Master's Thesis

Title

Design, Implementation, and Evaluation of Active Video-Quality Adjustment Method for Heterogeneous Video Multicast

Supervisor

Prof. Hideo Miyahara

Author

Tatsuya Yamada

February 12, 2003

Department of Informatics and Mathematical Science

Graduate School of Engineering Science

Osaka University

Master's Thesis

Design, Implementation, and Evaluation of Active Video-Quality Adjustment Method for Heterogeneous Video Multicast

Tatsuya Yamada

Abstract

By introducing video-quality adaptation mechanisms into intermediate network equipment using active network technologies, we can provide users with video distribution services that take into account client-to-client heterogeneity in terms of available bandwidth, performance of client systems, and user preferences for video quality.

In this thesis, we design and implement an active video-quality adjustment node on a network processor-based equipment. In the design, we consider the architecture of an active video quality adjustment node within the framework of an active network technology. We first consider the functional design of the active video-quality adjustment node. Then, four functional layers of the active node, i.e., an Active Application layer, an Execution Environment layer, a NodeOS layer, and a Router layer, are assigned to processors on the node, i.e., microengines, a StrongARM processor core, and a Pentium processor. The active video-quality adjustment node can adapt the video rate to the requested level on a packet-by-packet basis. The target rate is specified in an ANEP header of a video packet. We conduct experiments and verified the practicality of the video-quality adjustment within a network. It is shown that the active video-quality adjustment node can adapt the video rate to the dynamically changing requested level on a packet-by-packet basis. Furthermore, the active video-quality adjustment node can simultaneously generate video streams of different quality levels from the same video stream.

Keywords

video quality adjustment, network processor, active network, heterogeneous video multicast, MPEG-2

Contents

List of Figures

List of Tables

1 Introduction

With the proliferation of broadband access to the Internet, video distribution services such as video streaming or live transmissions are now being widely deployed. Video services reach users who are heterogeneous in terms of the capacity of access links, available network bandwidth, performance of client machines, and user preferences for perceived video quality. Therefore, it is essential to introduce mechanisms that ensure the provided video stream can meet each user's environment and preferences.

In [1], we proposed mechanisms for video multicast services in which diverse client requests are simultaneously satisfied while network resources are efficiently used. Our mechanisms were developed on the basis of active network technologies where intermediated network nodes, called active nodes, adapt the video rate to the desired level.

By using the active network technology, new network services can be developed and deployed flexibly and easily according to the demands of network administrators, users, or applications [2, 3]. Each packet passing through network equipment, called an active node, is processed based on a program that is contained in the packet itself or has been preloaded at the node. By introducing programs, active nodes can perform highly intelligent packet processing from lower-layer functions such as QoS routing to application-layer functions that manipulate user data in the packet payload. In our video multicast mechanisms proposed in [1], we coped with client-to-client heterogeneity by appropriately configuring chosen active nodes to adapt the rate of an incoming video stream to the desired level by means of video-quality adjustment as illustrated in Figure 1.

In [4], our research group compared several video-quality adjustment methods for real-time MPEG-2 video multicast, namely frame discarding, low-pass, and requantization filters. Algorithms were proposed for these video-quality adjustment methods to adapt the video traffic to the specified target rate. Our research group conducted several experiments and concluded that the low-pass filter, which provides rate reduction by progressively eliminating high-frequency com-

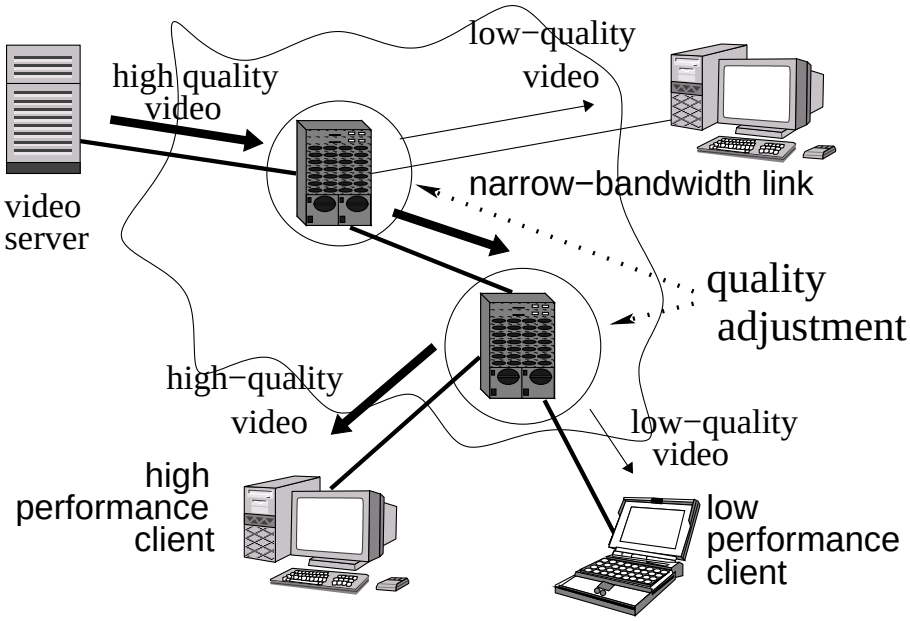


Figure 1: Heterogeneous video multicast

ponents of the video signal, is the most effective method for suppressing the quality degradation and granularity of the rate adaptation.

To verify the practicality of video-quality adjustment within a network, we implemented our low-pass filter for real-time MPEG-2 video multicast on an IXP1200-based network node in [5]. We discussed the packetization of an MPEG-2 video stream. The IXP1200 is an off-the-shelf network processor that has an architecture specialized for packet processing, and it has a general-purpose processor and some specialized processors, which are programmable. From the results of evaluation experiments using MPEG-2 video, we demonstrated that we can build a video-quality adjustment node on network-processor-based equipment. The node can adjust an incoming video stream to the desired rate on a per-packet basis. However, the target rate of video-quality adjustment must be specified in advance, at the time of generating the program code. Thus, the node in [5] cannot react to the dynamically changing target rate, which reflects changes in the available network bandwidth, performance of client machines, and user preferences.

This thesis describes how we designed and implemented the active video-quality adjustment node. The active video-quality adjustment node employs active network technologies to enable flexible setting of target rate, dynamic initiation and termination of video-quality adjustment operations, and appropriate node configuration. The functionalities of the active video-quality adjustment node are arranged on the basis of an active network architecture. Four functional layers of the active node, i.e., an AA (Active Application) layer, an EE (Execution Environment) layer, a NodeOS layer, and a Router layer, are assigned to processors on the node, i.e., microengines, a StrongARM processor core, and a Pentium processor. The performance of the node in terms of the accuracy of rate adaptation and the throughput was evaluated through experiments using an MPEG-2 video stream. In this thesis, we give details of the design, implementation, and results of the evaluation experiments.

The organization of this thesis is as follows. In Section 2, we briefly introduce the MPEG-2 coding algorithm, the IP packetization technique of MPEG-2 video streams, the implemented low-pass filter, and the rate adaptation mechanism. In Section 3, we describe the active node architecture and functional design of the active video-quality adjustment node. In Section 4, we explain the features of the Intel IXP1200 network processor and describe the functional assignments among processors as well as system implementation. We present experimental results and discussion in Section 5 and conclude the thesis in Section 6.

2 MPEG-2 Video-quality Adjustment

2.1 MPEG-2 video coding algorithm

Our active video quality adjustment is intended for the MPEG-2 video Elemental Stream [6] of the MPEG-2 Program Stream format [7], where a video and an audio stream are multiplexed into a single media stream. Figure 2 shows the hierarchical structure of MPEG-2 video data.

