Compression Performance of the Xremote Protocol

John Danskin * Pat Hanrahan

Department of Computer Science, Princeton University

Abstract

The Xremote protocol is a compressed transformation of the X Window System protocol, designed to efficiently implement X connections across relatively slow serial lines. Using an Xremote simulator and 11 traces of X sessions, we found that Xremote's overall compression ratio is 2.4:1. This figure varies widely depending on the trace. A study of compression ratio as a function of message type shows text based messages commonly achieving 3:1 compression, while geometric messages usually achieve only 1.6:1 compression. By examining bandwidth requirements and compression performance as a function of time, we see that Xremote performs adequately for some applications which are text based or which use small geometric datasets, except at application startup where more bandwidth is required. Further work is required to adequately support the initialization stage of X applications and medium to large geometric databases.

1 Introduction

Window systems provide an abstraction for the display screen, and arbitrate between applications competing for display space. The X Window System version 11 [7] (usually known as X11 or just X) also provides *network transparency*. Network transparency decouples the application's user interface from its computer without application intervention. Usually, remote X connections are made over the ethernet; however, modem performance has improved to the point where it is possible to use X across normal telephone connections.

Whenever network bandwidth is a limiting factor, there is an opportunity for compression. This is the case for X connections using normal telephone service, ISDN, and even sometimes ethernet. Xremote [4] is a transformation of the X protocol designed (by NCD) as a way to efficiently implement X connections over serial lines. There is a follow-on to Xremote called LBX [6], but it is still in the design stage. In this paper, we will study the performance of the Xremote protocol with the aim of finding opportunities for further compression.

X has a large user community, which makes X worthy of study in its own right, but there are other reasons to study X. As the communications infrastructure in the US improves, we expect to see networked graphics applications in the home. These applications will vary from networked multiplayer games, to online catalogs, stock quotes, and trading services. An X-like

*Funded by an AT&T PhD fellowship and a generous grant from Apple Computer

protocol could support all of these applications. Research into efficiently sending X commands across low bandwidth networks will directly support these imminent applications.

The problem of compressing a graphics protocol such as X differs from classical compression problems such as image compression or text compression in several ways:

1. X supports many different types of data including images, geometric objects, and text.

2. X supports interactive applications. Humans can detect, and are affected by delays exceeding 100ms [3]. Delays can arise either from the compression architecture, or from insufficient network bandwidth.

3. X connections support traffic in two directions, it could be that the two streams are best served by different compression strategies, or that there is a way to use messages traveling in one direction (stimulus) to make predictions about messages traveling in the other direction (response).

4. X applications generate a lot of messages when they start up. This traffic is usually a small fraction of the total traffic generated by an application over the minutes or hours the application is in use, but it can make a big impression on the user if it contributes to sluggish application startup. Other messages, which may seem more important because of total bulk, may not need compression because the interactions which they support already appear instantaneous to the user.

One way to address the client-server bandwidth issue is to allow applications to dynamically extend the protocol by downloading macros, display-lists, or even procedures into the server. For example, the NeWS window system [1] allows applications to download postscript programs into the server. We are interested in protocols, like the X protocol, where the division of labor between client and server is fixed.

This study is based on our simulation of the Xremote protocol over 11 traces of X sessions. A summary of the traces is presented in Table 1. Motivation for the selection of these traces, the trace gathering methodology, and an analysis of the contents of the traces can be found in our previous work on profiling the X protocol [5].

According to our traces, Xremote's overall compression ratio is 2.4:1. This figure varies widely depending on the trace. A study of compression ratio as a function of message type shows text based messages commonly achieving 3:1 compression, while geometric messages usually achieve only 1.6:1 compression. By examining bandwidth requirements and compression performance as a function of time, we see that Xremote performs adequately for some applications which are text based or which use small geometric datasets, except at application startup, where more bandwidth is required. Further work is required to adequately support the initialization stage of X applications and medium to large geometric databases.

2 X and Xremote

X clients (applications) communicate with the X server using a single reliable byte-stream connection such as a Berkeley socket using the TCP/IP network protocol. When a client wishes to update the display or query the X server, it sends a structured message asynchronously, that is, the client is not suspended until the requested action completes. There is a library interface for X which hides some of the details of the network connection, and also buffers client output until the client waits for input. This means that groups of messages frequently appear together on the network. The X server sends replies to queries, errors, and events to the client. Events usually denote user input such as key presses or mouse movements. Examples of common X messages can be found in Figure 4.

