Reducing Delay With Dynamic Selection of Compression Formats Extended Abstract

Chandra Krintz Brad Calder
Department of Computer Science and Engineering
University of California, San Diego
{ckrintz,calder}@cs.ucsd.edu

1 Introduction

Compression is used to reduce network transfer time of files by decreasing the number of bytes transfered through the use of compact file encoding. The resulting size of compressed files (compression ratio) is dependent upon the complexity of the encoding algorithm. Similar complexity is also required for decompression of the file prior to its use. Techniques with a high compression ratio are time consuming to decompress. Alternately, techniques with fast decompression rates are unable to achieve aggressive compression ratios and thus the transfer times are slower.

Compression is commonly used to improve the performance of applications that transfer over the Internet for remote execution, i.e. *mobile* programs. The overhead imposed by this compression-based execution is similar to mobile execution without compression: it includes the time for mobile code requests (program invocation and dynamic loading) and for transfer. However, since compression techniques must trade off compression ratio for decompression time, the latter must also be considered a source of delay since it occurs on-line while the program is executing. We refer to the combined overhead due to file request, transfer, and decompression as **Total Delay**.

To minimize total delay, a compression technique should be selected based on the underlying resource performance (network, CPU, etc). Moreover, since such performance is highly variable [1, 9], selection of the "best" compression algorithm should be able to change dynamically for the *same* link. Such adaptive ability is important since the selection of a non-optimal format may result in substantial total delay (8-10 seconds in the programs studied) at startup or intermittently throughout mobile program execution. Much prior research has shown that even a few seconds of interruption substantially effects the user's perception of program performance [2].

To address this selection problem, we introduce **Dynamic Compression Format Selection** (DCFS), a methodology for automatic and dynamic selection of competitive compression formats. Using DCFS, mobile programs are stored at the server in multiple compression formats. DCFS is used to predict the compression format that will result in the least delay given the bandwidth predicted to be available when transfer occurs. We use the Java execution environment for our DCFS implementation since it is the most common language for this computational paradigm (remote execution). We incorporate an extension to Network Weather Service (NWS) [9] for network performance prediction.

The key contribution this work makes is the design and implementation of a system for the exploitation of the benefits available from existing compression techniques without the associated costs. Empirically, DCFS reduces 36% (5.5 seconds) of total delay on average, for the range of network characteristics and compression techniques studied. DCFS achieves 52% (10 seconds) reduction in total delay on average over the commonly used Java archive (jar) format.

Program	Static Class Count	Size (kB) Static Classes	Application Decompression Rate (kB/sec) (Compressed Size in kB) PACK		JAR		TGZ	
Antlr	118	417.9	15.8	(57.8)	740.8	(221.8)	5111.2	(172.3)
BIT	53	147.6	14.5	(18.1)	615.5	(84.5)	1798.5	(57.0)
Jasmine	207	404.1	12.7	(34.1)	695.0	(219.0)	4161.9	(127.7)
Javac	76	548.3	14.8	(49.1)	810.1	(275.8)	5315.6	(179.2)
Jess	151	387.3	12.4	(22.7)	552.0	(185.4)	4724.7	(164.8)
JLex	20	84.9	13.4	(13.9)	238.1	(47.6)	909.5	(37.8)
AVG	104	331.7	13.9	(32.6)	608.6	(172.4)	3045.2	(123.1)

Table 1: Benchmark Statistics. The first two columns of data contain static class count and total application size. The final three columns show the decompression rate and compressed size (parenthesized) of the benchmarks encoded with three compression techniques PACK, JAR, and TGZ.

2 Compression Techniques

In this paper, we consider three commonly used compression formats for mobile Java programs: JAR, PACK, TGZ. We evaluated others as part of this study but include these three for brevity. The first is the Java archive (jar) format (referred to from here forward as JAR). JAR is the most common tool for collecting (archiving) and compressing Java application files and is based on the standardized PKWare zip format [5].

PACK [7] is a tool from the University of Maryland for the compression of JAR files. This utility substantially reduces redundancy by exploiting the Java class file representation and by sharing information between class files. The compression ratios achieved by this (Java-specific) tool are far greater than any other similar utility. However, PACK has very high decompression times since the class files must be reconstituted from this complex format.