The highest layer is called the sequence layer. A sequence is composed of several Groups of Pictures (GoPs). A GoP is a sequence of three types of pictures: I (Intra-coded), P (Predictive-coded), and B (Bidirectionally predictive-coded). A GoP starts with an I picture, followed by several P and B pictures. A picture is composed of 16-pixel-high stripes, called slices. All sequences, GoPs, pictures, and slice layers begin with a 32-bit start code that is used for error recovery and for rewind and fast-forward functions.

Each slice consists of one or more macroblocks. Each macroblock corresponds to a 16x16-pixel

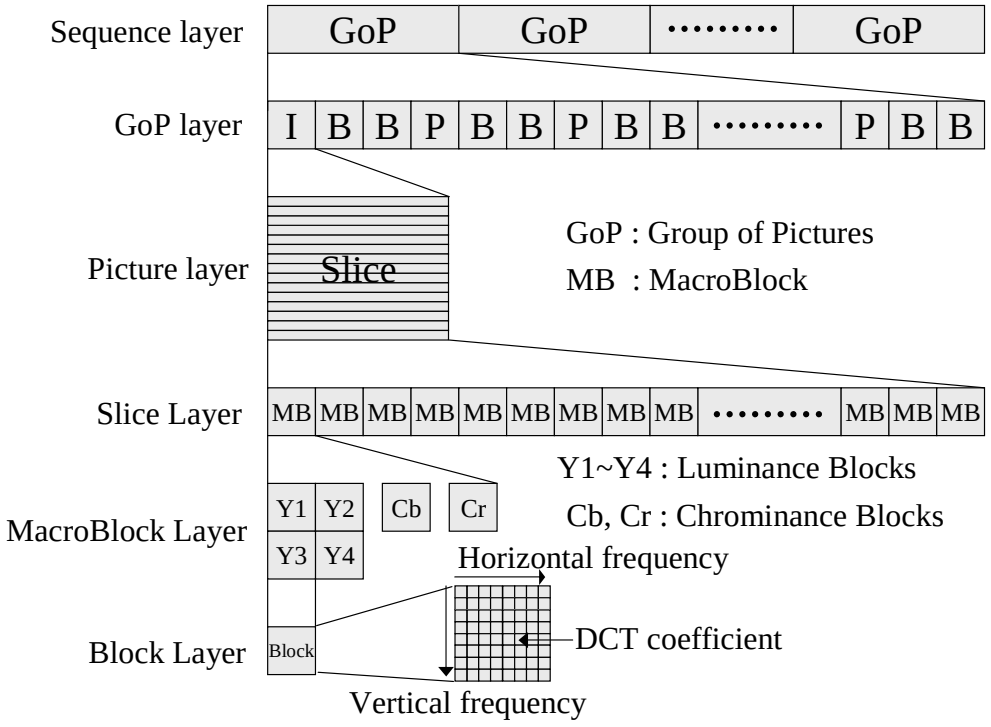


Figure 2: MPEG-2 video data

square and is composed of four 8x8-pixel luminance (Y) blocks and two 8x8 chrominance (Cb, Cr) blocks. Each block is transformed to the frequency domain using discrete cosine transform (DCT). The DCT coefficients in a block are arrranged in ascending order of horizontal and vertical frequency.

2.2 Packetization of MPEG-2 video stream for video-quality adjustment

In order to attain low-latency and on-the-fly video-quality adjustment, a packet-basis mechanism is indispensable. Thus, an MPEG-2 video stream should be segmented into a sequence of independent packets. In a previous work [5], we used a start code to divide a stream into multiple units of data. From the sequence layer to the slice layer, we decided to packetize one stream per slice. For example, an MPEG-2 video stream in a profile of MP@ML, i.e., 720x576 pixels and 30 fps, has 36 slices per picture. If it is coded at the coding rate of 8 Mbps, it follows that each packet amounts to 7 Kbits on average.

2.3 Low-Pass Filter

In [4], our research group compared several video-quality adjustment methods for real-time MPEG-2 video multicast, namely frame discarding, low-pass, and requantization filters. Algorithms were proposed for these video-quality adjustment methods to adapt the video traffic to the specified target rate. Our research group conducted several experiments and concluded that the low-pass filter, which provides rate reduction by progressively eliminating high-frequency components of the video signal, is the most effective method for suppressing the quality degradation and granularity of the rate adaptation. In this implementation, we use the low-pass filter as a video-quality adjustment method. Details of the low-pass filter is given below.

To achieve rate reduction, the low-pass filter eliminates an appropriately determined number of DCT coefficients from the high-frequency ones that comprise a luminance or chrominance block. We call the parameter *low-pass parameter* to the number of DCT coefficients left in each block

after quality adjustment. At the beginning of each GoP, initial low-pass parameter values are set independently for I, P and B pictures, according to the following formulas:

$$l_I = \left\lfloor -6.17329 + 59.7498 r_{G_i} - 112.427 r_{G_i}^2 + 111.905 r_{G_i}^3 \right\rfloor \tag{1}$$

$$l_P = \left\lfloor -11.8626 + 85.5488 r_{G_i} - 159.667 r_{G_i}^2 + 139.499 r_{G_i}^3 \right\rfloor \tag{2}$$

$$l_B = \left\lfloor -71.9536 + 360.75 r_{G_i} - 590.353 r_{G_i}^2 + 353.265 r_{G_i}^3 \right\rfloor, \tag{3}$$

where l_I, l_P, and l_B are low-pass parameter values for I, P and B pictures, respectively. r_{G_i} is the compression ratio for the i-th GoP. These equations were obtained while investigating the relationship between the low-pass parameter and the resultant picture size, and they give an approximation of the low-pass parameter that can produce the desired compression. For example, with $r_{G_i} = 0.5$, they are $l_I = 17$, $l_P = 14$, $l_B = 7$. We obtain r_{G_i} from the formula below:

$$r_{G_i} = \frac{T_i}{G_i - H_i}. \tag{4}$$

G_i is the predicted size for the i-th GoP in bits, which is calculated from the measured size of the $(i-1)$-th GoP, g_{i-1}, by the following formula:

$$G_i = \frac{7}{8} G_{i-1} + \frac{1}{8} g_{i-1}, \quad i \geq 2, \quad G_1 = g_0. \tag{5}$$

H_i is the predictor for the total bits used by header data in the i-th GoP, which is derived from the measured header size of $(i-1)$-th Gop h_{i-1}:

$$H_i = h_{i-1}, \quad i \geq 1. \tag{6}$$

T_i is the number of bits allowed for the current GoP, and it is calculated from a specified target rate R (bps), the number of pictures in a GoP, N (frames), the frame rate F (fps), and an adjustment value a_i:

$$T_i = \frac{R \times N}{F} - a_i - H_i. \tag{7}$$

This formula implies that the rate averaged over a GoP-time is regulated by the target rate. However, the result of per-packet adjustment does not necessarily match the target rate. To make up the balance, we introduce the adjustment value a_i.

The adjustment value a_i is calculated by using:

$$a_i = \sum_{k=\max(0,i-5)}^{i-1} \frac{T_k - f_k}{5}, \tag{8}$$

where f_k is the size of the filtered k-th GoP.

