Higher Bandwidth X
(Extended Abstract)
Submitted to ACM MULTIMEDIA '94

John Danskin *
Princeton University
Computer Science Department
Princeton NJ, 08540
jmd@cs.princeton.edu

Abstract

Network bandwidth has always been a key issue for multimedia protocols. Many potential users of networked multimedia protocols will continue to have low bandwidth network connections for some time: copper wire ISDN, infra-red, cellular modems, etc.. Compression provides potential relief for users of slow networks by increasing effective bandwidth. HBX introduces a new technique, based on arithmetic coding and statistical modeling, for compressing structured data. Applied to the X networked graphics protocol, this technique yields 4.5:1 compression across a representative set of traces, performing twice as well as the popular LZW-based Xremote compression protocol. HBX's coding techniques are generally applicable to the graphics and imaging subset of multimedia protocols. Future work will determine whether HBX's coding techniques can be applied to audio and video streams as well.

1 Introduction

Multimedia network protocols need to support many different datatypes: audio, video, images, text and geometry. Graphics network protocols such as The X Window System version 11 (X) [11, 12, 6], also support images, text and geometry, a critical subset of any multimedia system. Many existing networks are too slow to run standard interactive X sessions. These include telephone connections (normal and ISDN), wireless networks, and cellular connections. My new system, HBX, introduces a unified framework for compressing the structured data in the

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

X protocol in a content sensitive way using techniques from statistical modeling and arithmetic coding. This technique yields 4.5:1 compression across a representative suite of traces, while the popular LZW-based Xremote compression protocol [3] only achieves 2.3:1 compression on the same set of traces.

Graphics protocols such as X present a different compression challenge from audio and video streams for several reasons:

- X must be compressed losslessly. Unlike audio and video streams, *all* of the application's bits are precious.
- X streams include heterogeneous data types: images, geometry, input, text, etc. For good compression, all of these must be compressed effectively.
- X packets are small and include mixed types: Most X packets are less than 100 bytes long[4]. An effective compression protocol cannot waste bits shifting gears between compression techniques.

In the rest of the paper, Section 2 discusses the outlook for software network compression. Section 3 surveys strategies for managing bandwidth in networked graphics protocols. Section 4 is a short introduction to the X protocol. Section 5 introduces the trace suite. Section 6 covers the architecture and performance of NCD's Xremote protocol. Section 7 introduces some necessary concepts from coding theory. Section 9 covers the coding techniques used in HBX and the resulting performance. Section 10 summarizes the paper.

2 Software Network Compression

The opportunity for software network compression is proportional to the mismatch in performance between the computer and the network. When the network transmits data as fast as the computer can produce it, spending cycles on compression is counterproductive. When the computer must wait for the network, cycles spent on compression are well-spent.

Networks which are too fast for software compression today include the ethernet at 800 KBps and much faster networks such as the Atomic at 62 MBps.

Networks where compression is often appropriate include:

- ISDN: 64KBps
- Wireless (infra red): 100KBps
- Telephone with modem: 14.4 Kbps = 1.75 KBps
- Cellular modem: 4.8 Kbps = .6 KBps

There is a popular scenario today in which all networks become fast enough and compression becomes useless. This could happen in limited domains where improvements in network performance may outstrip improvements in computer performance for some time. However, as long as there are shared networks, and networks where users are charged for use, compression will increase efficiency and lower costs.

In the world of wireless networks, the outlook is not so rosy. There is a limited total amount of usable bandwidth, and the number of users is increasing. As the available bandwidth per user decreases, we will need more efficient transport mechanisms. These mechanisms will continue to include compression.

3 Strategies for Managing Bandwidth

Networked graphics has a relatively long history, and bandwidth has been a recurring issue. Two major techniques for managing bandwidth have emerged:

- Explicit support: The best example of this technique is the Networked extensible Window System [1] (NeWS) in which applications can load postscript programs into the server. Application writers can reduce their communication requirements to a minimum by breaking their applications at the appropriate point and redefining the network protocol to minimize bandwidth requirements.

