Adaptive Compression for Remote Visualization

Zoran Constantinescu

Department of Computer and Information Science
Norwegian University of Science and Technology
zoran@idi.ntnu.no

ABSTRACT

Using large data sets for scientific visualization provides more detail and precision, thus making possible greater insights to researchers. However, larger data sets require more computing resources for rendering, resources usually not available locally. Remote visualization techniques using client-server environments allow users to access such large datasets.

One possible solution for remote visualization is the use of compression techniques, where images are generated and compressed at the servers side, then the encoded images are transfered over a data network, decompressed and displayed at the clients side.

One of the problems in remote visualization is to increase the frame rate for the user. A possible solution is to reduce the amount of data transfered over the network, in our case to choose an efficient compression algorithm. However, the better a compression algorithm is, the more computing is necessary for both compression and decompression, increasing the time needed to process the image. The choice of the most efficient compression depends also on the content of the image.

We propose an adaptive compression method for selecting different image compression algorithms for remote visualization. The selection is made based on the performance of previously compressed frames and network transfer delays. We use a reinforcement learning technique to select the compression algorithm for each individual frame. The algorithm was tested using the SGI OpenGL Vizserver, but it can be easily adapted to other remote visualization systems.

Keywords: Remote Visualization, Compression, Vizserver

1 INTRODUCTION

Advances in computing power allows researchers to generate and utilize progressively larger and more accurate data sets. Usually, such data sets are generated using modern supercomputers or clusters of commodity PCs. These computers typically have many CPUs, large amounts of memory, increased I/O capability, and may have specialized graphics hardware. Obtaining the insights offered by these data sets is becoming more and more challenging for the researchers. Transferring the full data set to the researcher's desktop for visualization purposes is most of the time impossible, due to the lack of memory and storage space of local desktop computers.

Scientific visualization research applies a client-server approach to this problem. Remote visualization can be done using different strategies. In a first scenario, the server renders the images and streams them to the client. In a second scenario, the server is doing some of the rendering calculations, such as geometry transformations or visibility determination, while the client is doing the final rendering. Another scenario is where the client is doing all the rendering computations.

Each of these scenarios has tradeoffs. For example, performing the rendering completely on the client side requires high-end desktop computers, not always available to researchers. Performing some of the rendering on the server can greatly improve the visualization, however the low-end client resources may not provide sufficient power to finish the rendering in time.

In the rest of the article we consider only the first scenario, where the server is doing all the computations including the rendering, and the client is responsible only with the display of the final image. Image streaming makes possible remote visualization using low-end desktop computers (thin clients), and can be made independently from any visualization algorithm used.

However, image streaming can require significant network bandwidth. For example, if we consider that the resolution of displayed image is 640x512 pixels with 4 bytes for the RGB colors and the alpha channel, then the size of such an image is 1.25 MBytes. The maximum theoretical frame rate which can be obtained using a 100 MBps bandwidth network is 10 frames per second, considering that the full bandwidth could be used. If we consider remote visualization over a wide area network, then the achievable frame rate will be much lower.

One possible solution to this problem, when using image streaming over the network, is to use compression algorithms. The server renders the image, then compresses that image and sends it over the network. The client is then responsible for decompressing the encoded image and display it. Using different compression techniques for the images, the amount of data transfered over the network can be significantly reduced. How much an image can be compressed depends essentially on the image content. This means that by using different compression algorithm for the same image, different compression rates can be obtained. The problem is to find an automatic way of selecting the right compression algorithm.

Different methods based on analyzing the image and using the compression algorithm that gives the best compressed size are presented in the literature. However, most of these methods can be used only for certain types of images. In this paper we present an algorithm for selecting the compression methods during a remote visualization session without analyzing the content of the image.

We propose an adaptive algorithm for dynamically selecting one of the compression algorithms to be used for each individual frame. The selection is done using a reinforcement learning algorithm, and it is based on different performance measures from the environment: past and present frame rates, compressed image sizes, compression times, estimated bandwidth. The compression method which increases the overall frame rate is chosen. However, from time to time, other compression methods are also used for short periods of time, in order to estimate the potential benefit of selecting them.