Gzip is a standard compression utility commonly used on UNIX operating system platforms. Gzip does not consider domain specific information and uses a simple, bit-wise algorithm to compress files. As such, applications in this format can be decompressed very quickly but do not achieve the same compression ratios as more complex compression algorithms. The TGZ format refers to files that are first combined (archived) using the UNIX tape archive (tar), then compressed with gzip. Tar combines a collection of class files, uncompressed and separated by headers, into a single file in a standardized format.

Characteristics of each format are shown in Table 1 for the benchmarks used in the empirical evaluation of the techniques presented in this paper. The benchmarks are from the SpecJvm95 [8] benchmarks and other Java studies [6, 3]. The class count and the total static size (in KB) of all class files in each application are shown in the first two columns of data. The inherent trade-off made by compression techniques (compression ratio for decompression time) is exhibited in the final three columns of the table. The first number in each column is the decompression rate (KB/sec) for the application. The second, parenthesized number shows the compressed size (in KB) of the application from which the compression ratio can be computed (original application size (column 3) over compressed size). For example, the PACK format requires over 2.3 seconds to decompress the applications on average (JAR and GZIP require 89% and 98% less time, respectively), yet it enables a compressed file size that is 81% and 74% smaller than JAR and GZIP archives, respectively.

This data indicates that for slow networks PACK should be used due to its compression ratio, and for fast links TGZ should be used since it is inexpensive to decompress. *No* single utility enables the least total delay (request, transfer, decompression time) for all network performance characteristics and applications. In addition, each format is able to offer substantial benefit under certain circumstances. The choice of compression format should therefore be made dynamically, depending upon such circumstances to enable the best performance of mobile programs. To do this, we introduce **Dynamic Compression Format Selection** (DCFS), a technique that automatically and dynamically selects the format that results in the least total delay.

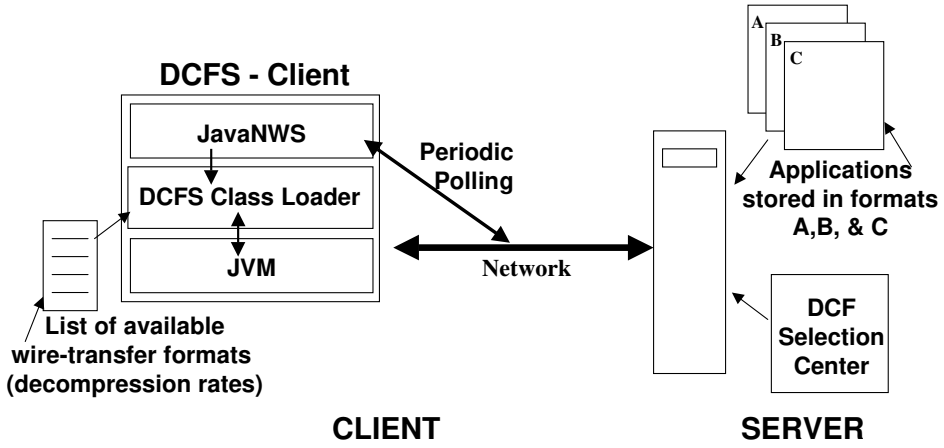

Figure 1: The Dynamic Compression Format Selection (DCFS) Model.

3 Dynamic Selection

Figure 1 exemplifies our DCFS model. The client-side Java Virtual Machine (JVM) incorporates a DCFS class loader. When an executing program accesses a class file for the first time (or the program itself is invoked), the request made to the JVM is forwarded to the DCFS class loader. Concurrently, a network performance measurement and prediction tool (called JavaNws) monitors the network connection between the client and the server at which the application is stored. The DCFS class loader acquires the network (as well as, possibly the CPU) performance value from the JavaNws and forwards the value(s) with the request to the server. With the initial server request, the DCFS class loader also includes the compression formats for which the client machine has decompression utilities.

At the server, applications are stored in multiple compression formats. When a server receives a request for an application or file, it uses the information sent (predicted resource performance value(s) and available compression formats) to calculate the potential total delay for each format. Therefore, given the predicted performance of measured by the client, the server determines the format that results in the least total delay for the file(s) requested. Total delay, again, consists of the request, transfer, and decompression time. The selected format is the one in which the application or class files are sent back to the client.

