QoS Control for High-Speed Communication in a Group

Takuya Tojo†, **Hiroshi Yamamoto**†, **Tomoya Enokido**††, **and Makoto Takizawa** ††

†*NTT Service Integration Laboratories*
{tojo.takuya, yamamoto.hiroshi}@lab.ntt.co.jp

††*Dept. of Computers and Systems Engineering*
Tokyo Denki University
{eno, taki}@takilab.k.dendai.ac.jp

In group communications, multiple processes first establish a group and then each process sends a message to multiple processes while receiving messages from multiple processes in the group. In addition, messages are required to be causally/totally delivered to each process. Due to the limited amount of computation and communication resource, processes cannot send and receive as many messages as the processes would like. We newly propose a notification-based data transmission procedure with two-phase slow start (TPSS) to efficiently exchange multimedia messages in a group so as to satisfy QoS requirement. In TPSS, the transmission rate of a process is increased by transmitting redundant data so that no data is lost even if some packets are lost.

高速グループ通信のための品質制御

東條琢也 † **山本浩司** † **榎戸智也** †† **滝沢 誠** ††

†*NTT* サービスインテグレーション基盤研究所

†† 東京電機大学理工学部情報システム工学科

マルチメディアデータを扱うグループ通信では、必要とされるサービス品質 (QoS) をグループ内の各アプリケーションプロセスに提供せねばならない。グループ通信では、プロセスは複数のプロセスからメッセージを受信し、かつ複数のプロセスにメッセージを送信する。計算機資源が限られていることから、プロセスは他のプロセスから送信されたメッセージをすべて受信できない場合がある。本論文では、高速ネットワーク上でグループ内のプロセス間に必要な QoS を提供するために、通知方式とデータ多重化による二相スロースタート方式によるマルチメディア送信について論じる。

1 Introduction

In distributed applications, multimedia data like voice and image is exchanged among processes in high-speed networks like Gigabit Ethernet [6] and ATM [1]. High-speed protocols like XTP [4] and multimedia protocols like RTP [9] and RSVP [3] are developed so far, by which a large volume of multimedia data can be efficiently *unicast* to one process or *multicast* to more than one process. In these protocols, inter-message gap is controlled to be kept so long that a receiver process does not overrun the buffer. One-to-one (unicast) and one-to-many (multicast) protocols to satisfy requirement of Quality of Service (QoS), delay time, bandwidth, and loss ratio are discussed [1, 3, 9]. On the other hand, a process exchanges messages with multiple processes in a group [2, 5, 7, 11], i.e. sends each message to multiple processes while receiving messages from multiple processes. For example, video and voice data are distributed to every remote site in teleconferences.

In a traditional approach, every site first sends multimedia data to a controller. Then, the controller forwards the data to every remote site. Since it takes at least two rounds to deliver a message from a site to another site, the centralized approach is not suited to real-time applications in a wide-area network. We take a *fully distributed* approach [7] where every process directly sends a message to destination processes without any centralized controller in order to satisfy real-time constraints of multimedia data. Every process sends a message to each destination process while receiving messages from multiple processes so that QoS requirement is satisfied. The authors [13] discuss synchronous/asynchronous and QoS-balanced/unbalanced multicast/receipt models for exchanging multimedia messages in a group. Each process spends computation resource to exchange messages with multiple processes. For example, if a process allocates most computation resource to receive messages from one process, the process cannot receive messages from other processes due to the lack of computation resource. It is not easy to estimate bandwidth for each process and adopt the feedback-based control taken by other protocols [4, 8, 9] in the group communication. In addition, the transmission rate of a process is increased by more redundantly transmitting data [14]. Even if a destination process overruns the buffer and some data is lost, the data can be recovered from the redundant data and be delivered without retransmission. In this paper, we newly propose *a notification-based data transmission protocol* (*NQCP*) with *two-phase slow start* (*TPSS*) in a fully distributed group communication.

