Report Number: WUCS-92-15

1992

An Overview of Segment Streaming for Efficient Pipelined Televisualization

Fengmin Gong and Gurudatta M. Parulkar

The importance of scientific visualization for both science and engineering endeavors has been well recognized. Televisualization becomes necessary because of the physical distribution of data, computation resources, and users invovled in the visualization process. However, televisualization is not adequately supported by existing communication protocols. We believe that a pielined televisualization model (PTV) is suitable for efficient implementation of most visualization applications. In order to support this model over high speed networks, we are developing a segment streaming interprocess communication (IPC) mechanism within the Axon communication architecture. Important aspects of this development include: the segment streaming paradigm which supports low-overhead... **Read complete abstract on page 2.**

An Overview of Segment Streaming for Efficient Pipelined Televisualization

Fengmin Gong and Gurudatta M. Parulkar

Complete Abstract:

The importance of scientific visualization for both science and engineering endeavors has been well recognized. Televisualization becomes necessary because of the physical distribution of data, computation resources, and users invovled in the visualization process. However, televisualization is not adequately supported by existing communication protocols. We believe that a pielined televisualization model (PTV) is suitable for efficient implementation of most visualization applications. In order to support this model over high speed networks, we are developing a segment streaming interprocess communication (IPC) mechanism within the Axon communication architecture. Important aspects of this development include: the segment streaming paradigm which supports low-overhead communication as well as concurrency between the communication and local computation; a two-level flow control method for distributed pipeline synchronization; and an application-oriented error control method which allows error control to be optimized for different applications. This paper describes a set of ideas that lead to the design of this IPC mechanism.

An Overview of Segment Streaming for Efficient Pipelined Televisualization

Fengmin Gong and Gurudatta M. Parulkar

WUCS-92-15

March, 1992

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
St. Louis MO 63130-4899

This work was supported in part by the National Science Foundation, and an industrial consortium of Bellcore, BNR, DEC, Italtel SIT, NEC, NTT and SynOptics.

An Overview of Segment Streaming for Efficient Pipelined Televisualization[†]

Fengmin Gong*
lfg@wucs.wustl.edu
(314) 935-4163

Gurudatta M. Parulkar*
guru@flora.wustl.edu
(314) 935-4621

Abstract

The importance of scientific visualization for both science and engineering endeavors has been well recognized. Televisualization becomes necessary because of the physical distribution of data, computation resources, and users involved in the visualization process. However, televisualization is not adequately supported by existing communication protocols.

We believe that a pipelined televisualization model (PTV) is suitable for efficient implementation of most visualization applications. In order to support this model over high speed networks, we are developing a segment streaming interprocess communication (IPC) mechanism within the Axon communication architecture. Important aspects of this development include: the segment streaming paradigm which supports low-overhead communication as well as concurrency between the communication and local computation; a two-level flow control method for distributed pipeline synchronization; and an application-oriented error control method which allows error control to be optimized for different applications. This paper describes a set of ideas that lead to the design of this IPC mechanism.

1. INTRODUCTION

The 1987 NSF report *Visualization in Scientific Computing* [17] defines visualization as a method of computing which transforms symbolic information into geometric information, enabling researchers to gain better insight into their simulations and computations. It has been further pointed out that visualization is a critical tool for discovery and understanding, as well as a tool for communication and teaching [7].

Scientific visualization has emerged as a major computer-based field of research due to the following three key factors: (1) large-scale computations and high bandwidth data sources (e.g. medical

[†]This work was supported in part by the National Science Foundation, and an industrial consortium of Bellcore, BNR, DEC, Italtel SIT, NEC, NTT, and SynOptics.

*Computer & Communications Research Center and Department of Computer Science, Bryan 405 - Box 1115, Washington University, One Brookings Drive, St. Louis MO 63130-4899.

scanners and satellite imaging) generate huge amounts of data that scientists cannot analyze in numerical form in reasonable time [11, 21]; (2) it is believed that the human vision system has a data bandwidth on the order of gigabits per second, but this bandwidth is not exploited by text-oriented data presentation methods which limit the data bandwidth made available for interpretation [26]; and (3) images can convey information far more effectively than numerical data [17].

The use of visualization for many scientific applications has been well documented [18]. New visualization applications have been rapidly emerging with developments in visualization methodology and underlying technologies. In short, visualization is beginning to revolutionize the way research is done in various disciplines of science and engineering.

Televisualization is visualization that utilizes data and computing resources that are physically distributed. There are three components in the visualization process, namely data, computation, and user interaction. As long as the locations of these components are not all the same, the need for televisualization arises. For example, in a three-tiered computing facility environment such as the one at NCSA [11] and as envisioned in the NSF report, a large simulation is run on a supercomputer while a scientist views the visual output and steers the simulation at a workstation. Furthermore, the scientist may wish to do parts of the visualization computation on separate machines in order to distribute the computation load and achieve better performance. Therefore, we believe a significant fraction of practical visualization will be televisualization.

It is interesting and important to note that many visualization applications fit into a pipeline model. For example, steps of a visualization process such as data preparation, mapping of raw data to attributes of visual data, and graphics rendering are readily identifiable stages of a pipeline. The adoption of pipeline model for visualization applications is also apparent in two existing visualization systems (apE [9] and AVS [25]). Since televisualization is important, and a pipeline is an appropriate computation model for many televisualization applications, a natural question is:

Can we successfully use the pipeline model across high speed networks to make demanding televisualization applications possible?

