PAPER

Evaluation of High Speed Multimedia Communication Architecture in ATM Networks

Hiroshi ESAKI†, Yoshiyuki TSUDA † *Members and* Kumiko KANAI † *Nonember*

SUMMARY This paper discusses and evaluate a high speed multimedia communications in ATM networks. Communication platforms below AAL layer in ATM networks can provide fairly high speed communication channel for ATM-UNI. On the other hand, the conventional data processing architecture in the work stations can not provide sufficient throughput compared to the throughput of ATM networks at this time. Also, regarding error-free peer-to-peer data communications, conventional transport protocol, e.g., TCP, can not provide small latent communication, when the product of communication bandwidth and propagation delay is large. This paper focuses on protocol architecture, especially on error control policy at SSCS in AAL layer, and on performance evaluation on high speed multimedia communication based on the current data processing architecture.
key words: ATM, ATM-LAN, TCP, UDP, X-window, FEC, SSCS

1. Introduction

ATM, Asynchronous Transfer Mode, network will be the high speed communication platform both for B-ISDN and for high speed LAN/MAN [1],[2]. The interface speed to be provided by ATM networks to the user interface (UNI; User Network Interface) will be from dozens Mbps to few Gbps. ATM network handles any user information through a unique fixed sized packet, called cell. It is said that ATM network can provide flexible communication channel for the wide variety of communication requests, e.g., real-time video communication. ATM network is the connection oriented network, which provides a virtual circuit (VCC; Virtual Channel Connection) between two SAPs (Service Access Point). And, the ATM network will provide a QOS (Quality Of Service) to the guaranteed QOS connection [2]. Therefore, it is said that ATM network is the solution of multimedia communication platform.

Regarding the computer communication platform, both the data processing speed of computers and the speed of communication medium are getting faster and faster [3]. Many distributed applications, e.g., client-server applications and object oriented applications, executed in the current computer systems require high speed data communications among the processes belonging to the applications. Since the processes are not generally executed within the same computer, the distributed applications require high speed communication channels among the regarded computers. For the current computer system, the conventional communication platform (e.g., Ethernet) has been becoming the bottle-neck of system. One purpose of the paper is clarifying the bottle-neck of current and future distributed computer communication system. As evaluated in this paper, using the current computer architecture, we can not obtain a high speed data communication, even though ATM network can provide fairly high speed communication channel to applications.

When the communication speed (bandwidth) is large, the conventional error control (e.g., retransmission in TCP) can not provide a fast data-unit delivery, especially for wide area network (WAN). As shown in Refs.[4],[5], when the product of bandwidth and propagation delay is large, we need a forward error correction (FEC) protocol to get a high speed and small latent data-unit delivery.

This paper evaluates the performance of peer-to-peer data communications both for error-free and for allowing error. Section 2 discusses about error-free peer-to-peer data communications in high speed WAN, and Sect.3 discusses about peer-to-peer data communication that need not (exact) error-free characteristics, i.e., UDP and video stream. The rest of this paper is as followed. In Sect.2, the performance of FEC based error control at SSCS (Service Specific Convergence Sub-layer) of AAL layer in high speed wide area ATM networks is evaluated. And, in Sect.3, the end-to-end communication speed for user applications, between the (UDP) ports and video communication topping on X-window system, is evaluated and discussed to clarify the system bottle-neck, from the view point of processing overhead. As discussed in Sect.3.3, we can obtain sufficient throughput for user application level, when we can reduce the bottle-neck processing overhead. Sect.4 gives a brief summary.

Manuscript received April 18, 1994.
Manuscript revised June 21, 1994.
†The authors are with R&D Center, TOSHIBA CORPORATION, Kawasaki-shi, 210 Japan.

2. Error-Free Data Communications in ATM Networks

2.1 Error Control Capability in ATM Networks

The purpose of error control in data networks are to provide an error free data-unit delivery between a pair of SAPs. Error detection and error correction are the key functionalities of error control. Every layer does not necessarily equip both functionalities. TCP has both functions, however, ATM network fundamentally have only error detection capability.

The performance matrix of error control should be throughput and latency of data delivery. The conventional error control schemes, e.g., go-back-N, can not yield good performance for wide area high speed network [4],[5]. In this section, we propose and evaluate the error control schemes based on FEC scheme in SSCS of AAL Type 5 to provide high throughput and small latent data delivery. As well as throughput, the latency of end-to-end data delivery is very important performance matrix for error control. The conventional error correction algorithm (e.g., go-back-N) could yield enough performance in the network where the product between the bandwidth and the propagation delay is not large. However, in the network where the product between the bandwidth and the propagation delay is large or the data error/loss ratio is large, the conventional error correction algorithm can not yield enough performance of latency. Therefore, the performance of the proposed SSCS protocol is evaluated focusing on the end-to-end latency of data-unit delivery.

Table 1 shows error detection and error correction capability below CPCS sub-layer in ATM networks.

Table 1 Error detection/correction in ATM Network

		Error Detection		Error Correction	
		Payload(SDU)	Header	Payload(SDU)	Header
	PHY	No	Yes(HEC)	No	1bit(ATM)
	ATM	No	Yes(HEC)	No	1bit
S	Type1	No	Yes(SNP)	No	1bit(SN)
A	Type3/4	Yes(CRC-10)	Yes(CRC-10)	No	No
R	Type5	No	No	No	No
C	Type1	No(optional)	No(optional)	No(optional)	No(optional)
P C	Type3/4	Yes(LI for lost)	No	No	No
S	Type5	Yes(CRC-32/LI)	Yes(CRC-32)	No	No

As shown, error(s) both header field and for payload is detected [1],[6]. On the other hand, error for payload (CPCS-SDU) is not basically corrected, excepting the option of AAL Type 1. SSCS of AAL could have error correction capability for assured mode operation [6]. However, at this time, neither ITU-T nor ATM Forum has been discussed about detailed protocol specification of error correction functionality in SSCS.

Error control is necessary for data delivery. On the other hand, for voice or video communication, the error control may not be necessary. Regarding data delivery service, AAL type 5 will be used for many cases. Since AAL type 5 does not have error correction capability, the dropped or collapsed data-unit (CPCS-SDU) will be retransmitted by the request of upper layer protocol.

ATM network discussed in ATM Forum will provide two types of connection. One is the guaranteed QOS connection, which is subject to CAC (Connection Admission Control). The other is the best effort connection, which may not be subject to CAC. This means that best effort connection could be always established (i.e., no rejection of connection set up request), and each terminal could submit cell to the network through the best effort connection with their own peak bit rate at any time. Since, there may be no CAC policy for the best effort connections, the congestion may occur frequently. And, then, the service quality (QOS) of best effort connection will be degraded. This is serious for data communication, which requires error free data delivery, through best effort connections.

