the mechanical system described above. If the rate of data entering the queue is greater than the rate of data leaving the queue, the former should be decreased and the latter should be increased by doing more compression. If the rate of data entering the queue is less than the rate of data leaving the queue, the former should be increased and the latter should be decreased by doing less compression.

Algorithm 2 tries to maximize throughput by adjusting the rates of data entering the queue and leaving the queue so that they are equal. The algorithm uses a target compression ratio, C_{calc}, which it expects will equalize the queue input and output rates. Algorithm 2 tries to achieve this target compression ratio by deciding which mode of data transfer to use for each block of data to be transferred. This target compression ratio is adjusted once in every W blocks according to the condition of the network queue. W is one of the parameters of this algorithm. Algorithm 2 implements a discrete approximation of the stopper control mechanism illustrated

in 3.2. If the compression ratio is approximately constant for all the W blocks in the window, there are only $W+1$ choices of possible average compression ratios that correspond to stopper positions in the analogy.

Ideally, the algorithm would increase C_{calc} if it observed an increasing backlog of data awaiting transmission in the queue. However, the data awaiting transmission is maintained within the operating system and is not visible to the algorithm in the implementation. The only feedback the algorithm receives from the operating system concerning the state of the transmission queue is a flag that is set when an attempt is made to add more data onto a full queue.

To guess the state of the transmission queue, the algorithm keeps track of whether or not the queue is full when it attempts to send each block. The percentage of the recently-transmitted blocks for which the queue was full is denoted by T_{meas}. The algorithm attempts to adjust C_{calc} to keep T_{meas} as close as possible to a specified target value, T_{theory}. For all of the experiments reported in Chapter 5, we have set T_{theory} equal to 50%.

Algorithm 2 works as follows. Let

- C_{calc} be the target compression ratio
- C_{last} be the compression ratio of the most recent block that uses the regular mode
- C_{min} be the minimum compression ratio to be used
- C_{max} be the maximum compression ratio to be used
- k be the compression adjustment factor

Before block i is transmitted, the algorithm uses the following rule to determine which mode to use for that block.

```
   C_meas = the average compression ratio achieved over the W most recent
blocks
   if (i mod W = 0) then
          /* do this once for every W blocks */
          T_meas = the percentage of blocks amongst W most recent
                   blocks for which the queue was full
          if (T_meas < T_theory)
                /* decrease compression */
                C_calc ← max(C_min, C_meas - kC_last/W)
          else
                /* increase compression */
                C_calc ← min(C_max, C_meas + kC_last/W)
   if (C_meas < C_calc)
          use regular mode
   else
          use transparent mode
```

The algorithm assumes that regular mode increases network throughput. For this assumption to hold, the files have to be compressible. To accommodate non-compressible files, a preliminary compressibility test may be added to the algorithm. If a file is not compressible, the transparent mode should be used for all of the file's blocks. Algorithm 2 is also unable to distinguish between a slow network and a slow receiver. If the receiver is a bottleneck, there will be a backlog in the network queue. When Algorithm 2 senses the backlog, it will try to increase compression at the sender. This makes matters worse for the receiver because when data arrive at the receiver, the receiver has to decompress them. For this reason, Algorithm 2 may not work well when there is a heavy load on the receiver.

The minimum value, C_{min}, that C_{calc} can take is set to 1.0 since files are assumed to be compressible. The maximum value, C_{max} is set to a little higher than the highest compression ratio achieved for any transmitted block (i.e. 1 + highest compression ratio achieved). As more blocks are transmitted, the value of C_{max} may be adjusted. The value of C_{max} is set this way because this is a compression ratio that that particular type of data can achieve, and so it is more realistic. As a result, even if the value of C_{calc} is adjusted upwards, its value will not reach unrealistic values.

The speed with which the algorithm reacts to changes in its environment can be affected by controlling the size of the periodic adjustment to C_{calc}. Suppose that an average compression ratio of C is obtained for the blocks that use regular mode. An increase of 1 in the number of blocks that use regular mode in each window will increase the overall compression ratio for the window by $\frac{C-1}{W}$. As shown, the algorithm uses $\pm\frac{C_{last}}{W}$, a simplified version of the formula above, as the increment for C_{calc}. Larger values (for example, integer multiples of $\frac{C}{W}$) could also be used to cause C_{calc} to change more quickly by using a larger value of k. The compression adjustment factor, k, is another parameter of Algorithm 2.

The difference between Algorithm 1 and Algorithm 2 is that the basic idea of the former is a compress/no-compress decision based on past performance (aside from the overhead that Algorithm 1 has to pay). That is, aside from the fact that the worse-performing mode is used once every W blocks, with no load or a constant load at the sender, one of the modes always performs better than the other and so the mode that performs better is always used for the next block. Algorithm 2, however, has a target compression ratio which may take on values between C_{min} (no compression) and C_{max} (compress all blocks). To achieve its target, Algorithm 2 may compress some, but not all, of the blocks.