In section 2, we discuss the two-phase slow start (TPSS) algorithm. In section 3, we discuss the notification-based data transmission procedure (NQCP). In section 4, NQCP with TPSS is evaluated.

2 Two-Phase Slow Start

2.1 System Model

A *group* G is a collection of multiple processes $p_1, \ldots, p_n (n>1)$ which are cooperating by exchanging messages in a network. A process sends each message to multiple processes while receiving messages from multiple processes in a group. Let $src(m)$ and $dst(m)$ denote a source process and collection of destination processes of a message m, respectively. The network supports Quality of Service (QoS); bw(bandwidth [bps]), pl(packet loss ratio [%]), and dl(delay time [msec]). A message m is decomposed into a sequence $pkt(m)$ of *packets* $pk_1, \ldots, pk_l (l \geq 1)$. A packet is a unit of data transmission in the network. A destination process receives packets and assembles the packets into a message. Then, the message is delivered to the application process by the destination process.

2.2 Redundant data transmission

In a high-speed network, packets are lost due to buffer overrun and overflow in each destination process. In addition, packets are lost in a burst manner. In order to tolerate the burst packet loss, we transmit redundant packets. An application data is decomposed into a sequence of *segments*. Each segment is furthermore decomposed into a sequence of *data* packets $pk_1, \ldots, pk_l$ with a *parity* packet pk_o [Figure 1]. A data packet carries application data. The parity packet includes the data obtained by calculating the *exclusive or* (XOR) of the packets $pk_1, \ldots, pk_l$. Here, all the data packets $\langle pk_1, \ldots, pk_l \rangle$ can be obtained only if more than one packet in the segment $\langle pk_1, \ldots, pk_l, pk_o \rangle$ is not lost. That is, even if one data packet pk_i is lost in a segment, every application data in the segment can be recovered. A segment is a unit of recovery from packet loss.

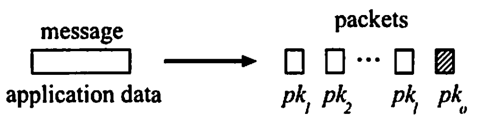


Figure 1. Segment.

2.3 Increasing transmission rate

Suppose a group G is established among multiple processes $p_1, \ldots, p_n$. If a process would like to send packets, the process makes a decision on a type of packet delivery and the initial transmission rate to each destination process. There are two types of packet delivery, *priority-based* and *fair* one. In the priority-based packet delivery, processes are prioritized. Packets received are delivered in priority order of sender process. In the fair packet delivery, processes are peers. Packets from each process are delivered at same priority. Another process p_i starts data transmission to the process p_j at initial transmission rate. The initial transmission rate is so small that no buffer overrun occurs in the process p_j. Let $rate_{ij}$ denote the transmission rate from a process p_i to a process p_j. The receiver process p_j allocates the transmission rate $rate_{ij}$ to each sender process p_i so that p_j does not overrun buffer even if all the other processes send packets to the process p_j. Then, p_j notifies each process p_i of $rate_{ij}$. That is, the summation of the rates $rate_{1j} + rate_{2j} + \ldots + rate_{nj}$ is smaller than the maximum receipt rate of the process p_j. Then, the process p_i sends packets to the process p_j at the rate $rate_{ij}$.

If a sender process has to send packets at higher transmission rate, the sender process monotonically increases the transmission rate in traditional slow start algorithms [4, 8]. If the sender process is notified of packet loss by a receiver process, the transmission rate is decreased and the packets lost are retransmitted by the sender process. This strategy is not suitable for real-time applications since it takes time to deliver a packet lost. We propose a novel slow start data transmission strategy named *two-phase slow start* (TPSS), which aims at reducing the number of packets retransmitted even if the packets are lost. TPSS is composed of *redundant* and *non-redundant* transmission phases. In the traditional slow start algorithms, a sender process increases the transmission rate of packets, i.e. the transmission rate is increased by transmitting more volume of application data for a unit time. Hence, the destination process cannot receive application data in the packets lost. In our approach, the transmission rate is increased by more redundantly transmitting application data while application data itself is transmitted at the original rate [Figure 2]. The higher the transmission rate, the more redundantly application data is transmitted. Even if a destination process cannot receive a packet due to overruns, no application data is lost since application data in packets lost are redundantly carried by other packets. Since packets lost are not retransmitted, the total transmission rate can be increased by increasing the redundancy of the data transmission while the transmission rate of the application data is invariant.