The JavaNws [4] utility at the client is an extension of the Network Weather Service (NWS) [10], a resource monitoring and prediction tool from the University of Tennessee. The JavaNws makes periodic measurements of the network performance between the client and the server which are used by a set of forecasting techniques to make short-term predictions of bandwidth and round-trip time. The forecasting techniques are further described in [9].

3.1 DCFS Results

To evaluate DCFS, we implemented both of the DCFS modules (DCFS client and server). However, to enable repeatability of results, we execute the modules on the same machine and simulate different networks between them. Instead of using JavaNws prediction, we use bandwidth and round-trip time averages from network traces. This enables the presentation of the upper-bound potential of DCFS. We provide a discussion of the impact of incorporating prediction in the next section.

Table 2 shows the bandwidth and round-trip time values used in this study and the corresponding networks. The networks include a 28.8 baud modem (MODEM) and an integrated services digital network link (ISDN) between a residence and university, a cross-country common-carrier

Name:	From:	To:	Bandwidth:	RT Time:
MODEM	Residence (East)	UTK (East)	0.03 Mb/s	95.0 ms
ISDN	Residence (East)	UTK (East)	0.11 Mb/s	48.0 ms
INET	UCSD (West)	UTK (East) (Internet)	0.53 Mb/s	79.0 ms
VBNS	UCSD (West)	UTK (East) (vBNS)	0.71 Mb/s	93.0 ms
LAN	UCSD (West)	LAN (10Mb/s Ethernet)	4.01 Mb/s	8.0 ms

Table 2: Description of the networks used in this study. Each network is represented by the average bandwidth value and round-trip (RT) time shown in the final two columns. These values were obtained from actual trace data collected using the connection.

Internet connection (INET), a cross-country vBNS connection (VBNS), and a 10Mb/s local area Ethernet connection (LAN). ISDN is a system of phone connections that allows data to be transmitted simultaneously using end-to-end digital connectivity. The vBNS is an experimental transcontinental ATM-OC3 network sponsored by NSF that can be used for large-scale and wide-area network studies. The table includes the average bandwidth and round-trip time measurements in the final two columns from 24-hour, JavaNws trace data for each network.

To compute total delay using this execution environment, upon client program invocation the server computes the sum of the average round-trip time (for the request), the transfer time (the average bandwidth value multiplied by the size of the compressed application), and the decompression time (the decompression rate multiplied by the size of the compressed application). The decompression rate is supplied by the client as part of the initial request. The total delay is calculated by the server for each of the compression formats, TGZ, JAR, and PACK, and the minimum is selected.

To illustrate the performance potential of dynamic format selection, we first present the percent reduction in total delay due to DCFS in Figure 2. The base case is the use of PACK compression (or JAR or TGZ, respectively for each bar of data presented) for each type of network without dynamic selection. The percent reduction is defined as $(TD_Base - TD_DCFS)/TD_Base$ for each network, where TD_Base is the total delay for the base case and TD_DCFS is the total delay using DCFS. Notice that the percent reduction can be zero when DCFS selects the base case and the base case results in the minimum total delay for that network. That is, when the base-case format is the optimal one, DCFS selects it and no additional improvement can be gained. Each bar in the figure represents a base case (compression format: PACK, JAR, or TGZ) for each network bandwidth and indicates the performance improvement a user would experience if DCFS were used instead of the base case. For example, if a user that consistently invokes a jar file for execution of the Jasmine program instead uses DCFS, he or she will experience an 80% reduction in total delay on a modem link, 45% using an Internet connection, and 50% on a local area network. The overall benefit from DCFS does not simply result from using a different compression utility, it results from selecting the **best** compression utility given the underlying network performance. The benefits achieved using DCFS are quite substantial for every benchmark and network performance rate.

We next present the total delay in seconds in Figure 3. A logarithmic scale is shown (for the y-axis) since the substantial total delay for the slow networks obscures fast-link results when a normal scale is used. For each network, the number of seconds required for request, transfer and decompression is shown for each compression technique (PACK, JAR, TGZ). The fourth (far right, striped) bar of each set shows the DCFS total delay. The DCFS bar is always equivalent to the minimum of the prior three bars since it is the "best" performing format. The bar that is equal to the DCFS bar in each graph is the zero-valued bar in the the respective graph in Figure 2. Averaged across all networks, DCFS reduces total delay from 0.3 to 1.6 seconds over PACK, from 2.1 to 16.1 seconds over JAR, and from 1.4 to 9.7 seconds over TGZ.