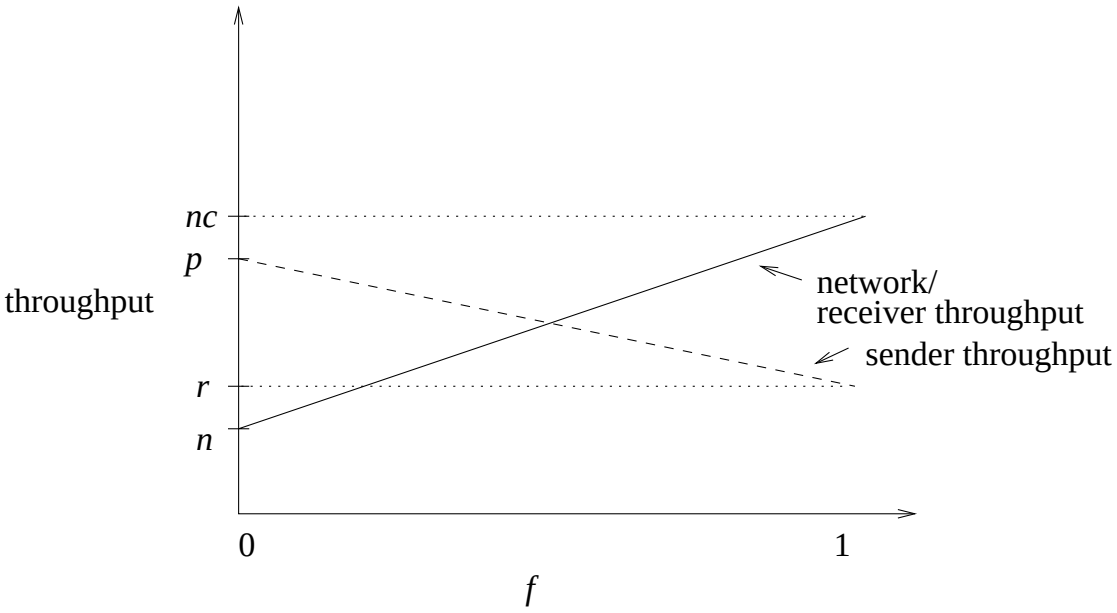

Figure 3.3: A graph with throughput plotted against the fraction of blocks that uses regular mode

Algorithm 2 is more complicated and difficult to use than Algorithm 1. A natural question to ask is whether it is worthwhile to use an algorithm capable of deciding to compress only a fraction of the data blocks. Let

- p be the rate at which the sender sends data using the transparent mode
- n be the rate at which data pass through network and receiver
- r be the rate at which the sender sends data using the regular mode
- c be the compression ratio ($c \geq 1$)
- f be the fraction of blocks in the window that uses the regular mode

Figure 3.3 shows a graph with the throughput plotted against f, the fraction of blocks for which regular mode is used. One line on the graph shows how the

network/receiver throughput changes with f. The endpoints of this line are n and nc because when f=0, no blocks use regular mode and so n is the minimum value of the network rate. When f=1, the network is transferring data at a rate of n times c, and so nc is the final and maximum point of this line. The other line shows how the sender's throughput varies with f. The endpoints of this line are p and r when f is 0 and 1 respectively. The throughput achieved by the whole system is the minimum of the sender's throughput and the network/receiver's throughput. The intersection of the two lines, if there is any, indicates the optimal value of f. If the sender's line is a lot higher than the line for the network/receiver, that is, if r is greater than nc, then there is no intersection between the two lines. In this case, the optimal value for f is 1. If the sender's line is a lot lower than the line for the network/receiver, that is, if r is smaller than nc, then there is also no intersection between the two lines. In this case, the optimal value for f is 0.

Using this graph, another graph was plotted with r against n to illustrate the conditions under which it is desirable to compress some, but not all, of the blocks. Figure 3.4 shows such graph. The parameter space can be divided into several regions.

- Region 1 ($r \geq nc$ and $r < p$):

  If the rate at which the sender sends data using the regular mode is greater than rate at which compressed data pass through the network/receiver, then regular mode should be used for all the blocks.

- Region 2 ($p < n$ and $p \geq r$):

  If the rate at which the sender sends in transparent mode is slower than the rate at which data pass through the network/receiver, and the sender can

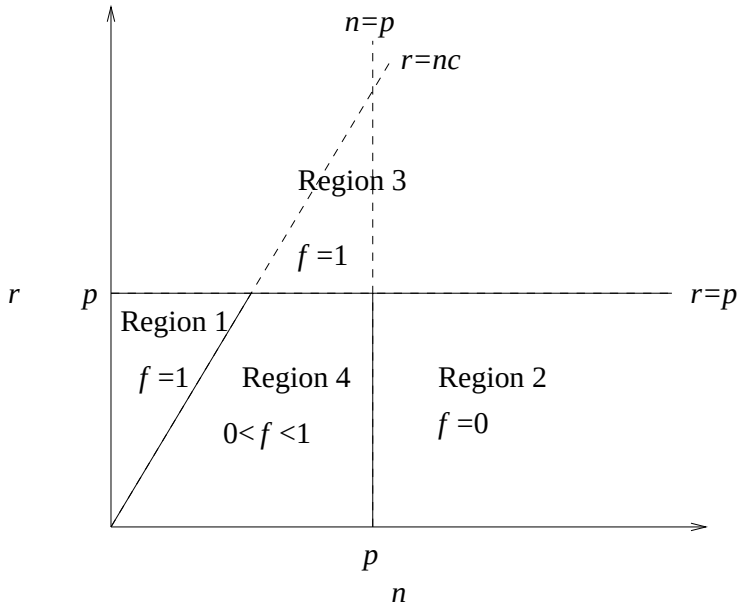

Figure 3.4: A graph with rate plotted against the fraction of blocks that uses regular mode

send faster in transparent mode than in regular mode, then all the blocks should use the transparent mode.