The Xremote protocol includes a transport mechanism for compressing and multiplexing multiple X connections onto a single byte-stream, and file access mechanisms for starting up a diskless X terminal. Both of these protocols are layered on a reliable datagram protocol which is also part of Xremote. We will only be discussing the compression and multiplexing mechanisms, and we will only discuss the multiplexing mechanism briefly.

On Unix machines, X clients use an environment variable to determine the network address of the X server to connect to. In order to use Xremote, this environment variable should be set to the address of an Xremote pseudo server instead. Clients connect to the pseudo server as if it were the real X server. Figure 1 shows how clients connect to the pseudo server, the pseudo server connects to the pseudo client, and the pseudo client connects to the X server. The pseudo server multiplexes messages on multiple connections by using three additional messages:

1. NEWSTREAM
2. CLOSESTREAM
3. CHANGESTREAM

These three extra messages are sufficient to manage the multiplexing and demultiplexing tasks.

Figure 2 shows an overview of Xremote's compression component. The delta encoder remembers the last 16 messages of length less than or equal to 64 bytes. Most X messages are in this category [5]. Incoming messages are compared to cached messages of the same length. If there are fewer than 8 differing bytes, then a difference message is output instead of the original message, as long as the difference message is not longer than the original message.

The LZW coder [9] is modified so that pointers have varying size as in the LZC scheme [8]. The protocol allows for varying dictionary and pointer sizes, but only 4096 entry dictionaries and pointers up to 12 bits are supported. The dictionary is cleared and rebuilt from scratch whenever it fills, allowing the coder to adapt to changing data.

The figure is slightly simplified in that whenever the LZW coder outputs a chunk of more than 120 bytes, the uncompressed chunk is sent instead and both the coder and the decoder's dictionaries are cleared.

3 Xremote Simulator

Our simulator duplicates the Xremote protocol down to but not including Xremote's physical transport layer. The simulator can run on traces or on a live X session. If coalescing and chunking are enabled (stages 3 and 4 in Figure 2), then message boundaries are lost after LZW compression; we cannot see how effective LZW compression is for each message type.

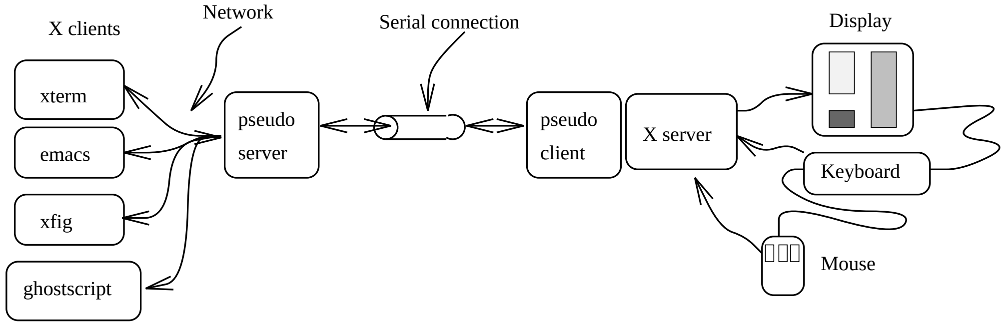

Figure 1: Xremote architecture: X clients connect to pseudo server which multiplexes connections and forwards coded messages to the pseudo client. The pseudo client decodes and demultiplexes messages before forwarding to the X server. Messages from the X server to the X client move in the opposite direction in exactly the same manner. The pseudo server appears to be the X server to the X clients, and the pseudo client appears to be the X clients to the X server.