The initial low-pass parameter value is changed dynamically for each of the following intra-macroblocks in the GoP by the following:

$$l_j = \begin{cases} l_j + 1, & r_{G_i} \times o_{MB_{j-1}} - f_{MB_{j-1}} > 0 \\ l_j - 1, & r_{G_i} \times o_{MB_{j-1}} - f_{MB_{j-1}} < 0 \\ l_j, & r_{G_i} \times o_{MB_{j-1}} - f_{MB_{j-1}} = 0 \end{cases}, \tag{9}$$

where l_j is the low-pass parameter to apply to the j-th macroblock, $o_{MB_{j-1}}$ is the macroblock size before filtering, and $f_{MB_{j-1}}$ is the size of the filtered macroblock.

Using the above algorithm, the low-pass parameter value for each macroblock is appropriately determined. By eliminating the specified number of DCT coefficients, it is possible to produce a video stream that has the desired rate R.

3 Functional Design of Active Video-quality Adjustment Node

Our active video-quality adjustment node has the hierarchical structure illustrated in Fig. 3. A router layer is responsible for classification of packets and packet forwarding [8]. An active node layer is for functionalities that make a node “active”. Each functional layer in the active node corresponds to a NodeOS layer, an EE (Execution Environment) layer, and an AA (Active Application) layer [3].

3.1 ANEP

ANEP (Active Network Encapsulation Protocol) [9] is a communication protocol for active networks. User data with an ANEP header is encapsulated in an IP packet using the protocol number 107, which is assigned by IANA (Internet Assigned Number Authority) [10]. The ANEP header format is shown in Figure 4. Type ID is defined by ANANA (Active Networks Assigned Number Authority) as Type ID Registry [11]. For example, ANTS [12] of MIT has Type ID 19, and NetScript [13] of Columbia University has Type ID 31-35.

An ANEP header can have one or more option fields. The first bit of an ANEP Option indicates whether the ANEP Option follows the common global format specified by ANANA or a private

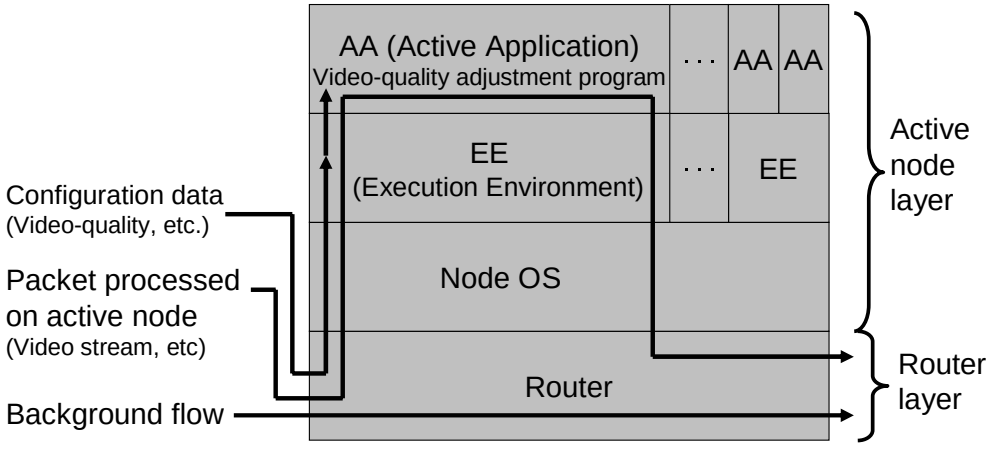
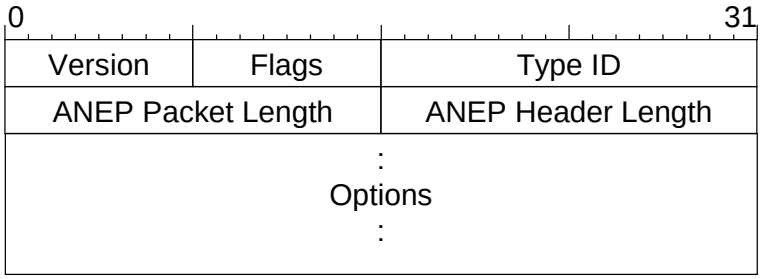
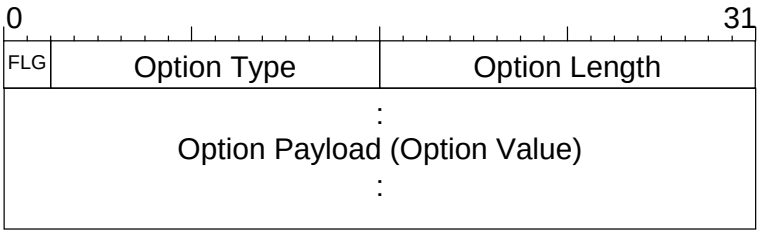


Figure 3: Active video-quality adjustment node



ANEP header



ANEP Option

Figure 4: ANEP header format

format specified by the Execution Environment indicated by Type ID. The ANEP Global Option is analyzed by a NodeOS and the ANEP Private Option is analyzed by an EE. The Option Type specifies the type and format of the ANEP Option payload.

3.2 Active application layer

An AA is a program given by users, network service developers, or network administrators. It is contained in a packet itself or dynamically obtained from other active nodes or servers.

We implemented the video-quality adjustment program described in Subsection 2.3 as an AA. The video-quality adjustment AA is initiated and terminated by an underlying EE and is notified of a target rate by the EE. It has an input channel and an output channel from and to the EE for exchanging MPEG-2 video packets. An AA receives an MPEG-2 video packet through the input channel from the EE, adjusts the quality of the packet in accordance with the specified target rate, and then sends the packet to the EE through the output channel. An AA also has a control channel

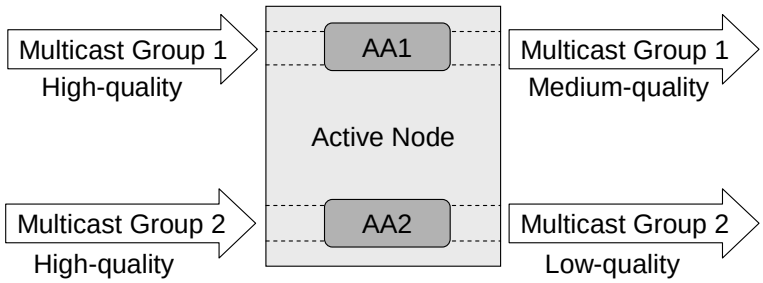


Figure 5: Different videos are quality-adjusted simultaneously.

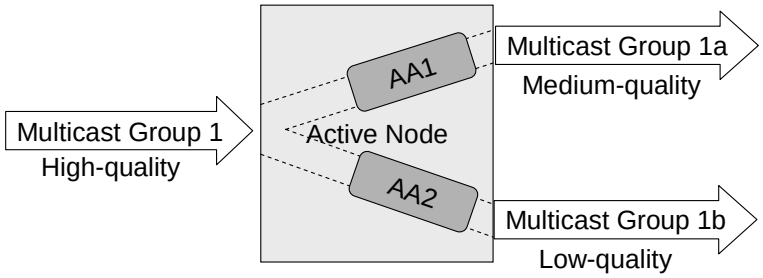


Figure 6: Videos of different quality levels are generated from the same video.

from the EE through which the EE dynamically changes the target rate. As illustrated in Figs. 5 and 6, an AA is bounded to one multicast group. Two or more AAs are initiated when a node adjusts the rate of two or more independent video streams and/or adjustment is made for two or more multicast groups.

3.3 Execution environment layer

An EE is an execution environment or a virtual machine for executing programs on an active node. Furthermore, it provides a programming interface. An EE manages the initiation, execution, and termination of AAs and allocation of resources among AAs. The NodeOS is hidden from AAs by an EE.

