A Retransmission Control Algorithm for Low-Latency UDP Stream on StreamCode-Base Active Networks

Hideki Otsuki[1] and Takashi Egawa[2]

[1] Communications Research Laboratory,
4-2-1, Nukui-Kitamachi, Koganei, Tokyo, 184-8795 Japan,
eiji@crl.go.jp,
[2] NEC networking laboratories,
4-1-1 Miyazaki, Miyamae-ku, Kanagawa, 216-8555 Japan,
t-egawa@ct.jp.nec.com

Abstract. We propose an algorithm for real-time data stream that achieves low latency as well as low packet loss rate. In our algorithm a sequence number is included in each packet, and it is renumbered as if there is no packet loss in the upper stream when a packet loss and corresponding packet retransmission occurs. This conceals the packet loss and its recovery from succeeding nodes in the downstream, thus simplifies packet processing in succeeding nodes. This algorithm can easy be implemented on capsule-type active networks, and we evaluated this algorithm on StreamCode based active networks. It is implemented with 1.0k line in-packet programs plus node initialization programs, and the evaluation results show that 27.1% end-to-end packet loss rate is suppressed to around 5.6%.

1 Introduction

Real-time multimedia applications such as video conferencing and live-video distribution are becoming popular on the Internet. Applications of this kind are sensitive to latency as well as packet loss. Packets for these applications must be delivered in 'allowable delay' to replay audio and video data, and if a packet does not arrive in time, the retransmission of the packet do not contribute to the application quality improvement.

Ordinary TCP is not appropriate for these applications. TCP is designed to achieve perfect error recovery [1]. This means that a transmission failure of a packet can block other packet transfer, and can drastically increase latency. Moreover, the retransmission of TCP is processed in an end-to-end manner, which causes long RTT and thus large latency.

There are a few kinds of proposals that aim at satisfying the requirements of these applications. One is to use UDP/IP with Forward Error Correction (FEC). In this method lost packets are recovered from redundant information included in other packets. This enables to avoid retransmission, which shortens latency. However, huge amount of redundant data must be attached to recover bursty packet loss, which is common in the Internet.

The second proposal is to insert TCP/IP bridge in the TCP/IP's end-to-end feedback loop, and to make them an accumulation of multiple shorter TCP/IP loops. This shortens

N. Wakamiya et al. (Eds.): IWAN 2003, LNCS 2982, pp. 92–102, 2004.

retransmission delay and overall latency [2]. However, this method still aims at perfect error recovery, which means that a packet can block other packet transfer.

We therefore have to develop an algorithm that enables partial error recovery that suppresses packet loss to acceptable level.

In designing partial error recovery algorithms, the method to detect packet loss is important. Sequence number is commonly used for the detection, and if we use it in the same manner for partial recovery, a packet loss that is never recovered generates multiple NACK packets in every downward nodes, none of them contributes to application quality improvements.

Similar problem occurs in reliable multicast, and [3] proposed to use broadcast for NACK suppression. [4] proposed to establish an independent session for packet retransmission to distinguish retransmitted packets. However, these methods consume much node resources and bandwidth. We believe we should use more simple method because perfect error recovery is not required. We therefore propose a novel transport protocol algorithm for UDP data that suppress these problems by

- retransmitting lost packets at every intermediate node,
- simplifying the protocol by limiting the message type to NACK (negative acknowledge) only,
- limiting the number of packet retransmission of each packet, and
- renumbering the sequence number to hide retransmission from downstream nodes.

This paper is composed as follows. In chapter 2 we describe the details of this proposed algorithm. We then explain this algorithm is appropriate for capsule-type active networks, and describes the method to implement it on StreamCode-based active networks in chapter 3. The detailed implementation of experimental system and its evaluation results are stated in chapter 4, and in chapter 5 we conclude the paper.

2 Proposed Algorithm

2.1 Application and Network Model

We assume that applications for this algorithm contain a transmitter and a receiver. As shown in figure 1, packets from the transmitter to the receiver contain application level sequence number. The receiver has an application level buffer, and if the sequence of a packet reverses, it corrects the order before using it. The size of the buffer is decided to store packets for a certain amount of time.

As for the network, we assume that it consists of several routers, some of which have buffers for this algorithm. These routers with buffers (we name this relay nodes) implement the algorithm described in this chapter, and if a relay node detects packet loss it tries to recover the loss by requesting retransmission to the previous relay node as shown in figure 2.