2.2 Performance Evaluation of FEC Method

2.2.1 SSCS Protocol for FEC Control

Figure 1 shows the proposed protocol structure. AAL type is AAL Type 5. Since, the proposal is about SSCS of AAL type 5, only the processing sequence of SSCS is described blow.

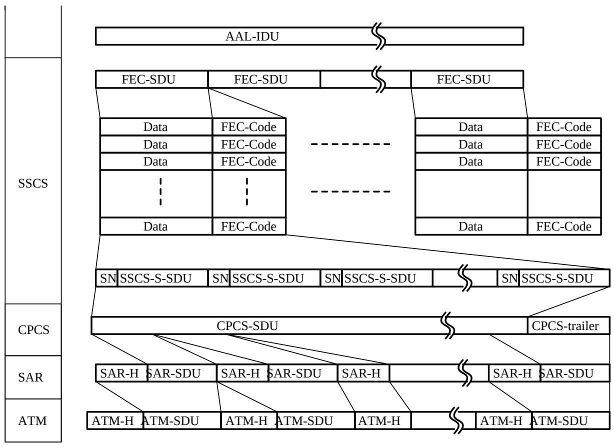

Fig 1 Protocol structure applying FEC-SSCS

SSCS-SDU (AAL-IDU) may be equivalent to IP segment, whose default MTU size is 9180bytes in ATM networks [8]. SSCS-SDU is divided into multiple FEC-SDUs. Each FEC-SDU is stored in the FEC frame, and FEC code is calculated to be attached, e.g., [5],[6]. When the data in the FEC frame is read out, certain interleaving algorithm is applied [10]. When the data (SSCS-S-SDU) is read out from FEC frame, SN (Sequence Number) is inserted in order to identify which SSCS-S-SDU(s) is(are) dropped at the SSCS entity of destination terminal. Also, some error detection field (e.g., checksum) would be attached to SSCS-S-SDU.

When the number of errored SSCS-S-PDUs is larger than the capability of applied FEC, the whole of FEC frame is retransmitted. It is possible to apply the selective repeat for only the corresponding SSCS-S-SDU(s) to recover the FEC frame, however, in this simulation, selective repeat for SSCS-S-SDU level is not applied.

Regarding the error control policy for FEC-SDU level, the selective repeat is applied. This means that only the FEC-SDU(s), that could not be correctly decoded by FEC algorithm, is(are) retransmitted until the FEC-SDU(s) is(are) decoded correctly by the destination SSCS entity.

2.2.2 Performance Evaluation

Referenced error control algorithm in SSCS is the following two. Here, it is assumed that the data-unit (denoted as control data-unit) for retransmission against the error is the same size as FEC-SDU. This means that, for the referenced control algorithm, AAL-IDU (maybe equivalent to IP segment) is divided into multiple control data-units.

- Go-Back-N
  Go-back-N algorithm is applied for error recovery of control data-unit(s). Regarding the control data-unit, only error detection capability is provided by CPCS.
- Selective Repeat
  Selective repeat algorithm is applied for error recovery of control data-unit(s). Again, regarding the control data-unit, only error detection capability is provided by CPCS.

Figure 2 shows the end-to-end latency of data-unit (SSCS-SDU) delivery versus the cell loss ratio in the ATM network. Here, the performance of proposed method is donated as "FEC". The evaluated latency is from when the AAL-IDU is submitted to the SSCS buffer of source terminal to when the error-free AAL-IDU is come out from the SSCS buffer of destination terminal. This means that the latency before submitting to SSCS buffer (queuing before SSCS buffer) is not included. Here, the latency is represented by the *unit-time*. One *unit-time* is defined as the time one cell is transmitted by the ATM link. For example, when ATM link is STS-3c (156Mbps), one *unit-time* corresponds to about $2.7\mu sec$. End-to-end propagation delay is 75 *unit-time* ($\simeq 202\mu sec$). Here, 75 *unit-time* approximately corresponds to the propagation delay of 55.5km.

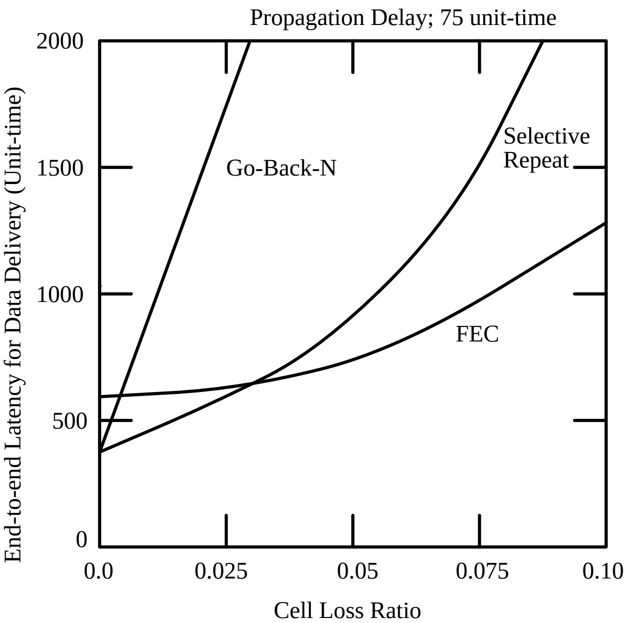

Fig 2 End-to-end latency versus cell loss ratio

Source terminal generates the data continuously, and has a infinite buffer to transfer the data. The data transmission speed (peak speed of VCC used by the source terminal) is 156Mbps. The size of IP segment (AAL-IDU) corresponds to 256cells ($\simeq 12Kbytes$), and the size of FEC-Frame (FEC-SDU and the control data-unit) corresponds to 16 cells ($\simeq 756bytes$). 12Kbytes of IP segment may not so large in ATM network. As specified in Ref.[6], the maximum size of AAL-IDU is 64Kbytes, and as discussed in Ref.[8], the default MTU size of IP segment in ATM network is 9180bytes ($\simeq 190cells$).

The capability of FEC is that only one cell drop/error is allowed within FEC frame, and recover it without any data retransmission. When more than one cell is dropped/errored, the whole of FEC frame is retransmitted. The latency of coding and decoding for FEC is assumed as 16 *unit-times*.

In the simulation, it is assumed that the cell is dropped in ATM network at random, and no bit error is assumed.

Figure 3 shows the end-to-end latency of AAL-IDU (IP segment) delivery versus the end-to-end propagation delay in the ATM network. The simulation would correspond to in congestion status. Generally, in ATM networks, the cell loss ratio during congestion status is so large, which would be sometimes few % [5]. In the simulation, the cell loss ratio in the ATM network is 2%. Here, the horizontal axis represents end-to-end propagation delay between end terminals. For example, 100 *unit-time* ($\simeq 272\mu sec$) approximately corresponds to the propagation delay of 74km. Here, similar results can be obtained, even when the cell loss ratio is much less value, if the propagation delay is large.