In our active video-quality adjustment node, an EE manages the initiation, execution, and termination of video-quality adjustment AAs. An EE receives a video packet from a NodeOS, sends it to an appropriate AA, receives the video packet from the AA after application of video-quality

adjustment, and finally sends it to the NodeOS. The AA used by the EE to exchange a packet is specified in the packet header by a set consisting of a source IP address, a source port number, a multicast group address for sending a video packet, and a destination port number.

3.4 NodeOS layer

A NodeOS manages node resources such as processors, channels, and memory. The NodeOS forwards packets received from the router layer to an appropriate EE on the basis of protocol number, IP addresses, port numbers in an IP header, and/or Type ID in an ANEP header. It also receives packets from EEs and sends them to the router layer.

3.5 Router layer

A router layer classifies packets that require processing by the active node layer from conventional IP packets. It forwards packets received from input ports and the active node layer to the next nodes through appropriate output ports.

If an ANEP header is added to a video packet, a router layer can easily distinguish one video packet from the others by the protocol number in the IP header. However, classification based on protocol number requires a server to generate an ANEP header, attach it to video data, and encapsulate them in an IP packet. Clients need to recognize the existence of an ANEP header from the protocol number, decapsulate video data with the ANEP header from the received IP packet, and detach the ANEP header from video data. If a server or clients cannot handle ANEP, the active node closest to a conventional end-system is responsible for addition or removal of ANEP headers so that active networks and clients can process video streams in an appropriate way.

On the other hand, if ANEP is not used, a router layer should classify the packet on the basis of a set consisting of a sender IP address and port number and a multicast group address and port number. For this purpose, a router layer must maintain and investigate a table of sets of identifiers

every time it receives a packet.

In our implementation, we use ANEP for packet classification to avoid the excessive load introduced by table management and lookup. We should note that our router layer is only responsible for simple packet forwarding. The routing table is statically given by hand.

3.6 Design of signaling protocol

The initiation, termination, and changing target rate of our active video-quality adjustment AA is performed using the ANEP Private Option. Option Types 0, 1 and 2 are assigned to the termination, initiation, and changing target rate, respectively.

To initiate an AA, an ANEP header is attached to the packet. This header uses a private format for the ANEP Option, has an Option Type 1, and contains IP addresses and port numbers of sender and destination multicast groups in the Option Payload. On receiving the packet, an EE initiates an AA for the multicast group if there is no corresponding AA under execution. For this purpose, an EE maintains and investigates a table of sets of identifiers of AAs. The EE informs the AA of changes in the target rate when it receives a packet with an ANEP Option Type 2. If the ANEP header is attached to video data, the new target rate specified in the packet itself is applied to the data. The AA is terminated by the EE after it receives a packet whose ANEP Option Type is 0.

If there is no AA targeted by an ANEP Option Type 0 or 2, the EE will ignore the ANEP Option. After processing all ANEP Options in a video packet, the EE sends the video packet to the appropriate AAs. If there is no appropriate AA, the EE drops the video packet.

4 Implementation of Active Video-quality Adjustment Node

4.1 Intel IXP1200 network processor overview

We implemented our active video-quality adjustment node on programmable network equipment built on an Intel IXP1200 network processor [14-16].

The structure of the IXP1200 network processor chip and the other devices connected to it is illustrated in Fig. 7. The IXP1200 network processor chip, which is surrounded by a thick rectangle in Fig. 7, has a StrongARM core processor and six microengines running at 232 Mhz. An embedded version of Linux with restricted functions runs on the StrongARM. The microengine used is a simple RISC processor optimized for packet forwarding, and there are limitations on its number of registers and executable program size. Each of the six microengines can execute four program threads concurrently, so up to 24 threads can be executed in parallel.

The outer rectangle in Fig. 7 corresponds to the ENP-2505 board [17] of RadiSys Corporation that we used in our implementation. Of the onboard SDRAM, 70 MB is devoted to the Linux system on the StrongARM. Linux uses 62 MB as main memory and the remaining 8 MB as a

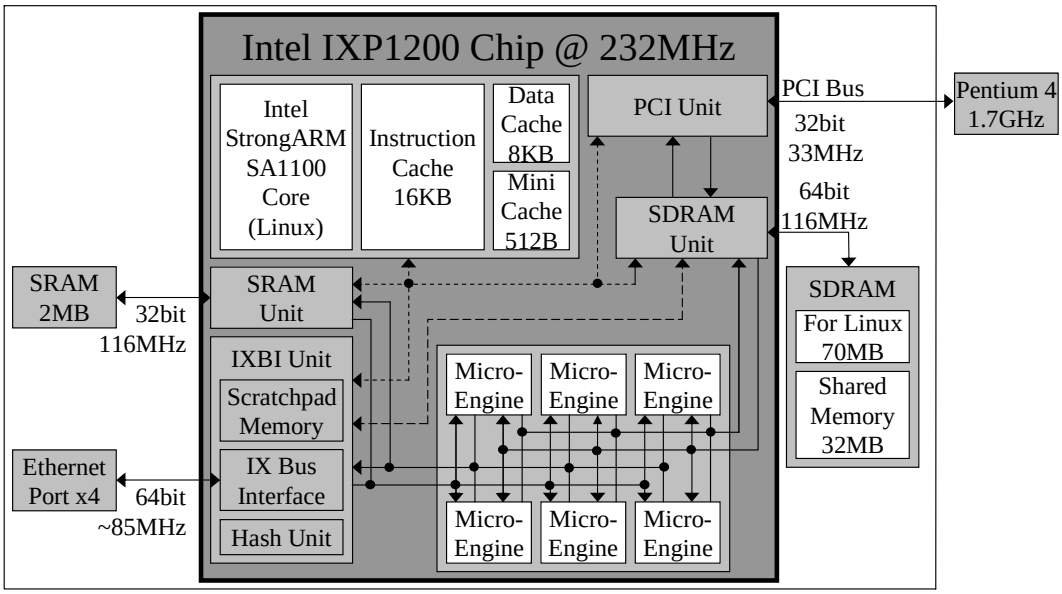


Figure 7: IXP1200 block diagram

ramdisk. Together, the StrongARM and the microengines use 32 MB of the onboard SDRAM, 2 MB of the onboard SRAM, and 4 KB of the internal scratchpad memory. The StrongARM and the microengines can exchange data and information through those shared memories. In order to perform more advanced packet processing, we can insert a board in the computer through a PCI bus and thus attach a Pentium processor to the computer. In our implementation, Redhat Linux 7.2 runs on the Pentium processor. The ENP-2505 board is equipped with four full-duplex 10/100Base-TX Ethernet ports.

4.2 Functional assignment on active video-quality adjustment node

Four functions of our active video-quality adjustment node, i.e., router, NodeOS, EE, and AA, are assigned to microengines, StrongARM processor core, and Pentium processor.

As mentioned in Subsection 3.5, a router layer is responsible for receiving packets from the NodeOS and input ports, classifying ANEP packets from conventional IP packets, and forwarding packets to the NodeOS and output ports. Microengines are simple but optimized for packet processing. The number of threads, four, is the same as the number of the Ethernet ports that the ENP-2505 is equpped with. Therefore, a router layer is implemented on two microengines. One microengine is responsible for processing packets received from the Ethernet ports and classifying video packets from the other conventional IP packets. The other microengine is responsible for forwarding packets received from the microengine for input process and the NodeOS to the appropriate Ethernet port. Each thread on a microengine is bound to an Ethernet port. In Section 5, we evaluates the validity of this implimentation.