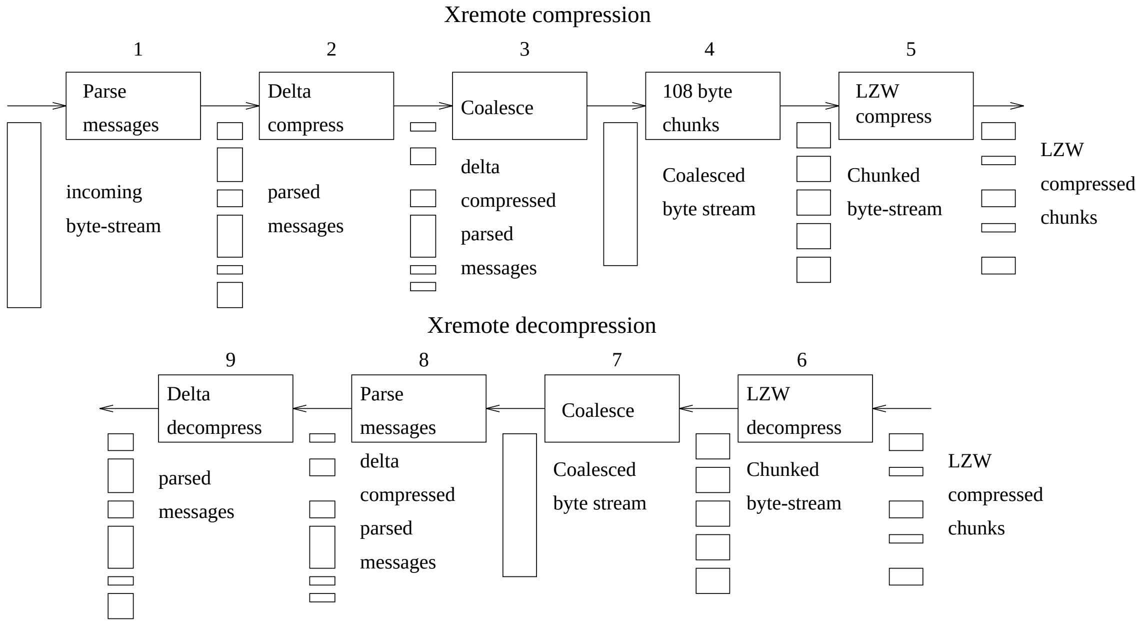

Figure 2: Xremote compression architecture: bytes available from the network are parsed into X messages (1) and delta compressed (2). Output from the delta compressor is gathered together (coalescing) (3), and broken into chunks no larger than 108 bytes (chunking) (4). These chunks are run though an LZW compressor (5) and sent out across a serial line. Compressed chunks incoming from the serial line are run through the LZW decompressor (6), gathered together into a byte stream (7), parsed into X messages (8), run through the delta decompressor (9), and sent out to the network.
This figure is simplified by the omission of stages irrelevant to compression.

Trace	color?	Size	Duration	Description
idraw	bw	4.5 Mb	11 min	A 2D drawing program, making a very large model by duplicating a small one repeatedly.
xfig	bw	5.6 Mb	17 min	A 2D drawing program, making a very large model by duplicating a small one repeatedly.
ghostscript	bw	30 Kb	10 sec	A postscript previewer, previewing a single page
x11perf	bw	1.9 Mb	33 sec	An X benchmark program configured to test point drawing performance.
xbattle	bw	7 Mb	14 min	A multiplayer interactive wargame.
xtetris	bw	350 Kb	11 min	The classic game
grad1	bw	13 Mb	6 hrs	A graduate student, preparing a paper and previewing graphs.
grad2	color	14 Mb	45 hrs	A fast typing graduate student.
grad3	bw	2.4 Mb	54 min	A graduate student doing software development.
grad4	color	21 Mb	11 hrs	A graduate student, making significant use of a dvi file previewer
grad5	bw	10 Mb	110 hrs	A graduate student who reads mail and writes papers.

Table 1: We collected traces of complete X sessions by interposing a logging process *XTee* on the network between client and server. *XTee* and the traces are described fully in [5].

Our simulator optionally disables these stages so that we can measure LZW compression as a function of message type. Later, we will show that exercising this option has little effect on overall compression.

4 Results

Chunking and Coalescing We wanted to study compression as a function of message type, but with Xremote's message coalescing enabled, this is impossible. We were curious to see how much effect chunking and coalescing had on compression. We expected that coalescing alone might help, by providing longer input strings for the LZW coder. We expected that chunking alone would hurt, by shortening the coder's input strings. Finally, we expected that chunking and coalescing together would hurt because the LZW coder would no longer start off aligned with the beginning of each message. Table 2 shows the result of our experiment with chunking and coalescing. Chunking and coalescing almost never have a significant impact on compression, probably because most messages, and even most groups of messages eligible for coalescing are smaller than the chunking theshold of 108 bytes [5].