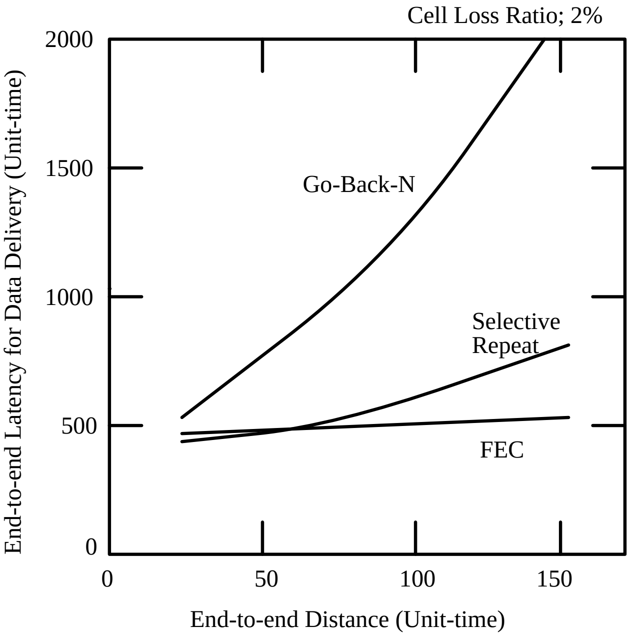

Fig 3 End-to-end latency versus propagation delay

As shown, either when cell loss ratio increases or when the end-to-end propagation delay increases, the end-to-end latency of IP segment delivery using the proposed method is smaller than the referenced control methods. These results indicate that, when the product between the cell loss ratio and the propagation delay is large, the proposed error control scheme can yield smaller latency of IP segment delivery, compared to the referenced scheme can.

2.2.3 Discussion

In the above evaluations, the performance between SSCS entities is discussed, but the performance between end-to-end application interface points (e.g., TCP ports) has not been discussed. Therefore, regarding the above evaluation, it is equivalent as that no flow control policy is applied by the upper layer. In the actual system, some flow control policy is applied in the upper layer. One is between the layer entities (e.g., window control between TCP entities). The other is between upper layer, let say TCP/IP, and SSCS. The later flow control is generally ACK/NACK based control. TCP/IP can write the IP segment to the SSCS buffer, when there is no NACK signal. However, when there is NACK signal, TCP/IP can not write IP segment to the SSCS buffer.

In the actual system, SSCS is equipped in the interface boad connecting with the network, and the upper layer entity (e.g., TCP/IP) is equipped in the operating system. This means that ACK/NACK based flow control will be performed between operating system and interface buffer, i.e., between TCP/IP and SSCS. We have not discussed about the flow control policy between TCP entities, in this paper. The performance evaluation when TCP/IP is operated topping on the proposed SSCS should be for further study.

The goodput is defined in ATM Forum to evaluate the performance of service using best effort connection. Goodput could be equivalent to actual throughput between a given SAPs. Regarding TCP/IP, for example, goodput could be the throughput between a given TCP ports pair. Below, for the convenience, TCP/IP is assumed for upper layer protocol.

As shown in Table 1, ATM network does not have error correction capability. Therefore, when a CPCS-SDU (may be equal to AAL-IDU in usual case due to null SSCS) is errored, AAL-IDU (may be equal to IP segment) will be retransmitted. There are two problems for TCP/IP control policy. One is window control to regulate a IP segment flow, and the other is go-back-N retransmission control for error correction. These two control policy will cause serious congestion in the network, and goodput will be degraded. IP segments in the window (dynamic or static) will be submitted to the network continuously, even when IP segment(s) in the window is(are) dropped or errored. When IP segment(s) is(are) dropped or errored, the IP segment(s) after the first dropped or errored IP segment in the window will be retransmitted. This means that, even when many IP segments are correctly delivered to the destination terminal, IP segments after the dropped or errored IP segment must be transmitted again (go-back-N policy). Moreover, the window size is modified after the completion of data delivery within the current window.

There is another problem regarding goodput. Since, IP segment (AAL-IDU) is composed of multiple cells, IP segment is errored even when only one cell belonging to the IP segment is dropped or errored. By Ref.[8], the default MTU size of IP segment on ATM network would be 9180bytes that corresponds to 190cells.

Through the above consideration, it is easy to understand that TCP/IP connection topping on AAL type 5 (with null SSCS) could not yield enough goodput. This is because that the cell loss ratio will be increase rapidly, due to the serious loop of cell retransmission. Only when all of cells belonging to an IP segment are correctly delivered to the destination terminal, and when all of IP segments in the TCP window before they are correctly delivered to the destination terminal, the retransmission of IP segment (cells) will not occur.

In order to improve goodput, many control schemes are proposed, e.g.,[7], in ATM Forum. The proposed schemes in ATM Forum require the modification of transport protocol, generally. However, in the actual computer, the transport protocol, e.g., TCP, is built-in the operating system. This means that it is generally difficult to modify the algorithm of transport protocol. Therefore, the algorithm to improve goodput and end-to-end latency of data delivery should be implemented in SSCS, rather than in higher layer that is generally fulfilled in operating system.

Since the proposed control scheme could reduce the number of retransmitted cells totally, the goodput of the ATM system would be improved, as well as the end-to-end data-unit delivery would be improved.

3. Video and Data Communications in ATM Networks

In this section, the end-to-end communication throughput for user application level, between the (UDP) ports and video communication topping on X-window system, is evaluated and discussed in order to clarify the system bottle-neck, from the view point of processing overhead. The discussed data communications do not need (exact) error-free characteristics. As discussed in Sect.3.3, we can obtain sufficient throughput for user application level, when we can reduce the clarified bottle-necked processing overhead.

It is assumed that the operating system used in the evaluated system is the usual UNIX model [12]. As discussed in Ref.[13], in the high speed communication platform, the data transferring speed of operating system is the bottle-neck of the end-to-end communication throughput. In the following evaluations, we clarify the system bottle-neck of end-to-end communication throughput for the current and the future computers. We evaluate the communication between UDP ports and the video communication using X-window system. Here, the reason why we assumed UDP, instead of TCP, is the evaluation is focusing on the end-to-end communication within LAN (not for inter-LAN communication). Also, we have focused on real-time video communication, which does not need (exact) error-free characteristics. Actually, both real-time voice (vat) and real-time video (nv) discussed in IETF do use UDP, rather than TCP [16].

The details of system assumption is described in appendices. It is assumed that both the bandwidth of system bus and the access speed of memory (DRAM) will increase, in accordance with the increase of CPU's operation speed. Three models are evaluated; 25MHz CPU model (for current or few years ago), 100MHz CPU model (for about 1994 or 1995) and 500Mhz CPU model (for about 2000) [3]. CPU with 25MHz yields approximately few decades of MIPS, CPU with 100MHz would yield 100MIPS, and CPU with 500MHz would yield 500-1000MIPS.