- Region 3 ($r > p$):

  If the rate at which the sender sends data using the regular mode is greater than that at which the sender sends data using the transparent mode, then all the blocks should use the regular mode. This situation does not normally occur in practice. It is included for completeness.

- Region 4 ($r < nc$ and $p \geq n$):

  If the rate at which the sender sends data using the regular mode is slower the the rate at which compressed data pass through the network/receiver, and the rate at which the sender sends data using the transparent mode is

faster than the rate at which data pass through the network/receiver, then some blocks should use regular mode and some should not.

Region 4 is the situation where the potential benefits of Algorithm 2 are visible.

Chapter 4

The Experiments : Setup and Environment

Experiments were designed and performed to demonstrate that the controlled- compression algorithms are able to pick correctly which mode to use in different environments and to compare the algorithms. Each experiment involved transmitting files from one machine to another. Two pairs of machines were used as transmission endpoints — one pair with a low-bandwidth connection through the Internet, and the other pair with a relatively high-bandwidth connection over a local-area network (LAN). To allow for more control over the load of the machines, the experiments were done when the sending and receiving machines were otherwise idle.

In this chapter, the general setup of the experiments and the environment in which they were performed are described.

4.1 The Setup and Environment

Three machines were used in the experiments. The sender, S, is a SUN Sparc- 5 workstation running Solaris 2.3. The receivers, R_L and R_H, are Sun workstations running SunOS 4.1.3. Both R_H and S are located at the University of Waterloo, and are connected by a 10Mbps local-area network. R_L is located at the University of Maryland in College Park, MD. The bandwidth available between S and R_L is substantially lower than that between S and R_H.

Each experimental run consisted of four transfers of a file between S and one of the receivers. The file was first transferred using the regular mode for all blocks, and then again using the transparent mode. Finally, the file was transferred using each of the two algorithms described in the previous chapter. For each run, a variety of statistics were collected, including the realtime required for the transfer, and the different resource usages such as the system CPU time and user CPU time. All of the experiments were done in the evenings since the network load is relatively low in the evenings, and since the load on both the network and the machines is less variable then.

The file used was a text file which compresses by a factor of 2.66 to 1 when compressed using our Lempel-Ziv 77 implementation in regular mode. The size of each block of data to be transferred was 64000 bytes, and there were about 180 blocks in the file for which statistics could be gathered and performance could be analyzed. For Algorithm 1, any value smaller than 10 for the history window size would impose a large overhead on the algorithm. So, the history window size was arbitrarily set to be 20 so that the overhead would not be large. For Algorithm 2, the compression adjustment factor, k, was set to 1 for simplicity, and the history window size was also set to be 20 for the sake of consistency with Algorithm 1.

The procedures, results and the interpretation of the results of the experiments are described in detail in the next chapter.

Chapter 5

Evaluation of the Algorithms

In this chapter, the procedures for the experiments designed to evaluate the performance of Algorithms 1 and 2 are described. The results and the interpretation of the results are stated, and the performance of the two controlled-compression algorithms is evaluated.

5.1 Experiments

5.1.1 Constant Load on the Sender Machine

The procedures described in this subsection demonstrate that the controlled-compression algorithms are able to pick the correct mode to use under conditions such as different loads on S. In order to introduce additional load on S, programs are run concurrently when data are transferred from S to one of the receivers. The program used is a compression program that reads data from a file, compresses it using the Lempel-Ziv 77 algorithm and then outputs it to another file. Several copies of

this program are run concurrently to create load on S. Define a unit of load on the sender machine as the execution of a single compression program.

Performance statistics for 15 experimental runs between the sender S and the receiver R_H were collected with no additional load on S. The 15 runs were spread over 3 evenings, with 5 runs per evening. An additional 15 runs were then performed over 3 more evenings, this time with 4 units of load added to the sender.

The above was done once using S and R_H, and once using S and R_L.

5.1.2 Variable Load on the Sender Machine

Several additional experimental runs were performed to illustrate the dynamics of the two algorithms under changing load conditions. During these runs, statistics (including the realtime, system CPU time, and user CPU time required to transfer each block of data, the compression ratio, and, for Algorithm 2, C_{calc}, C_{meas}, T_{meas}) were gathered for each block of the transferred file. Each of these runs began with no load on the sending machine, S. After part of the file was transferred, 4 units of load were applied to S. This was at about block 25 for Algorithm 1 and about block 40 for Algorithm 2. Shortly before the transfer was complete, the load was removed. This was at about block 130 for Algorithm 1 and about block 100 for Algorithm 2. These experiments were run between S and R_H.