A summary of the results is shown in Figure 4. The format of this graph is the same as Figure 2

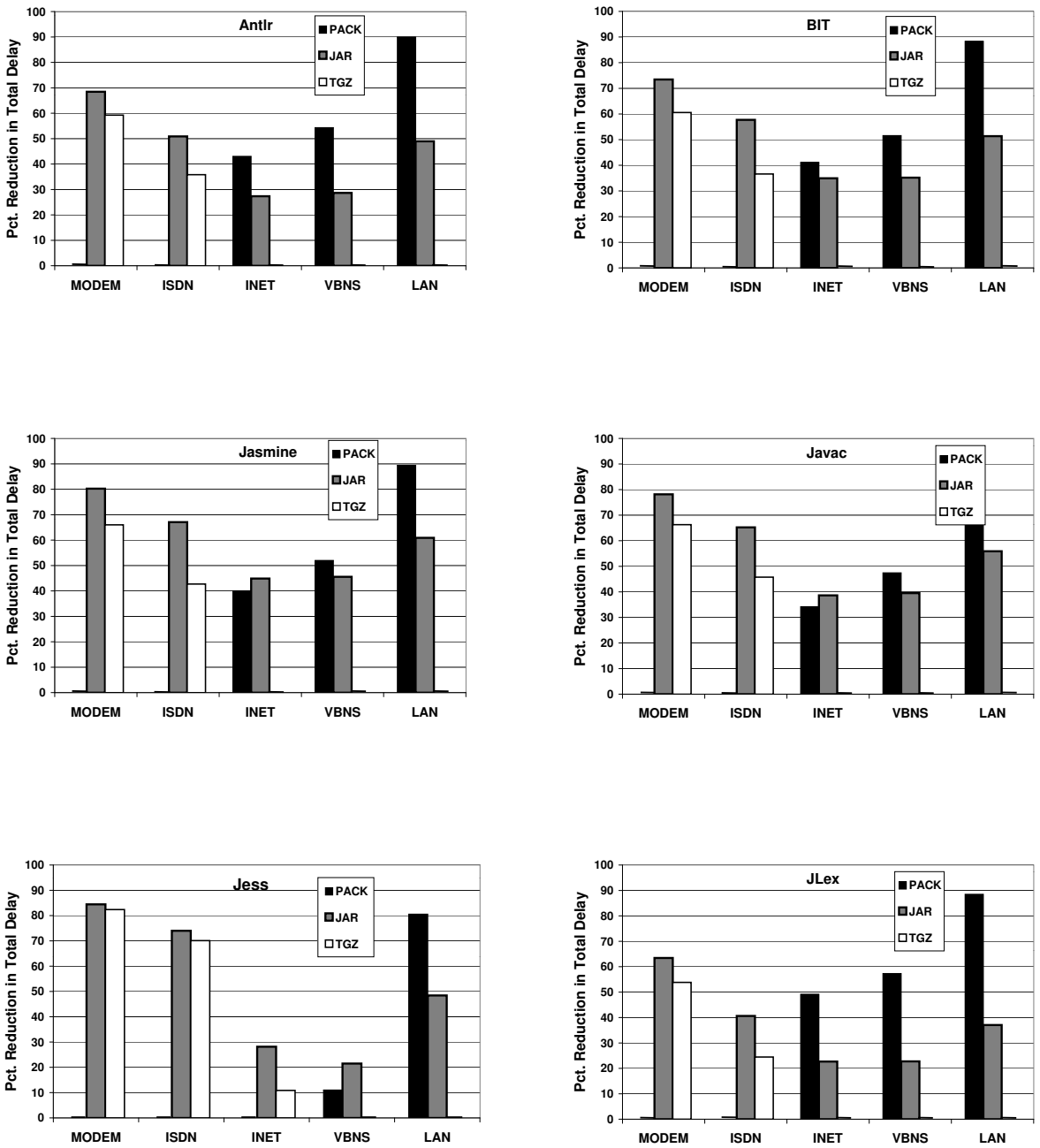

Figure 2: Percent reduction in total delay due to DCFS. Each graph is data for a different benchmark. Each bar is a different compression format (base case) and represents the percent reduction in total delay (y-axis) when DCFS is used instead of that format alone. The choice made by DCFS for a given network (x-axis) is represented as zero-valued (missing) bars, i.e., when the format chosen is the base case, DCFS enables no further reduction since it has selected the optimal format given the ones available. In every case, DCFS correctly determines and uses the format that requires the minimum total delay and significantly reduces it.