We believe the answer to this question is yes and our undertaking is to develop the necessary support and to demonstrate the viability of the pipelined televisualization (PTV) model. The purpose of this paper is to discuss the requirements of PTV and outline a solution for supporting it. The rest of the paper is organized as follows: Section 2 discusses the requirements of a visualization development environment. Since there are two major components to PTV: pipelined computation and communication, we will describe the computation model in section 3 and leave the discussion of the communication model to section 4; Section 5 presents major components of our solution; Section 6 concludes the paper.

2. VISUALIZATION ENVIRONMENTS

It has been pointed out by many that in order to make scientific visualization a practical tool for every scientist, a visualization development environment which is user-friendly, functionally complete, and

efficient is necessary [8]. There are at least two existing visualization environments, apE and AVS, that are aimed at achieving these objectives. Both systems use a graphical user interface and allow scientists to build their visualization applications by selecting appropriate function modules and connecting them in a general pipelined fashion. For example, consider Figure 1, which is a screen dump from apE version 2.1 showing a typical user interface in apE. In the center is the graphical display of a visualization pipeline, which was built by dragging and connecting appropriate icons from the *palette* window on the left. For each module in the pipeline, more detailed information (e.g. input, output, and control) can be displayed in the *element info* window. In particular, there is a *host* box for users to specify which host the module should be executed on. The current *element info* window on the top of the screen is showing information on the module *forizon*, which calculates a floating horizon hidden-line surface. The executing host is *localhost* which is the default. Below the pipeline display, two control dials for the *forizon* are also displayed along with a control panel for the normalization module *norm*. The pipeline is invoked by clicking on the *start* button in the center window, and module parameters can be adjusted from the control surface while the pipeline is running. The pipeline is stopped by clicking on the *stop* button. Windows to display additional information or images can be created as needed. We assume that this kind of user interface will be built on our IPC solution to support PTV.

It is obvious that this kind of window-based graphical interface together with the module-based programming paradigm makes such environments user-friendly and easy to use. Moreover, these environments allow users to add new functionality by designing new modules, and thus tailoring the system to any applications. However, whether a visualization application can be executed with efficiency is a big question. Although efficiency depends on many factors such as the algorithms and the implementations of the modules, effective speeds of compute engines used, and efficiency of the interprocess communication support, IPC support is becoming the most critical factor, as single processors approach their speed limits and the most efficient algorithmic implementations are achieved. In an environment like apE, we expect that most visualization applications will need to run different stages of the pipeline on distributed machines. Although apE does provide support for distributed execution, it will not be able to satisfy the IPC requirement for efficient PTV. In section 4, we will consider these requirements and look at the current apE support for distributed pipeline execution in more detail. First, we take a closer look at the computation model of the PTV.

3. PTV COMPUTATION MODEL

There is a significant class of visualization applications whose computation contains several well-defined stages [11, 25]. We can generalize them into the following five-stage description:

Data generation. This is the step in which the original data is generated. Numerical simulation and image scanning are two examples.

Data preparation. This stage derives visualization data from the observational, experimental, or simulation data. For example, for visualization of a simulation, additional data points may be