A less powerful technique is exemplified by PEX, a network graphics protocol for 3D graphics, which allows applications to define structures, which are essentially geometric macros, in the server.

While these techniques allow for terrific compression, effective use can place a heavy load on the application programmer.

- Implicit support: The best example of this technique is the HDTV system where the application developers (film producers, for example) are not concerned with the bandwidth necessary for effective transmission. Where there is a low bandwidth network, such as an ISDN connection, the system provides compression.

  Another example of implicit support for managing bandwidth is the X protocol layered on a transparent compression protocol such as Xremote or HBX. X does not allow applications to extend the protocol at runtime. When there is a low bandwidth connection, an underlying system must provide compression.

  Implicit support eases application development by releasing programmers from the burden of managing bandwidth explicitly. The challenge for an implicit system like HBX is to provide the power of an explicit system like NeWS or PEX with none of the pain.

4 About X

X applications (or clients) connect to the X server using a reliable byte stream connection such as TCP/IP and Berkeley sockets, as shown in Figure 1. Typical X clients include terminal emulators, text editors, 2D drawing programs and postscript previewers. The X server selectively forwards user input to the clients, and updates the display according to client requests.

There are 172 different types of X messages, making X a fairly complex protocol. Messages from the client to the server are called *requests*. There are 120 *requests* including CreateWindow, PutImage, GetImage (generates a reply), Polyline, and ImageText8. Messages from the

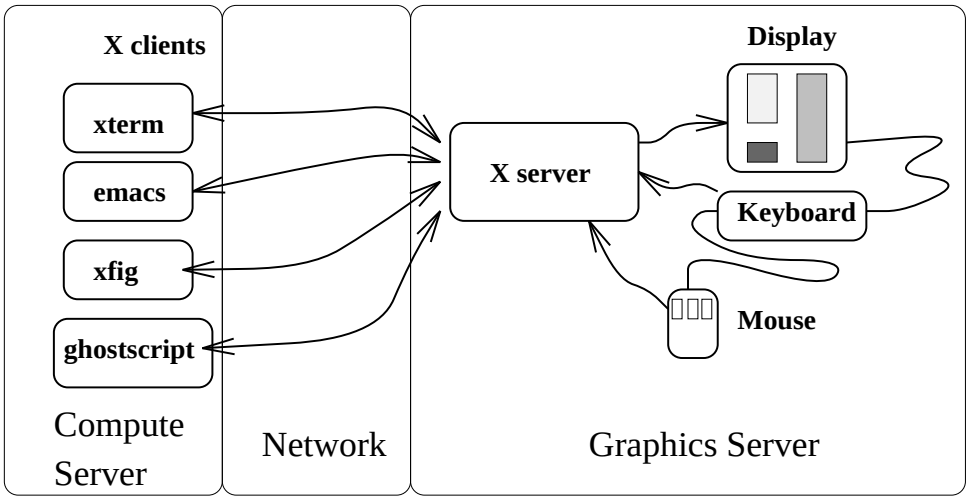

Figure 1:

server to the client fall into three classes: *errors* (17), *events* (35) and *replies* (counted with reply generating requests). Some important events are: ButtonPress, MotionNotify (mouse movement), and Expose (window needs redrawing).

5 Traces

X sessions are interactive: they involve a human. This means that X sessions are not repeatable in the way that standard computer benchmarks are repeatable. In order to assess the performance of X compression protocols in a way that permits fair comparisons, it is necessary to gather traces. A previous paper [4] discussed tracing techniques and the choice of traces, introduced a taxonomy of X messages, and characterized X applications and quantified their network bandwidth requirements.