3.1 Communication between UDP Ports

3.1.1 Evaluation Model

The datagram transmission from a sender host (computer) to a destination host is as follows.

1. When the application process in the sender host sends a datagram to the destination host, the application process calls a system call. A trap in the system call changes the model of application process to kernel mode, and the process copys the data from a buffer owned by the application to the system buffer.
2. Kernel mode process (in Operating System) performs UDP/IP to transmit it to the destination host.
3. After the processing of UDP/IP, the datagram is copied to the interface buffer through DMA.
4. The interface buffer access the communication medium to transmit the datagram toward the destination host. And, the interface buffer in the destination host receives the datagram.
5. When the interface buffer in the destination host receives the datagram, the operating system is interrupted to deal with the received datagram.
6. Operating system copies the received datagram from the interface buffer to the kernel memory space of operating system through DMA.
7. Operating system performs UDP/IP for the received datagram.
8. After the completion of UDP/IP processing, the received datagram is copied from the kernel memory space to the user memory space.

Here, some techniques were proposed to reduce the copy overhead between the kernel memory space and the user memory space [14],[15]. However, in this paper, the usual data processing operation in the UNIX system mentioned above is assumed.

Also, it is assumed that there is no other process excepting the evaluated process. This means that no context switching overhead is not considered in the evaluation; ideal model.

CPU examines the cache memory to access the data, first. When the cache is missed, the data is loaded from the main memory. Therefore, the path that transmitted data passes are as followed.

1. User memory space in main memory of sender host
2. CPU through cache memory of sender host

3. Kernel memory in main memory of sender host through CPU
4. CPU through cache memory of sender host to perform transport protocol
5. Kernel memory in main memory of sender host
6. Interface buffer of sender host
7. Interface buffer of destination host
8. Kernel memory in main memory of destination host
9. Cache memory of destination host to perform transport protocol
10. Kernel memory in main memory of destination host
11. Cache memory of destination host
12. User memory space in main memory of destination host

The detailed evaluated models are shown in Appendix A.

3.1.2 Performance Evaluation

Table 2 shows the end-to-end communication speed of UDP ports, copy overhead, and required cycles per one word to transmit the data.

As shown, the copy overhead is about 60% - 70% of data processing. Due to the large copy overhead of data processing, we can not yield high speed data transmission. For instances, even when we use 100MHz CPU (32 bits oriented), the end-to-end throughput will be about 52Mbps. And, we can recognize that, even in the future computer that uses synchronous DRAM for main memory, the copy overhead to process the data will be about 70%.

Table 2. End-to-end communication speed of UDP ports

Model	Throughput	Operation (cycles/word)	Copy overhead
25MHz CPU	13.1 Mbps	61 cycles	59%
100MHz CPU	52.4 Mbps	61 cycles	59%
500MHz CPU	355.6 Mbps	90 cycles	72%

3.2 Video Communication Using X-Window System

3.2.1 Evaluation Model

The data (video image) transmission from a sender host (computer) to a destination host is as follows. Transport protocol used in the evaluation system is UDP, since the communication is assumed as within a LAN.

1. Video image is transferred from video camera to main memory
2. Video image is transformed to image data for X-window system
3. Transformed data is copied to interface buffer, after the processing of UDP/IP
4. Interface buffer accesses the communication medium to transmit the data toward the destination host. And, the interface buffer in the destination host receives the data.
5. Received data is copied from the interface buffer to the kernel memory space, and UDP/IP processing is performed.
6. After the completion of UDP/IP processing, the received data is copied from kernel memory space through user memory space to the frame buffer.

Again, it is assumed that there is no other process except the evaluated process. Therefore, no context switching overhead is not considered in the evaluation. Here, there is no copy operation between kernel space and user space. This is because video camera and frame buffer are dealt with by the kernel as external devices. In other words, they can be seen as the same level of devices as user space.

The detailed evaluation models are shown in Appendix B.

3.2.2 Performance Evaluation

Tables 3 and 4 show the end-to-end video communication speed using X-window system, copy overhead, and the number of video windows to be able to be provided. Table 3 is for bi-directional video communication, and table 4 is for uni-directional video communication. Here, three types of video window is assumed; (1) Small window (160× 120 pixel), (2) NTSC broadcast quality (360× 480 pixel), (3) NTSC studio quality (720×480 pixel).

As shown, the copy overhead is more than 50% of data processing. Due to the large copy overhead of data processing, we can not yield high speed data transmission. For instances, even when we use 100Mhz CPU (32 bit oriented), only one or two small window(s) could be handled. And, we can recognize that, even in the future computer that uses synchronous DRAM for main memory, only one NTSC studio signal can be handled.

Table 3. Bi-directional video communication by x-window

Model	Throughput	Copy overhead	Small Window	NTSC (broadcast)	NTSC (studio)
25MHz CPU	1.99 Mbps	55.4%	0.43	0.048	0.024
100MHz CPU	8.06 Mbps	55,4%	1.75	0.194	0.097
500MHz CPU	70.4 Mbps	67.1%	15.30	1.696	0.848

+ Small window; ; 4.6Mbps(160x120x8bitsx30frame)
+ NTSC(broadcst) ; 41.5Mbps(360x480x8bitsx30frame)
+ NTSC(studio) ; 83.0Mbps(720x480x8bitsx30frame)

Table 4. Uni-directional video communication by x-window

Model	Throughput	Copy overhead	Small Window	NTSC (broadcast)	NTSC (studio)
25MHz CPU	2.38 Mbps	56.8%	0.52	0.057	0.0287
100MHz CPU	9.52 Mbps	56.8%	2.07	0.229	0.115
500MHz CPU	86.6 Mbps	69.5%	18.82	2.087	1.043

+ Small window; ; 4.6Mbps(160x120x8bitsx30frame)
+ NTSC(broadcst) ; 41.5Mbps(360x480x8bitsx30frame)
+ NTSC(studio) ; 83.0Mbps(720x480x8bitsx30frame)

3.3 Discussion

In this subsection, the methods to increase the throughput between the application interface points (e.g., UDP ports) are discussed briefly.

3.3.1 Reducing Copy Overhead

As shown, the copy overhead in the operating system is more than 50% of total data processing in the evaluated applications. The most effective method to increase the throughput is reducing the copy overhead.

There are some methods and some architectures to reduce the copy overhead in the operating system. For example, Ref.[4] proposed the technique that emulates the copy operation through pointer operation, in the conventional operating UNIX system. E.g., in Ref.[15], a novel operating system, which essentially does not require the data copy operation, has beed proposed. By using this novel operating system, we could obtain greater throughput than the conventional operating system does.