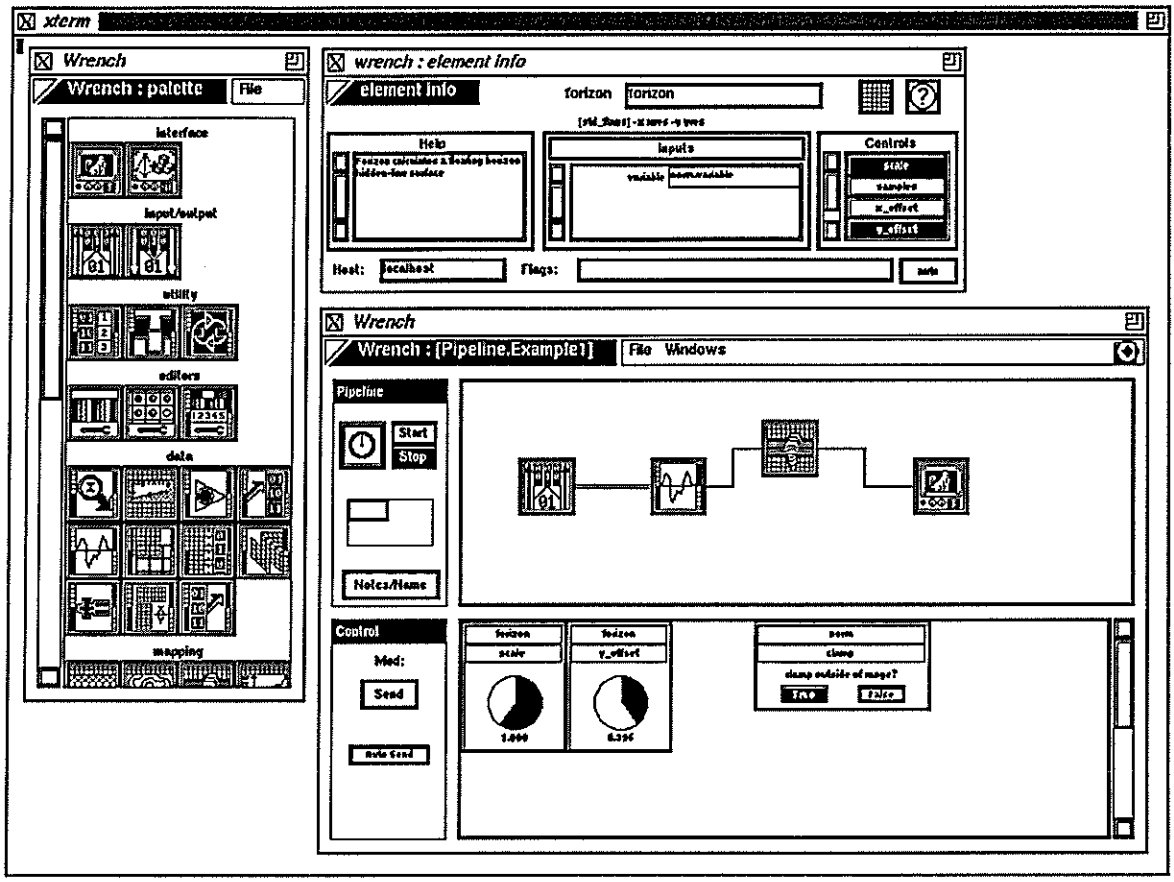

Figure 1: An apE Interface Screen Dump

interpolated to transform an irregular grid into a regular one.

Visualization mapping. This stage constructs abstract visualization objects (AVO [11]) from the data derived at the previous stage. Specifically, it maps the derived data (e.g. pressure, temperature, intensity) to the attributes of the AVO (e.g. geometry, color, opacity) for graphics rendering.

Graphics rendering. Abstract visualization objects generated in the last step are rendered into images for display at this step. The actual operations may include image processing, surface rendering, and volume rendering.

User interaction. The user interacts with the visualization process through the image display and input devices such as a keyboard, a mouse, and a set of dials. Interpreting user input and interfacing with rest of the computation stages are functions of this stage.

These stages can be readily mapped onto a pipeline. In addition, computation at some stages may be further partitioned, thus creating more stages for the pipeline.

The pipeline principle has been successfully used in many ways and at different levels of abstraction to achieve high performance. For example, instruction pipelines and arithmetic pipelines are ubiquitous in the designs from microprocessors to supercomputers. Pipelining is also one of the major principles used in systolic array and wavefront array processors [14, 15]. Many methodologies for the design and operation of hardware pipelines have been developed [6, 13]. The proposed use of the pipeline model across networks for televisualization, however, is a new idea and requires significant research. For instance, most of the existing pipelines are synchronous pipeline (i.e., all stages of the pipeline are synchronized by a single global clock). On the other hand, the PTV pipeline is asynchronous and coordination between stages has to be accomplished via explicit message exchange. Accurate performance modeling of such pipelines still represents a challenge. Furthermore, intrastage computation, interstage communication, and their interaction become important factors affecting overall PTV performance.

There are two major requirements for efficient pipelining: engineering the pipeline to achieve minimum and equal cycle time for all stages; and operating the pipeline with minimum delay or interruption. To satisfy the first requirement in televisualization pipelines, visualization computation has to be partitioned with respect to the communication overhead and allowed to overlap as much as possible with the communication process. For the second requirement, an efficient IPC paradigm with appropriate flow and error control mechanisms is necessary to minimize the effect of the network latency and errors on the operation of the pipeline.

4. PTV COMMUNICATION MODEL

We discuss the communication aspect of pipelined televisualization in this section.

4.1. Protocol Hierarchy

In a pipelined televisualization process, different stages of the pipeline are executed on separate processors, communication protocols are needed for the interprocess communication. Figure 2 illustrates, at a high level, the communication model for one visualization process (VP) at stage k to pass data to another VP at stage $k+1$. The two stages are separated by an internetwork which is abstracted as a direct link in the figure. Interprocess communication is provided through a set of communication protocols, which logically consists of the transport protocol (TP), the internetwork access protocol (IAP), and the network access protocol (NAP). In essence, the network access protocol provides communication connections within a homogeneous network while the internetwork access protocol provides connections across a set of heterogeneous networks, and the transport protocol provides "reliable" end-to-end communication over the underlying internetwork.

In practice, performance of the underlying internetwork hardware and the communication protocols will have direct impact on the performance of pipelined televisualization. We look at the availability of networking support next.