The set of traces used to generate the results presented here is shown in Table 1. Game traces are especially interesting because games are expected to be among the first applications of interactive TV technology. Although the visual display for a game could be distributed across the network in the form of video, it would be far more economical in terms of network bandwidth

	Trace	Size	Duration	Dominant traffic	Description
Geometry	Idraw	4.5 MB	11 min	geometry	*Xfig* 2D drawing program, making a very large model by iteratively duplicating a small one.
	Xfig	5.6 MB	17 min	geometry	As above but with *xfig* program.
	Mult	3.6 MB	1 hr	mouse movement	*Xfig* program, duplicating a figure from a book.
	X11perf	1.9 Mb	33 sec	geometry	An X benchmark program configured to test point drawing performance.
Real life	Grad1	13 Mb	6 hrs	text & geometry	A graduate student, preparing a paper and previewing graphs.
	Grad3	2.4 Mb	54 min	text & geometry	A graduate student doing software development.
Games	Xbattle	7 Mb	14 min	geometry	Map oriented multi-player wargame.
	Xtetris	350 Kb	11 min	geometry	Real time 2D binpacking
Images	Ghostscript	30 Kb	10 sec	images	A postscript previewer, previewing a single page

Table 1: Traces of complete X sessions were collected by interposing a logging process *XTee* on the network between client and server. *XTee* and the traces are described in [4].

for the media server to send graphics primitives. Sending graphics commands instead of video also releases the media server from the burden of generating large numbers of video streams simultaneously.

6 Xremote

Xremote is the name of a system and a protocol developed by NCD for compressing the X protocol for transmission across a low bandwidth network. The Xremote protocol is described by Cornelius [3]. Danskin and Hanrahan implemented an Xremote simulator from NCD's spec and measured its compression performance across a suite of representative traces [5].

Neither clients nor server need be aware of Xremote. As shown in Figure 2, clients connect to Xremote's pseudo server as if it were the X server, and Xremote's pseudo client connects to the X server as if it were the clients.

The compression portion of the Xremote protocol is comprised of a delta compressor composed with an LZW compressor. Xremote's delta compressor remembers the previous 16 mes-

sages of 64 bytes or less. When a new message arrives, the delta compressor generates difference packets for all of the retained packets of the same size, sending the smallest of the new packet and the difference packets. Xremote's LZW compressor is a bytestream compressor similar to that used in the UNIX compress utility.

Figure 3 shows how the components of Xremote performed separately and together. Across all of the traces, the lack of synergy between the delta compressor and the LZW compressor is disappointing. Although the delta compressor does well on some of the traces, it never improves overall compression by more than about 7% of the input size. Xremote's performance on geometry and images is also disappointing. On the Idraw and Xfig traces, which are both geometry bound, compression is well under 2:1. The X11perf benchmark is extremely repetitive, yet Xremote achieves only 2:1 compression. Fax protocols regularly achieve 10:1 compression on documents, yet the ghostscript trace sees less than 3:1 compression. Xremote performs poorly on geometry and images: important components of a graphics protocol.

7 Compression: a technical aside

HBX uses arithmetic coding and statistical modeling to achieve its high compression rates. The beauty of this technique is the separation of modeling and coding. In this method, the sender and receiver each have a predictive model which generates a table of probabilities for the next input symbol. An arithmetic coder on the sender side accepts the table and the symbol s_i which actually occurred in the input, and communicates the symbol to the decoder in just $-\log_2(p_i)$ bits, where p_i was the estimated probability of symbol s_i appearing at this position in the input. The arithmetic decoder in the receiver decodes the coded bits using its identical table of probabilities as an inverse map. Bits are shared between adjacent messages, which means that individual messages can be expressed in fractional bits. For example, when $p_i = .5$ then 1 bit is necessary as expected, but when $p_i = .75$ then only $\approx .41$ bits are necessary to update the receiver.