The transmitter and the receiver lie at each end of the network. It is not necessary for every router to be a relay node. Since such ordinary routers do not affect to the behavior of this algorithm, we do not discuss this issue in more details in this paper.

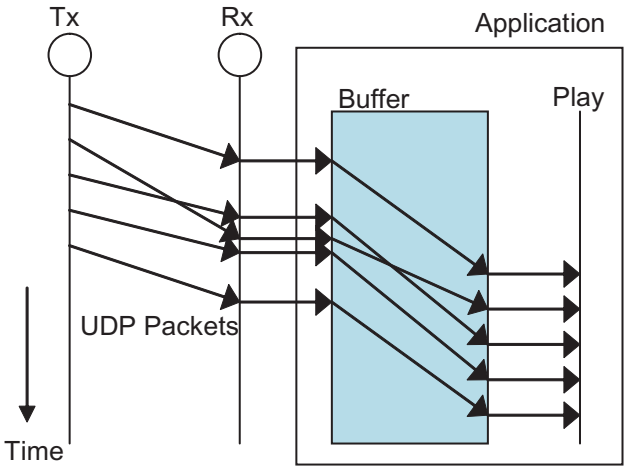


Fig. 1. Target application model

2.2 Algorithm Overview and Its Basic Features

The algorithm described here is a generalized version of [5] by eliminating allowable delay parameter. As stated in Figure 2, packets are buffered at every relay node. Each packet contains a sequence number that is independent of the application level sequence number mentioned in Figure 1.

Figure 3 shows the overview of the retransmission and corresponding sequence number renumbering. When a packet arrives at a relay node, the node checks the sequence number. If it detects packet loss(es) by a jump of the sequence number, it asks the previous relay node to retransmit the lost packet(s). Then the node renumbers the sequence number of the arrival packet so that the next relay node will receive packets whose sequence number do not jump, and transfers it to the next node. Thanks to this renumbering, a packet loss between a relay node pair is hidden from succeeding nodes, which prevents redundant retransmission requests by succeeding nodes.

If the previous node retransmits the lost packet and the packet is recovered successfully, the sequence number of the recovered packet is also renumbered in the same manner. This hides the fact that the packet is a retransmitted one, and the succeeding nodes can process it as an ordinary packet. This simplifies the packet processing in routers, and enables packet processing without waiting lost packets. This shifts the burden of packet renumbering to relay nodes to end hosts, which is compliant to the end-to-end argument.

In this algorithm packets are retransmitted between intermediate routers, which shortens the retransmission delay compared with the delay of end-end retransmission.

This algorithm do not aims at perfect packet loss suppression. If a packet retransmission request fails for some reason (e.g., the packet has already discarded in the previous router or the request packet is lost), the succeeding routers do not do anything. This may seem insufficient, but we believe that this is the right recover policy for real-time applications because even if a packet loss is recovered thanks to several retransmission