Overall Compression Table 2 also has data on overall compression efficiency. On average, client data compressed to 44% of its original size while server data compressed to about 28% of its original size. On average, server data accounted for 19% of each trace with client data accounting for the remaining 81% of the transferred bytes. In an average trace, compression would have reduced the amount of transferred data to 41% of its original size, giving a compression ratio of 2.4:1.

Compression by message type We measured compression as a function of message type for Xremote's two compression stages, separately and together.

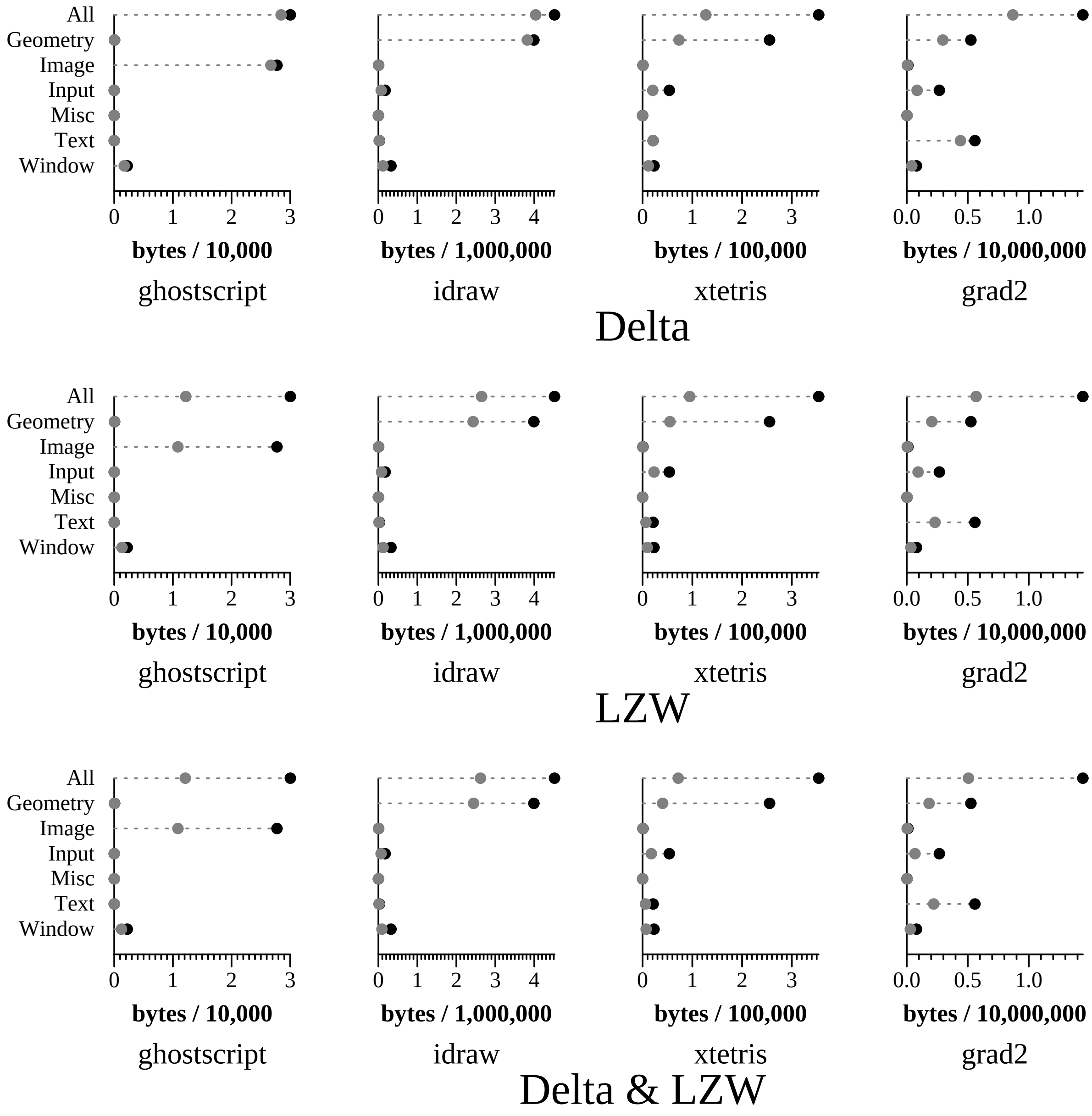

Figure 3: Magnitudes correspond to total bytes in each category. Black dots show the size of the input data while grey dots show the size of the compressed data. For details of the categories see [5].
The top graph shows how well the delta compactor works by itself, the middle graph shows how well the LZW encoder works by itself, and the bottom graph shows how well they work together.
Note that each trace has the same horizontal scale in all three rows, but that horizontal scales vary between traces, since each graph is scaled to the size of its trace.

		output bytes as a % of input bytes								
		Client to Server				Server to Client				
Coalescing		off	off	on	on	off	off	on	on	
Chunking		off	on	off	on	off	on	off	on	
Trace	Client bytes									Server bytes
idraw	4315792	59%	59%	59%	59%	38%	38%	38%	48%	203196
xfig	4895392	73%	74%	73%	75%	31%	31%	31%	39%	795568
ghostscript	28488	39%	39%	39%	41%	58%	58%	58%	58%	1548
x11perf	1960896	51%	51%	51%	52%	11%	11%	11%	12%	27960
xbattle	6541936	34%	34%	34%	35%	8%	8%	8%	8%	1450416
xtetris	273012	17%	17%	17%	18%	32%	32%	32%	32%	80976
grad1	10442396	50%	50%	50%	50%	27%	27%	27%	26%	2676090
grad2	11192330	38%	38%	38%	38%	27%	27%	27%	28%	3254696
grad3	1889556	39%	39%	39%	38%	29%	29%	29%	30%	555216
grad4	18094626	43%	43%	43%	44%	31%	31%	31%	32%	2997507
grad5	8124392	40%	40%	40%	40%	26%	26%	26%	27%	2207958
Average	6159892	43%	44%	43%	44%	28%	28%	28%	30%	1295557
Std Dev	1660298	4.4	4.4	4.4	4.4	4.0	4.0	4.0	4.3	384293

Table 2: This table illustrates both the range of overall compression one is likely to get from Xremote, and that the difference in compression performance resulting from coalescing messages after delta compaction, or from breaking messages into maximum sized chunks, or both is not significant. (Averages are unweighted.)