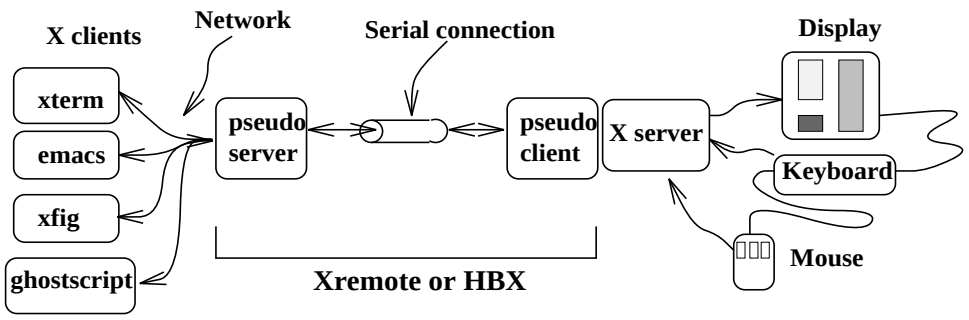

Figure 2: This is the architecture of both Xremote and HBX. Clients think the pseudo server is the X server. The X server thinks the pseudo client is the clients. Traffic between the pseudo server and the pseudo client is multiplexed and compressed.

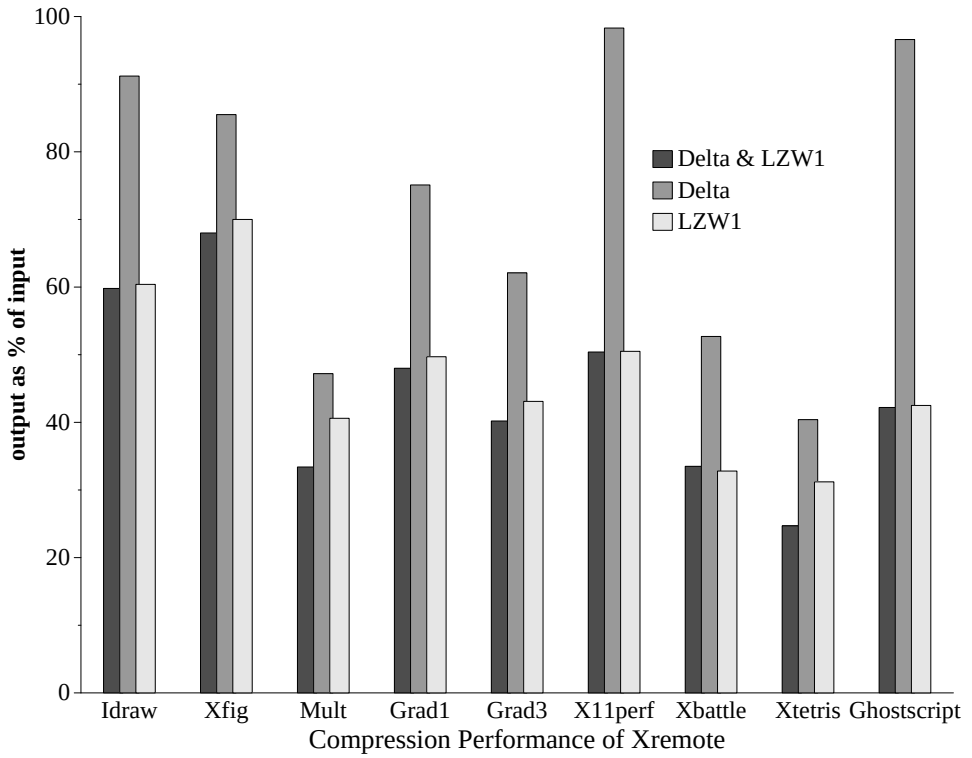

Figure 3: This graph shows output size as a function of input size for each of the traces when compressed by Xremote's two compression stages, separately and together. Note how little improvement the complete Xremote protocol (Delta&LZW1) is over the LZW1 protocol by itself. Also note Xremote's poor compression of the geometry in the Idraw, Xfig and X11perf traces.

To the extent that the model's probability estimates are accurate and fully express all available knowledge, this arithmetic coding procedure is optimal. In practice, unless the source is controlled by radioactive decay or some other truly random source, we accept that the predictive models which we implement are approximations to what is known about the source. Generating powerful predictive models is still hard.

