

Enhanced End-system Support for Multimedia Transmission over the Differentiated Services Network

Haisang Wu^{1*}, Huai-Rong Shao², Xing Li¹, Wenwu Zhu², Ya-Qin Zhang²

¹Dept. of Electronic Engineering
Tsinghua University, Beijing, P.R.China 100084

²Microsoft Research China
No. 49 Zhichun Road, Beijing, P.R.China 100080

Abstract. Proposed as a scalable solution to provide soft Quality of Service (QoS), Differentiated Services (DiffServ) architecture enables service providers to offer each customer a range of services that are differentiated on the basis of performance. However, up to now little work discusses how to provide effective communication support for multimedia applications over the differentiated services network. In this paper we present a new enhanced communication approach at the end system to support distributed multimedia applications. Two new mechanisms are highlighted in our approach: enhanced communication stack to support differentiation within one flow, and the network-awareness provision for applications at the end system. Our approach improves the capability of both error resilience and flexible rate control for transmitting compressed multimedia bitstreams, particularly those using scalable coding technologies. We also develop an object-based video transmission system as an application instance to take advantage of the enhanced communication approach. Experimental results demonstrate the effectiveness of our proposed methods.

1. Introduction

Current Internet provides best-effort (BE) service to end-users and does not make any service quality commitment. However, most multimedia applications are sensitive to available bandwidth and delay experienced in the network. To satisfy these requirements, two frameworks have been proposed by IETF: the Integrated Services (IntServ) [1, 6], and the Differentiated Services (DiffServ) [10].

The IntServ model provides per-flow QoS guarantee and RSVP (Resource ReserVation Protocol) [5] is suggested for resource allocation and admission control. However, the processing load is too heavy for backbone routers to maintain states of thousands of flows. DiffServ model is designed to scale to large networks and gives a class-based solution to support relative QoS. The main idea of DiffServ is to

* Corresponding author. E-Mail: hswu@public.bjnet.edu.cn Tel: (86-10) 62785814~525; Fax: (86-10) 62785933.
Mail address: Room 224 Central Building, Tsinghua University, Beijing 100084 P. R. China.

minimize state and per-flow information in core routers by placing all packets in fairly broad classes at the edge of network. Core devices perform differentiated aggregate treatment of these classes based on the marking performed by the edge devices. A single byte in each packet is used to do this, called the DS byte (the Type of Service byte in IPv4 and the Traffic Class byte in IPv6), which can be set by the end station or ingress edge router to indicate the class of service desired. Since it is highly scalable and relatively simple, DiffServ model may be promising to dominate the backbone of the next generation Internet in the near future.

One of the most important research topics in multimedia networking is how various applications such as multimedia streaming and video conference can take full advantage of differentiation capability provided by DiffServ. Inherently, different kinds of information in the compressed bitstream may have different importance levels for the decoder to reconstruct the multimedia playback data. For instance, in MPEG4 shape and motion information is more important than texture for a P frame [15]. If shape and motion information is lost during transmission, the decoder cannot reconstruct the P frame successfully. However, if partial texture information is lost without the loss of shape and motion, it is still possible to reconstruct the P frame with acceptable quality [11, 15]. Another example is scalable coding such as layered approach [2, 4, 8] or Fine Granularity Scalability [14], in which enhancement layers are much less important than the base layer information. It seems that there is a natural mapping between different kinds of information with different classes of network packets, but unfortunately today's differentiation approaches can only support differentiation among flows, i.e., all packets belonging to the same flow have to be mapped to the same packet class. To satisfy the requirements of scalable multimedia, we propose an enhanced communication stack that can provide differentiation within one flow.

Currently, layered transmission and multicasting approach for scalable media can also differentiate base layers and enhancement layers by using multiple IP sessions. However, it is complicated for network and end system to maintain multiple sessions semantically bundled. Particularly, it is difficult to control the synchronization between these semantically bundled layers [2, 15]. On the other hand, with the functionality of differentiation within one flow, multiple-layer information can be transmitted within one IP session. Moreover, different kinds of information within one layer can also be easily differentiated in our approach.