Reinforcement learning is a computational approach to learning whereby an agent tries to maximize the total amount of reward (the frame rate in our situation) it receives when acting with a complex, uncertain environment. As opposed to other machine learning methods, in this method the learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them. In many cases, the actions may affect not only the immediate reward, but also the next situation and, through that, all subsequent rewards.

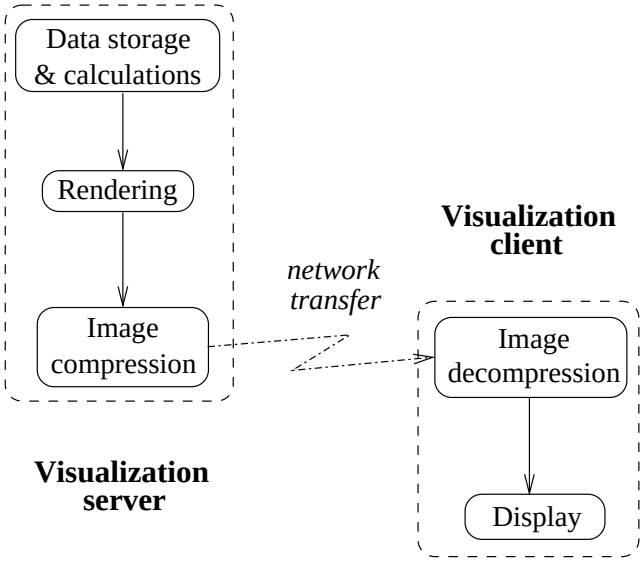

Figure 1: Remote visualization.

In the following sections, we first discuss related work, then present some of the basics of image compression and reinforcement learning. We then describe our adaptive algorithm for compression used for remote visualization. We then present some experimental results using our adaptive algorithm for choosing between four lossless compression methods (BZIP2, ZLIB, LZO, RLE). We use a volume data set with a volume visualization software which is using hardware based texture rendering for creating a typical visualization session. Finally, we discuss our conclusions and give suggestions for future work.

2 Related Work

Renderer implementations exploiting image compression have mostly adopted relatively simple lossless schemes which rely on frame differencing and run-length encoding. While these techniques can deliver acceptable frame rates over local area networks, their compression ratios are highly dependent on image content, and are insufficient in slower networks.

SGI OpenGL Vizserver [7] is a product developed by Silicon Graphics, Inc., to enable remote-visualization applications. Specifically, OpenGL Vizserver is designed to provide users remote access to graphics pipelines of Onyx2 Infinite Reality machines so that they may view rendered output from visualization applications at geographically remote locations while utilizing the powerful pipeline and memory of an Onyx2 machine located at a some centralized place.

OpenGL Vizserver uses programmable compression modules to compress and decompress frames of the rendered scene. It comes with five standard modules (CCC, ICC, SCC, SICC, LCC) and an API that provides the capability to develop new modules with user-defined functionality. Each compression module has the capability of taking advantage of frame-to-frame coherency inherent in most visualizations by implementing an inter-frame compression scheme where only the changing portions of each frame are compressed and sent to the clients. The CCC, ICC, SCC, and SICC compression modules implement lossy compression algorithms. These four schemes are derived from the Block Truncation Coding (BTC) algorithm that compresses a 4x4 pixel block down to two colors plus a 4x4 pixel mask. In addition to lossy compressors, there is also a lossless compression module called LCC. This preserves the original image quality while still saving bandwidth. In many cases the savings are as high as 4x without any reduction in image quality.

A similar solution is presented in [3]. The framework provides remote control to Open Inventor or Cosmo3D based visualization applications. It allows transparent access to remote visualization capabilities and allows sharing of expensive resources. A visualization server distributes a visualization session to Java based clients by transmitting compressed images from the server frame buffer. Visualization parameters and GUI events from the clients are applied to the server application by sending CORBA (Common Object Request Broker Architecture) requests.

Both of these two solutions require the user to explicitly select the compression algorithm to be used. In most of the situations, the user does not have any knowledge about the compression algorithm.