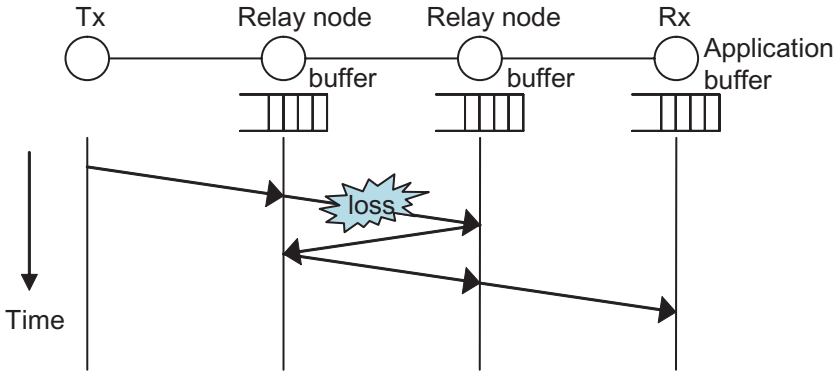


Fig. 2. Network model

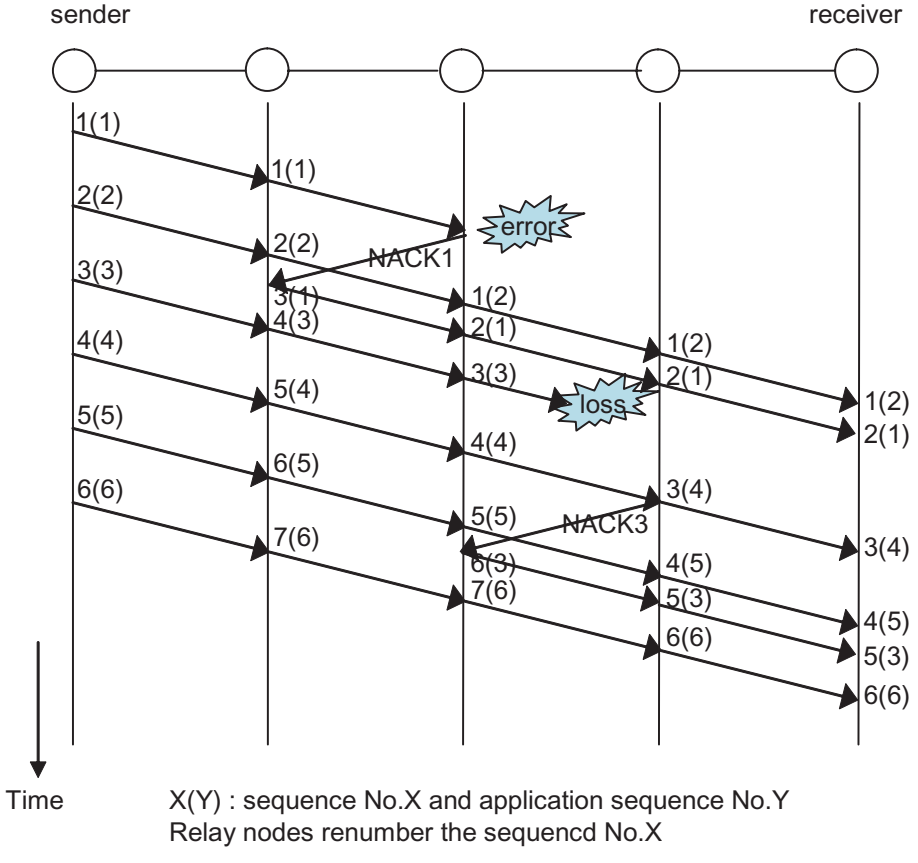


Fig. 3. Packet sequence renumbering

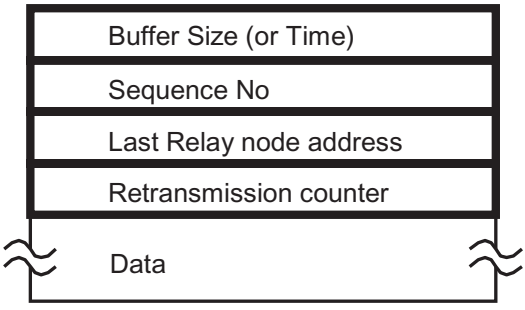


Fig. 4. Required infomations for this algorithm

trials, it is often useless because retransmission trial takes much time and the latency of the packet becomes so large for the application.

This algorithm requires all packets to go through the same route, because the packet loss detection depends on the sequence number. We believe that in active networks it should be done by the routing algorithm of active networks. This problem is out of the scope of this paper.

2.3 Processing on a Node

Figure 4 shows the information required in the packet.

Using this information, each node processes packets as shown in Figure 5.

- If a node receives a data packet, the node (re)writes the sequence number in the packet so that the sequence number does not jump in the next node, changes the node address to request NACK in the packet to itself, and forward it to the next node. Before forwarding the packet, the node makes a copy of the packet and stores it in the buffer with expiration time information. It also checks the sequence number of the received packet to see whether there is a jump in the sequence number, which means that packet(s) is lost.
  - If packet(s) loss is detected, the node sends a NACK to the previous node and returns to an idle state.
  - If there is no packet loss, the node returns to an idle state.
- If the node receives a NACK packet, the node checks the buffer whether the packet is stored. If stored it restores the lost packet, the node creates the sequence number so that the sequence number does not jump in the next node, and sends it out to the next node if retransmission counter is smaller than predetermined threshold. If the packet has already been discarded and is not stored in the node, or retransmission counter is larger than threshold, it just returns to the idle state. The total number of retransmission among links is therefore suppressed by this threshold in end-to-end basis.
- If the time out of a packet stored in the buffer arrives, the node discards the packet from the buffer. In our current implementation this is virtually realized by checking the expiration time of the packet before sending the restored packet out on receiving a NACK packet.