Because the X protocol defines so many different types of messages (156 by our count), we divided the different types of messages into 6 major categories [5]:

1. Geometry: line drawing, polygons, etc.
2. Image: image and cursor manipulation.
3. Misc: access control, the bell, errors, X extensions, and IPC facilities such as cut and paste.
4. Text: character drawing and font manipulation.
5. Window: colormaps, graphics state, window manipulation, and X's database facility.

Figure 3 reports on the compression achieved by each type of message for each arrangement of compression stages.

Delta compression The Delta compression stage should work well on small messages which are repeated almost verbatim. Input messages (keystrokes, mouse movements) fall into this category, as does xtetris' geometry (a few rectangles moving around). The images in the ghostscript trace, the complicated geometry in the idraw trace, and the text in the grad2 trace were not compressed by the delta compressor.

The text in grad2 trace would probably have been more amenable to compression if the user had used a terminal emulator other than *xwsh*. *Xwsh* redraws the whole line at each keypress instead of just drawing the new character.

The reader will notice that text and geometry are roughly of equal importance in the grad2 trace, even though the user spent most of his time typing. This is because the terminal emulator uses rectangles to clear the space behind each character and to implement a cursor.

LZW compression The LZW compression stage achieves good compression on every important datatype in this trace. The best compression (better than 3:1) was on the repetitive xtetris trace. The worst compression was on the idraw trace which is dominated by complicated geometry. This geometry is very regular because it was created by making many copies of a simple geometric dataset. Unfortunately, the copies are translated and the coordinates are absolute, so the LZW algorithm cannot recognize the similarities. We conjecture that LZW was able to compress this geometry somewhat because the coordinates are represented as pairs of 16 bit numbers, but the visible screen was small enough so that fewer than 10 bits were used.