3.3.2 Hardware Processing

Regarding video communication, some processing could be fulfilled by external hardware, rather than by software (operating system). There would be two approaches. One is using an external hardware transforming from NTSC camera signal to X-window's image signal, the other is using MPEG chip. Former approach is transmitting bit-mapped video image, and the later is transmitting compressed video image.

- Bit-Mapped Video Transmission
  In the evaluation of Sect.3.2, the signal transformation from NTSC camera signal to X-window's image signal could be fulfilled by external hardware.

Tables 5 and 6 show the modified end-to-end video communication speed using X-window system, when the signal transformation from NTSC camera signal to X-window's image signal could be fulfilled by external hardware. Again, the number of video windows to be able to provided is shown. Table 5 is for bi-directional video communication, and Table 6 is for uni-directional video communication. Once again, three types of video window is assumed; (1) Small window ($160\times$ 120 pixel), (2) NTSC broadcast quality ($360\times$ 480 pixel), (3) NTSC studio quality ($720\times$ 480 pixel).

Here, third column shows the processing overhead of transformation from NTSC signal to X-window signal. And, it is assumed that processing speed of data transformation from NTSC signal to X-window signal is sufficiently fast and that it could be ignored.

As shown, the processing overhead of transformation from NTSC signal to X-window signal is more than 70% of data processing. Therefore, using the external hardware transforming from NTSC signal to X-window signal, the communication speed increases. For instances, when we use 100MHz CPU (32 bit oriented), four or five of small windows could be handled. And, when we use the future computer (500MHz CPU), some NTSC studio quality video windows can be handled.

Table 5. Modified bi-directional video communication

Model	Software Processing Model		Modified Model (Hardware processing NTSC to X-win)			
	Throughput	Overhead (NTSC to X-win)	Throughput	Small window	NTSC (broadcast)	NTSC (studio)
25MHz CPU	1.99 Mbps	72.7%	4.57 Mbps	0.99	0.110	0.055
100MHz CPU	8.06 Mbps	72.7%	18.3 Mbps	3.98	0.441	0.221
500MHz CPU	70.4 Mbps	71.7%	227.0 Mbps	49.35	5.470	2.735

+ Small window; ; 4.6Mbps(160x120x8bitsx30frame)
+ NTSC(broadcst) ; 41.5Mbps(360x480x8bitsx30frame)
+ NTSC(studio) ; 83.0Mbps(720x480x8bitsx30frame)

Table 6. Modified uni-directional video communication

Model	Software Processing Model		Modified Model (Hardware processing NTSC to X-win)			
	Throughput	Overhead (NTSC to X-win)	Throughput	Small window	NTSC (broadcast)	NTSC (studio)
25MHz CPU	2.38 Mbps	87.2%	7.4 Mbps	1.609	0.178	0.089
100MHz CPU	9.52 Mbps	87.2%	29.6 Mbps	6.435	0.713	0.356
500MHz CPU	86.6 Mbps	88.2%	377.4 Mbps	82.043	9.094	4.547

+ Small window; ; 4.6Mbps(160x120x8bitsx30frame)
+ NTSC(broadcst) ; 41.5Mbps(360x480x8bitsx30frame)
+ NTSC(studio) ; 83.0Mbps(720x480x8bitsx30frame)

- Video Transmission Using MPEG 1/2
  MPEG 1 works at 1.5Mbps (CBR) for output of codec. MPEG 2 may work about 10Mbps (both CBR and VBR). In the following evaluation, it is assumed that output speed of MPEG2 codec
  is assumed that output speed of MPEG2 codec

is 10Mbps with CBR. Using the results of the throughput of UDP evaluated in Sect.3.1, we could evaluate how many video windows could be handled by workstation equipping many MPEG chips. For the convenience, the context switching overhead in the OS because of handling multiple video streams is ignored. The detailed evaluation, including such a overhead, should be for further study item.

Table 7 shows how many video windows could be dealt with simultaneously by workstation.

As shown, when we use MPEG chip for real-time video communication, we could provide some real-time video windows simultaneously. For example, for 100MHz CPU model, we could deal with about 2 real-time (bi-directional) windows.

Table 7. Video communication speed using MPEG1/2

Model	Throughput	MPEG1 (1.5Mbps)		MPEG2 (10Mbps)	
		uni-direction	bi-direction	uni-direction	bi-direction
25MHz CPU	13.1 Mbps	8.7	4.3	1.3	0.6
100MHz CPU	52.4 Mbps	34.9	17.5	5.2	2.6
500MHz CPU	449.1 Mbps	299.4	149.7	44.5	22.4

4. Conclusion

This paper discussed about a high speed multimedia communications in ATM networks. This paper focuses on the protocol architecture, especially on the error control policy at SSCS at AAL layer, and on the performance evaluation of high speed multimedia communication based on the current data processing architecture.

Regarding SSCS in AAL layer, we has proposed FEC based protocol and has evaluated the performance. There are many further study items, however, we could show that the proposed protocol works well in the high speed wide area network. The proposed SSCS may be appropriate for best effort service, as well as the application that requires small latency for data delivery using the guaranteed QOS connection.

Regarding the discussion of end-to-end communication speed, we has clarified the system bottle-neck. The system bottle-neck was copy overhead of operating system. By reducing such a processing overhead, we could obtain high speed end-to-end communication as user application level. The detailed performance evaluation, when a new data processing scheme or a new operating system (e.g., [15]) is applied, is for further study.

Acknowledgement

The authors express special thanks to Mr.Mitsuo Saito and Mr.Shigehiro Asano, who work for Toshiba Corporation, for their beneficial comments.