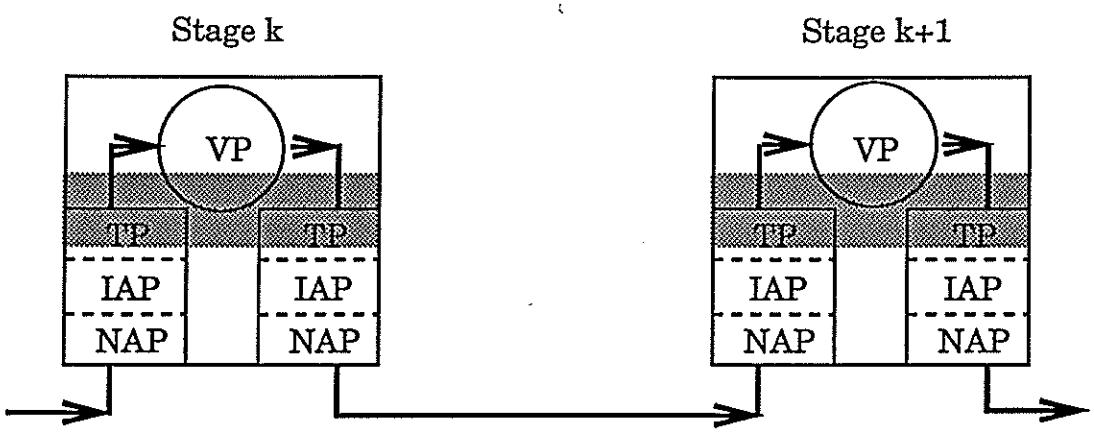

Figure 2: Interstage Data Communication Path

4.2. Networking Support

Suitable network support is critical to the success of televisualization. Trends in networking suggest that the necessary support will be possible. Fast packet switching networks are being developed in many research laboratories (e.g. Broadcast Packet Network at Washington University [24]). These networks can support communication channels with high transmission rates and low queueing delay. They will have some resource allocation capability and provide performance guarantees. In order to make this guaranteed network bandwidth available to applications, the host-network interface must be designed to support high data rates. A number of research groups have also begun addressing these issues [2, 12, 22].

However, it is important to note that host-to-host connections with high bandwidth and performance guarantees are necessary but not sufficient conditions for successful deployment of demanding distributed applications. As a result of decreased queueing delay in newer networks, the speed of light propagation delay will increasingly dominate the communication latency in wide area networks. The networks will continue to have more packet errors and losses than a local environment. Moreover, the new networks have much higher bandwidth-delay products[‡] which affect flow and error control strategies at the application and transport levels. It is the responsibility of applications and the transport protocols to cope with these conditions and to translate the high bandwidth into high performance for the applications. It has been recognized that suitable solutions can be found by developing deep understanding of the communication requirements of various classes of applications [19]. In the case of PTV, the pipelined processing and high communication latency coupled with the demand for high performance present a unique set of interprocess communication (IPC) requirements that have not been adequately addressed by existing research on networking or visualization.

[‡]The bandwidth-delay product of a network is the product of the network peak bandwidth (bits/sec) and the maximum transmission delay (sec) between two hosts on the network. It is a measure of how much data can be in transit before a feedback message reaches the other end of a connection. For example, if a wide area network has a peak bandwidth of 1 gigabits per second and a maximum host-to-host delay of 100 milliseconds, the bandwidth-delay product is 100 megabits.

4.3. IPC Requirements

We focus our energy on the IPC mechanism, which deals with overlapped issues between the application and the transport protocol (corresponding to the shaded areas in Figure 2). The internetwork access protocol will be assumed to be a multi-point congram-oriented high performance internet protocol (MCHIP) available from a related research effort [16, 20]. This IAP will provide connection-oriented internetwork services with plesio-reliability[§]. The requirements of the IPC mechanism are discussed in the next two paragraphs.

A pipeline achieves parallelism by overlapping processing among different stages [6, 13]. In a distributed pipeline such as the one for PTV, due to the physical distribution of pipeline stages, interstage communication incurs significant delay. A stage that is the slowest becomes the bottleneck of the pipeline and limits the pipeline speed. In order to achieve efficient pipelining, the interstage communication has to satisfy a number of conditions:

- A visualization application typically involves a large number of data segments of considerable size. These segments have to be streamed through the pipeline to allow overlapped processing.
- Each physical stage of the pipeline has to concurrently perform the receiving, processing and transmitting of data items to achieve overlapping.
- Interstage segment transfer time should match closely with the computation time at a stage, to avoid wasting communication bandwidth or computing power.
- An application should be able to specify its reliability requirement to the IPC mechanism and the IPC mechanism should enforce it only to the degree that is necessary to satisfy the requirement, thus avoiding any unnecessary error control overhead.
- Buffer overflow in the pipeline should be avoided because loss of partially processed data due to overflow may require restarting the pipeline from an earlier stage, thus wasting computing cycles and introducing unnecessary (and unacceptable) delay.

ApE's current IPC support for distributed execution is based on the socket library in the Unix system. The socket library supports the message-passing IPC paradigm by providing a set of library functions for creating sockets, establishing connections between sockets, and sending and receiving data from the sockets. Specifically, apE uses a stream socket in the Internet domain for communication between distributed modules of the pipeline. Since this type of socket is built on top of the TCP (Transmission Control Protocol), which is designed to provide 100% reliable communication regardless of the "cost" and application needs, it is impossible for applications to trade error tolerances with the associated control overhead. It should also be noted that the only flow control function available in the socket mechanism is the one provided by TCP, and this is not suitable for PTV pipelines. It is possible to implement the pipeline flow control on top of the socket mechanism, but

[§] *Plesio* comes from the Greek word *plesios* which means close to or almost.

that puts extra burden on applications and will cause more overhead due to the necessary context switching. Finally, we recognize that the TCP/IP protocols were not designed with PTV applications in mind, and are not suitable for our IPC mechanism [4, 5, 27]. In the next section, we present our IPC solution to these problems.