Network-awareness support at the end system is another key issue for multimedia transmission over DiffServ network. With the development of network technologies and users' requirements, today's multimedia applications (such as MPEG4 applications) become more and more complicated. It is required that resources should be allocated and managed in a cooperative way, which means network resources can be dynamically coordinated among applications and flows within the applications adaptive to network fluctuations. We propose network-awareness support at the end-system to solve this problem successfully.

The rest of the paper is organized as follows. Section 2 presents the mechanism of differentiation within one flow. Section 3 discusses the network-aware support at the end system. In Section 4, we describe the experiments and the performance testing results of our approach. Section 5 introduces an application instance running on the proposed communication stack. Finally we conclude this paper.

2. Differentiation within One Flow

2.1. Design of the Enhanced Communication Stack

For complex multimedia applications such as MPEG4 programs, usually there are multiple objects of the same media or different media [15]. If these objects have different QoS requirements, generally each object is served by an individual IP session or even several sessions in the case of multiple layers. In general it is difficult for the end-system and network to maintain so many sessions for one application. By default, packets are marked based on a mapping from the service type associated with a flow, and all packets within one flow have the same marking value. However, if differential marking within a flow is supported, layers belong to the same object or different objects can be multiplexed into one IP session.

A possible problem resulting from differentiation within one flow is disorder of packets within the same flow, but this exists even without the function of differentiation within one flow because of the connectionless characteristic of IP protocol. To support the new differentiation functionality, an extension needs to be made on the protocol stack of the end-system.

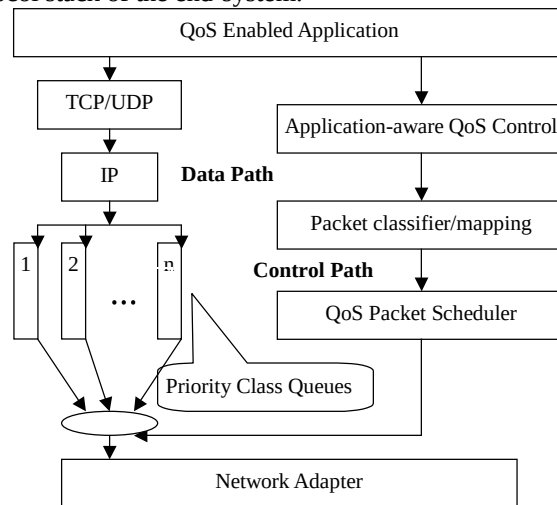


Fig. 1. Logical Architecture of the enhanced communication stack

We propose a new marker mapping mechanism in the host protocol stack to support differentiation within one flow. We introduce the multiple queue mechanism at the end-system and each queue buffers packets with a particular priority. When the IP header is added at the IP layer, the priority is mapped to the DSCP (DiffServ Code Point) byte.

To achieve our proposal, we intend to find the different priorities of packets according to DSCP value marked by QoS-aware applications, and then packets are classified into different classes according to specific rule. All this can only be

performed through TCP/IP stack within operating system. Application should call the kernel to set the DSCP value in IP packets header, and the kernel should finish all following queuing and scheduling.

After kernel sorts the packets generated by QoS-enabled applications into different classes according to their DSCP value, DiffServ scheduler in operating system sends out different classes of packets according to certain scheduling algorithm. Thus Priority Class Queues are used to implement packets scheduling.

The logical architecture of proposed system is shown in Figure 1. It shows a host system, including applications, operating system and hardware (network adapter). In the operating system, the TCP/IP stack will perform the mechanism proposed and send the resulting packets into network adapter. It can be seen from Figure 1 that the data path and the control path are separated, and the control capability is improved.

2.2. Implementation of the Enhanced Communication Stack

We implemented our enhanced communication stack based on the open source code of Microsoft Research IPv6 stack release 1.4 (MSRIPv6 release 1.4).

Network protocols in Windows NT are dynamically loadable device drivers, much like any other device driver in Windows NT. It is possible to add a new protocol to the system by writing two new components: a kernel-level driver (tcpip6.sys) that exports the TDI interface and uses the NDIS interface, and a user-level helper (wship6.dll) to support access to the driver via sockets [7].

We made needed modification to MSRIPv6, most of which is done within these two components. Our main purpose is to add modules to set Traffic Class Value in IPv6 packet header and to queuing the packets according to their TC values. As a prototype, this would not add to the overhead of kernel.