References

[1] ITU-T Recommendation I.150, "B-ISDN Asynchronous Transfer," 1990
[2] ATM Forum, "ATM User-Network Interface Specification," Aug., 1993.
[3] Kodama, T., "ATM Private Network," *Proc. of IEICE SSE/IN Workshop*, pp.33-40, Mar., 1993.
[4] Kikuchi, Y., Inai, H., Murata, M. and Miyahara, H., "Comparative Evaluation for Error Recovery Schemes for High Bandwidth-Delay Product Networks," *IEICE Technical Report,* **SSE92**-199, Mar., 1993.
[5] McAuley, A.J., "Reliable Broadband Communication Using a Burst Erasure Correcting Code," *ACM SIGCOMM90*, pp.297-306, 1990.
[6] ITU-T I.363, "B-ISDN ATM Adaptation Layer (AAL) Specification," Jul., 1993.
[7] Ikeda, C., Suzuki, H., "Congestion Control Schemes for Best Effort Services in ATM-LAN," *IEICE Technical Report*, **IN93**-109, Jan., 1994.
[8] Atkinson, R., "Default IP MTU for use over ATM AAL5," *Internet-Draft*, draft-ietf-atm-mtu-06.txt, Jan., 1994.
[9] Comer, D.E., "Internetworking with TCP/IP," Prentice Hall, 1991.
[10] TU-T SG15 Rapporteur's Group on part of Q.2/15, (Documents AVC-459, AVC-521, AVC-569), 1993.
[11] Doeringer, W., Dykeman, D., Kaiserwerth, M., Meister, B., Rudin, H. and Williamson, R., "A Survey of Light Weight Transport Protocols for High Speed Networks," *IEEE Trans. on Communications*, vol.38, no.10, pp.2025-2039, Nov., 1990.
[12] Tanembaum, A.S., *Computer Networks*, Prentice Hall, 1988.
[13] Rudin, H., "Protocols for High Speed Networks - Transport Level Protocols," *Tutorial of SIGCOMM90*, Sep., 1990.
[14] Kimata, H. and Nishida, T., "Implementation for High Speed Protocols Processing in Workstations," *Proc. of the 1993 IEICE Fall Conference*, B-634, Sep., 1993.
[15] Okamoto, T., Segawa, H., Shin, S.H., Nozue, H., Maeda, K. and Saito, M., "A Micro Kernel Architecture for Next Generation Processors," *Proc. of the USENIX workshop*, on Micro-Kernels and Other Kernel Architectures, Apr., 1992.
[16] Schulzrinne, H. and Casner, S., "RTP: A Transport Protocol for Real-Time Applications," *Internet-Draft*, draft-ietf-avt-rtp-04.txt, Oct., 1993.

Appendix A: Evaluation of UDP/IP Connection

Appendix A:1 25MHz CPU model

The following model is assumed.

- CPU clock; 25MHz (32 bits)
- Bus clock; 25MHz (32 bits)
- Cache like; 16 bytes (=4 words)
- Cache miss; 20 cycles (for 4 words)
  - Load from main memory; 20 cycles (for 4 words)
  - Store from cache to main memory; 20 cycles

- UDP/IP processing; 25 cycles/word
- DMA between main memory and I/O buffer; 12 cycles (for 4 words)

The following is the analysis for UDP data segment processing.

1. User memory → cache memory; 11 cycles/word
   1+20/4+20/4=11 cycles/word
   Since cache line is 4 words (16bytes), cache will be missed once for 4 words copying from user memory (in main memory) to cache memory. Store operation to cache is 1 cycle, cache-miss routine is 20/4 cycle/word, and load from user memory is 20/4 cycles/word.
2. Cache memory → kernel memory; 11 cycles/word
   1+20/4+20/4=11 cycles/word
   Cache will be missed once for 4 words copying from cache memory to kernel memory (in main memory). Store operation to main memory (kernel space) is 1 cycle cache-miss routine is 20/4 cycles/word, and store from cache memory to main memory (kernel space) is 20/4 cycles/word.
3. Kernel memory → cache memory; 11 cycles/word
   1+20/4+20/4=11 cycles/word
4. UDP/IP processing; 25 cycles/word
5. Kernel memory → interface buffer; 3 cycles/word
   12/4=3 cycles/word

Total cycles to transmit the data is 61 cycles/word. Since one word is 32 bits and one cycle is 40 nsec (=1/25MHz), the throughput is 13.1Mbps.
$1/[(61/32)\times 40\times 10^{-9}] = 13.1\times 10^6$
Here, the copy overhead is 36 (=61-25) cycles, which is about 59%.

Appendix A:2 100MHz CPU Model

It is assumed that, as well as increase of system clock, the operation speed of every module increases four times as fast as the 25MHz model.

1. User memory → cache memory; 11 cycles/word
2. Cache memory → kernel memory; 11 cycles/word
3. Kernel memory → cache memory; 11 cycles/word
4. UDP/IP processing; 25 cycles/word
5. Kernel memory → interface buffer; 3 cycles/word

Total cycles to transmit the data is 61 cycles/word. Since one word is 32 bits and one cycle is 10 nsec (=1/100MHz), the throughput is 52.4Mbps.
$1/[(61/32)\times 10\times 10^{-9}] = 52.4\times 10^6$
Here, the copy overhead is 36 (=61-25) cycles, which is about 59%.

Appendix A:3 500MHz CPU Model

The following 64bits oriented system model is assumed. And, it is assumed that synchronous DRAM is used for main memory.

- CPU clock; 500MHz (64bits)
- Bus clock; 125MHz (128 bits)
- Cache line; 64 bytes (=8 words)
- Cache miss; 50 cycles (for 8 words)
- Access time for main memory; 30 cycles
- Load from main memory; 4 cycles (for 1 word)
- Store to cache; 20 cycles (for 8 words)
- Load from cache; 20 cycles (for 8 words)
- Store from cache to main memory; 4 cycles (for 1 word)
- UDP/IP processing; 25 cycles/word
- DMA between main memory and I/O buffer; 200 cycles (for 128 bytes)

The following is the analysis for UDP data segment processing.

1. User memory → cache memory; 17.5 cycles/word
   $1 + 50/8 + (30 + 4\times 8)/8 + 20/8 = 17.5 cycles/word$
   Since cache line is 8 words (64bytes), cache will be missed once for 8 words copying from user memory (in main memory) to cache memory. Store operation to cache is 1 cycle, cache-miss routine is $(30 + 4\times 8)/8$ cycle/word, and load from user memory is 20/8 cycles/word.
2. Cache memory → kernel memory; 17.5 cycles/word
   $1 + 50/8 + (30 + 4\times 8)/8 + 20/8 = 17.5 cycles/word$
   Cache will be missed once for 8 words copying from cache memory to kernel memory (in main memory). Store operation to main memory (kernel space) is 1 cycle cache-miss routine is $(30+4\times 8)/8$ cycles/word, and store from cache memory to main memory (kernel space) is 20/8 cycles/word.
3. Kernel memory → cache memory; 17.5 cycles/word
   $1 + 50/8 + (30 + 4\times 8)/8 + 20/8 = 17.5 cycles/word$
4. UDP/IP processing; 25 cycles/word

5. Kernel memory $\rightarrow$ interface buffer; 12.5 cycles/word
   200/(128/8)=12.5 cycles/word

Total cycles to transmit the data is 90 cycles/word. Since one word is 64 bits and one cycle is 2 nsec (=1/500MHz), the throughput is about 356Mbps.
$1/[(90/64) \times 2 \times 10^{-9}] \simeq 355.6 \times 10^6$
Here, the copy overhead is 65 (=90-25) cycles, which is about 72%.

Appendix B: Evaluation of Video Communication by X-Window

The system model is as same as in Appendix A. Regarding the description of system model, only the system assumption associated with video data processing is described.

Appendix B:1 25MHz CPU Model

The following model is assumed.