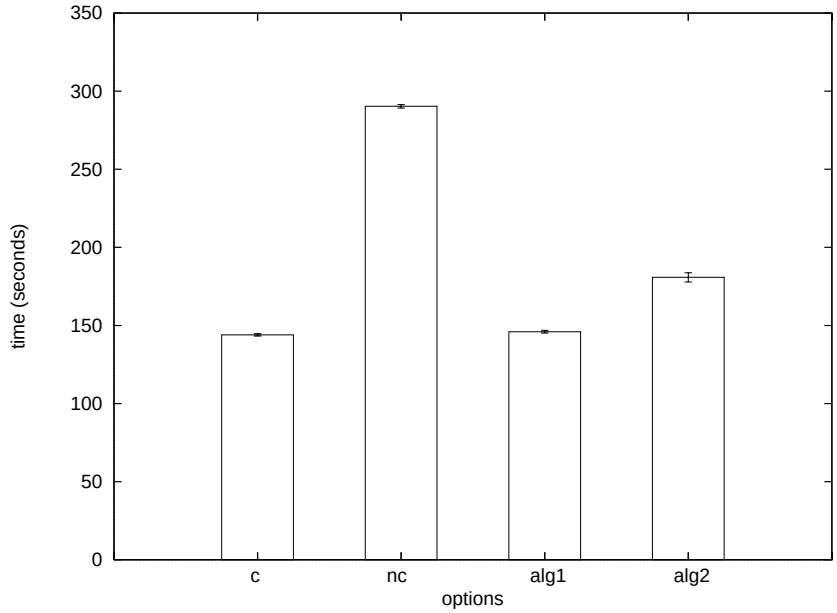

Figure 5.1: Results of data transfer from S to R_H with no additional load on S

5.2 Results and Interpretation

5.2.1 Constant Load on the Sender Machine

Figures 5.1 shows the average file-transfer time under each of the 4 options of data transfer for 15 runs when data were transferred from S to R_H. Figure 5.2 shows the same information when data were transferred from S to R_L. Zero units of additional load were added to S in both cases. In the figures, "c" means that regular mode is used for all the blocks; "nc" means that transparent mode is used for all the blocks; "alg1" means that Algorithm 1 is used; and "alg2" means that Algorithm 2 is used. The error bars indicate the 90% confidence interval for each average value.

In these experiments, the regular mode performed better than the transparent mode, and both Algorithm 1 and Algorithm 2 were closer to the regular mode than to the transparent mode.

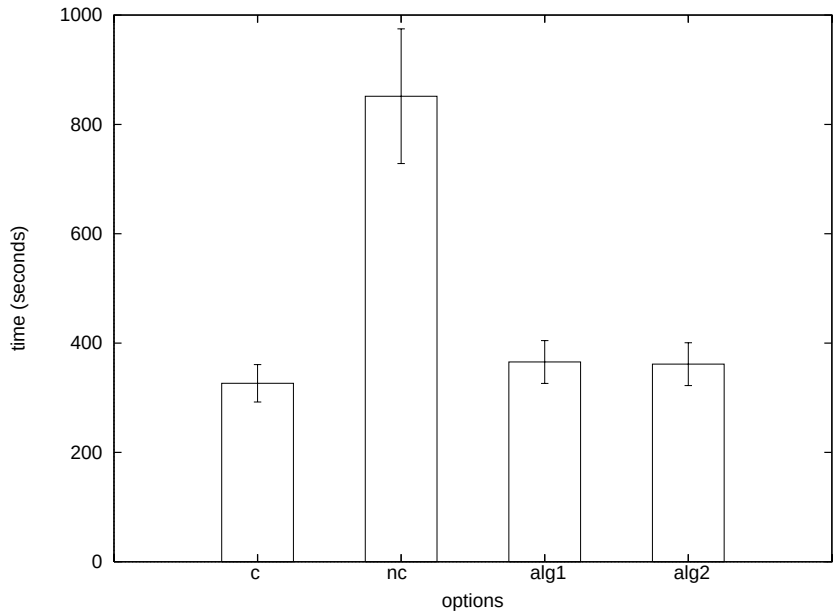

Figure 5.2: Results of data transfer from S to R_L with no additional load on S

Figure 5.3 and 5.4 show the results of the experiments in which 4 units of load were applied to S and data were transferred to R_H and R_L respectively.

In the case of data transfer from S to R_H, the transparent mode outperformed the regular mode. This time both Algorithm 1 and Algorithm 2 were closer to the transparent mode than to the regular mode. In the case where data were transferred from S to R_L, the regular mode still performed better. An explanation to why the regular mode performed better than the transparent mode in this case is that the additional load on the sender machine was not high enough to compensate for the relatively low network throughput. In this case, both Algorithm 1 and Algorithm 2 were closer to the regular mode. The above indicates that no matter which mode of data transfer was better in a given situation, Algorithms 1 and 2 performed about as well as that mode. Algorithm 1 performed better than Algorithm 2 between S and R_H, but the performance of both algorithms was similar between S and R_L.