3. Network-aware End System

Figure 2 shows a framework example of the streaming server with intelligent resource control and management for multimedia applications. This framework considers the transmission of multiple-object video programs and other types of media such as audio and data. Each video object is compressed first and corresponding elementary stream is generated. Then information within each elementary stream is classified based on importance and assembled into packets with different DiffServ classes. Network Monitor is responsible for estimating the available network bandwidth dynamically through probing or feedback-based approach. Packet Forwarder forwards the packets to the network. We do not discuss these two blocks in details in this paper. The other functional components are described as follows.

- Priority Mapping and Marking Agent

This component is responsible for the interaction between applications and the DiffServ networks. It assigns DSCP marks to packets and maps them to the corresponding DiffServ classes.

- Application Collaborator

The Application Collaborator is responsible for resource coordination among multiple objects within one application and among multiple applications. It receives information from Application Profiles, Remote Users Interactions, and Network Monitor to make the decision. In addition, the Application Collaborator tells how to map packet priorities from individual encoders into network classes. The receivers can interact with the server through user-level signaling.

- Application Profiles

This component records the semantic information of the applications such as which media and flows are included in an application and their relative importance levels.

- Remote User Interactions:

A user can interact with the video player or the server in several ways such as mouse clicking, mouse moving, fast forward, fast backward, object zoom-in, object zoom-out, add or delete. Some of these interactivity behaviors require dynamic adaptation of the bit rate of each video object and dynamic resource allocation coordination among multiple video objects. In object-based video multicast applications, different clients can have different views and interactions for the same video.

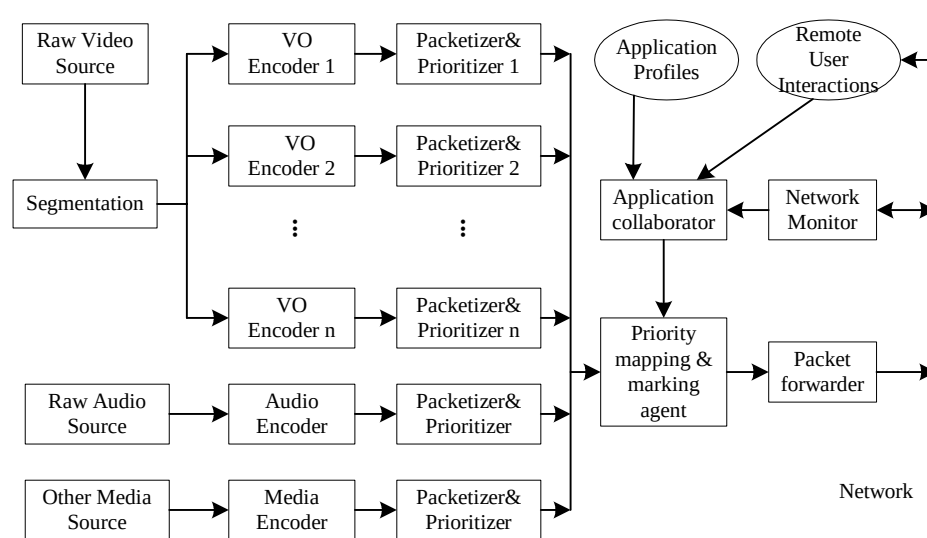


Fig. 2. Multimedia Communication Framework in the End-system

4. Performance testing of the enhanced communication Stack

4.1. Testing Environment

The experimental testbed can be seen in Figure 3, which is an IPv6 testbed.

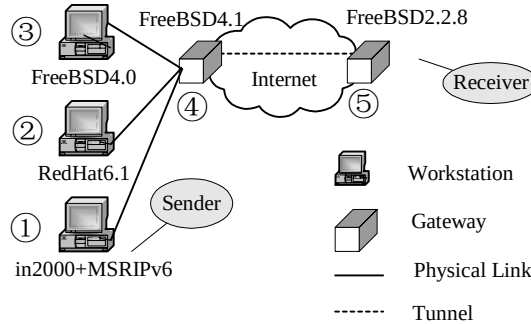


Fig. 3. Testing Environment

Box 1 is a PC installed with Win2000 operating system and MSRIPv6 stack. The others are installed with Linux (Redhat) and UNIX (FreeBSD). All the boxes are dual-stack, i.e., IPv4 and IPv6 stacks. The “Internet” in the figure is a network comprised of several PCs, routers and switches. Box 4 is the default gateway of box 1, box 2 and box3. Box 5 is connected to box 4 via IPv6 over IPv4 tunnel, which is virtual link and different from physical links among the other boxes.