- One pixel of video camera image; 3 byte
- Memory space for video camera image; 4 bytes/pixel
- Load from camera to cache; 30 cycles/pixel
- Operation for load from video camera to cache; 6 cycles/pixel
- Store from cache to main memory; 30 cycles (for 16 bytes)
- One pixel of X-window's image; 1 byte
- Transformation from camera image to X-window image; 25 cycles/pixel
- Operation for store data to frame buffer; 15 cycles (for 32 bits)
- Store from main memory to frame buffer; 5 cycles (for 32 bits)

The followings are the evaluation for video communication.

1. Transfer video image from video camera to main memory; 66 cycles/pixel

   Store from video camera to cache is 16 cycles (=4 pixels) for one operation (24 cycles = 4 pixel × 6 cycles/pixel), and cache is missed once for 4 pixel data copy. Therefore, 41 cycles/pixel is required to copy from video camera to cache.
   $30 + 20/4 + 6 = 41 cycles/pixel$
   Here, copy overhead is 35 cycles/pixel.
   $30 + 20/4 = 35 cycles/pixel$

2. Transform video image to image data for X-window system; 32.25 cycles/pixel

   For one pixel processing, camera image is loaded once, transformation table is loaded three times (for R/G/B), and transformed data (1 byte) is stored. Since, camera image should be loaded when it is copied from video camera, cache will always hit.

   Since, the transformed data is 1 byte, regarding store the transformed data, cache will be missed once in 16 time of store of transformed data. We could assume that, for table look up to transform camera image, the cache-miss probability is small, let be 10%. This is because the adjacent pixel would have large correlation. When the operation code for data transformation is 25 cycles/pixel, the total cycles for data transformation is 32.25 cycles/pixel.
   $25 + 20 \times (1/16) + 3 \times 20 \times 0.1 = 32.25 cycles/pixel$
   Here, the copy overhead is 8.25 cycles/pixel
   $1 + 20 \times (1/16) + 3 \times 20 \times 0.1 = 8.25 cycles/pixel$

3. Processing UDP/IP and copy to interface buffer; 10.5 cycles/pixel

   Regarding load of data to perform UDP/IP, cache will hit. Data is processed every 32 bits (= 4bytes). Here, 4bytes corresponds to 4 pixels of transformed image data. Therefore, UDP/IP processing is 7.5 cycles/pixel
   $25 \times (1/4) + 20 \times (1/16) = 7.5 cycles/pixel$
   Data copy from main memory to interface buffer is performed by DMA (12 cycles/word). Therefore, 3 cycles/pixel is required.
   $12 \times (1/4) = 3 cycles/pixel$
   Totally, 10.5 (=7.5+3) cycles/pixel is required. Here, the copy overhead is 4.5 cycles/pixel.
   $1 \times (1/4) + 20 \times (1/16) + 12 \times (1/4) = 4.5 cycles/pixel$

4. Data transmission by ATM network (150Mbps); 1.3 cycles/pixel

   One pixel is 8 bit (=1 byte). Therefore, 52 nsec ($= 8 \times (1/150M)$) is required to transfer one pixel. 53 nsec corresponds to 1.3 (=53/40) cycles.

5. Copy from interface buffer to kernel memory space and UDP/IP processing; 11.75 cycles/pixel

   From interface buffer to kernel memory is 3 (=12/4) cycles/pixel. For UDP/IP processing, cache is missed to read data. Therefore, the required cycles to perform UDP/IP is 8.75 cycles/pixel.
   $25 \times (1/4) + 20 \times (1/16) + 20 \times (1/16) = 8.75 cycles/pixel$

Totally, 11.75 (3+8.75) cycles/pixel is required. Here, the copy overhead is 6.5 cycles/pixel.
$12/4 + 1 + 20 \times (1/16) + 20 \times (1/16) = 6.5 cycles/pixel$

6. Copy from kernel memory to frame buffer; 5 cycles/pixel

   Copy to frame buffer is performed by 32 bits (=1 word) oriented. When 15 cycles is required for program loop and write to frame buffer require 5 cycles, 5 cycles/pixel is required.
   $(15 + 5) \times (1/4) = 5 cycles/pixel$
   Copy overhead is 1.5 cycles/pixel.
   $(1 + 5) \times (1/4) = 1.5 cycles/pixel$

Therefore, sender requires totally 84 (=41+32.5+10.5) cycles/pixel, network requires 1.3 cycles/pixel, and receiver requires totally 16.75 (=11.75+5) cycles/pixel.

When bi-directional video communication is performed, 100.75 (=84+16.75) cycles/pixel is required. 100.75 cycles/pixel corresponds to 1.99Mbps.
$1/[(100.85/8) \times 40 \times 10^{-9}] = 1.99 \times 106$
Here, copy overhead is 55.75 cycles/pixel, and it is about 55.4% (=55.75/100.75).
$35+8.25+1.5+3+3+3.5+1.5 = 55.75 cycles/pixel$
When uni-directional video communication is performed, the bottle-neck will be the entity that requires the largest processing cycles. In this case, the bottle-neck is sender that requires 84 cycles/pixel. Therefore, the throughput of uni-directional video communication is about 2.38 Mbps.
$1/[(84/8) \times 40 \times 10^{-9}] = 2.38 \times 106$
Here, copy overhead is 47.75 cycles/pixel, and it is about 56.8% (=47.75/84).
$35 + 8.25 + 1.5 + 3 = 47.75 cycles/pixel$

Appendix B:2 100MHz CPU Model

Compared to 25MHz CPU model, only the clock speed of every entity is four times faster than 25 MHz CPU model.

Therefore, as same as the previous evaluation, sender requires totally 84 cycles/pixel, network (150 Mbps ATM network) requires 53 (= 1.3×4) cycles/pixel, and receiver requires totally 16.75 cycles/pixel.

When bi-directional video communication is performed, 100.75 (=84+16.75) cycles/pixel is required. 100.75 cycles/pixel corresponds to 8.06Mbps.
$1/[(100.85/8) \times 10 \times 10^{-9}] = 8.06 \times 106$
Here, copy overhead is 55.75 cycles/pixel, and it is about 55.4% (=55.75/100.75).
$35+8.25+1.5+3+3+3.5+1.5 = 55.75 cycles/pixel$
When uni-directional video communication is performed, the bottle-neck will be the entity that requires the largest processing cycles. In this case, the bottle-neck is sender that requires 84 cycles/pixel. Therefore, the throughput of uni-directional video communication is about 9.52 Mbps.
$1/[(84/8) \times 10 \times 10^{-9}] = 9.52 \times 106$
Here, copy overhead is 47.75 cycles/pixel, and it is about 56.8%.

Appendix B:3 500MHz CPU Model

The following 64 bits oriented system model is assumed. And, it is assumed that synchronous DRAM is used for main memory.