5. SEGMENT STREAMING

To support efficient interstage communication for PTV, we proposed a new IPC mechanism. The mechanism is based on a new IPC paradigm, called segment streaming, and it is supported by a special ALTP (Application-oriented Light weight Transport Protocol) within the framework of Axon [22]. Axon is a communication architecture designed to support a high performance data path that delivers high (inter)network bandwidth directly to applications. The two distinguishing components of the ALTP are a two-level flow control method, and an application-oriented error control method. We will first describe the segment streaming paradigm, and then the two-level flow control and the application-oriented error control in two subsections.

A segment is a logical unit of data that is independently processed by application processes. *Segment streaming* is an IPC paradigm for supporting exchanges of large number of segments among processes with high bandwidth and low latency. First of all, by taking advantage of the fact that the data to be exchanged is known and large in volume, it allows a single system call (by an application process) to perform the request for all segments; each segment will be transmitted when ready across a connection without the latency of request or setup. Moreover, segment streaming supports a set of options which applications can use to optimize the IPC for their particular needs. Segment streaming is provided through two transport level operations: *send-stream* and *get-stream*, which are invoked by applications by making the corresponding system calls. The set of options can be summarized as follows:

Repeated/Sequential Transfer. This option allows the application to specify if the stream consists of repeated transfers of a single segment or sequential transfer of segments in a group.

Flow Control. This option allows applications to specify their choice of flow control methods. Flow control determines when to initiate the transmission of each segment in the stream. This transmission can be initiated at fixed time intervals (*interval synchronized*), triggered by the execution of a specific program call (*program synchronized*), or determined by a two-level flow control mechanism which will be described later.

Error Control. An application can specify its tolerances for different types of errors. The error control mechanism will ensure that the tolerances are satisfied with minimal overhead. More details on the error control will be presented later. However, it is important to point out that applications will be able to specify error tolerances ranging from accepting every error to accepting no error at all.

Send-stream is used to send a segment stream to a remote host. Besides the options described above, application specifies segments to send, and their destination. Upon invocation, a local control block for the stream is created and a control packet is sent to the remote host, using a connection on top of MCHIP, which must have been established. The packet contains the name of the segment group and destination user and process information. After an acknowledgement is received from the remote host, a series of data packets will flow, which corresponds to segment(s) as they are streamed. There is no further request or setup overhead for the whole segment group. Figure 3 is a high level view of *send-stream* operation. The actual time lapse between two successive segment transmission, labeled t, will be determined by the particular flow control method used. *Get-stream* is used to retrieve a segment stream from a remote host. At the invocation of this operation, the application process specifies the name of the segment group, the host where the group resides and the access information (e.g. remote segments access path, access authorization, and estimated size) along with the other options. The operation is very similar to that of *send-stream* and will not be repeated here.

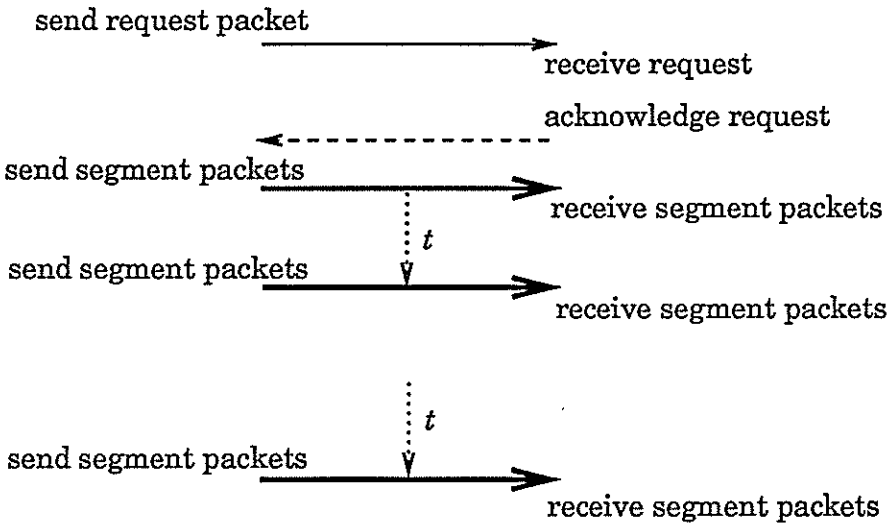

Figure 3: Send-Stream Flow

It is clear that *segment streaming* supports segment prefetching and allows overlapping between intrastage computations and interstage communications. This provides a necessary condition for efficient operation of the pipeline. With the flow control function provided as part of the IPC mechanism at the transport level, applications only need to make appropriate choices. For a typical PTV application, *get-stream* call can be used with sequential transfer mode. A segment stream is established between a pair of neighboring stages. A segment group is defined as the set of data items to be processed, with each data item corresponding to one segment. Flow will be controlled by the two-level flow control mechanism. The exact settings for error control option will depend on the particular application.

5.1. Two-Level Flow Control

In this section, we present the two-level flow control method and explain why it is appropriate for PTV.

As pointed out earlier, *segment streaming* allows more overlapping among intrastage computations and interstage communications in the pipeline. But it also makes the flow control a more critical issue than in other IPC paradigms (e.g. remote procedure call and synchronous message passing) due to the queueing of data at each stage. Especially, since each pipeline stage is usually a multiprogrammed system, change of multiprogrammed tasks will cause a corresponding change in the effective speed of the pipeline process. Moreover, underlying network connections can only provide statistical guarantees for bandwidths but not absolute rates. Without an effective flow control, buffer overflow may occur at bottleneck stages. Any overflow can in turn cause significant performance degradation to the application. Our objective for flow control is to maintain the flow rate of the pipeline at that of the bottleneck stage and avoid any overflow. To achieve this objective, the following issues need to be understood and resolved:

- What capabilities will the next generation (inter)networks have and how do they affect the flow control?
- How does the large bandwidth-delay product in high speed (inter)networks affect flow control strategy?
- How can the interferences between flow control, error control, and congestion control functions be minimized?