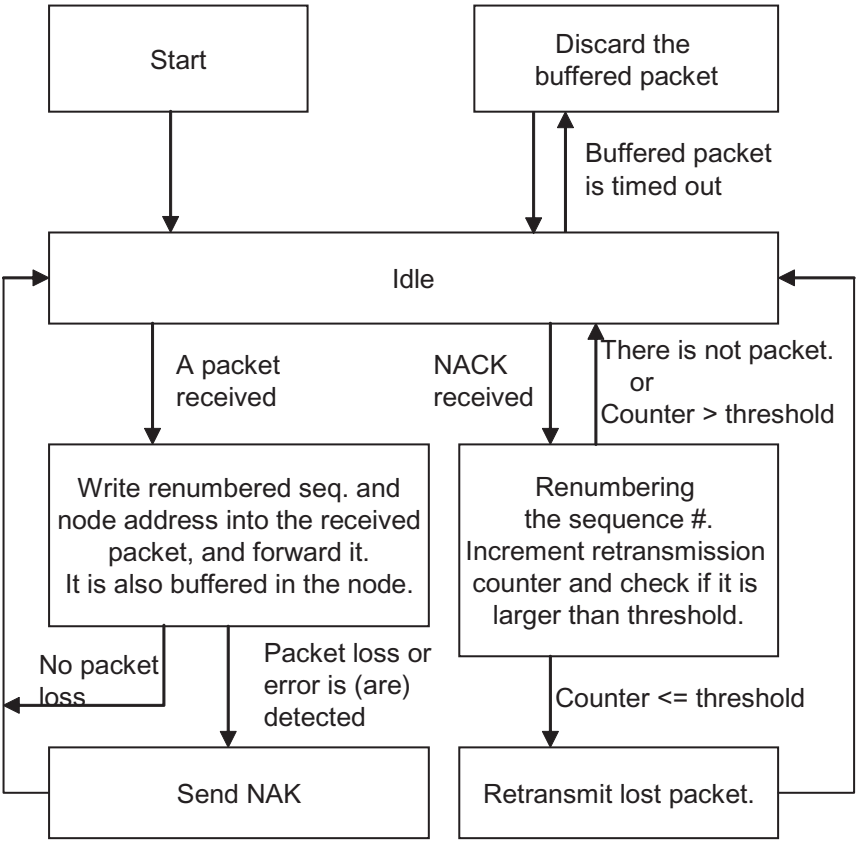


Fig. 5. Node behavior

3 Implementation on StreamCode Based Active Networks

3.1 Merit of Capsule-Based Active Network Implementation

We decided to use StreamCode-based active network [6,7] as the platform for the implementation of the proposed algorithm.

StreamCode is a capsule-type active networks in which packet processing algorithms are defined with in-packet programs that is written in StreamCode instruction set, a set that is defined for secure and high-performance packet processing.

There are two reasons why we used this system for the implementation.

1. Easiness of implementation.
   A new algorithm can easily be implemented on SC-based active networks. The proposed algorithm is implemented in merely 1.0k line in-packet programs except for the node initialization programs.
2. Flexibility against the network condition changes.
   On StreamCode network, we can change the processing algorithm for each packet

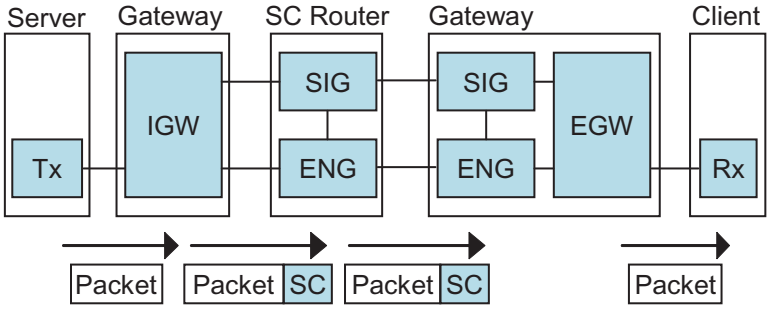


Fig. 6. Stream Code architecture

by attaching different programs to each packet. This means that we can choose optimal algorithm considering various network conditions, e.g., congestion, topological change or the occurrence of failures. For example, if the network is not congested we can choose no-retransmission algorithm and can enjoy less delay and less buffer consumption on each node. If the network congestion changes we can switch to our proposed algorithm without any modification in the network and can suppress packet losses.

3.2 Detailed Implementation Method on StreamCode Networks

Figure 6 shows the current StreamCode-based network architecture.