The lower bound on message size ($-\sum_{i=1}^{n} \log_2(p_i)$) is due to Shannon [13], while an efficient technique for achieving this lower bound using arithmetic coding was discovered by 3 independent groups around 1977 [10, 7, 9]. For background on the history, theory, and implementation of arithmetic coding see Bell, Cleary and Witten[2].

As an introduction to the predictive models used by HBX, here are some examples from text compression. These models are adaptive, which means that probability estimates are revised after each input character.:

Model	Example	Size
0 order context	$p(\text{“e”}) \gg p(\text{“j”})$	256 frequencies
1st order context	$p(\text{“u”})$ after “q”$> .99$	256 0 order contexts
2nd order context	$p(\text{“ ”})$ after “xy” $> .5$	256 1st order contexts

This type of model compresses English text most effectively at 3rd or 4th order [2]. A complete 4th order model would incorporate 2^{40} frequencies, so successful implementations make provisions for blending estimates from lower order models with estimates from higher order models when higher order statistics are incomplete. HBX uses a technique called lazy exclusions [2] for blending these probabilities.

8 HBX

HBX's connection level architecture was shown in Figure 2.

Since the values or symbols appearing in each field in each X message type may have different distributions, HBX associates a unique predictive model with each field in each X message type. Since there are many of these fields, the predictive models are instantiated lazily. So that the

group of instantiated predictive models does not grow in proportion to the size of the input, the frequencies of the symbols in the models are periodically halved (exponential decay). Symbols whose frequencies reach zero are deleted, and empty models are deleted. There is an escape mechanism for gracefully dealing with symbols whose estimated probability is zero.

The key to optimizing the compression of structured data is to find a good predictor for each field. X is a graphics protocol, supporting text, geometry and images. Other types of data are also important, such as window-ids, timestamps, and sequence numbers. Here are the predictive models used by HBX for each of these datatypes:

Text model: 3rd order context. (This is Moffatt's PPMC' [8, 2])

Geometric model: 3rd order context on (ΔX, ΔY) coordinates. Since shape is important, and it is common to move objects around on the screen without deforming them, we transform absolute coordinates (X, Y) into relative coordinates (ΔX, ΔY), and then predict them using a separate 3rd order model for each coordinate.

Image model: I have not yet implemented this model for image compression, but for bitmap images I plan to use separate 0 order contexts for black run lengths, and for white run lengths. This is an adaptive arithmetic coding interpretation of the group 3 FAX protocol. [NOTE to reviewers: I expect to have this implemented in time for the final paper.]

Incrementing fields: 0 order context predicting Δ values. Some important X messages contain timestamps and sequence numbers. A simple 0 order context will not effectively predict new values, but the Δ (difference) values usually fall into a small range and can effectively be predicted by a 0 order context.

Default model: 0 order context. Fields effectively using this model include: window-ids, graphics-context-ids, and length fields.

There are many X messages. Specifying predictive models for each field in each message would be a dreary task. Fortunately, most X traffic is generated by just a few messages[4]. This means that we can reap most of the benefits of optimizing all of the messages by optimizing just a few important ones. By default, HBX assigns a 0 order context model to each field in each message. During the optimization task, I assign different models to fields in important messages which did not compress well with the default models. An X message profile isolates important messages. Once an important message has been isolated, I generate profiles like those in Figure 4, which show which fields in the MotionNotify event are compressing and which are not. The profile on the left shows the MotionNotify event before optimization. Many fields compressed well, but the incrementing fields: *Seq. number,* and *Timestamp*, and the coordinate fields *RootX, RootY, EventX,* and *EventY* are not compressing very well at all. Changing the predictive models used by these fields to a model predicting relative changes, rather than absolute values, resulted in the profile on the right. Optimization resulted in an *improvement* of 3:1 for an overall compression ratio of 12:1 for the fields in this message. (The delivered compression rate, which is what the other figures use, will be worse than the weighted average of the fields, because HBX adds overhead in the form of length and tag bits to each message.)