Those packets with the protocol number 107 are passed to the NodeOS, and the other packets are sent to the microengine for output processing. Packet exchange among those two microengines and the StrongARM is achieved using the SRAM and the shared SDRAM. There are slots of 2 KB for packets on the SDRAM. Each slot has a dedicated area on the SRAM, called a packet descriptor, which is used to maintain a slot. We make a list of packet descriptors to contruct a packet

queue. Queues are built for each of the four Ethernet ports and the StrongARM. When the input microengine receives a packet with an ANEP header, it adds a packet to the tail of the queue of the StrongARM. Then the StrongARM dequeues the packet from the queue, processes its payload, and puts the packet into the queue of one of Ethernet ports. Finally, the output microengine reads the packet from the queue and sends it out to the next queue.

Packet exchange temporarily uses an SDRAM area of 2 KB per packet. Packet descriptors on the SRAM are used to manage the free memory area of the SDRAM. Each descriptor corresponds to an SDRAM area of 2 KB. In addition, the packet descriptors can make a transmission queue by using a link list of the packet descriptors. The transmition queues are addressed to ports or the StrongARM. A packet descriptor's addresses of heads and tails of transmition queues are managed on the SRAM. In packet exchange, the sender enqueues the packet to the tail of the queue and the receiver dequeues the packet from the head of the queue.

We consider three assignments of NodeOS, EE, and AA layers to processors.

4.2.1 Active video-quality adjustment with StrongARM

In the implementation using only StrongARM, all of the NodeOS, EE, and AA functionalities are running as user programs of the Linux system on the StrongARM. Packets are exchanged among the NodeOS and the router layer on microengines over the SRAM and the shared SDRAM. The NodeOS uses Linux kernel functions and drivers to access those shared memories. Packet exchange among layers on the StrongARM, that is, between the NodeOS and the EE and between the EE and the AA, are performed over socket communications based on the localhost IP address. An active video-quality adjustment AA is initiated and terminated by the EE using the Linux system's call functions.

4.2.2 Active video-quality adjustment with StrongARM and microengines

Processing DCT coefficients in video-quality adaptation is a simple but high-volume operation. As an example, 75% of video data coded at 8 Mbps are from the block layer, and 56% of the time is spent in processing DCT coefficients when a low-pass filter is implemented on a PC. Communications among the StrongARM and the microengines are performed through the scratchpad memory and the shared SDRAM.

First, the StrongARM moves the data needed for processing the block layer on the microengine to the shared SDRAM area. The data include, for example, macroblock type, low-pass parameter, macroblock pattern, and DCT coefficients. Then, it writes the SDRAM address in the scratchpad memory. Each microengine thread periodically checks the scratchpad memory, with an address assigned to the thread. If the SDRAM address is written in the scratchpad memory by the StrongARM, the microengine thread reads the shared memory address and writes "0" on the same address in the scratchpad memory to notify the StrongARM that it is processing the given layer data. The microengine thread obtains data from the SDRAM and additional information such as the quantization table from the scratchpad memory. The microengine processes the DCT coefficients and writes the results on the given SDRAM address. Then the microengine notifies the StrongARM of the completion of processing by using the scratchpad memory with another address assigned for this. The StrongARM takes note of this completion by periodically checking the scratchpad memory, reads the shared memory address, and writes "0" to the same address in the scratchpad memory. Finally, the StrongARM reads the results of processing DCT coefficients from the shared SDRAM area.

For the video-quality adjustment, we assign 16 MB of the shared SDRAM area and four microengines. The number of microengines used for each active video-quality adjustment AA is specified beforehand.

4.2.3 Active video-quality adjustment with Pentium processor

In this architecture, an ENP-2505 board is attached to PC, and all layers except the router layer are running in Redhat Linux 7.2 on a Pentium processor. The relaying program on the StrongARM is resonsible for exchanging packets between the router layer on the microengines and the NodeOS on the Pentium processor by using the SRAM, the shared SDRAM, and the socket with the localhost IP address.

5 Evaluation

In this section, to verify the practicality and applicability of the active video-quality adjustment within a network, we evaluate the accuracy of the rate adaptation and the throughput of video-quality adjustment.

Figure 8 illustrates our experimental System 1. It consists of a video server, two clients, and our active video-quality adjustment node. The video server reads video data, divides them into slices, attaches a UDP header to a slice, attaches an ANEP header to the UDP datagram, encapsulates them in an IP packet, and sends the packet to a multicast group. The active video-quality adjustment node receives a packet from the port that the server is connected to, applies active-node processing to the packet, and sends it to one or two clients connected through the other ports. The target rate can be changed on a GoP-by-GoP basis. The active video-quality adjustment node detaches an ANEP header from an IP packet. For a wider deployment of services and systems, users must be able to easily receive the service without being troubled with the extra configuration of client systems. Therefore, in our experimental systems, clients only receive video packets in UDP/IP format and process them in a conventional manner. Using System 1, we first evaluate the accuracy of rate adaptation. Then we consider the throughput of the video-quality adjustment.

The other experimental system, System 2 illustrated in Fig 9, is used to verify the performance of the active video-quality adjustment node when there exists background traffic that requires conventional packet forwarding of the node. In this system, two switching hubs and a packet generator are introduced. Two of the four Ethernet ports of the active video-quality adjustment node are connected to the packet generator. The third port and its input link are shared among video traffic originating from the video server and background traffic injected by the packet generator. The fourth port is used by both the client and the packet generator.

We used an MPEG-2 PS stream whose average rate is 8 Mbps. A GoP has 15 frames and thus lasts 0.5 seconds.