The network is composed of a server, a client, two gateways and StreamCode-based routers. Server sends out UDP/IP packets. StreamCode program is attached to each UDP/IP packet at the ingress gateway, and removed at the egress gateway. In this figure, the ingress gateway has Ingress Gateway Process (IGW) only. It is possible to implement SIGnaling process (SIG) and stream code ENGine (ENG) processes on the gateway, but we assume they are not implemented because of the simplicity. A StreamCode-based router has a StreamCode engine that has two functions. One is a function to process in-packet StreamCode programs, and the other is a function to process signaling that establishes sessions among StreamCode routers and gateways. By establishing a session among the related nodes, buffers for this algorithm is allocated and initialized on each node. Authentication information for each data packet is also distributed during the session initialization, and it enables the authentication and authorization of each packet to access the allocated buffer.

Shown in figure 7 is the essence of the algorithm of in-packet StreamCode program for data packets, and the information stored in the packet.

SC

```
START
  IF (source address !=0) {
     packet loss check;
     IF ( packet is lost ) {
}        send NACK SC;
     }

  Write sequence number from service table
      into this packet;
  Increment sequence number in service table;
  Forward this packet itself;
  Write time to packet table;
  Write this packet into packet table;
END
```

SC data

Sequence No.
Retransmission counter
Allowable delay
Originate address
Service ID

Service table

Service ID
Route ID
Sequence Number
Source address
Max packets
Max packet size
NACK SC size
NACK SC
Packet table (buffer area) : : :

NACK SC

```
START
  LOOP (The Number of lost packets) {
    Get packet from packet table;
      IF (packet is buffered) {
         Write sequence no of service table
         into the packet;
         Inclement sequence no of service table;
         Decrement retransmission counter in
         the packet;
         IF ( retransmission counter >=0 ) {
            retransmit the packet;
         }
      store the packet into packet table;
      }
  }
END
```

SC data

Destination address
Lost packet sequence (start)
Lost packet sequence (End)

Fig. 7. Stream Code for proposed retransmission control

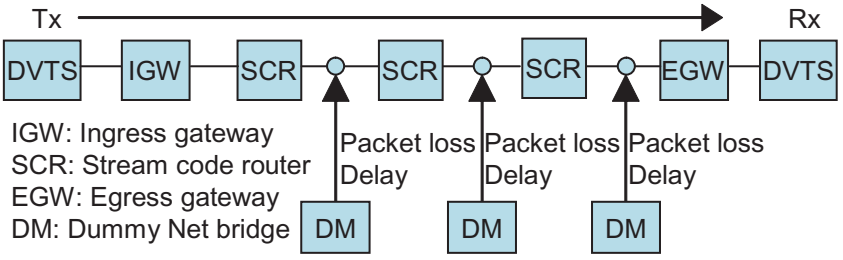


Fig. 8. Test system

A NACK packet also contains a StreamCode program called NACK SC. When it arrives the previous node the program checks the retransmission counter of the packet stored in the buffer, and if the counter is under the limit the program generates retransmission packet(s). In the current implementation one NACK packet can regenerate multiple packets that corresponds to a consecutive packet loss. It suppress the number of NACK packets, but a loss of a NACK packet may cause a consecutive loss. NACK packet should therefore be protected (e.g., duplicate NACK) more if the better packet loss rate is necessary.

4 Experimental System and Its Evaluation Results

4.1 Experimental System

Shown in Figure 8 is the experimental system to evaluate this proposed algorithm. It is composed of the StreamCode-based active network described in chapter 3.2, and two terminals on which Digital Video Transport System (DVTS) [8] is working. DVTS is an application that transmits and receives Digital Video stream on RTP [9]. Between some active routers we inserted dummynet [10] that can emulate packet loss and link delay.

4.2 Evaluation Method and Its Results

DVTS itself has little buffer and cannot adapt to the reversal of packet sequence, thus is not compliant to the application model described in chapter 2.1. We therefore made a program that buffers packets of 500msec stream, and reorders the packet sequence if a reversal occurs. Thanks to this program application-level packet loss does not occur if retransmission completes in 500msec.

Using this system we measured the end-to-end packet loss with our proposed algorithm and compared it with the theoretical packet loss ratio without retransmission, changing the link delay of each link using dummynets. The link delay and the packet loss rate of three dummynets are the same. The results are shown in Table 1.