In it's current state, HBX has 244 lines of code mapping predictive models onto message fields. Of these lines, 124 were necessary to parse the messages so that default models could be applied to each field. The other 120 lines of code constitute my optimization efforts to date. This code is written in a special low level language which enforces interfaces, and makes it easy to use the same mapping code for encoding and decoding, which removes a potential source of error from the system. This mapping code could potentially be written by a specification compiler or an RPC stub generator.

The relative performances of Xremote and HBX are shown for each trace in Figure 5. HBX's average performance was just about twice Xremote's and HBX outperformed Xremote on every trace. The relative difference was most apparent for geometry, as in the Idraw and

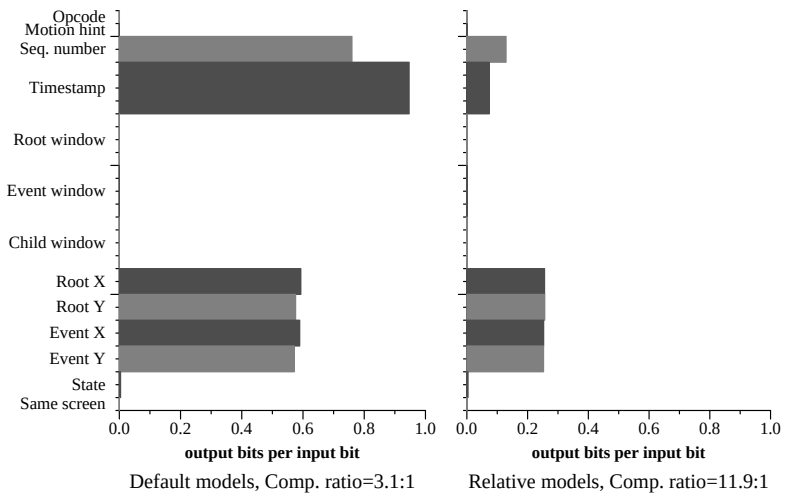

Figure 4: Output bits per input bit for each field in a mouse movement event. The area of each bar is proportional to its compressed size. To the left, all of the fields were compressed using default 0 order contexts. To the right, the *Seq. number, Timestamp*, and coordinate fields were compressed using relative contexts, which predict differences rather than absolute values.

Xfig traces, and especially the X11perf trace. Xremote's performance on the X11perf trace was disappointing since the trace was very repetitive; HBX was able to recognize this regularity and achieve 169:1 compression on the this trace. Certain types of 3D applications generate similarly repetitive traffic since it is common to iterate on sending a transformation matrix and some geometry, with only the transformation matrix changing from iteration to iteration.

So far, we have examined only the average behavior of HBX. Figure 6 shows the instantaneous behavior of HBX across a ten second excerpt from the Mult trace. As you can see, the most important message type in this interval is MotionNotify, with about 30K bits, compressing to around 5K bits for a 6:1 ratio overall. The bits/sec line plot in the middle of Figure 6 shows uncompressed X traffic attaining a peak bandwidth of almost 80K bits/sec, while HBX compressed X traffic attains only about 15K bits/sec. Based on this plot, I believe that a mouse based X applications using HBX compression would be usable at the bit-rates available with cellular modems.