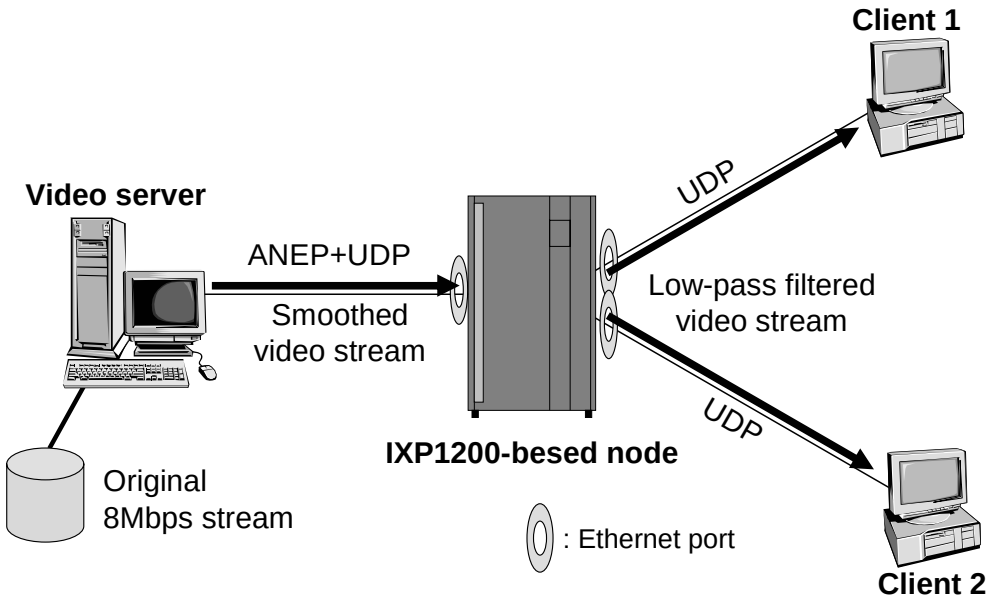


Figure 8: Experimental System 1

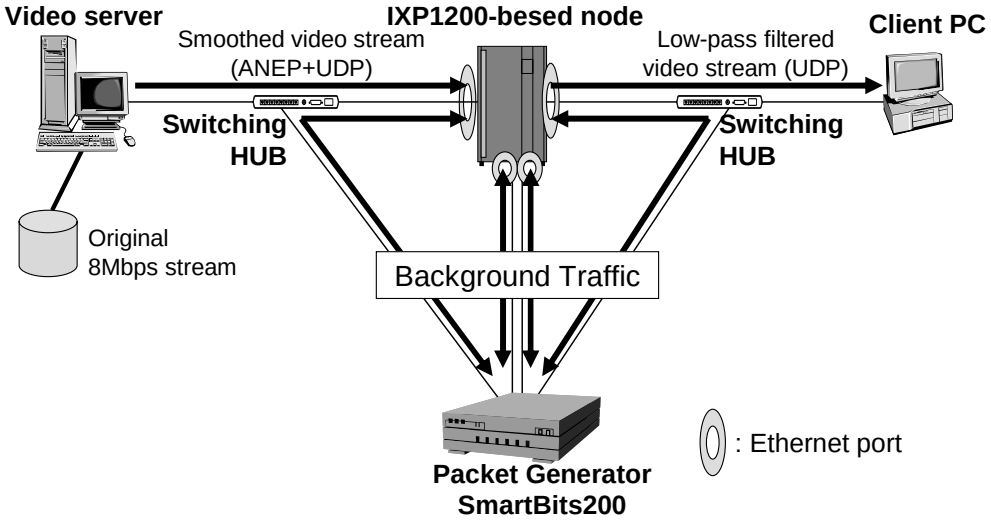


Figure 9: Experimental System 2

5.1 Accuracy of rate adaptation

We first applied the active video-quality adjustment to an original video stream of 8 Mbps sent from a server to a client. An active video-quality adjustment AA was implemented on the StrongARM and four microengines. We should note here that our active video-quality adjustment node cannot perform the video-quality adjustment at the rate of 8 Mbps, as will be shown later. Therefore, in evaluating the accuracy of the rate adaptation, an original video stream of 8 Mbps injected into the active video-quality adjustment node was sent out from the video server at the rate of 1 Mbps by a smoothing buffer so that all video packets could be received, adjusted, and sent by the active video-quality adjustment node. Figure 10 shows the rate variations of video streams adjusted to 2, 4, and 6 Mbps. The horizontal axis shows time in a video stream. This figure indicates that our system can successfully adapt the video traffic to the desired rate. The variations observed, which sometimes exceed the target rate, are due to the slice-based packetization and the variable length coding of MPEG-2.

In the next experiment, a video stream was sent from the server to two clients that belong to different multicast groups. Accordingly, two AAs were running on the node, each of which was assigned two microengines for processing DCT coefficients. The results are summarized in Fig. 11. For video packets from 0 to 14.5 seconds in video time, one AA with the specified target rate of 6 Mbps is initiated and executed on the active video-quality adjustment node. Then, an ANEP header attached to the video packet leading the 30th GoP, i.e., 15 seconds in video time, changes the target rate to 2 Mbps. At the same time, by having another ANEP Option for a new destination multicast group, the ANEP header initiates another AA with the target rate of 5 Mbps. The new AA is terminated by the server at 35 seconds in video time. From 25.5 to 35 seconds in video time, the target rate of the former AA is gradually increased by 0.1 Mbps per GoP (0.5 sec). As shown in Fig. 11, our active video-quality adjustment node can appropriately adapt the video rate to the desired rate on a packet-by-packet basis, even when two AAs simultaneously generate

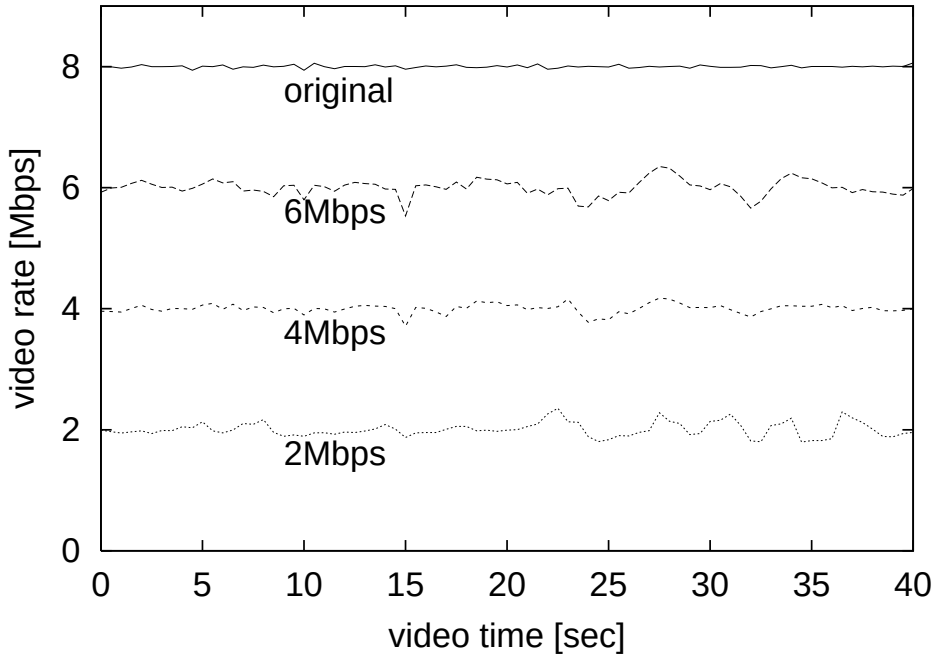


Figure 10: Video-quality adjustment at fixed target rate

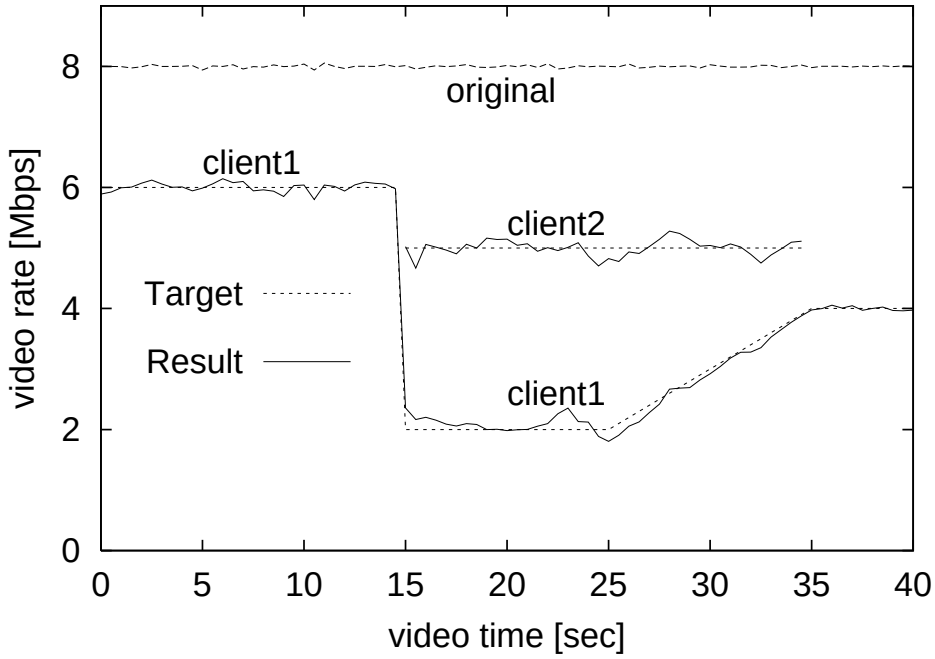


Figure 11: Video-quality adjustment at variable target rate for two clients

video streams of different quality levels from one video stream and the target rate is drastically or gradually changed.