We would not normally expect LZW compression to achieve the kind of compression on images that we see in the ghostscript trace. This image was a black and white page from a document, which is presumably an easier tagret for lossless compression than a landsat image, for example.

Delta and LZW compression Delta compression together with LZW compression is much better than delta compression alone, and slightly better than LZW compression alone. The improvement over LZW compression alone is especially noticeable for the insignificant looking but all important input category. We conjecture that delta compression was included in the Xremote protocol entirely for the sake of input.

Compression over time We were interested to see how Xremote responded to differing application requirements over time. Figure 4 shows the first 8.3 minutes of the 11.5 minute idraw trace. At the left, we see *idraw* starting up. There is a flurry of activity at startup time, with an almost 5 second period where the bit-rate averages 75Kbps, but the compressed bit-rate would be only about 25Kbps. If we supposed a serial line connection with a 14400 bps modem and a low overhead network protocol, Xremote would speed application startup by a factor of three over the vanilla X protocol, but startup would still take nearly twice as long as it would if there were a fast network connection. Through the next 400 seconds, the compressed bit-rate stays under the 14400 mark, while the raw bit-rate drives well past this mark. Eventually, as the number of polygons and lines in the idraw database increase geometrically, geometry dominates the network and the raw bit rate passes 100,000 bps. At this stage, a slow serial line would clearly be the limiting factor even with Xremote's compression. Looking to the graph of compression ratios as a function of message type at the far left of the figure, we see that the polylines and polygons flooding our hypothetical serial line are also the most poorly compressed of all of our messages.