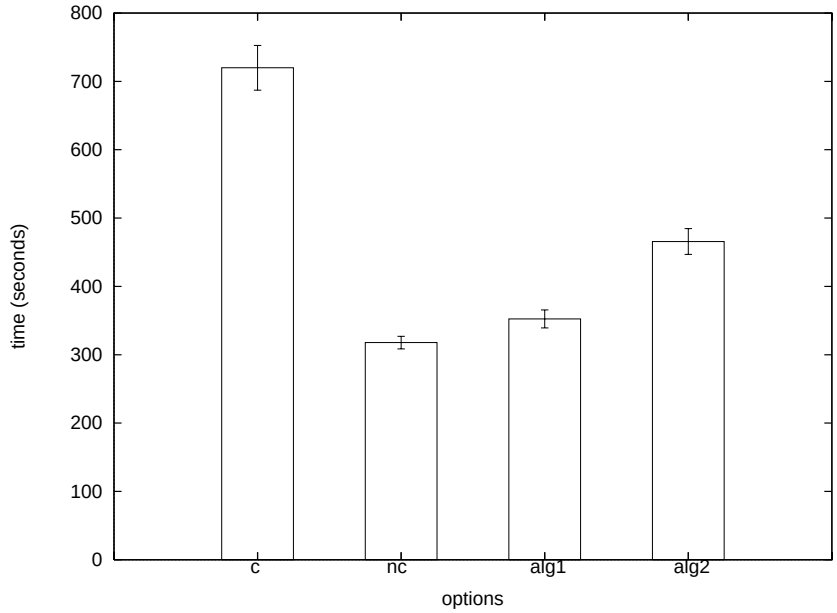

Figure 5.3: Results of data transfer from S to R_H with 4 units of additional load on S

However, this comparison would certainly be affected by changes to the tunable parameters of Algorithm 2.

5.2.2 Variable Load on the Sender Machine

In Figure 5.5, one curve illustrates the time to transmit blocks in regular mode. A point is plotted for block i if block i was sent in regular mode. The value plotted at i is the average of the times to transmit regular-mode blocks with indices in the range i through $i-W+1$. This averaging was performed to generate a smoother plot. The other curve in Figure 5.5 illustrates the same information, except that it is for the blocks that use transparent mode. Figure 5.6 shows the percentage of blocks that use regular mode. The value plotted at block i is the percentage of blocks that use the regular mode with indices in the range i through $i-W+1$.

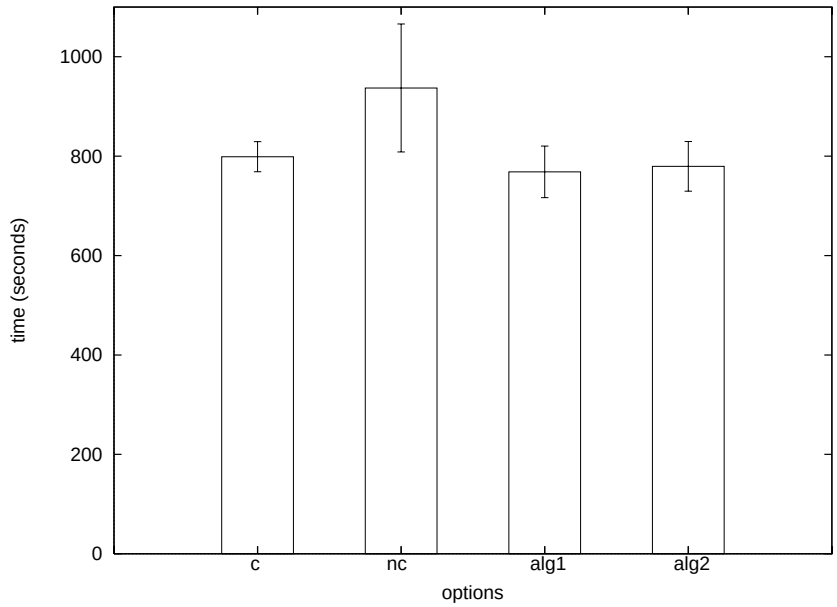

Figure 5.4: Results of data transfer from S to R_L with 4 units of additional load on S

Figure 5.7 and 5.8 show the same information as 5.5 and 5.6, respectively, for Algorithm 2.