An adaptive compression algorithm for medical images was presented in [4]. The adaptive algorithm presented is based on a classification of digital images into three classes and followed by the compression of the image by a suitable compression algorithm.

The content of the image is analyses based on a validation of the relative number and absolute values of the wavelet coefficients. A comparison between the original image and the decoded image will be done by a difference criteria calculated by the wavelet coefficients of the original image and the decoded of the first and second iteration step of the wavelet transform.

Compression of images was used in [5] for visualizing time-varying volume data over a wide area network. The rendering was done on a remote parallel computer and compression of the images was used for significantly reducing the cost of transferring output images from the parallel computer to the local display. They used lossy compression methods combined with lossless compression methods, which were capable of providing acceptable image quality for many applications, while retaining desirable properties such as efficient parallel compression and fast decompression. They experimented with different combinations of the JPEG, BZIP and LZO compression algorithms, and then selected the combination of JPEG and LZO as giving the best frame rates for their system.

3 IMAGE COMPRESSION

The use of image compression algorithms can significantly improve the amount of data transmitted over the network. All compression algorithms are based on the same principle: compressing data by removing redundancy from the original data. Any nonrandom collection data has some structure, and this structure can be exploited to achieve a smaller representation of the data, where no structure is discernible. This is the case of using lossless compression algorithms. An important feature of image compression is that in many situations it can be lossy, being acceptable to lose image features to which the human eye is not sensitive. Images can be lossy-compressed by removing irrelevant information even if the original image does not have any redundancy.

Different image compression algorithms can be used for different types of images. Each type of image may feature redundancy, but they are redundant in different way. This is why any given compression method may not perform well for all images, and why different methods are needed to compress the different image types.

The choice of the best algorithm is not trivial, most of the time requiring a certain experience with the algorithms. During a visualization session, the type of image can also change, making even more difficult to choose the appropriate algorithm.

One important factor which is important in choosing the compression algorithm is the amount of computation needed for both compressing and decompressing the image. More efficient algorithms, capable of generating smaller compressed images are usually requiring more CPU power. This becomes very critical, especially for high resolution images. There is a tradeoff between the amount of computation time needed to generate the compressed image and the amount of time used to transfer it over the network. There are cases when an investment in a more efficient compression algorithm can result in a higher frame rate, especially when the remote visualization is done over low bandwidth networks.

In many situations, the actual network bandwidth available which can be used is less that the maximum bandwidth. This is the case when the remote visualization is done without having a dedicated network connection between the visualization server and client, especially when using wide area networks for visualization over long distance. An additional problem is that this available network bandwidth can change significantly during a remote visualization session. This can be due to other data traffic in the network.

For our study, we used four lossless compression algorithms. The choice was mainly made based on the performance of these algorithms for general image compression and the availability of optimal implementations as software libraries.

The first algorithm (ZLIB) is the so-called "deflation" algorithm, which is used in the popular programs zip and gzip. This is a dictionary based compression method: it selects strings of symbols and encode each string as a token using a dictionary. It is based on the LZ77 compression method combined with static Huffman encoding. The compression time and image sizes are pretty good, however for certain image type compression can be very poor.

The second algorithm called Lempel-Ziv-Oberhumer (LZO), an optimized dictionary based method, which is more suited for real-time compression-decompression. It offers pretty fast compression and very fast decompression, however it favors speed over compression ratio. The resulting compressed images can be very large, thus increasing the transfer time over the network.

The third algorithm used (BZIP2) is based on the Burrows-Wheeler method, which is a compression method using block sorting. The input stream is read block by block and each block is encoded separately as one string. The main idea is to start with a string S of n symbols and to scramble (permute) them into another string L which satisfies: (1) any area of L will tend to have a concentration of just a few symbols; (2) it is possible to reconstruct the original string S from L. The method is a general purpose method, which works well on images and can achieve very high compression ratios. The disadvantage of this algorithm is that it requires a lot of computing, both compression and decompression being slow. Since the algorithm is compressing individual blocks independently, it is possible to use a parallel version of the compression to reduce the time.