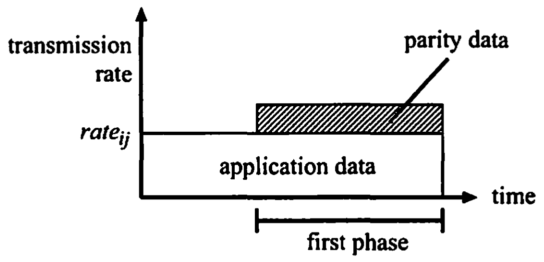
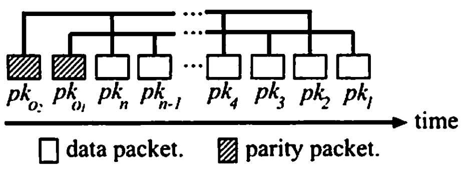


Figure 2. First phase.

2.4 Burst packet loss

In high-speed communication, packets are lost in a burst manner. For example, if CPU is not available to receive packets for ten milliseconds [msec] in a Gigabit Ethernet, a process loses about forty packets (30kbytes / packet). Hence, there must be more than forty packets between every pair of packets which carry replicas of same application data. Otherwise, every replica of the application data is lost in one burst failure. The number of contingent packets lost is referred to as *burst loss length*. For example, Figure 3 shows how to transmit a pair of parity packets pk_{o_1} and pk_{o_2}. The application data is decomposed into a *segment* of n packets $pk_1, \ldots,$ pk_n. A sequence of the data packets $\langle pk_1, \ldots, pk_n \rangle$ is furthermore decomposed into a pair of *subsegments* $s_1 = \langle pk_1,$ $pk_3, \ldots, pk_{l_1} \rangle$ and $s_2 = \langle pk_2, pk_4, \ldots, pk_{l_2} \rangle$ where $l_1 = n$ - 1 and $l_2 = n$ for even number n, $l_1 = n$ and $l_2 = n$ - 1 for odd number n. The parity packets are transmitted after all the data packets. Each parity packet pk_{o_i} is created by calculating XOR (exclusive or) of data packets $pk_i, pk_{i+2}, \ldots, pk_{l_i}$ (i=1, 2). If the packets are decomposed into k segments, each segment s_j is a sequence $\langle pk_j, pk_{j+k}, pk_{j+2k}, \ldots \rangle$ of packets. A segment is realized by interleaving k subsegments. That is, packets are transmitted in a sequence $\langle \ldots, pk_j, pk_{j+1}, \ldots,$ $pk_{j+k-1}, pk_{j+k}, \ldots \rangle$ where each packet pk_{j+i-1} comes from a subsegment s_i (i=1, ..., k). Here, the number k is referred to as *packet distance*. In the example of Figure 3, the packet distance is 2. The packet distance is required to be longer than or equal to the burst loss length. Here, even if two contingent packets, say pk_3 and pk_4 are lost, at most one packet is lost in every subsegment. Each of the packets lost are recovered in each subsegment. Hence, the application data can be delivered to the destination process even if packets in a segment are lost in a burst manner.



pk_{o_2} pk_{o_1} pk_n pk_{n-1} pk_4 pk_3 pk_2 pk_1
time
data packet.
parity packet.

Figure 3. Transmission of segments.

If a destination process p_j can receive all of the packets from a process p_i in the redundant transmission phase for a round-trip time, the destination process p_j assigns the receipt rate $rate_{ij}$ at the redundant phase to the sender process p_i and the second non-redundant phase starts [Figure 4]. The sender process starts non-redundantly sending packets at the assigned transmission rate $rate_{ij}$.