Box 1 is the sending machine and box 5 is the receiving machine in the testing. As we made our modification in sending module and leave the receiving module untouched, we only need to testify the sending functions, and the box 5 with UNIX installed should not affect the testing results.

With the original MSRIPv6 stack we are not able to set the value of TC field, but the TC value can be set with the modified stack. Thus the testing can be performed under four conditions. The composition is shown in table 1.

Table 1. Testing Condition

	Not setting TC (So TC=0)	Setting TC Randomly
Original Stack	√ Condition ①	Not available
Modified Stack	√ Condition ②	√ Condition ③

We define the condition 1 as “not setting TC value with the original stack”, and condition 2 as “not setting TC value with modified stack”, and so on. In the following paragraphs, we use “test 1” to represent “test carried out under condition 1”, and it is the same with “test 1” and “test 3”.

In the testing procedure, the sender, box 1 sends packets to the receiver, box 5 consecutively. Before sending, each packet’s TC field is marked with one number randomly chosen, and the packet content is filled with the same value as TC field in its header.

4.2. Experiment 1: Delay

One aspect we consider is delay. We perform testing under the three conditions respectively, consecutively sending 40 packets to see the delay. The result is shown in figure 4 (left). At first the network link is with relatively light load. We find that the delay in test 1, 2 and 3 has no radical difference, and the delay time of each packet is around 1 millisecond except for a few bizarre points, which can be easily seen in the curves.

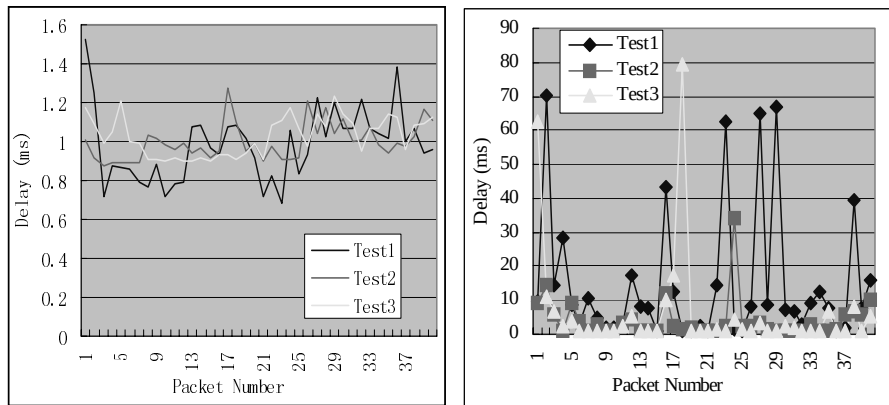


Fig. 4. Results of Delay Experimenting. (Left: delay testing result without congestion; Right: delay testing result in congestion condition)

Then we perform the same testing with heavy link load. Several other services assumed to consume much link resource such as FTP are started to create the link congestion. The testing result is shown in figure 4 (right). This time the delay of each packet and the variation within each condition are apparently much larger than before. But in the results we still cannot find the distinction among test1, test2 and test3.

4.3. Experiment 2: Packet Loss

The other aspect we consider is packet loss. The first problem that we care is whether adding queues in the stack to schedule different classes of packets would have negative effect on the performance of the stack. As described before, when applications do not set TC value, they leave this field to be zero, which is the default value in the original stack. Thus in case of not setting TC value, i.e., in condition 1 and condition 2 respectively, it would be more reasonable for us to compare results with different IPv6 stacks.

We create congestion with some traffic such as FTP to consume link resource. Some packet losses do occur, but very tiny, for our methods in testing environment is not enough to simulate the complex conditions of WAN. Both in test 1 and test 2 we tried 10 times to count packet loss in each time. The result is shown in figure 5 (left).