- CPU clock; 500MHz (64bits)
- Bus clock; 125MHz (128 bits)
- Cache line; 64 bytes (=8 words)
- Cache miss; 50 cycles (for 8 words)
- Access time for main memory; 30 cycles
- Load from main memory; 4 cycles (for 1 word)
- Store to cache; 20 cycles (for 8 words)
- Load from cache; 20 cycles (for 8 words)
- Store from cache to main memory; 4 cycles (for 1 word)
- UDP/IP processing; 25 cycles/word
- DMA between main memory and I/O buffer; 200 cycles (for 128 bytes)

The followings are the evaluation for video communication.

1. Transfer video image from video camera to main memory; 6.25 cycles/pixel

   DMA copy from video camera to cache is 128 bytes (=32 pixels) by one operation (200 cycles), and cache is always missed to read operation, because of DMA copy. Here, program loop to read the video image data from video camera can be enoughly small to be ignored. Since one pixel of video camera's image is 4 bytes, the required clocks is 6.25 cycles/pixel.
   $200 \times (4/128) = 6.25 cycles/pixel$

2. Transform video image to image data for X-window system; 34.5 cycles/pixel

   It is assumed that CPU processes two pixels by one program loop. This is because 64 bits (=8 bytes) corresponds to two pixels. For two pixels processing, camera image data is loaded once, transformation table is loaded 6 (= 3×2) times (for (R/G/B), and transformed data (2 bytes) is stored. Camera image data is represented by 4 bytes. Since, cache line is 64 bytes that corresponds to 8 pixels, cache will be missed once in 4 times of read operation.

Since, the transformed data is 1 byte, regarding store the transformed data, cache will be missed once in 32 (=64/2) time of store of transformed data. We could assume that, for table look up to transform camera image, the cache-miss probability is small, let be 10%. This is because the adjacent pixel would have large correlation. When the operation code for data transformation is 25 cycles/pixel, the total cycles for data transformation per two pixel is 69 cycles.

$25+50\times(1/4)+50\times(1/32)+3\times2\times20\times0.1 \simeq 69 cycles/pixel$

Therefore, it is 34.5 cycles/pixel.

Here, the copy overhead is 22.5 cycles/pixel

$(1+50\times(1/4)+50\times(1/32)+3\times2\times50\times0.1)\times(1/2) \simeq 22.5 cycles/pixel$

3. Processing UDP/IP and copy to interface buffer; 5.46 cycles/pixel

Regarding load of data to perform UDP/IP, cache will hit. Data is processed every 64 bits (= 8bytes). Here, 8bytes corresponds to 8 pixels of transformed image data. Regarding store of data, cache will be missed once in 64 times write operation. Therefore, UDP/IP processing is 3.9 cycles/pixel

$25\times(1/8)+50\times(1/64) = 3.9 cycles/pixel$

Data copy from main memory to interface buffer is performed by DMA (200 cycles/word). Here, one word for DMA is 128 bytes. Therefore, 1.56 cycles/pixel is required.

$200\times(1/128) = 1.56 cycles/pixel$

Totally, 5.46 (=3.9+1.56) cycles/pixel is required. Here, the copy overhead is 2.47 cycles/pixel.

$1+1\times(1/8)+50\times(1/64)+200\times(1/128) \simeq 3.47 cycles/pixel$

4. Data transmission by ATM network (600Mbps); 6.6 cycles/pixel

One pixel is 8 bit (=1 byte). Therefore, 13.3 nsec ($= 8\times(1/600M)$) is required to transfer one pixel. 13.3 nsec corresponds to 1.3 (=13.3/2) cycles.

5. Copy from interface buffer to kernel memory space and UDP/IP processing; 6.25 cycles/pixel

From interface buffer to kernel memory is 1.56 (=200/128) cycles/pixel. For UDP/IP processing, cache is missed to read data. Therefore, the required cycles to perform UDP/IP is 4.69 cycles/pixel.

$25\times(1/8)+50\times(1/64)+50\times(1/64) \simeq 4.69 cycles/pixel$

Totally, 6.25 (=1.56+4.69) cycles/pixel is required. Here, the copy overhead is 3.25 cycles/pixel.

$1.56+(1+50\times(1/8)+50\times(1/8))\times(1/8) \simeq 3.25 cycles/pixel$

6. Copy from kernel memory to frame buffer; 4.38 cycles/pixel

Copy to frame buffer is performed by 64 bits (=1 word) oriented. When 20 cycles is required for program loop and write to frame buffer require 15 cycles, 4.38 cycles/pixel is required.

$(20+15)\times(1/8) = 4.38 cycles/pixel$

Copy overhead is 2.63 cycles/pixel.

$(1+20)\times(1/8) = 2.63 cycles/pixel$

As a result, sender requires totally 46.21 (=6.25+34.5+5.46) cycles/pixel, network requires 6.6 cycles/pixel, and receiver requires totally 10.6 (=6.25+4.35) cycles/pixel.

When bi-directional video communication is performed, 56.81 (=46.21+10.6) cycles/pixel is required. 56.81 cycles/pixel corresponds to 70.4Mbps.

$1/[(56.81/8)\times2\times10^{-9}] = 70.4\times106$

Here, copy overhead is 38.1 cycles/pixel, and it is about 67.1% (=38.1/56.81).

$6.25+3.47+3.47+3.25+2.63 = 38.10 cycles/pixel$

When uni-directional video communication is performed, the bottle-neck will be the entity that requires the largest processing cycles. In this case, the bottleneck is sender that requires 46.21 cycles/pixel. Therefore, the throughput of uni-directional video communication is about 86.6 Mbps.

$1/[(46.21/8)\times2\times10^{-9}] \simeq 86.6\times106$

Here, copy overhead is 32.2 cycles/pixel, and it is about 69.5% (=32.2/46.21).

$6.25+22.5+3.47 = 32.2 cycles/pixel$

Hiroshi Esaki received the B.S. and M.S. degrees from Kyushu University, Fukuoka, Japan, in 1985 and 1987, respectively. In 1987, he joined Research and Development Center, Toshiba Corporation, where he engaged in the research of ATM systems. He has been at Bellcore in New Jersey (USA) as a residential researcher from 1990 to 1991, and has engaged in the research on high speed computer communications. He is currently interested in ATM internet architecture and high speed computer networks.

Yoshiyuki Tsuda joined Research and Development Center of Toshiba Corporation in 1989. His research interests have included real-time systems, operating systems and communication systems. He is currently interested in software architecture suitable for high speed networks. He received the B.S. and M.S. degrees from University of Tokyo, Tokyo, Japan, in 1987 and 1989, respectively.

Kumiko Kanai received the B.S. degree in Computer Science and the M.S. degree in Computer Science in 1990 and 1992, respectively from Gunma University. In 1992, she joined Research and Development Center, Toshiba Corporation, where she has been working on communication systems. She is currently interested in ATM network architecture. Miss Kanai is a member of Information Processing Society of Japan.