During the first phase, every receiver process p_j is notified of the transmission rate required by a sender process p_i. Every receiver process allocates computation resource to receive packets. Then, the process p_j notifies the process p_i of the rate $rare_{ij}$ at which p_i can send packets to p_j. Then, the second phase starts. Hence, most packets can be received by every destination process at the second phase. Thus, the size of application data to be lost can be more reduced than the traditional slow start algorithms even if some number of packets are lost in a burst manner.

3 Data Transmission Procedure

3.1 Notification approach

In traditional feedback-based protocols, the transmission rate of each process is controlled to prevent packets from being lost due to lack of receiver buffer and network congestions, i.e. window-based flow control [8] and rate-based congestion control [4]. If packets are lost due to lack of computation resource like CPU, the receiver process considers that packets are lost due to congestion. Then, the receiver process notifies the sender process that the transmission rate is reduced in the congestion control. In the group communication, a process receives packets from multiple processes while sending messages to multiple processes. The number of packets to be lost can be decreased by changing the delivery rate of packets sent by the other sender processes. Therefore, we newly take a *notification-based* approach, where each process p_i notifies every process p_j of the transmission rate $rate_{ij}$ at which the process p_j is required to receive packets from the process p_i. Here, the process p_j may receive packets from other processes in addition to the process p_i. If the buffer overruns at the destination process p_j in the two-phase slow start, the packet transmission rate $rate_{ij}$ is reassigned to each sender process p_i based on the type of packet delivery.

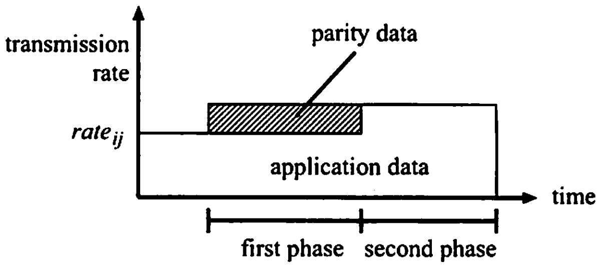


Figure 4. Second phase.

A process p_s receives a message m from an application. A sequence $pkt(m)$ of packets $pk_1, \ldots, pk_l$ of the message m is stored in a message queue MQ of the process p_s. Here, pk_1 and pk_l are referred to as *top* and *last* packets in the sequence $pkt(m)$, respectively. Each process p_s manipulates the following parameters:

1. D : data size [packet] stored in the queue MQ.
2. TT : transmission time D / TR [time unit].
3. HT : holding time [time unit].

The process p_s dequeues a packet from the queue MQ every $\frac{1}{TR}$ time units. TR [packet/time unit] shows the transmission rate of the message m, which is lower than the notified transmission rate $rate_{ij}$. HT indicates the processing time to change the delivery rate in a process. The QoS requirement of the message m is sent to all the destination processes on time when the top packet of the message m is enqueued into the message queue MQ of p_s. Even if MQ is empty when packets are enqueued, the packets are not soon transmitted in the network. Each packet stays in the queue MQ for at shortest HT time units. HT is *holding time* showing how long each packet stays in the queue MQ. The transmission of the message m is delayed until the QoS information of the message m is surely delivered to the destination processes [Figure 5]. QoS information is in a form $<D, TT, HT>$.