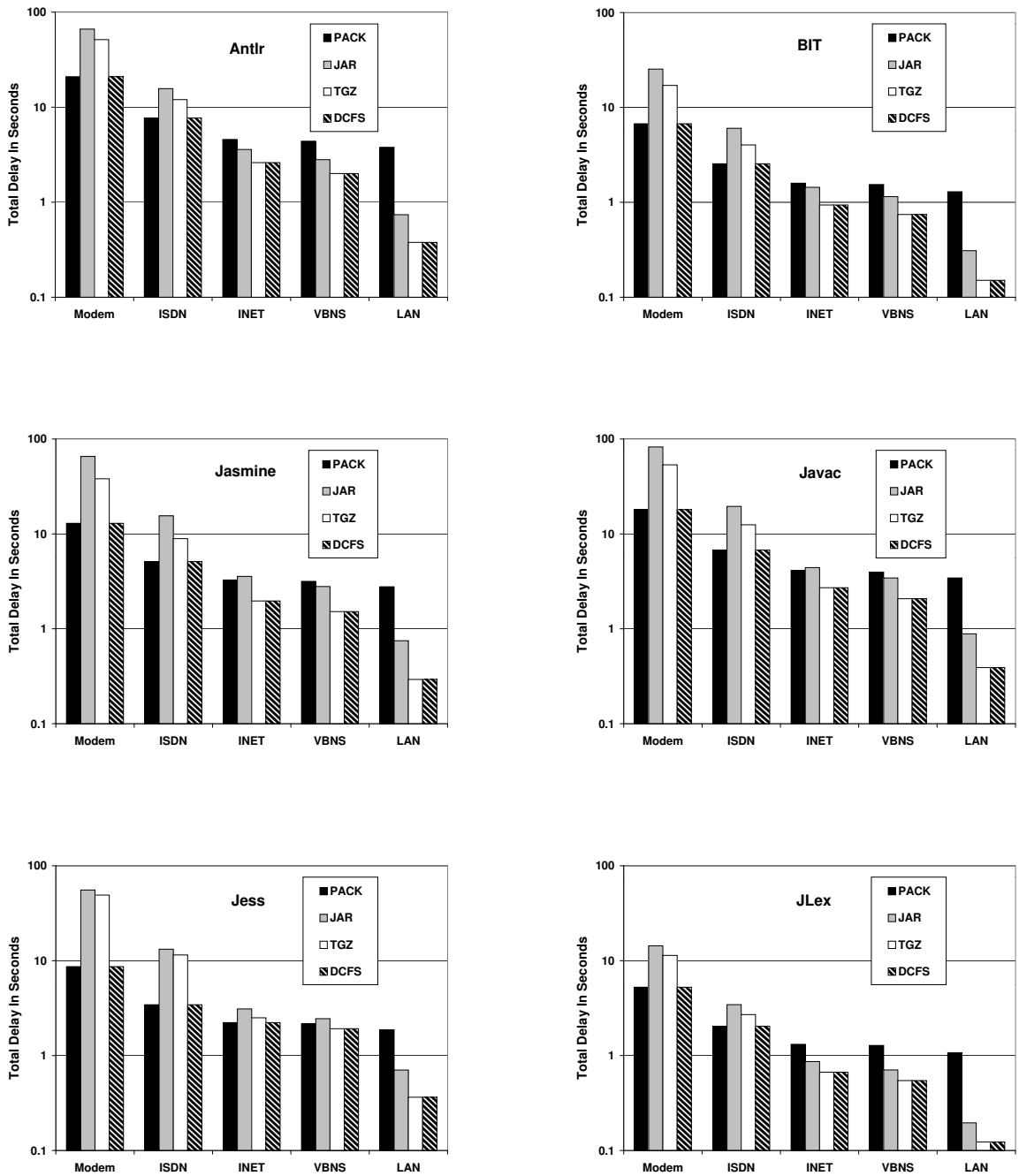

Figure 3: Total Delay in seconds (on a logarithmic scale) without and with (striped bar) DCFS. This set of graphs shows (for each benchmark) the total number seconds required for request, transfer, and decompression using each compression technique (bar), PACK, JAR, TGZ. The fourth (striped) bar of each set is the total delay when using DCFS. DCFS selects the format that results in the minimum total delay.