Next generation (inter)networks will provide connection-oriented services with statistical guarantees on average bandwidth, packet loss rate, and delay. The connection-oriented service is very appropriate for the interstage communication because the data exchange sessions between stages are usually long lived. However, users (such as transport protocols and applications) of these services need to ensure that their traffic output conforms to the specification given at the connection setup time. Therefore, a rate control mechanism is adopted at the transport level which regulates the traffic at the (inter)network interface, according to the specification agreed upon at the connection setup.

While the rate-based flow control may be sufficient for supporting applications such as remote file transfer, rate control alone cannot provide efficient flow control for distributed applications such as the PTV pipeline. The reason is that for these applications, there are frequent needs to adjust the effective flow rate in order to account for user interactions and speed variations in distributed processors. It is infeasible to achieve this adjustment by modifying only the connection rate in a large bandwidth-delay product environment. The solution then is to use another level of flow control above the rate mechanism.

A credit-based flow control method is necessary to allow continuous flow of data but avoid any overflow. Under this scheme, the receiver of a connection grants credit to the sender according to

the space available and the sender can send a data segment only if it has a matching unit of credit. It is also realized that large data granularity should be used in order to achieve high utilization of connections in high bandwidth-delay product environments. A simple explanation is that, since the delay is large with respect to bit transmission time (reciprocal of the rate: bits/sec), the data unit associated with a single control message exchange should be large enough to keep the sender (and the connection) busy before the next control action is taken. We provide this functionality by using a simple window based mechanism. The window has a variable size of W and it slides over a set of consecutive sequence numbers. By choosing the window unit to be an application data segment, overhead of mapping between these two is avoided. Since data segments of applications are expected to be on the order of megabytes, the resulting window unit is large and allows the flow control overhead to be low. The flow control effect is achieved by manipulating two window-related parameters: (1) the time to send a control message (called *permit*) to advance the window; and (2) the size of the newly advanced window.

The combination of rate and window based flow control has lead to the two-level flow control mechanism as illustrated in Figure 4.

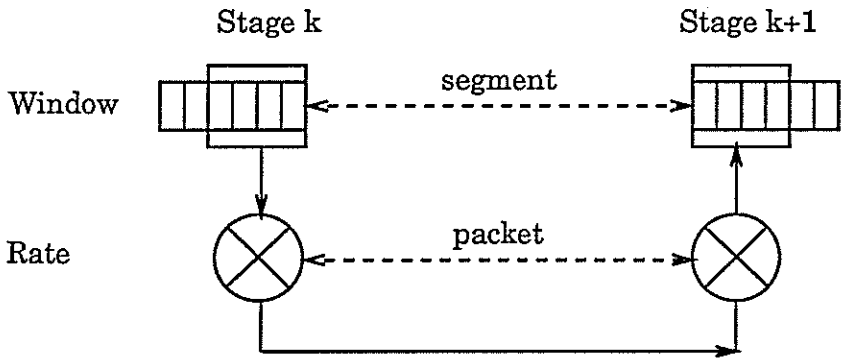

Figure 4: Structure of Two-Level Flow Control

The rate specification will be in the form of average bandwidth, peak bandwidth, and burst factor which characterizes the burstiness of the data traffic. The rate enforcement will be based on derivatives of the *leaky bucket* model [1, 3]. We explain briefly how the flow control works in this paragraph, with emphasis on the window control portion. During the setup phase of the pipeline, the desired speed of the pipeline (e.g. segments/sec measured at the last stage) is translated into specifications of rate for different stages of the pipeline. These specifications are used to request and negotiate for connections from the underlying networks. The rate control mechanism is to enforce the data flow to the rate specification. The window control is used to adjust the stage-to-stage flow within the limit of rates. The receiver starts by sending a control packet (*permit*) to the sender specifying the next expected segment number N and the credit limit W (the maximum number of segments the sender can send without further permit). The sender can send a segment only if there are available window slots $(N, N+1, \ldots, N+W-1)$. A segment is sent as a sequence of packets, each carrying a segment number, a packet number within the segment, and the most recently received credit limit. During pipeline operation, the sender keeps track of the consumption of credits and

the newly issued credits. The receiver accepts only packets with valid credit and makes decisions as to when to grant new credit and how much credit to grant.

The two-level flow control mechanism above successfully addresses the flow control needs for PTV. First of all, the rate control ensures that the sources (visualization processes in PTV) do not use more bandwidth than requested during the connection setup. The next generation networks (e.g. ATM) require this of data sources in order to manage network resources effectively and to provide guaranteed services to applications. Such a performance guarantee from the network is essential to the success of PTV. At the second level, the window control is used to adjust data flow between two neighboring stages to account for changes in the data consumption speed. Large data granularity for each window slot is used to achieve efficiency in networks with high bandwidth-delay product. Window update at the sending end is controlled by a special *permit* from the receiver, not by the ACKs as in a traditional sliding window protocol. Therefore, the flow control can be quickly activated by withholding new *permits*. The credit-based window control together with buffers distributed among pipeline stages prevents data overflow and minimizes delay to the pipeline. Since the window mechanism serves only the purpose of stage-to-stage flow control, it is simple and free of the interferences from both error and congestion conditions in the underlying network. The rate control has already been used in the design of Axon and is supported by the Axon host-network interface [23]. The proposed window control is also simple and we believe that the two-level flow control mechanism can be conveniently implemented using the hardware of the Axon host-network interface.