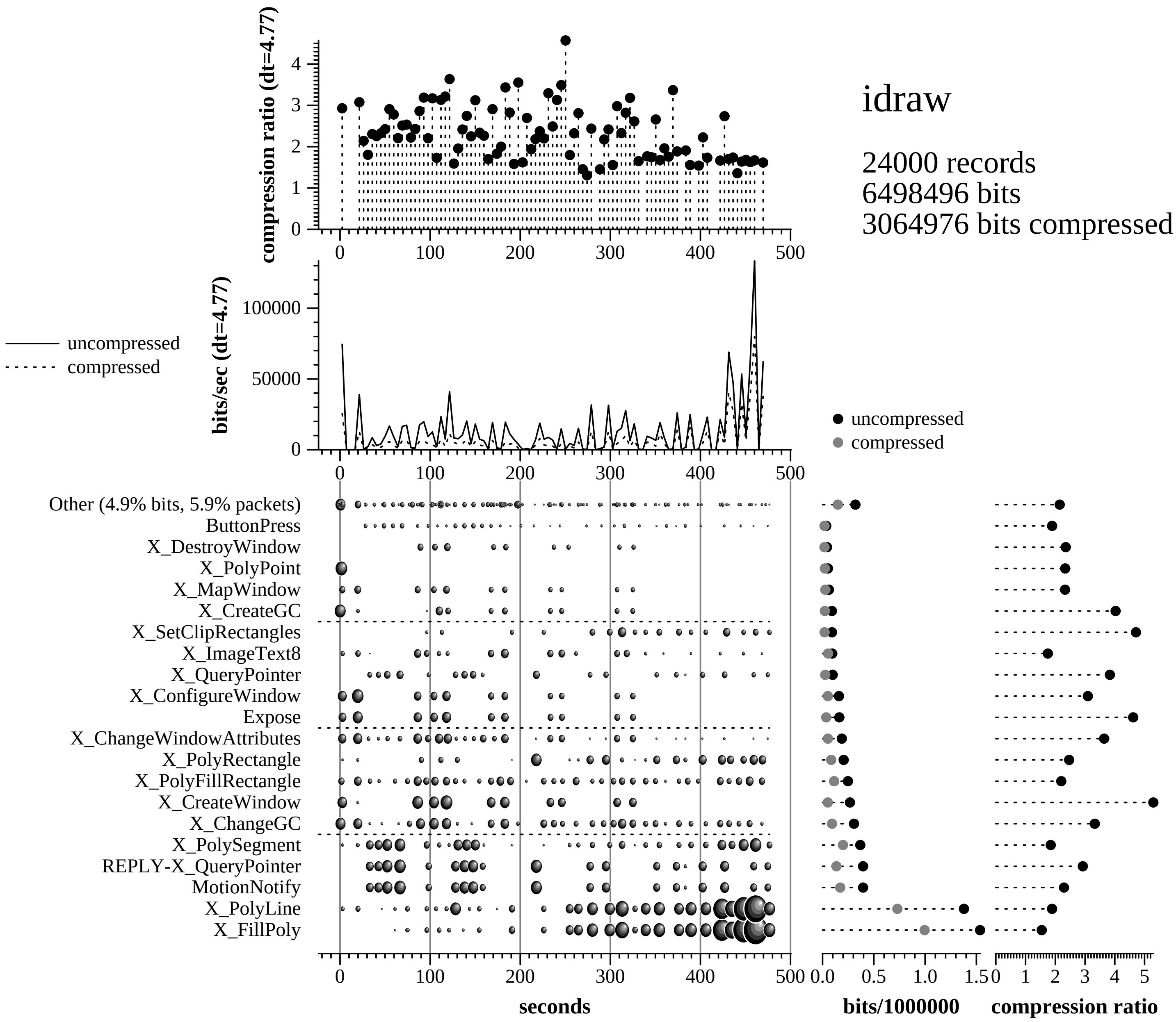

Figure 4: This graph covers the first 8.3 minutes (and 29% of the messages) from the 11.5 minute idraw trace. The X axis along the bottom of the central area is seconds since session startup. The Y axis in the main area is marked off by message type. The volume of spheres in the data area is proportional to the size of the messages which they represent (before compression). Spheres which are vertically stretched represent multiple nearly simultaneous messages, other spheres represent single messages. The dash-dot graph to the middle right shows the total number of bits for each type of packet. The dash-dot graph to the far right shows the compression ration for each type of packet. The line graph at the middle top shows the number of bits per second as a function of time. The integration interval is listed in the Y-axis legend. The dash-dot graph at the tip-top shows the compression ratio as a function of time.

5 Discussion

In our view, Xremote just scratches the surface of what is possible in graphics protocol compression. We would like to see reply caching, better image compression (both proposed in [6]), better geometric compression, and a way to take advantage of message structure.

Just as file compression makes disks bigger, network compression enhances network bandwidth. We believe that compression algorithms tuned specifically for graphics protocols will perform far better than generic network compression because they can apply image compression algorithms to images, geometry compression algorithms to geometry, and intelligent caching algorithms to queries.

Enhanced bandwidth graphics protocols are applicable whenever networks are too slow. Our experience is that networks are always too slow. Extreme cases of slow networks include phone lines (even ISDN), cellular links, proposed interactive TV links, and networks connected to supercomputers. Research in enhanced bandwidth protocols will enable new applications on all of these networks.

References

[1] Arden, Michelle J., James Gosling, and David S. H. Rosenthal, "The NeWS book : an introduction to the Networked extensible Window System," Springer Verlag, 1989.

[2] Bell, Timothy C., John G. Cleary, and Ian H. Witten, "Text Compression," Prentice Hall, Englewood Cliffs, New Jersey, 1990

[3] Card, Stuart K, Thomas P Moran, and Allen Newell, "The psychology of human-computer interaction," Hillsdale, N.J., L. Erlbaum Associates, 1983

[4] Cornelius, David, "XRemote: a serial line protocol for X" 6th Annual X Technical Conference, Boston, MA, 1992

[5] Danskin, John and Pat Hanrahan, "Profiling the X Protocol," submitted to 1994 Sigmetrics Conference on Measurement and Modeling of Computer Systems.

[6] Fulton Jim, and Chris Kent Kantarjiev, "An update on low bandwidth X (LBX)," Proceedings of the 7th Annual X Technical Conference, January 1993, O'Reilly and Associates.

[7] Gettys Jim, Phil Karlton, and Scott McGregor, "The X Window System, Version 11," *Software Practice and Experience* vol. 20, no. S2, May 1991.

[8] Thomas, S. W., J. McKie, S. Davies, K. Turkowski, J. A. Woods, and J. W. Orost. "Compress (version 4.0) program and documentation," available from joe@petsd.uucp, 1985.

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