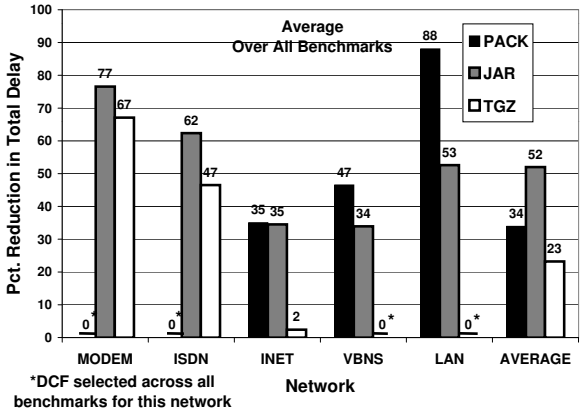

Figure 4: Average reduction in transfer delay over all benchmarks enabled by DCFS. Each bar shows the average percent reduction in total delay across all benchmarks for each of the three compression formats. Bars with zero values indicate that the base case was the format selected by DCFS, i.e., the base case was optimal and no additional benefits are possible. In every case, DCFS correctly determines and uses the format that requires the minimum total delay and significantly reduces it. The average reduction in total delay over all network types is shown by the rightmost three bars.

(percent reduction) with the average reduction across all benchmarks given instead of that for a specific benchmark. If PACK is used for non-MODEM bandwidths, e.g., LAN, then DCFS reduces total delay over PACK by almost 90% (2 seconds) on average across all benchmarks. For the LAN bandwidth, the optimal format is TGZ. However for MODEM and INET bandwidths, DCFS provides 67% and 46% average reduction (24 and 4 seconds), respectively, over TGZ. On average across all networks, 33% (1 second), 52% (10 seconds), and 23% (6 seconds) of the delay can be eliminated over PACK, JAR, and TGZ, respectively, if selection of the compression format is made dynamically.

Interestingly, JAR is never selected by DCFS (there are no zero-valued JAR bars in Figure 2) using the bandwidths examined. This implies that using DCFS with only two compression formats can improve (substantially, in many cases) the performance of programs compressed using jar, given any network technology. This also implies that only two formats need to be stored at the server for application download, given current compression technology, to achieve the substantial reductions in total delay presented here. As compression utilities change, however, providing additional DCFS choices enables additional opportunity for improved performance. On average, across all benchmarks and bandwidths, DCFS reduces total delay imposed by jar compression, the most commonly used Java application compression technique, by more than half.

4 DCFS Use and Implementation

In the previous section, we articulated the DCFS design and reported results to indicate the potential of dynamic selection of compression formats to improve mobile program performance. Our results use average performance values from real network traces. In this section, we discuss practical implementations of DCFS and the implications of incorporating predictions of future network bandwidth.

The results in the previous section show that DCFS is able to select the appropriate compression format that results in the minimum delay. For example, for modem links, DCFS commonly chooses PACK; similarly for LAN, DCFS chooses TGZ. However, the affect of variance is not represented by these results since we use a single network bandwidth value (the trace average). Such network