5.2 Throughput of video-quality adjustment

We first evaluated the throughput of the active video-quality adjustment node by using experimental System 1 and comparing combinations of processors. We set the target video rate to 4 Mbps, and only one client was used. We started with an injection rate of 50 Kbps and gradually increased it by 50 Kbps until the client observed packet loss during a 100-second video stream. Taking into account packet loss due to unexpected and accidental problems in the server, network, and client, we conducted four additional experiments when any packet loss had been observed. When no packet was lost in more than three out of the four experiments, the active video-quality adjustment node was regarded as capable of processing a video stream at that injection rate. The results are summarized in Table 1 and Fig. 12.

The throughput of video-quality adjustment using only StrongARM was 0.8 Mbps.

For the combination of StrongARM and one microengine, we conducted four experiments by changing the number of threads running on the microengine. Although the microengine takes part in video-quality adjustment, employing only one thread leads to lower throughput then that of the StrongARM-based implementation due to the overhead of exchanging data between the StrongARM and the microengine. To take advantage of the multi-processor architecture of a network processor and to improve performance, three or four threads should be used.

Consequently, we can expect further performance improvement by distributing takes among more microengines. However, as shown in the table and figure, employing more than three microengines, each of which uses four threads, adds little to performance. This is due to the shortage of memory bandwidth of the SDRAM on the scratchpad memory, the computational power of the StrongARM, or some other cause.

For communications to the StrongARM, an idle thread periodically investigates the scratchpad

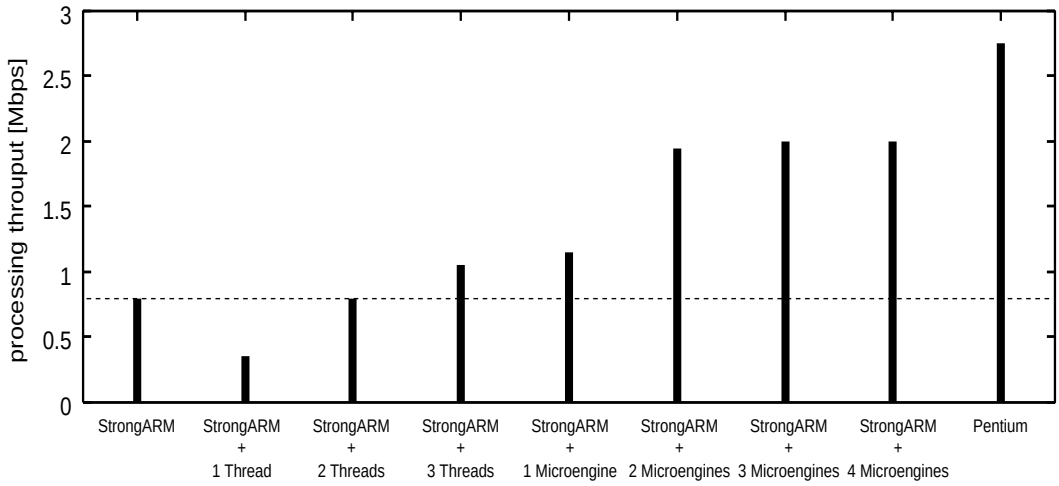


Figure 12: Throughput of video-quality adjustment

Processors used	Throughput (Kbps)
StrongARM	800
StrongARM and 1 microengine thread	450
StrongARM and 2 microengine threads	800
StrongARM and 3 microengine threads	1050
StrongARM and 1 microengine (4 threads)	1150
StrongARM and 2 microengines	1950
StrongARM and 3 microengines	2000
StrongARM and 4 microengines	2000
StrongARM and 2x2 microengines (2 streams)	1000x2
Pentium	2750

Table 1: Throughput of video-quality adjustment

memory as described in 4.2.2. A working thread also refers to the scratchpad memory to process DCT coefficients. Therefore, if the bandwidth of the scratchpad memory were insufficient and thus formed a bottleneck, throughput would be expected to decrease due to the competition among idle and working threads for memory. However, we observe no such performance degradation in Fig. 12. Therefore, we conclude that the scratchpad memory is not a bottleneck.

In the router layer, the shared SDRAM is used for exchanging packets between two micro-engines for input and output processing as described in 4.2. If the bandwidth of the SDRAM were a bottleneck, the throughput of the video-quality adjustment would deteriorate as the number of packets increased. However, as will be shown later, the throughput of the active video-quality adaptation does not decrease when much background traffic, which needs conventional packet forwarding, is injected into the node.

Consequently, we can assume that the computational power of the StrongARM impedes the performance improvement with multiple microengines. However, we cannot verify this conclusion because the embedded Linux system on the StrongARM has a limited set of instructions and cannot provide a way of measuring the computational load.

As the next experiment, we initiated two AAs, each of which is assigned two microengines and adjusts a video stream for different multicast groups. The throughput attained for each AA was the same at 1.00 Mbps. The total throughput of the node is equal to that of the case when a single AA with four microengines processes a video stream.

The throughput attained with a Pentium processor was 2.75 Mbps. Since the utilization of the Pentium processor was 20% or less, we assume that either the computational power of the StrongARM or the bandwidth of the PCI bus is a bottleneck.

Finally, we introduced background traffic using the experimental System 2 illustrated in Fig 9. The packet generator, SmartBits 200 of Spirent Communications [18], generates conventional IP packets that do not need to be processed by the active node layer. We first examined the maximum throughput of the active video-quality adjustment node that only relays packets. The

packet generator injected conventional IP packets into the node through four full-duplex Ethernet ports. The attained throughput on each port was 95 Mbps (Fig. 13). Thus, the router layer on two microengine has the capacity of 760 Mbps, which is close to the full wire rate. Here, the throughput takes into account Ethernet preambles, frames, and CRCs.

On the basis of this result, video traffic smoothed down to 2 Mbps, and background traffic of 93 Mbps is injected into the node thought the Ethernet port connected to a hub. The AA was configured to process the video data of 8 Mbps at target rate of 4 Mbps. In addition, the node handles three other flows of 95 Mbps (Fig. 14). During 50-second experiments, we did not observe any packet loss in any of the five flows. Therefore, we conclude that two microengines are sufficient for the router layer and are capable of handling video and a conventional packet at near-wire speed.