When the packet loss rate of each link is 10becomes 27.1% if there is no retransmission. This value greatly decreases to around 5.6% if the proposed retransmission algorithm is applied. When the rate of each link is 5% and 1%, the end-to-end packet

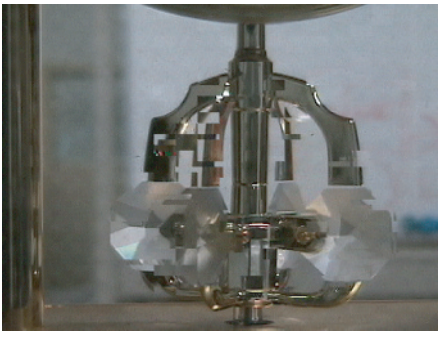
Table 1. End-End packet loss

Packet loss rate of each link	Theoretical end-end packet loss	delay of each link (msec)					
		50	60	70	80	90	100
10%	27.10%	5.58%	5.62%	5.65%	5.59%	5.67%	5.68%
5%	14.26%	1.47%	1.50%	1.48%	1.47%	1.49%	1.50%
1%	2.97%	0.08%	0.09%	0.09%	0.11%	0.09%	0.08%

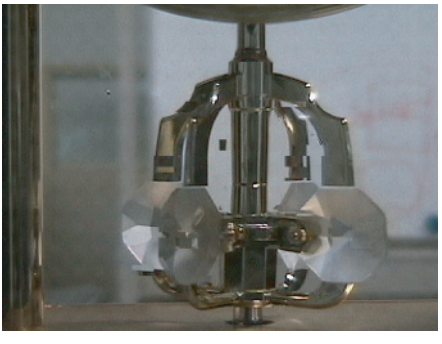
loss rate is suppressed from 14.26% to around 1.5% and from 2.97% to around 0.1%, respectively.

The packet loss should become slightly larger if the link delay of each link increases. However, thanks to the retransmission times limitation the packet loss rate was almost constant against link delay in our evaluation.

Figure 9 shows the pictures of DVTS with and without our proposed algorithms. The picture quality becomes higher in (b), the one with our proposed algorithm.



(a) UDP (packet loss rate=10%)



(b) Using proposing SC (Packet loss rate=10%)

Fig. 9. Application quality

5 Conclusion

We proposed an algorithm for real-time stream that achieves low latency as well as low packet loss rate. It renumbers the sequence number in each packet if a packet loss and the corresponding packet retransmission occurs, which simplifies the packet processing in succeeding nodes. This algorithm can easy be implemented on capsule-type active

networks, and we evaluated this algorithm on StreamCode based active networks. The evaluation results show that 27.1% end-to-end packet loss rate is suppressed to around 5.6%, which proves the effectiveness of our proposed algorithm.

Thanks to the implementation in StreamCode, we can easily change the algorithm if we sense the change of network conditions, e.g., congestion or failure. We therefore are planning to design an algorithm that dynamically changes buffer consumption as well as the latency and packet loss ratio by changing the StreamCode attached to the packet.

References

1. TCP Selecive Acknowledgement Options, RFC2018
2. Kammouane XONSIHAPANYA, Katsunori YAMAOKA, Yoshinori SAKAI: Improvement of Average Delay Performance in Packet Network by Introducing Intermediate Node, Proc. of ICOIN12 (1998) 25–28
3. Sally Floyd, Van Jacobson, Ching-Gung Liu, Steven McCanne, Lixia Zhang: A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing, IEEE/ACM Trans. on Networking Vol.5 No.6 (1997) 784–803
4. Sneha Kumar Kasera, Gisli Hjalmtysson, Donald F. Towsley, James F. Kurose: Scalable Reliable Multicast Using Multiple Multicast Channels, IEEE/ACM Trans. on Networking Vol.8 No.3 (2000)
5. Hideki OTSUKI, Katsunori YAMAOKA, Yoshinori SAKAI: A Realtime Media Stream Transfer Protocol with Finite Retransmission between Relay Nodes, Proc. of IEICE CQR2002(2002) 223–226
6. Takashi Egawa, Koji Hino and Yohei Hasegawa: Fast and secure packet processing environment for per-packet QoS customization, Proc. of IWAN 2001 (2001)
7. Takashi Egawa and Hideki Otsuki: Per-packet authentication for capsule networks; its pros and cons, Proc.of ANTA2003 (2003)
8. Digital Video Transport System, http://www.sfc.wide.ad.jp/DVTS/index.html
9. RTP:A Transport Protocol for Real-Time Applications, RFC1889
10. http://info.iet.unipi.it/ luigi/ip_dummynet/
11. Hideki OTSUKI, Katsunori YAMAOKA, Yoshinori SAKAI: Adaptive Protocol Relay Capability with Allowable Delay of Media Transmission, Proc. of IEEE PACRIM2001 (2001) 627–630