Next, suppose that a process p_t receives packets from another process p_s. The process p_t first receives QoS information. The process p_t allocates computation resource, i.e. CPU to receive packets from p_s. Let *top*(MQ) and *last*(MQ) de-

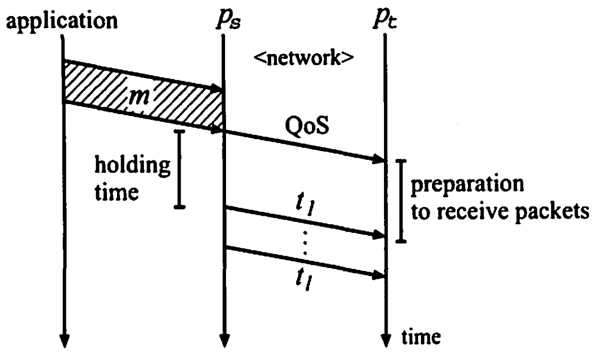


Figure 5. QoS notification.

note top and last packets in a queue MQ, respectively. A notation "$pk := dequeue(MQ)$" shows that the top packet $pk = top(MQ)$ is dequeued from the queue MQ. $enqueue(MQ, pk)$ means that a packet pk is enqueued into the queue MQ. $size(MQ)$ indicates the total amount of data of packets in MQ. $send(pk)$ shows that a packet pk is sent to all the destination processes. $current.time$ stands for current time in a process. $unique.no()$ shows a function where returns a value which is incremented by one each time the function is called. $pk.T$, $pk.N$, and $pk.Q$ show time, packet number, and QoS carried by a packet pk, respectively. $pk.Q$ includes $pk.Q.D$, $pk.Q.TT$, and $pk.Q.HT$ which show data size, transmission time, and holding time, respectively.

[Receipt of a message from application] A message m from an application is decomposed into a sequence $pkt(m)$ of packets $pk_1, \dots, pk_l$ in a process p_i:

```
• for $h$=1, ..., $l$, { $pk_h.T := current.time$;
      $pk_h.N := unique.no()$;  $enqueue(MQ, pk_h)$; }
• $pk := top(MQ)$;
  if $pk = \phi$, { create a control packet $c$ without data;
      $c.Q.D := size(MQ)$;  $c.Q.TT := c.Q.D / TR$;
      $c.Q.HT := HT - (current.time - pk_1.T)$;
      $send(c)$; }; □
```

[Transmission of packets]

```
• For each top packet $pk = top(MQ)$, {
    if $(current.time - pk.T) \geq HT$, {
      $pk := dequeue(MQ)$;  $c := top(MQ)$;
      $pk.Q.D := size(MQ)$;
      $pk.Q.TT := pk.Q.D / TR$;
      $pk.Q.HT := HT - (current.time - c.T)$;
      $send(pk)$; }; } □
```

3.2 Traffic estimation

On receipt of the QoS notification $\langle QN_1, \dots, QN_n \rangle$ from processes $p_1, \dots, p_n$, a process p_t estimates the total number of packets to be received for a time unit (10 milliseconds). Let $Trf_i[T]$ show number of packets received from a process p_i at time T. "T=0" means current time. $Trf_i[T]$ is obtained from the QoS notification as follows:

- $Trf_i[T] := QN_i.D \ / \ QN_i.TT$ for $T = QN_i.HT, QN_i.HT + 1, \dots, QN_i.HT + QN_i.TT$;

The total number of packets received at time T is $\sum_{i=1}^{n} Trf_i[T]$. If the maximum processing rate of a process p_t at time T is smaller than the total number ($\sum_{i=1}^{n} Trf_i[T] + TR$) of packets received and transmitted, the process p_t will overrun buffer at time T. In case, the process p_t does not deliver packets sent by less-prioritized processes in order to deliver packets from higher priority processes at time T.

4 Evaluation

4.1 TPSS

A module of the *Notification-based QoS Control Protocol* ($NQCP$) is implemented in C++ as processes on Solaris and Linux. In NQCP, a group is first established among multiple processes. Then, each process sends a packet to multiple processes and receives packets from each process in the group. $NQCP$ is evaluated compared with a traditional feedback-based approach with respect to how many packets are lost due to buffer overruns and how much throughput is obtained by redundant transmission.