5.2. Application-Oriented Error Control

Error control has two aspects: detection of packet corruption, loss, and duplication; and compensation for these error conditions. Traditionally, error control either corrects all error conditions to achieve 100% reliability as in TCP, or does nothing as in UDP (User Datagram Protocol). We argue that the goal of the error control should be to satisfy the error tolerance of applications with minimum error control overhead. There are several observations that support this argument.

- With the combination of large bandwidth-delay product of very high speed internetworks and demanding PTV applications, the delay for rigorous error control becomes a significant overhead and may not be acceptable for many applications.
- Visualization applications have very diverse reliability requirements from the underlying transport protocol. A visualization application may deal with different types of data (e.g. video frames, sensory data, text, and model data) at different stages of processing. The error tolerances for these data types spread over the whole spectrum from those of model data transfer to video distribution. A typical data transfer cannot tolerate any errors and thus requires recovery from every packet loss or corruption. A typical video distribution, on the other hand, requires no error recovery (as long as the error rate is reasonably low) because previous and subsequent frames provide a context that compensates for the loss in the current frame. Also,

transmission of real time data will have more tolerance for corruption than for time delay. Therefore, it is desirable for the transport protocol to provide variable grades of error recovery to avoid unnecessary control overhead.

We address this problem by using an application-oriented error control strategy which has several important aspects to it:

Detection at receiver. The loss detection is located at the receiving end of a connection. This is consistent with our objective to allow receivers to make application-dependent retransmission decisions.

Selective retransmission. In the case of errors, a selective retransmission scheme is used. A receiver saves correctly received data packets and requests retransmissions for only the missing or corrupted packets. Therefore, application processes can get access to partially received data without much delay, and extraneous retransmissions (as in a cumulative retransmission scheme) are avoided. While the unit of retransmission is a packet, we perform loss detection and send acknowledgment on a per segment basis, thus reducing the control overhead. Retransmissions preempt the primary data stream, which is consistent with the assumption that the application expects segments in the sequential order.

Application error tolerance. Since the main objective of the application-oriented error control is to satisfy an application's requirement using minimum retransmission, an accurate characterization of application error tolerance is necessary to determine the optimal control strategy. Application error tolerance has three key components: packet loss rate, burstiness of the loss, and end-to-end delay. They are specified as the length of a "window" W over which error requirement is enforced, E the maximum number of packet losses tolerable over W, and B the maximum length of error burst tolerable. We denote the tolerance as a triplet (B,E,W) $(0 \leq B \leq E \leq W)$. By satisfying the application tolerance with minimum retransmission overhead, we automatically achieve the minimum end-to-end-delay possible for the given connection. An application is expected to specify its tolerance, and the error control mechanism uses this tolerance to do application-oriented selective retransmission.

When a televisualization pipeline is set up for an application, the specific requirement for error control is registered by the transport service. During the pipeline operation, error control mechanisms (at both the sending and receiving ends of a connection) will respond dynamically to various error conditions and correct them just to the degree required by the application. With this scheme, applications will have the flexibility to choose appropriate error control strategy to suit their purpose and to endure transport overhead for error control only if necessary.

6. SUMMARY

The importance of televisualization is becoming well recognized. In this paper we have presented an efficient IPC solution for pipelined televisualization across high-speed internetworks. Our major effort

so far has been to better understand the communication requirements of distributed visualizations and to identify trade-offs in the design of IPC mechanism, with respect to the pipelined computation model, communication errors, and significant communication latency. This paper has outlined a novel IPC mechanism called segment streaming, which can meet the demands of PTV. Our aim is to engineer the IPC solution to provide efficient interstage communication so that true pipeline concurrency can be achieved for PTV applications.

We have completed the design of the flow and error control schemes. Preliminary analysis and simulation indicate that our schemes can support segment streaming with high channel utilization and low end-to-end delay. Detailed results of these studies will be presented in future publications. We have also developed a recurrence model for performance analysis of distributed asynchronous pipelines [10]. It will be used to study the direct impact of variations in processor speed, error retransmissions, and flow control strategies on performance of PTV pipelines.

References

[1] Akhtar, Shahid, *Congestion Control in a Fast Packet Switching Network*, Wash. U. CS Dept., M.S. thesis, St. Louis, Dec. 1987.

[2] Arnould, Emmaneul, et al., "The Design of Nectar: A Network Backplane for Heterogeneous Multicomputers", *ASPLOS-III (ACM SIGOPS OS Rev.)*, Vol. 23, ACM, New York, April 1989, pp. 205–216.

[3] Bovopoulos, Andreas D., *Performance Evaluation of a Traffic Control Module for ATM Networks*, Washington University Computer Science Department, technical report WUCS-91-22, St. Louis, February 1991.