The last algorithm we used is a simple Run Length Encoder (RLE). The idea behind this approach is the following: if a data item *d* occurs *n* consecutive times in the input stream, replace the *n* occurrences with the single pair *nd*. This is well suited for certain types of images, with large areas containing the same pixel value. The size of the compressed stream depends on the complexity of the image. The more detail we have, the worse the compression is. The algorithm being extremely simple, very efficient implementations could be implemented. It is also well suited for a parallel encoding.

4 REINFORCEMENT LEARNING

Reinforcement learning [8] is a computational approach for goal-directed learning from interaction. The learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them. Reinforcement learning is different from supervised learning, the kind of learning from examples provided by a knowledgeable external supervisor. In interactive problems it is often impractical to obtain examples of desired behavior that are both correct and representative of all the situations. In uncharted situations, where one would expect learning to be most beneficial, an agent must be able to learn from its own experience.

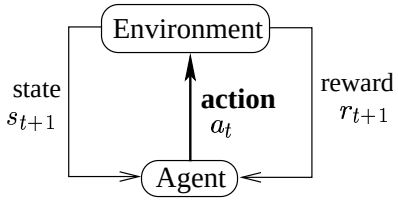

Figure 2: The agent-environment interaction.

In reinforcement learning, the learner and decision maker is called the *agent*. The thing it interacts with, comprising everything outside the agent, is called the *environment*. These interact continually, the agent selecting actions and the environment responding to those actions and presenting new situations to the agent. The environment also gives rise to rewards, special numerical values that the agent tries to maximize over time. More specifically, the agent and environment interact at each of a sequence of discrete time steps, t. At each time step t, the agent receives some representation of the environment's state, s_t, and on that basis selects an action, a_t. One time step later, in part as a consequence of its action, the agent receives a numerical reward, r_{t+1}, and finds itself in a new state, s_{t+1}.

At each time step, the agent implements a mapping from states to probabilities of selecting each possible action. This mapping is called the agent's policy. Reinforcement learning methods specify how the agent changes its policy as a result of its experience. The agent's goal, roughly speaking, is to maximize the total amount of reward it receives over the long run.

One of the challenges that arise in reinforcement learning is the trade-off between exploration and exploitation. To obtain a lot of reward, a reinforcement agent must prefer actions that it has tried in the past and found to be effective in producing reward. But to

discover such actions, it has to try actions that it has not selected before. The agent has to *exploit* what it already knows in order to obtain reward, but it also has to *explore* in order to make better action selections in the future. The agent must try a variety of actions and progressively favor those that appear to be best.

Another key feature of reinforcement learning is that it explicitly considers the whole problem of a goal-directed agent interacting with an uncertain environment. All reinforcement learning agents have explicit goals, can sense aspects of their environments, and can choose actions to influence their environments. It is usually assumed that the agent has to operate despite significant uncertainty about the environment it faces.

5 Adaptive Compression

The adaptive compression algorithm we are proposing is using a reinforcement algorithm as presented in the previous section.

We consider the frame rates as the rewards for each time step. An example of frame rate variation during a typical visualization session is presented in figure 3. The figure shows the current and average frame rates obtained by using the RLE (Run Length Encoding) compression algorithm for two situations: one 100 MBps and one 10 MBps network connection of the client to the LAN. There are large variations in the current frame rate, especially when there is enough available network bandwidth (in the left and right regions of the figure). The average is done using the last ten frame rates.

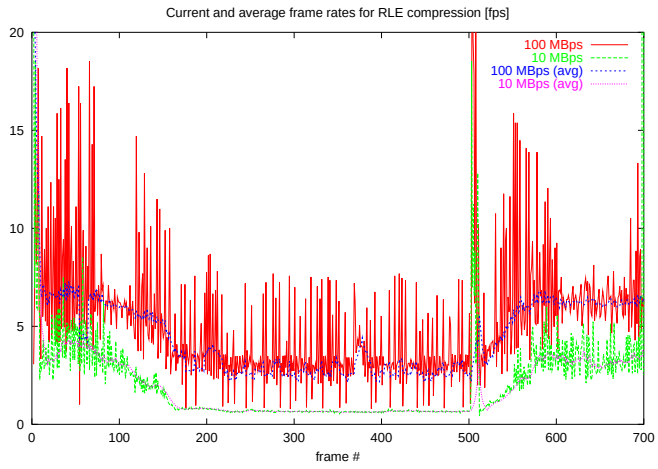

Figure 3: Frame rate variations and average.