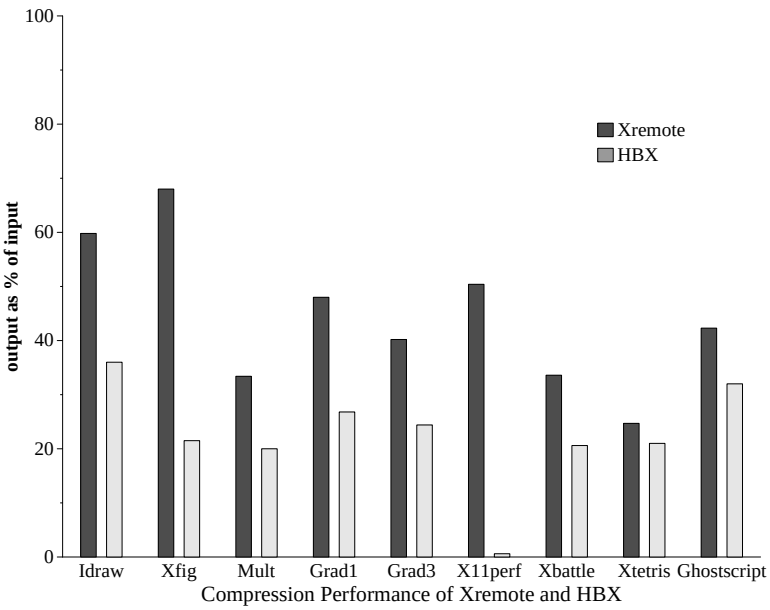

Figure 5: The height of the bars indicates output size as a fraction of input size. HBX outperforms Xremote on every trace.

9 Summary

By doubling or quadrupling the effective bandwidth available to multimedia applications, the HBX techniques allow applications to present a much more interesting class of interactive interfaces across slower network connections such as ISDN, infra-red, and cellular modems.

HBX exploits the separation of predictive model and coder in arithmetic compression to apply a custom compression technique to each field in each message in a graphics protocol, within a unified framework, and without wasting any bits in the transition from field to field. To the extent that the predictive models used accurately encapsulate all of the available knowledge about the fields in question, this technique is optimal. In practice, HBX achieved 4.5:1 compression over a suite of representative traces, effectively twice the compression achieved by the LZW-based Xremote protocol.

I am currently investigating expanded support for image compression in HBX. In the future I hope to investigate applying HBX to lossless compression of audio and video streams.