[4] Cheriton, David, "VMTP: A Transport Protocol for the Next Generation of Computer Systems", *SIGCOMM '86 Symposium: Communications Architectures and Protocols (Computer Communication Review)*, Vol. 16, No. 3, ACM, New York, 1986, pp. 406–415.

[5] Clark, David D., et al., "NETBLT: A High Throughput Transport Protocol", *SIGCOMM '87 Symposium: Frontiers in Computer Communications Technology (Computer Communication Review)*, Vol. 17, No. 5, ACM, New York, 1987, pp. 353–359.

[6] Dasgupta, Subrata, *Computer Architecture A modern synthesis, Volume 2: Advanced Topics*, John Wiley & Sons 1989.

[7] DeFanti, Thomas A., et al., "Visualization : Expanding Scientific and Engineering Research Opportunities", *Computer* , Vol. 22, No. 8, Aug. 1989, pp. 12–25.

[8] Droms, Ralph E., et al., "Report from the Joint SIGGRAPH/SIGCOMM Workshop on Graphics and Networking", to appear in *Computer Communication Review*, 1991.

[9] Dyer, D. Scott, "A Dataflow Toolkit for Visualization", *IEEE Computer Graphics and Applications*, July 1990, pp. 60–69.

[10] Gong, Fengmin, Zubin Dittia and Guru Parulkar, *A Recurrence Model for Asynchronous Pipeline Performance Analysis*, Washington University Computer Science Department, technical report WUCS-91-14, St. Louis, October 1991.

[11] Haber, Robert B., "Scientific Visualization and the Rivers Project at the Center for Supercomputing Applications", *Computer*, Vol. 22, No. 8, Aug. 1989, pp. 84–89.

[12] Kanakia, Hemant and David R. Cheriton, "The VMP Network Adaptor Board (NAB): High Performance Network Communication for Multiprocessors", *SIGCOMM'88 Symposium: Communications Architectures and Protocols (Computer Communication Review)*, Vol. 18, No. 4, ACM, New York, 1988, pp. 175–187.

[13] Kogge, Peter M., *The Architecture of Pipelined Computers*, Hemisphere Publishing Corporation 1981.

[14] Kung, H.T., "Why Systolic Architecture?", *Computer*, Vol. 15, No. 1, January 1982, pp. 37–46.

[15] Kung, S.Y., et al., "Wavefront Array Processor-Concept to Implementation", *Computer*, Vol. 20, No. 7, July 1987, pp. 18–33.

[16] Mazraani, Tony Y. and Gurudatta M. Parulkar, "Specification of a Multipoint Congram-Oriented High Performance Internet Protocol", *Proceedings of the Ninth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'90)* IEEE Computer Society, Washington D.C., June 1990, abridged from: Washington University Department of Computer Science, technical report WUCS-89-20, St. Louis, Aug. 1989.

[17] McCormick, Bruce H., et al., (eds.), "Visualization in Scientific Computing", *Computer Graphics*, Vol. 21, No. 6, November 1987.

[18] Nielson, Gregory M. and Bruce D. Shriver (eds.), *Scientific Visualization bringing data into focus*, special issue, Computer, Vol. 22, No. 8, Aug. 1989.

[19] Partridge, Craig (ed), *Proceedings of Internet Research Steering Group (IRSG) Workshop on Architectures for Very-High-Speed Networks*, January 24–26, 1990, Cambridge, Massachusetts.

[20] Parulkar, Gurudatta M., "The Next Generation of Internetworking" , *Computer Communication Review*, Vol. 20, No. 1, ACM SIGCOMM, New York, Jan. 1990, pp. 18–43, also: Washington University Department of Computer Science, technical report WUCS-89-19, St Louis, May 1989.

[21] Rosenblum, Lawrence J., "Visualization of Experimental Data at the Naval Research Laboratory", *Computer*, Vol. 22, No. 8, Aug. 1989, pp. 95–101.

[22] Sterbenz, James P. G., Gurudatta M. Parulkar, "Axon: A High Speed Communication Architecture for Distributed Applications", *Proceedings of the Ninth Annual Joint Conference of the IEEE Computer and Communication Societies (INFOCOM'90)* IEEE Computer Society, Washington D.C., June 1990, pp. 484–492, also: Washington University Computer Science Department, technical report WUCS-89-36, St. Louis, September 1989.

[23] Sterbenz, James P. G., Gurudatta M. Parulkar, *Axon: Application-oriented Lightweight Transport Protocol Design*, Washington University Computer Science Department, technical report WUCS-89-14, St. Louis, September 1989.

[24] Turner, Johnathan S., "Design of a Broadcast Packet Switching Network", *IEEE Transactions on Communication*, Vol.36, No. 6, New York, June 1988, pp. 734–743.

[25] Upson, Craig, et al., "The Application Visualization System: A Computational Environment for Scientific Visualization", *Computer Graphics and Applications*, Vol. Vol. 9, No. 4, July 1989, pp. 30–42.

[26] Winkler, Karl-Heinz A., et al., "On the Characteristics of a Numerical Fluid Dynamics Simulator", in *Supercomputers Algorithms, Architectures, and Scientific Computation*, (eds.) F.A. Matsen and T. Tajima, University of Texas Press, Austin 1986, pp. 416–429.

[27] Zhang, Lixia, *A New Architecture for Packet Switching Network Protocols*, MIT Laboratory for Computer Science, Ph.D. dissertation, July 1989.