Due to these large variations, the algorithm is making the selection of the compression algorithm based on these average values. For each selected compression method, at least 10 frames will be rendered using this method, providing this way a better estimate of performance of the algorithm.

The adaptive algorithm works as follows: it starts with one of the compression methods (LZO in our case) and it uses it for the next 10 frames to get an estimate of its performance. After that, it is trying in a similar way the other compression methods, and when all the methods are tested it is choosing the best of the algorithms.

From time to time, another compression method, different from the current one, is selected randomly and evaluated. If the new method is providing a better performance, i.e. increased frame rate, then it is selected as the next compression method. We used an interval of 50 frames between trying another compression method.

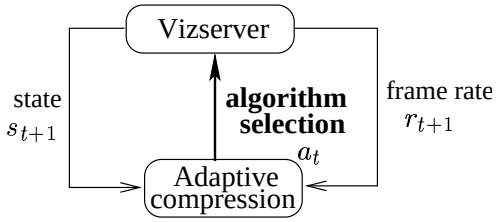

Figure 4: The adaptive algorithm.

6 Experimental Results

We conducted tests using an SGI Onyx2 2400 parallel computer as the remote visualization server. This computer consists of 32 R12000 RISC processors at 300 MHz, with a total memory of 16 GBytes and two Infinite Reality3 graphic pipelines. For the local visualization client we used a desktop PC with a Pentium 3 processor, running at 500 MHz, with 256 MBytes of memory. The operating system used was Linux with a 2.4.19 kernel.

As a remote visualization system, we used the SGI OpenGL Vizserver software. This software allows remote rendering of the images on the SGI server, which are then compressed and sent over the network to the client for display. The SGI Vizserver offers an API for writing additional compression modules to be used.

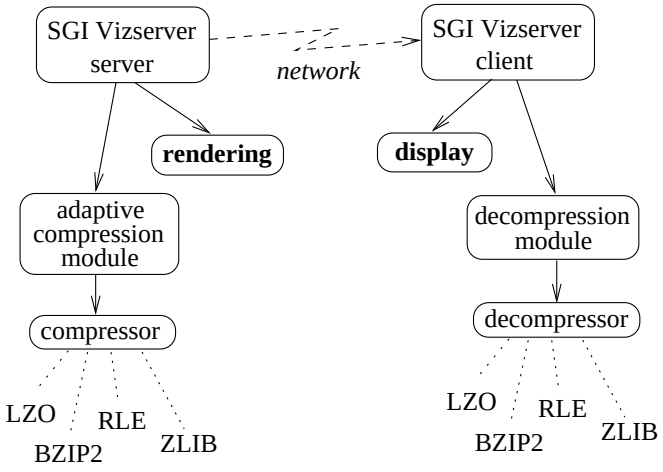

Figure 5: Vizserver architecture.

We implemented four compression modules using the four lossless methods described in section 3. These modules are basically wrappers for existing software libraries which implement the compression methods. The modules give a simple interface to both the compression and decompression, which is used by the adaptive algorithm. This is implemented as a compression module for the SGI Vizserver using the development API provided with the software. The adaptive algorithm was implemented using the C++ programming language.

We chose the SGI Vizserver for several reasons. First because we had access to an SGI parallel visualization server which had it available, and second, because the API used for the compression modules is very simple, making it very easy the implementation of different compression techniques. Another reason was that the use of the vizserver is transparent to the applications used.

There were however some problems we experienced. One of them is that the version we were using (3.1 beta) was quite unstable.