First, we evaluate the two-phase slow start (TPSS) algorithm. In TPSS, redundant packets are transmitted while packets lost are recovered. Hence, TPSS is evaluated in terms of the number of packets lost and the effective receipt rate of application data. The effective receipt rate means the rate at which application data is received. Since redundant packets are transmitted in TPSS, the effective receipt rate is smaller than the receipt rate at each process. A pair of processes p_1 and p_2 are realized on two Linux personal computers (PCs) (Xeon 1.7GHz x 2 and Xeon 2.0GHz x 2), respectively, interconnected in a Gigabit Ethernet. In addition, NIST Net [12] runs on the PCs in order to make a long fat pipe network. Each one-way delay time between the processes p_1 and p_2 is extended to 100 milliseconds to simulate a wide-area network in a local area network by NIST Net, i.e. RTT (Round Trip Time) = 200 [msec] and BDP (Bandwidth Delay Product) = 25 [Mbytes]. In the evaluation, the process p_2 transmits packets to the other process p_1 for 150 seconds. Here, each packet is 30kbytes long. We make the following assumptions in the evaluation:

1. If $pl > 5$ [%], the receiver process p_1 notifies the sender process p_2 of 80% decrease of the original transmission rate. Otherwise, p_1 notifies p_2 of 120% increase.
2. A parity packet is inserted every five packets and the packet distance is 20 in the redundant phase of TPSS. That is, one segment is 120 packets which is decomposed into 20 subsegments. Each subsegment is composed of 5 data packets with one parity packet.

Figures 6 and 7 show effective receipt rate and packet loss ratio of the receiver process p_1 in TPSS and traditional slow start approaches, respectively. Table 1 summarizes how many packets are delivered, lost, and recovered. This evaluation shows that fewer number of packets are lost in the TPSS approach than the traditional slow start approach. In the traditional approach, the receiver process overruns the buffer even while changing the transmission rate on receipt of a feedback message. The average packet loss ratio is 21.9% at average effective receipt rate 441Mbps. On the other hand, packets lost are recovered from redundant packets without retransmission when the buffer overruns at the receiver process in TPSS. The average packet loss ratio is 5.4% at average effective receipt rate 427Mbps. This evaluation shows that not only packets lost are more recovered but also the transmission rate can be more adaptively changed in TPSS than traditional slow start approach. The effective packet loss ratio can be reduced to one fourth of the traditional approach since packets lost can be recovered in TPSS. In addition, application data can be delivered without retransmission even if some packets are lost. This means TPSS supports more flexible data transmission against change of packet loss ratio.

Table 1. Evaluation result.

	TPSS	slow start
# of packets delivered	267257	275646
# of packet lost (average loss ratio)	15181 (5.4%)	77292 (21.9%)
# of packets recovered	2139	-

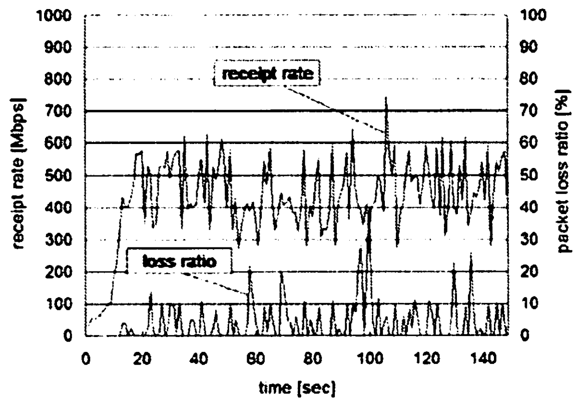


Figure 6. Two-phase slow start approach.

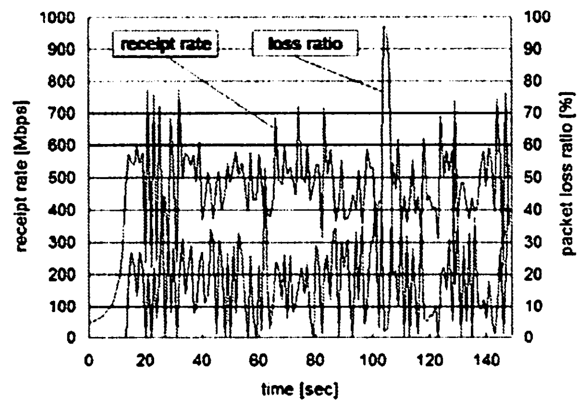


Figure 7. Slow start approach.