It can be seen that before 4 units of load were added, the time taken for data transfer of each block using the regular mode was shorter than that when the transparent mode was used. So most of the blocks during this period used regular mode. When 4 units of load were added to the sender machine, CPU and other resources had to be shared between the additional load and the encoding of data using regular mode. The time taken for data transfer of each block using the regular mode was longer than that when transparent mode was used. So during this time, most of the blocks used transparent mode. Then, after the load was taken away, the time taken for data transfer of each block using the regular mode was shorter than that when the transparent mode was used. So in Algorithm 1, most of the

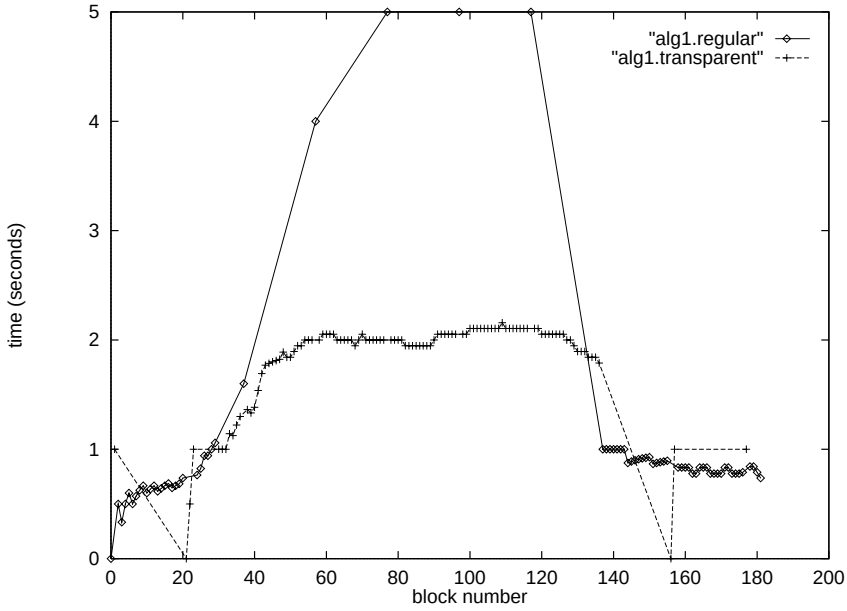

Figure 5.5: Effect of a variable additional load on the performance of Algorithm 1: S to R_H

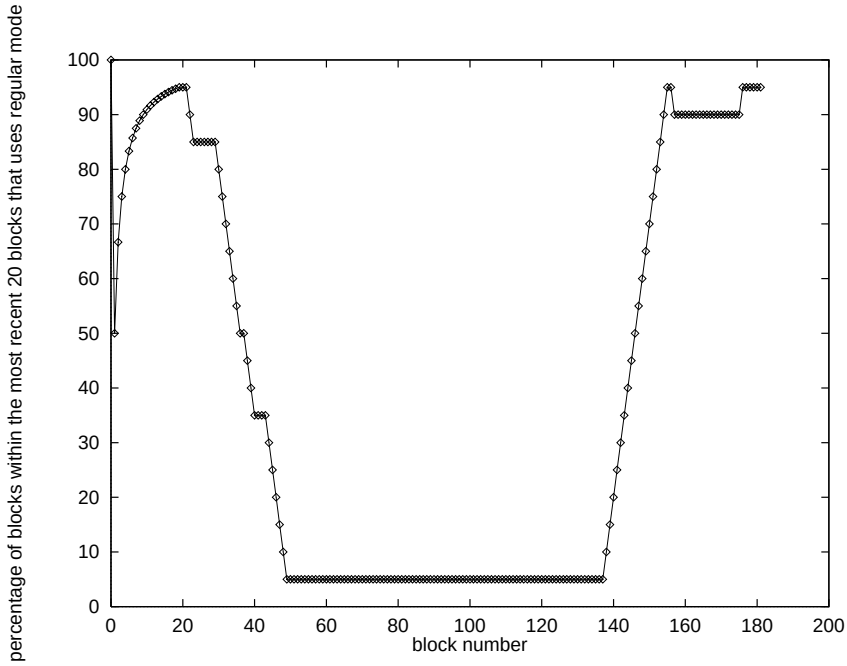

Figure 5.6: Percentage of the blocks using the regular mode

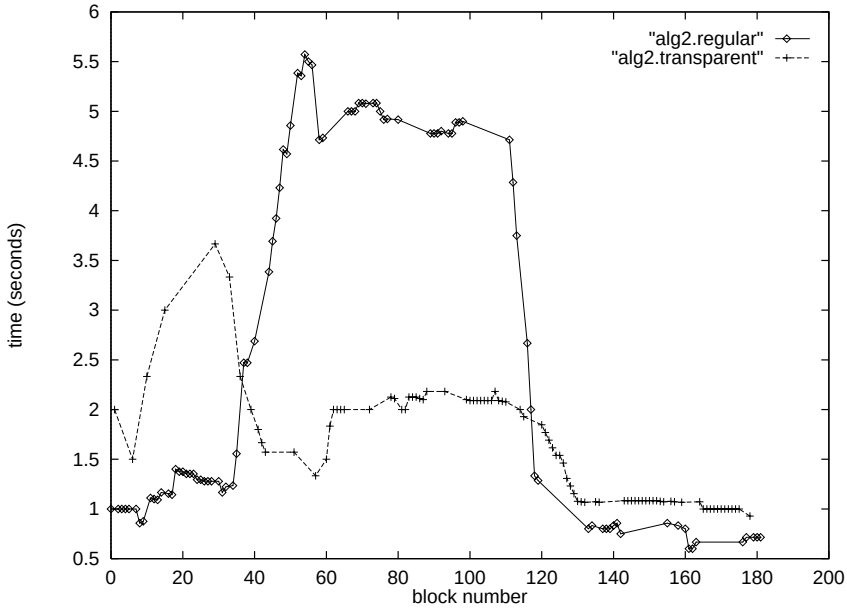

Figure 5.7: Effect of a variable additional load on the performance of Algorithm 2: S to R_H