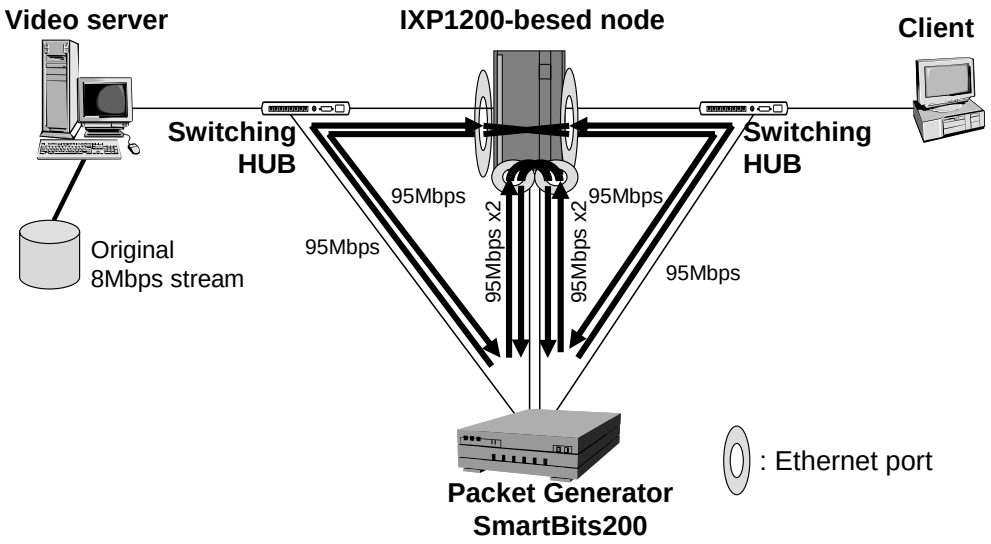


Figure 13: Routing throughput without video quality adjustment

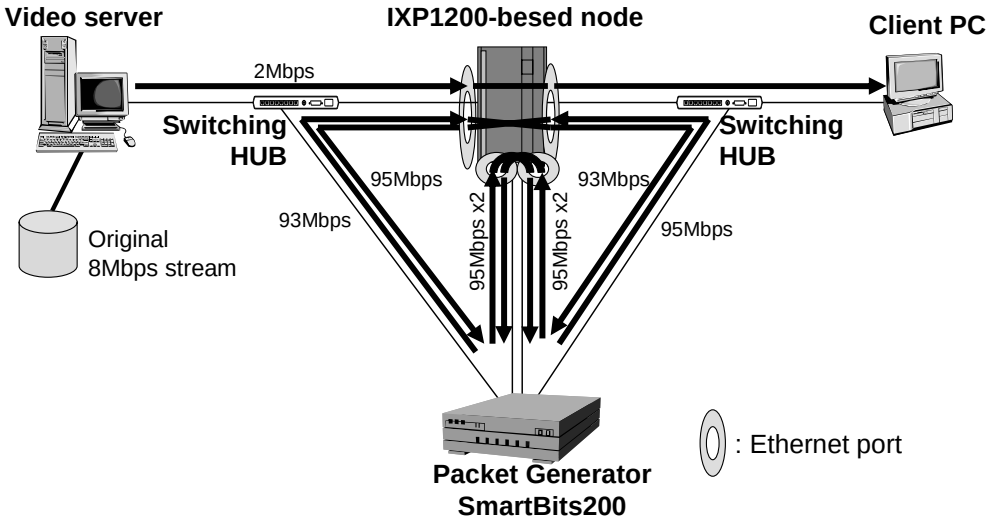


Figure 14: Routing throughput with video quality adjustment

6 Conclusions

In this thesis, we designed and implemented an active video-quality adjustment node on a network processor-based equipment by using active network technologies. We conducted experiments and verified the practicality of the video-quality adjustment within a network. The active video-quality adjustment node can adapt the video rate to the dynamically changing requested level on a packet-by-packet basis. Furthermore, the active video-quality adjustment node can simultaneously generate video streams of different quality levels from the same video stream.

At this time, our implemented system cannot perform video-quality adjustment at a practical rate. It is necessary to improve the throughput by, for example, reducing the load on the StrongARM Dynamic injection of an AA used for video-quality adjustment. In addition, extending the node functions is also a subject for future research.

The architecture of the IXP-1200 is sutable for implementing an active node. The StrongARM general purpose processor can be used for the NodeOS, the Execusion Environment, and the Active Applications, that process packets in a highly intelligent way. The microengines can take the responsibility for simple packet processing such as classification and forwarding, and some parts of the AA. However, limitations on the procesing power of the StrongARM and the microengines restrain the throughput of the active video-quality adjustment node. Introducing more powerful processors that run at higher rate and execute more complicated program enables truely practical and useful active video-quality adjustment node.

Acknowledgements

I would like to express my sincere appreciation to Professor Hideo Miyahara of Osaka University, who introduced me to the area of multimedia networks, in particular the issues discussed in this thesis.

None of the work in this thesis would have been possible without the support of Professor Masayuki Murata of Osaka University. It gives me great pleasure to acknowledge his invaluable assistance.

I would like to express my sincere appreciation to Associate Professor Naoki Wakamiya of Osaka University. None of the work in this thesis would have been possible without him.

I would like to express my deep gratitude to Mr. Akio Tomobe of Intel Corporation, who denoted the IXP1200 network processor. The implementation of this thesis would also not have been possible without him.

Thanks are also due to Professor Shinji Shimojo of Osaka University, who gave me much useful advice.

I am also indebted to Associate Professors Hiroyuki Ohsaki and Go Hasegawa and Research Associates Shinichi Arakawa and Ichinoshin Maki of Osaka University who gave me helpful comments and feedback.

Finally, I thank my many friends and colleagues in the Department of Informatics and Mathematical Science of Osaka University for their support.

References

[1] H. Akamine, N. Wakamiya, and H. Miyahara, “Heterogeneous video multicast in an active network,” *IEICE Transactions on Communications*, vol. E85-B, pp. 284–292, Jan. 2002.

[2] J. Smith, K. Calvert, S. Murphy, H. Orman, and L. Peterson, “Activating networks: A progress report,” *IEEE Computer*, vol. 32, pp. 32–41, Apr. 1999.

[3] K. Calvert, “Architectural framework for active networks,” *RFC Draft*, July 1999.

[4] H. Akamine, K. Nakada, N. Wakamiya, M. Murata, and H. Miyahara, “Implementation and evaluation of video filtering mechanisms for real-time multicast,” *Technical Report of the IEICE (NS 2001-50)*, pp. 13–18, June 2001.

[5] T. Yamada, N. Wakamiya, M. Murata, and H. Miyahara, “Implementation and evaluation of video-quality adjustment for heterogeneous video multicast,” in *Proceedings of The 8th Asia-Pacific Conference on Communications (APCC 2002)*, (Bandung), pp. 454–457, Sept. 2002.

[6] ISO/IEC 13818-2, “Information technology - Generic coding of moving pictures and associated audio infomation : Video,” *International Standard*, May 1996.

[7] ISO/IEC 13818-1, “Information technology - Generic coding of moving pictures and associated audio infomation : Systems,” *International Standard*, Dec. 2000.

[8] N. Shalaby, L. Peterson, A. Bavier, Y. Gottlieb, S. Karlin, A. Nakao, X. Qie, T. Spalink, and M. Wawrzoniak, “Extensible Routers for Active Networks,” in *DARPA Active Networks Conference and Exposition (DANCE)*, (San Francisco, CA), pp. 92–116, May 2002.

[9] D. S. Alexander, B. Braden, C. A. Gunter, A. W. Jackson, A. D. Keromytis, G. J. Minden, and D. Wetherall, “Active network encapsulation protocol (ANEP),” *RFC Draft*, July 1997.

[10] “Internet assigned number authority (IANA).” available at `http://www.iana.org/`.

[11] “Active netwroks assigned numbers authority (ANANA) type ID registry.” available at `http://www.isi.edu/~braden/anana/info/typeid.txt`.

[12] “Active network transport system（ANTS）.” available at `http://www.sds.lcs.mit.edu/activeware/`.

[13] “Netscript.” available at `http://www.cs.columbia.edu/dcc/netscript`.

[14] Intel Corporation, *Intel IXP1200 Network Processor Family Hardware Reference Manual*, Dec. 2001.

[15] Intel Corporation, *Intel IXP1200 Network Processor Family Programmer’s Reference Manual*, Dec. 2001.

[16] T. Spalink, S. Karlin, and L. Peterson, “Evaluating Network Processors in IP Forwarding,” Tech. Rep. TR–626–00, Department of Computer Science, Princeton University, Nov. 2000.

[17] Radisys Corporation, *ENP-2505 Hardware Reference*, Mar. 2002.

[18] “Spirent communications.” available at `http://www.spirentcom.com/`.