4.2 QoS notification

Next, we evaluate the QoS notification approach. A group is composed of four processes p_1, p_2, p_3, and p_4 on two Sun workstations (UltraSPARC-II 400MHz x 2 and UltraSPARC-III 900MHz x 2) and two Linux PCs (Xeon 1.7GHz x 2 and Xeon 2.0GHz x 2), respectively. The computers are interconnected in a Gigabit Ethernet. The processes p_1, p_2, and p_3 transmit packets to all the processes in the group while the process p_4 only receives packets as shown in Figure 8. One packet is 30kbytes long. Each packet stays in the message queue (MQ) for 100 milliseconds [msec], i.e. HT=100. Each queue MQ is so long that a process can deliver the QoS notification of each packet to every other process in 100 [msec] (=HT) until the packet is transmitted. If a process can have enough processing rate to receive all the packets sent by the other processes, there is no problem. It is problem how to receive packets if more number of packets arrive at a process than the process can receive. In NQCP, a process tries to receive all packets from higher-prioritized processes even if some packets from lower-prioritized processes are lost. We evaluate the QoS notification approach in terms of how many prioritized packets are delivered at the process p_4 on the following assumptions:

1. If $pl > 5$ [%], the receiver process p_4 notifies each sender process of 80% decrease of the original transmission rate. Otherwise, p_4 notifies each sender of 120% increase.
2. The process p_2 is the most highly prioritized in the process p_4. The process p_4 takes the priority-based packet delivery.
3. The process p_2 would like to send packets at 100Mbps.

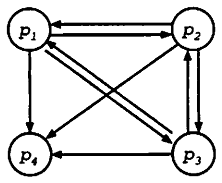


Figure 8. Communication among processes.

Table 2. Evaluation result.

		notification	feedback
p_1	# of packets delivered	21352	68173
	# of packet lost (average packet loss)	7254 (23.5%)	12468 (15.5%)
p_2	# of packets delivered	50106	47166
	# of packet lost (average packet loss)	2628 (5.0%)	6889 (12.7%)
p_3	# of packets delivered	20978	12790
	# of packet lost (average packet loss)	677 (3.1%)	564 (4.2%)

Figures 9 and 10 show the effective receipt rate and packet loss ratio of the process p_4 from the process p_2 obtained in

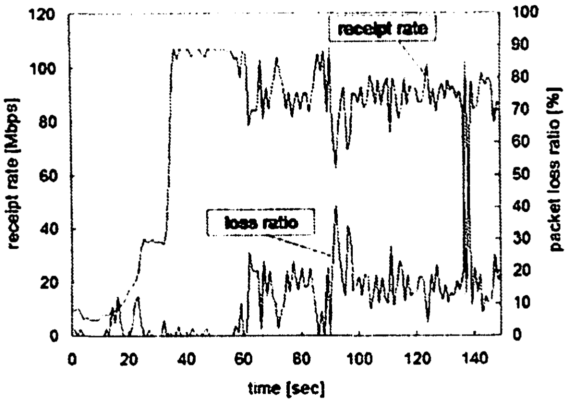


Figure 9. Feedback-based approach.