We had to go through many crashes of the server software while developing and experimenting with different compression algorithms. One of the disadvantages in using SGI Vizserver is that the server hardware must be an SGI computer. However, the algorithm we implemented for the adaptive compression, together with the four compression modules are very easy to adapt to other similar remote visualization systems, due to the modular of implementation.

One possible useful parameter we didn't have access to while using the SGI Vizserver framework was the effective time required to send each of the compressed frames over the network. The only available parameters we could use were the compression time for the frames and the time between two consecutive calls for the frame compression algorithm.

In our experiment, the size of each frame was 640x512 pixels with 4 bytes per pixel (RGB plus alpha channels). We used the Volview program for visualizing a volume data set of 256x256x77 voxels of a CT scan. The Volview is part of the SGI Volumizer2 software, and is using hardware accelerated 3D texturing for volume visualization This is a direct data visualization techniques using textured data slices which are combined is a specific order using a blending operator. This techniques takes advantage of graphics hardware and resources by using OpenGL 3D-texture rendering, allowing applications to obtain high interactive performances.

The experiments were conducted using two different network connections between the client and the 100 MBps LAN containing the server. In the first situation, we connected the client using a 100 MBps network card to the LAN. In the second situation, we used a 10 MBps network card for connecting the client.

Using a modified version of the Volview program, we recorded the translation and rotation vectors of the volume data for each frame generated during a typical interactive visualization session. We then played back the same session using the four different compression methods and then the adaptive algorithm. Frame rate averages for all five situations are presented in figures 6 and 7, for the two network connection situations.

In both situations, the adaptive algorithm is searching for the best algorithm in the beginning, thus giving low frame rates. However, when it finds the best algorithm, it keeps it for the rest of the visualization session.

7 Conclusions

As the amount and size of scientific data continues to increase, the demand for high-resolution imaging will also increase. Remote visualization is one solution for making accessible remote data sets to users with low capability desktop machines. Use of image compression techniques permits remote visualization of larger resolution images or over lower bandwidth networks.

There are different compression methods for different kind of images. Some of the methods give very good compression rates but only for a certain types of images, while for other types of images the compression is poor. The selection of the best compression algorithm is still a matter of experience. One other problem is that in most cases a better compression method also requires much more computational power. There is a tradeoff between the size of the compressed image and the amount of computation used in order to obtain the optimal frame rate using remote visualization.

In this paper we presented an adaptive algorithm based on reinforcement learning for choosing one of the available compression methods in order to maximize the frame rate. Our experiments show that such an algorithm can work in a dynamic and uncertain environment, consisting of a visualization server, a visualization client, and a network for transferring the compressed images between the server and the client.

One of the problems we experience with the current algorithm is that, in certain situations, one of the compression methods which is evaluated by the adaptive algorithm is giving really poor frame rates. This affects the interactive responsiveness of the application. One possible ' improvement of the algorithm would be to use a different selection algorithm for evaluating the next possible method, by making actions which give small rewards to be less likely to occur. In this way, compression methods which give poor frame rates will be less probable to be selected in the future.

The modules for the compression methods and the adaptive algorithm are available for download, both as source code and binary at the following web site:
`http://www.idi.ntnu.no/~zoran/vizserver`.

References

[1] M. Burrows and D. Wheeler. A block-sorting lossless data compression algorithm. Technical report, Digital Equipment Corporation, Palo Alto, California, 1994.

[2] T. Chu, J. E. Fowler, and R. J. Moorhead II. Evaluation and extension of sgi vizserver. In *Visualization of Temporal and Spatial Data for Civilian and Defense Applications III*, 2001.

[3] K. Engel, O. Sommer, and T. Ertl. A framework for interactive hardware accelerated remote 3d-visualization. In *Proc. TCVG Symposium on Visualization, VisSym'2000*, 2000.

[4] Sergei Hludov, Claus Shroter, and Christoph Meinel. Adaptive compression of image data. In *Broadband European Networks and Multimedia Services, SYBEN'98*, 1998.

[5] Kwan-Liu Ma and David M. Camp. High performance visualization for time-varying volume data over a wide area network. In *Proc. of Supercomputing 2000*, 2000.