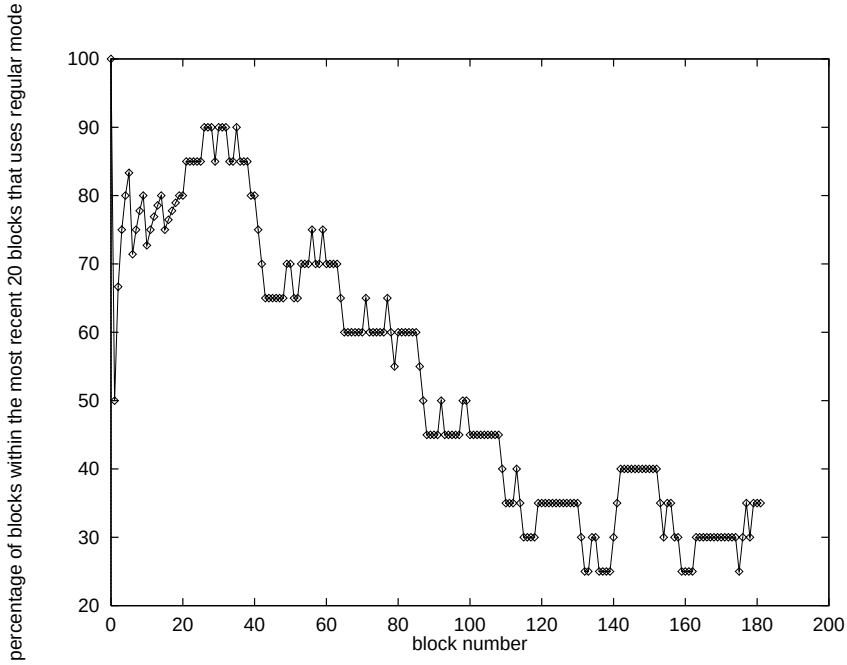

Figure 5.8: Percentage of the blocks using the regular mode

blocks used regular mode again.

For Algorithm 1, the percentage of blocks that used regular mode dropped when the load was added to the sender machine since regular mode became less efficient. Then it remained constant when the additional load was imposed on the sender machine since during this period, only 1 block out of 20 used the mode that performed worse as the overhead (in this case, it was the regular mode). When the load was taken away, the percentage of blocks that used regular mode increased again since regular mode performed better. In Algorithm 2, even when the load was taken away and regular mode performed better, a larger proportion of blocks used transparent mode than for Algorithm 1 in the same situation.

Algorithm 2 assumes that the probability of the queue being full, T_{theory}, should be 50%. In reality, 50% may not be the correct value to use. The value of T_{theory} should be adjusted downwards, say to 30%. By adjusting T_{theory} downwards, the probability of C_{calc} being adjusted upwards will be increased (i.e. the target compression ratio is increased), and hence more blocks will use regular mode to meet the higher target compression ratio.

The graph in Figure 5.7 gradually decreases in steps. The width of each step is approximately 20 blocks. The frequency at which C_{calc} is adjusted is once every 20 blocks, and this makes the step size 20 blocks. The graph is declining because C_{calc} is adjusted downwards once every 20 blocks to adapt to the additional load on S. Since C_{calc} is adjusted gradually, the percentage of blocks that use the regular mode in the most recent 20 blocks also changes gradually. This is also the reason that the curve in Figure 5.7 had not gone back up even after the load was taken away.

5.2.3 Drawbacks of the Algorithms

Algorithm 1 has the advantage of simplicity, but also has a drawback. The overhead that this Algorithm has to pay is that the mode that performs worse has to be used at least once every W blocks, where W is the size of the window in which past performance is evaluated.

Algorithm 2 has an advantage over Algorithm 1 in that there is no such overhead. However, it has several serious disadvantages. One of them is that Algorithm 2 needs the parameters, k, T_{theory}, and W, to be set correctly. For example, k and W affect how quickly and how much C_{calc} will change in response to changes in the network queue conditions. If the value of k is too large or the value of W is too small, C_{calc} may be too sensitive even to a very small change in network load. If the value of k is too small or the value of W is too large, Algorithm 2 may react too slowly to the changes of the network queue conditions. If T_{theory} is set too high, it will force C_{calc} to be below optimal value. On the contrary, if the value of T_{theory} is too low, C_{calc} will be forced to take a value above the optimal value. In general, tuning is more complicated in this algorithm. It may be difficult to determine a set of parameter values that will allow the algorithm to function effectively over a range of operating conditions.

Chapter 6

Related Work

Research on adaptive compression algorithms and rate-control methods have been in progress for some time. This chapter briefly describes both directions of research.