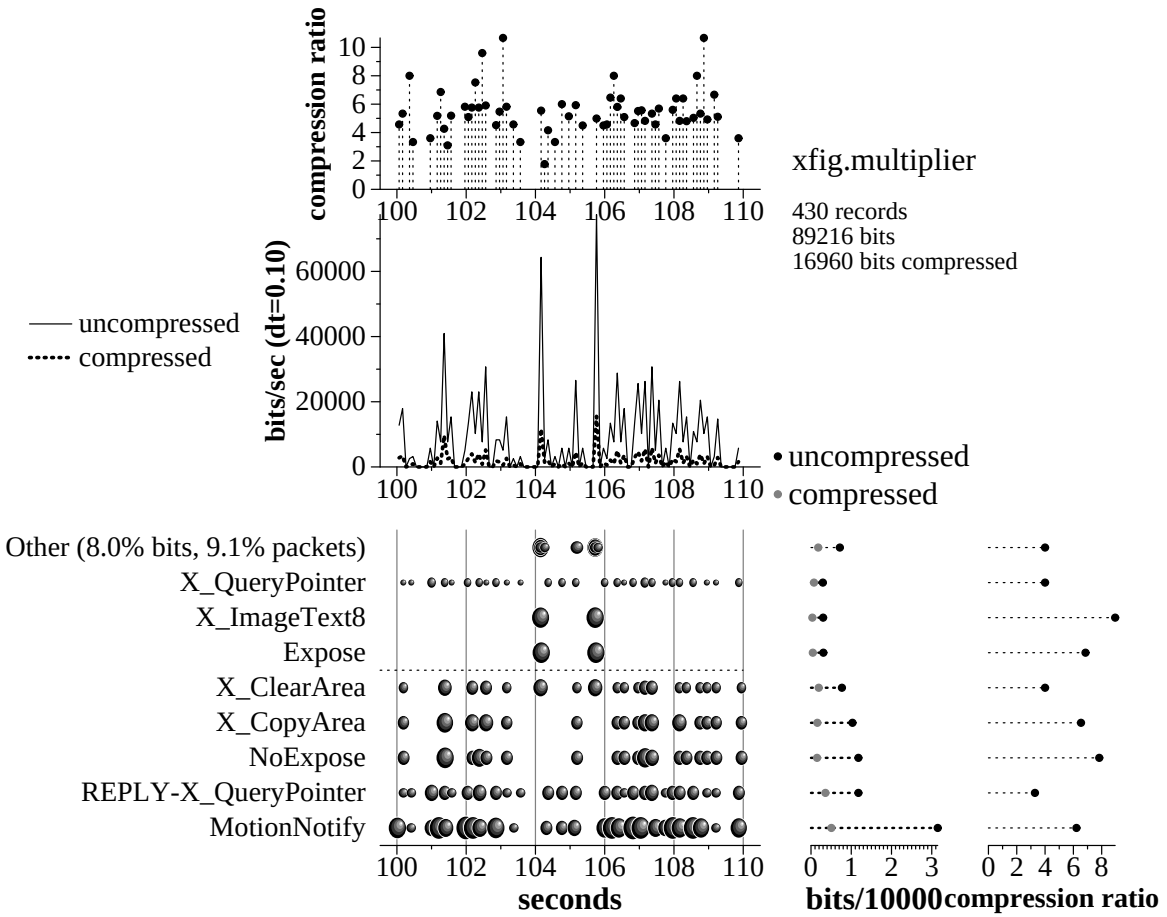

Figure 6: This graph is a ten second excerpt from the Mult trace, mostly showing mouse tracking. In the central area: The X axis indicates seconds since session startup, while the Y axis shows the types of the messages being sent. Each sphere represents a single message unless it is elongated vertically, in which case it represents multiple messages arriving nearly simultaneously. The volume of each sphere is proportional to the number of bits in the message(s) represented. The first dash-dot plot to the right of the spheres shows the total number of bits for each type of packet, while the dash-dot plot at the far right shows HBX compression ratios for each type of packet. The line plot above the spheres shows the number of bits per second as a function of time (solid line before HBX compression; dotted line after). The dash-dot plot at the top shows the HBX compression ratio as a function of time. The integration interval for time-wise plots is 100ms, the human interval of simultaneity.

References

[1] Arden, Michelle J. and James Gosling, "The NeWS book: an introduction to the Networked extensible Window System," Springer-Verlag, New York, 1989

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

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

[4] Danskin, John and Pat Hanrahan, "Profiling the X Protocol," *1994 SigMetrics conference on measurement and modeling of computer systems.* Full paper in Technical Report CS-TR-442-94, Department of Computer Science, Princeton University, Princeton, NJ, January 1994.

[5] Danskin, John and Pat Hanrahan, "Compression Performance of the Xremote Protocol," *1994 Data Compression Conference.* Full paper in Technical Report CS-TR-441-94, Department of Computer Science, Princeton University, Princeton, NJ, January 1994.

[6] Gettys, James, Philip L. Karlton, and Scott Mcgregor, "The X Window System, Version 11," *Software Practice and Experience* vol. 20(S2), S2/35-S2/67, October 1991

[7] Guazzo, M., "A general minimum redundancy source coding algorithm," IEEE Trans. Information Theory, IT-26 (1), 15-25, January 1980.

[8] Moffatt, A., "A note on the PPM data compression algorithm," Research Report 88/7, Department of Computer Science, University of Melbourne, Parkville, Victoria, Australia.

[9] Rissanen, J. J., and G. G. Langdon, "Arithmetic coding," IBM J. Research and Development, 23(2), 149-162, March 1979.

[10] Rubin, F., "Arithmetic stream coding using fixed precision registers," IEEE Trans. Information Theory, IT-25 (6), 672-675, November 1979.

[11] Scheifler Robert W., "The X Window System Protocol," M.I.T. Laboratory for Computer Science. 1988.

[12] Scheifler, Robert W. and Jim Gettys, "The X Window System," *Transactions on Graphics* 5(2), 79-109, April 1986, and *Software Practice and Experience* vol. 20(S2), S2/5-S2/34, October 1991

[13] Shannon, C. E., "A mathematical theory of communication," Bell System Technical Journal, 27, 398-403, July 1948