[6] David Salomon. *Data Compression: The Complete Reference*. Springer-Verlag, 2000.

[7] SGI. Opengl vizserver 3.0 white paper. Technical report, Silicon Graphics, Inc., 2003.

[8] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 1998.

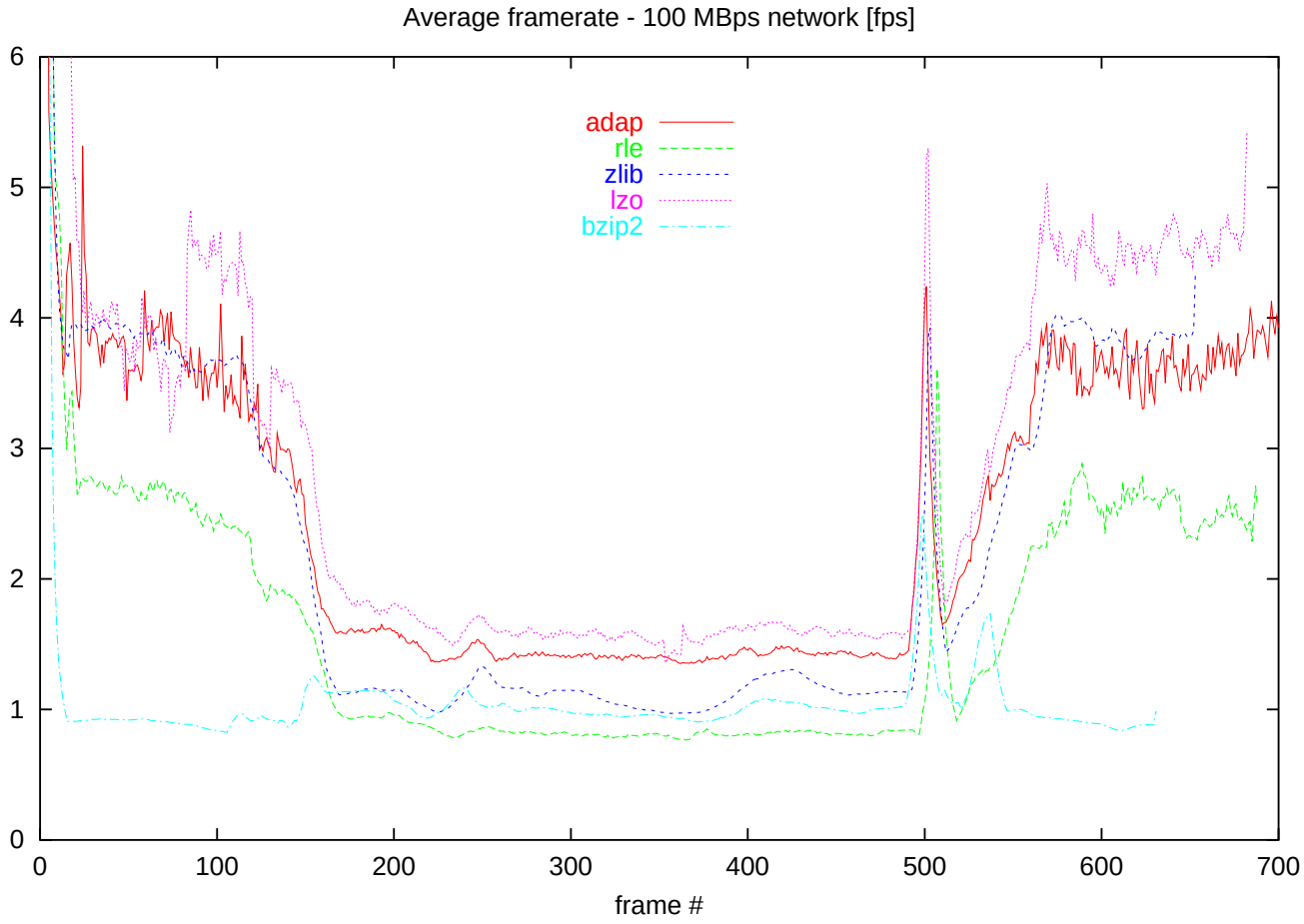

Figure 6: Average frame rate - 100 MBps network.