Adaptive source-coding techniques and constraints of communication systems are discussed in [Cla89]. This paper lists the design criteria for realtime or on-line communication systems and some adaptive methods for such systems. Some of the constraints are compression performance, avoidance of data expansion, and implementation complexity. To avoid data expansion after compression is applied, a preliminary compression-ratio test on the data is done. The Lempel-Ziv 77 compression algorithm is suggested here as the compression algorithm to use because of its simplicity. Some adaptive methods to enhance the Lempel-Ziv algorithm use a sliding-window mechanism or a dynamic library. Some hybrids of the two are also described and compared.

In [Ran93], a number of compression options and alternatives are suggested. The criteria for picking which compression options to use for wide-area networks in this paper are compression ratio, speed and size of memory required to implement

the protocol. Suggestions on which compression algorithm to use for a particular line speed of the data link are given.

The paper [Ran94] defines a method for negotiating data compression over PPP links. PPP is a communication protocol for the Internet. (Details of this protocol can be found in [Sim94].) Compression is an optional facility in the PPP protocol. It is negotiated after the data link has been established. Two methods of employing data compression when using multiple PPP links are also described.

Recommendations for the modem standard V.42*bis* in the paper [CCI90] indicate the need to use data-compression procedures to improve throughput, and the need to interoperate with data circuit terminating equipment (DCE) that do not provide data compression. This paper also suggests a particular data-compression procedure for a series of DCEs, namely the V-series. The proposed procedure includes a compressed mode of operation for data that can be compressed, and a transparent mode of operation for use when uncompressible data is detected. However, unlike algorithm 2 in this thesis, the data-compression facility is negotiated at the beginning of the data transfer and compressibility is the only factor considered in switching the mode of operation. The recommendation does not consider other factors, such as network traffic, that can affect the throughput. As well, error-control mechanisms are included in the data-compression procedure in the recommendation.

A feedback mechanism was used in the rate control of packet video in [BT94]. This is very similar to the mechanism behind Algorithm 2 in this thesis except that the encoding used is a lossy one and there is a trade-off between quality and speed.

A feedback toolkit that makes use of lossy data compression in order to manage the frame rate and jitter in a video presentation is suggested in [CCWP95]. This

paper also lists various ways to improve the quality of video presentation using the feedback toolkit.

Chapter 7

Conclusions and Future Work

7.1 Summary and Conclusion

The maximum throughput of data transfer from a sender to a receiver can be achieved by deciding whether or not compression should be used on a block-by-block basis during data transfer. Two controlled-compression algorithms are developed in this research. Algorithm 1 examines the past performance and picks the mode that performs better whereas Algorithm 2 monitors the fullness of the network queue. These algorithms are evaluated by performing experiments under different conditions. Although these algorithms have a different theory underlying them, both are, in general, able to pick the correct mode of data transfer to use and are able to perform well.

Algorithm 1 is simpler and requires less tuning. Algorithm 2, on the contrary, requires more complex tuning. If it is improperly tuned, Algorithm 2 may perform worse than Algorithm 1 in steady state and may react too slowly (or too quickly) when conditions change.

7.2 Future Work

The model used for Algorithm 2 is not able to distinguish between a slow network and a slow receiver. An extension to Algorithm 2 to allow this distinction to be made is to add a feedback loop from the receiver to the sender so that the sender is able to collect performance statistics from the receiver, and not just from the network queue alone. This way, the distinction between network load and the load on the receiver machine can be made and the performance of Algorithm 2 can be improved.

Bibliography

[BJLM92] M. Burrows, C. Jerian, B. Lampson, and T. Mann. On-line data compression in a log-structured file system. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 2–9, October 1992.

[BT94] J-C. Bolot and T. Turletti. A rate control mechanism for packet video in the internet. In *Proceedings of the IEEE Infocom*, volume 3, pages 1216–1223, June 1994.

[CCI90] CCITT. *Data Compression Procedures for Data Circuit Terminating Equipment (DCE) using Error Correcting Procedures*, 1990. URL: ftp://scitsc.wlv.ac.uk, under the directory : /pub/infomagic/standards.cdrom.Jan94/ccitt/1992/v/.

[CCWP95] C. Cowan, S. Cen, J. Walpole, and C. Pu. Adaptive methods for distributed video presentations. *ACM Computing Surveys*, 27(5), December 1995.

[Cla89] A. D. Clark. Adaptive source coding techniques for communications systems. In *Proceedings of the IEEE National Conference on Telecommunications*, pages 53–60, 1989.

[Ran93] D. Rand. Data compression for wide area networks. Technical report, Novell, Inc., 1993.

[Ran94] D. Rand. The PPP compression control protocol (CCP). Internet draft, Novell, Inc., 1994. URL: ftp://sgi.com/other/ppp-comp/.

[Sim94] W. Simpson. *The Point-to-Point Protocol (PPP)*, July 1994. URL: ftp://NIC.DDN.NIL, under the directory : /rfc/.

[Wel84] T. A. Welch. A technique for high-performance data compression. *IEEE Computer*, 17(6):8–19, June 1984.

[WMB94] I. H. Witten, A. Moffat, and T. Bell. *Managing gigabytes: Compressing and indexing documents and images*. Van Nostrand Reinhold, 1994.