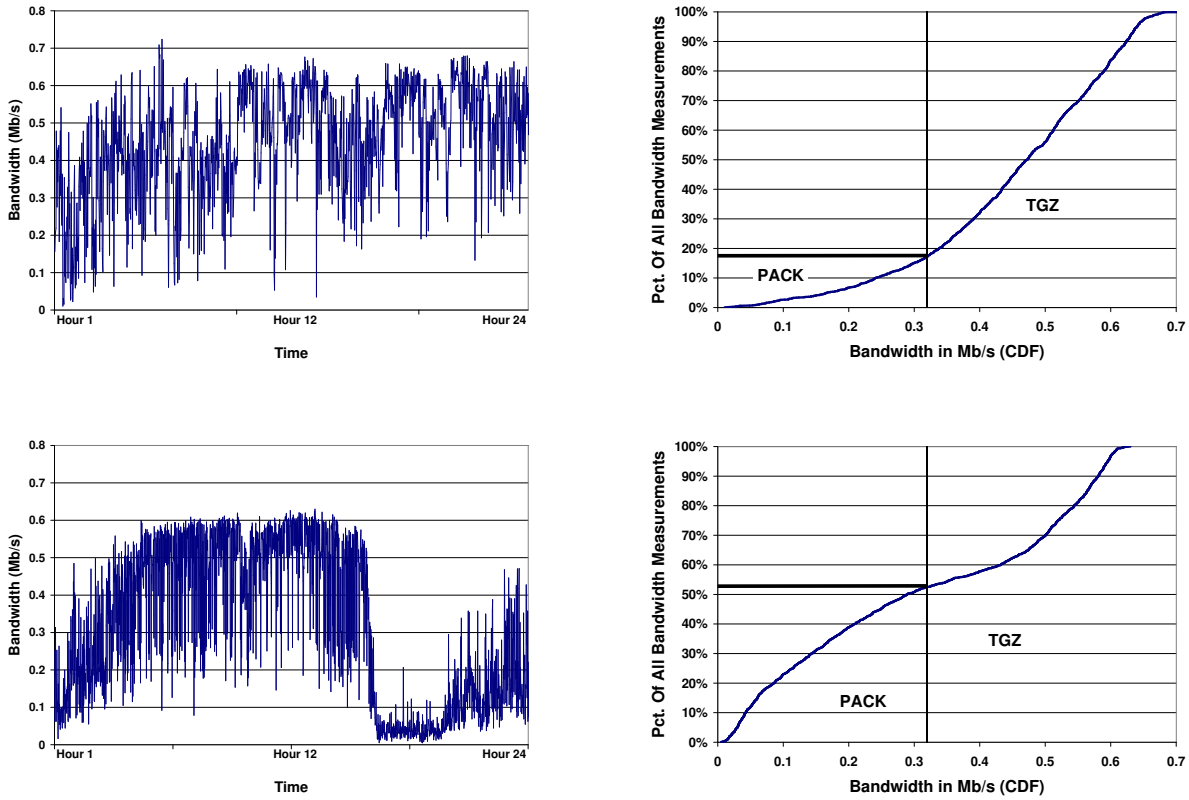

Figure 5: Raw data (left graphs) and cumulative distribution functions (CDF) (right graphs) for two data sets. The top pair is data from the INET trace used throughout this study. The bottom pair in also Internet data, however, between two different hosts. In the left graphs, the y-axis is bandwidth, and the x-axis is time. Both traces are of a single 24-hour period starting approximately at 8PM at night. The data indicates that these connections are highly variable and hence different DCF choices can be made for a single link. The right graphs indicate (given the average bandwidth value, 0.32Mb/s indicated by the vertical line, at which DCFS chose a different format over all benchmarks studied) that in the top pair, PACK is chosen less than 19% of the time. In the bottom pair, the number of times PACK is chosen by DCFS is about the same number as that for TGZ.

variance can cause DCFS to change the selection for a single link. For example, Figure 5 shows the bandwidth for two Internet connections. The first row of graphs is the data trace from which the INET bandwidth average was obtained. The second row provides data for a different Internet connection between the University of Tennessee and the University of California, San Diego. The left graph of each pair is the raw bandwidth measurement taken; the right is the cumulative distribution function (CDF) over all bandwidth values. Measurements were taken at just under one minute intervals over a 24 hour period that began at approximately 8PM.

In the right, CDF graphs, we have incorporated a vertical and horizontal line. The vertical line indicates the average bandwidth value 0.32 Mb/s at which the DCFS selection changes from PACK to TGZ over all of the benchmarks studied. For less than 42% of the values, PACK is chosen by DCFS for the link represented by the top pair of graphs; the remainder of time TGZ is chosen. For the link represented by the bottom pair, over 50% of the values on average, causes DCFS to select PACK. These results show that it is unclear as to which compression technique to use for this network. Hence, dynamic selection should be used to achieve the least total delay.