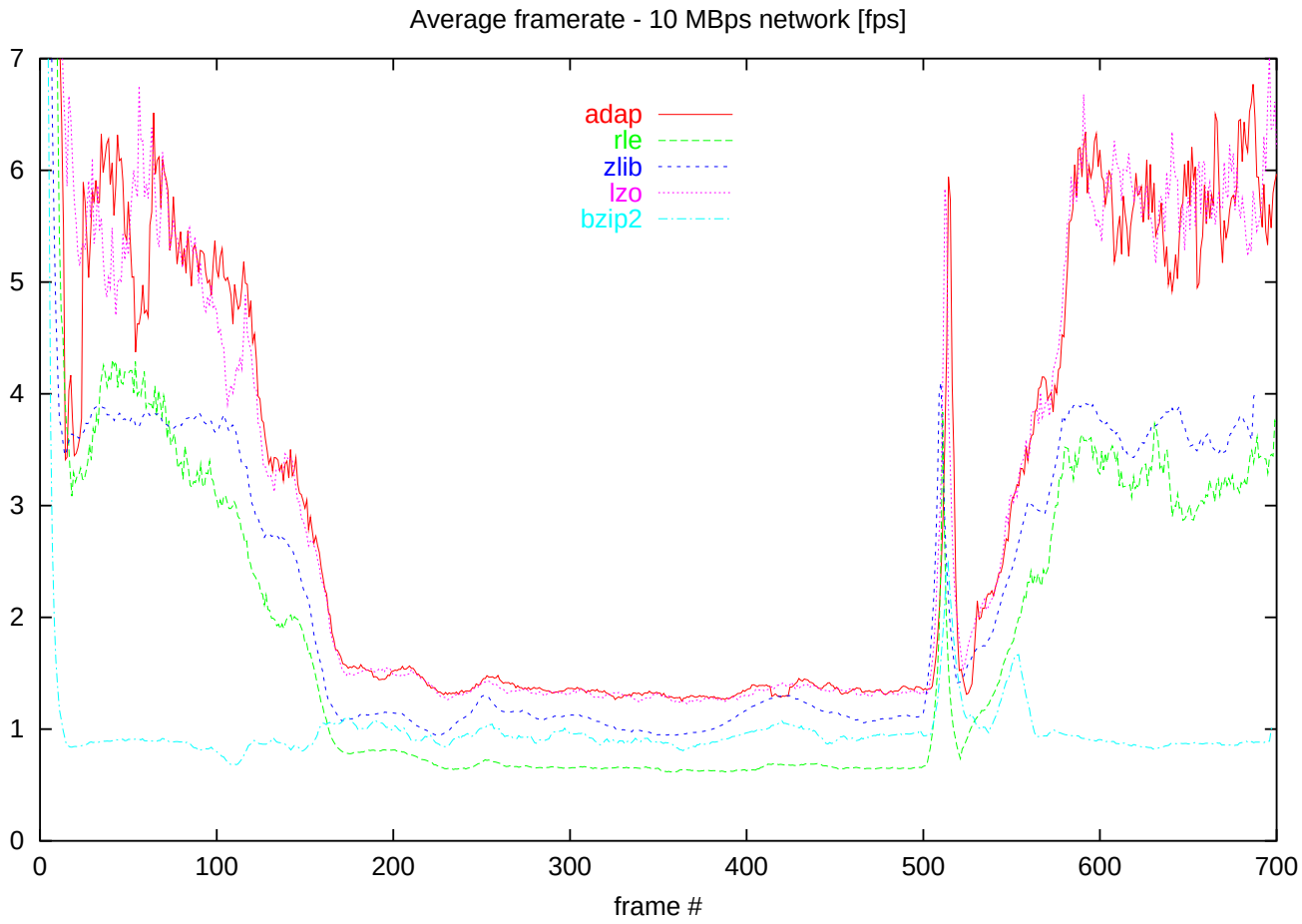

Figure 7: Average frame rate - 10 MBps network.