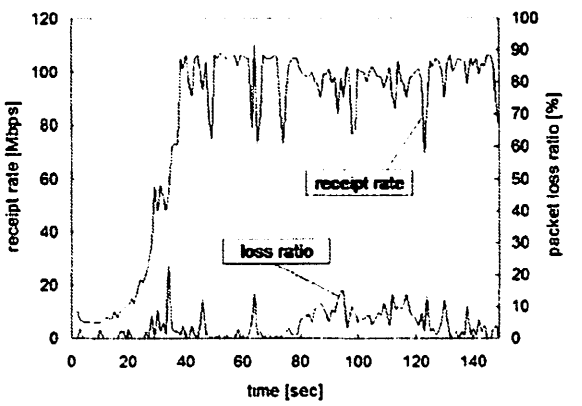


Figure 10. Notification-based approach.

the feedback-based approach and QoS notification-based approach, respectively. Table 2 summarizes how many packets from the processes p_1, p_2, and p_3 are delivered and lost in the process p_4. The process p_4 suffers from buffer overrun in the feedback-based approach. On the other hand, more number of prioritized packets can be delivered by changing delivery rate of packets sent by less-prioritized processes in the notification-based approach. Thus, the packet loss ratio between the processes p_2 and p_4 is 12.7% in the feedback-based approach while only 5.0% in the notification-based approach. In addition, the receipt rate of packets from the process p_2 in the notification-based approach is about 6.2% higher than the feedback-based approach as shown in Figures 9 and 10 because fewer number of packets are lost.

5 Concluding Remarks

Multimedia messages are exchanged among multiple processes in a group so as to satisfy QoS required by applications. We proposed the two-phase slow start (TPSS) data transmission algorithm where the transmission rate is increased by transmitting more redundant packets without increasing application data. One parity packet is transmitted for a subsegment of packets named segments. Packets in multiple segments are interleaved in transmission. Even if packets are lost in a burst manner due to the buffer overrun, at most one packet is lost in a segment. Destination processes can receive all application data by recovering data carried by packets lost. In addition, the notification approach is adopted to adjust the transmission rate in change of traffic. Packets of a message are sent on HT time units after notifying all the destination processes of the QoS information of the message. We implemented the protocol NQCP. We showed that more number of prioritized packets can be delivered in the NQCP than the traditional feedback-based approach in the evaluation.

References

[1] ATM Forum. Traffic Management Specification Version4.0, 1996.

[2] K. Birman. Lightweight Causal and Atomic Group Multicast. *ACM Trans. on Computer Systems*, 9(3):272-290, 1991.

[3] R. Braden. Resource ReSerVation Protocol. *RFC2205*, 1997.

[4] G. Chesson. XTP/PE Overview. *Proc. of the IEEE 13th Conf. on Local Computer Networks*, pages 292-296, 1988.

[5] C. Delporte-Gallet and H. Fauconnier. An Example of Real-Time Group Communication System. *Proc. of IEEE ICDCS-21*, pages 5-9, 2001.

[6] IEEE Computer Society. GigabitEthernet. *IEEE Std 802.3z*, 1998.

[7] A. Nakamura and M. Takizawa. Causally Ordering Broadcast Protocol. *Proc. of IEEE ICDCS-14*, pages 48-55, 1994.

[8] J. Postel. Transmission Control Protocol. *RFC793*, 1981.

[9] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real Time Applications, *RFC 1889*, pages 22-29, 1996.

[10] K. Shimamura, K. Tanaka, and M. Takizawa. Causally Ordered Delivery of Multimedia Objects. *Computer Communications Journal*, 25(5):437-444, 2002.

[11] T. Tachikawa, H. Higaki, and M. Takizawa. Group Communication Protocol for Realtime Applications. *Proc. of IEEE ICDCS-18*, pages 158-165, 1998.

[12] The Linux-based Network Emulation Tool NIST Net, http://snad.ncsl.nist.gov/itg/nistnet/.

[13] T. Tojo and M. Takizawa. QoS Control Model in Group Communication. *Proc. of DMS Workshop on Internet Computing and Multimedia (ICaM)*, pages 316-322, 2002.

[14] T. Tojo, T. Enokido, and M. Takizawa. Notification-Based QoS Control Protocol for Multimedia Group Communication in High-Speed Networks. *Proc. of IEEE the 24th International Conference on Distributed Computing Systems (ICDCS-2004)*, pages 644-651, 2004.