4.1 Prediction of Network Characteristics

In the real-world implementation of DCFS, the future performance of the network (at the time the compressed file transfers) is unknown. We must predict this value to determine which compression technique results in the least total delay. The JavaNws provides network performance prediction to DCFS. The JavaNws treats measurement values from network probes as time series. It applies a set of very fast[1], adaptive, statistical forecasting techniques to these time series to produce accurate, short-term predictions of available network performance [9]. For the use of DCFS to be practical, we must show that the performance improvements it is able to achieve are not outweighed by the use of prediction; which is possible if the predictions made are inaccurate.

To determine the accuracy of a predicted value, it is common to examine the error value: the difference between the predicted value and the actual value when it occurs. The average JavaNws prediction error from the top Internet connection data in Figure 5 is 7.5KB/s (for an average bandwidth value of 0.53 (INET) this is 111ms). Smaller average error improves the potential for correct selection (and improved performance) by the DCFS. Using 100 randomly selected bandwidth values from both sets of INET trace data presented in the figure, we achieve total delay reduction from DCFS using JavaNws prediction within 2% of that from DCFS using the actual trace values (the results presented in Figure 2).

5 Conclusion

Despite benefits provided by compression, transfer time continues to impede the performance of mobile programs. With this work we exploit the trade-off that must be made by compression techniques (compression ratio for decompression time). Since no single transfer format is best (in terms of transfer and decompression time) for every level of network performance (and such performance is highly variable for a single link), we introduce *Dynamic Compression Format Selection* (DCFS). By dynamically selecting the compression technique based on the underlying, available resource performance, we ensure that for any network bandwidth, the format resulting in the least total delay is used. We show that DCFS reduces total delay substantially: 52% (10 seconds) over jar, the most commonly used compression format for mobile Java programs. DCFS reduces total delay for fast links (T1) 90% (2 seconds) over PACK compression on average for the benchmarks studied. For slow links (modem) it can reduce total delay on average 59% (25 seconds) over TGZ (tar and gzip) compression.

To reduce total delay, the DCFS implementation requires that applications be stored in various formats at the server and that the server compute minimum load delay to select which format to use when transfer the file(s) back to the client. Currently, servers supply users with mirror sites to improve download times. Companies that manage servers are motivated by competition and continuously improve sites to ensure the satisfaction of customers/users. Our results motivate the need for compression format selection, and the storage of applications in additional formats on a server is a reasonable trade-off with the users reduction in load delay achieved from using DCFS.

References

[1] M. Crovella and A. Bestavros. Self-similarity in world wide web traffic: Evidence and possible causes. In *Proceedings of the 1996 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, 1996.

[2] W. Doherty and R. Kelisky. Managing VM/CMS systems for user effectiveness. *IBM Systems Journal*, pages 143–163, 1979.

[1] The JavaNws forecasting techniques require approximately 0.25ms to produce a single prediction, on the processor used in this study.

[3] C. Krintz, B. Calder, and U. Hölzle. Reducing transfer delay using Java class file splitting and prefetching. In *ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*, November 1999.

[4] C. Krintz and R. Wolski. JavaNws: The network weather service for the desktop. In *ACM JavaGrande 2000*, June 2000. to appear.

[5] Pkware inc. `http://www.pkware.com/`. PKZip format discription: `ftp://ftp.pkware.com/appnote.zip`.

[6] T. Proebsting, G. Townsend, P. Bridges, J. Hartman, T. Newsham, and S. Watterson. Toba: Java for applications a way ahead of time (wat) compiler. In *Proceedings of the Third Conference on Object–Oriented Technologies and Systems*, 1997.

[7] W. Pugh. Compressing Java class files. In *Proceedings of the SIGPLAN'99 Conference on Programming Language Design and Implementation*, May 1999.

[8] Spec jvm98 benchmarks.

[9] R. Wolski. Dynamically forecasting network performance using the network weather service. *Cluster Computing*, 1998. also available from http://www.cs.ucsd.edu/users/rich/publications.html.

[10] R. Wolski, N. Spring, and J. Hayes. The network weather service: A distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems*, 1999. available from `http://www.cs.utk.edu/~rich/publications/nws-arch.ps.gz`.