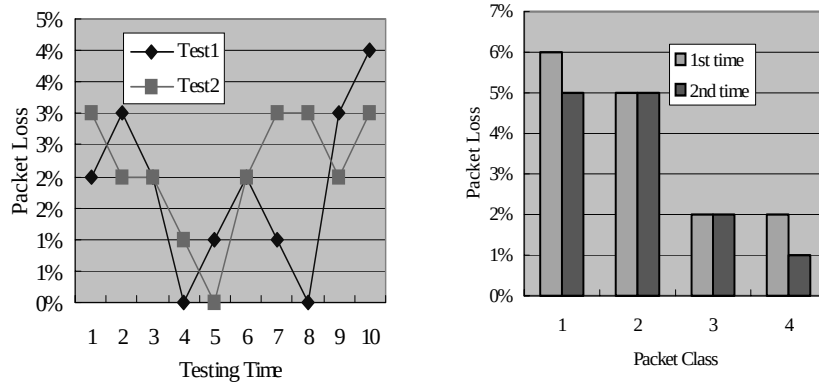


Fig. 5. Results of Packet Loss Experiment (Left: packet loss of different stacks Right: packet loss of different classes)

As the figure tells, there is no much difference in the results of test 1 and test 2, which may mean that queues added into the stack for class-scheduling do not impact too much negative effect on performance of the stack. But the testing environment and methods are too simple, and we cannot control sending speed of either the applications or the hardware (the network card). So the result in this figure does not mean that our modification would not affect the overall performance negatively in other conditions.

The other problem that we care is whether our mechanism can schedule different classes correctly, especially in congestion. We need to prove that with the scheduling mechanism, in congestion conditions packets with higher priorities can be sent out first. We performed two times of testing under condition 3. 100 Packets with TC value randomly set are sent out from box 1 to box 5 in a network link with very heavy load, and in box 5 the packet loss of each class is calculated. In this testing, we defined four classes; in which class 4 has the highest priority while class 1 has the lowest priority. Results of two times of testing are shown in figure 5 (right), from which we can see that packet losses of class 3 and class 4 are lower than those of class 1 and class 2. In the first time of testing, packet loss of class 1 is 6% and that of class 2 is 5%, but packet losses of class 3 and 4 are both 2%. In the second time, packet losses of class 1 and 2 are both 5%, but that of class 3 is 2% and 1% of class 1. This proves that the scheduling mechanism is effective in packet scheduling.

5. Application instance running on the enhanced communication stack

It can be seen from Section 4 that the enhanced communication stack can differentiate different classes of packets without much performance impairment. However, the enhanced stack can greatly benefit the transmission of scalable multimedia.

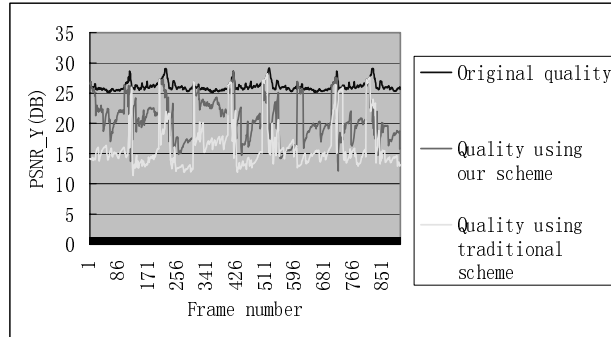


Fig. 6. Video quality comparison for Bream at 11.7% packet loss rate (actual bit rate=168kbps; original bit rate=187kbps).

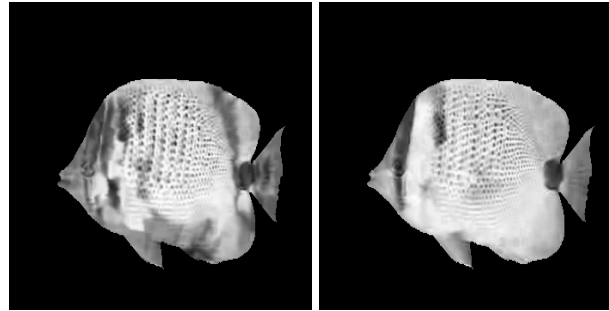


Fig. 7. Video frame example (Number: 150) Left: traditional approach Right: the proposed scheme.

We implemented a MPEG4 video streaming system and run it on the proposed enhanced communication stack within a simulated DiffServ network. In this system, we proposed a new bitstream classification, prioritization, and packetization scheme in which different types of data such as shape, motion, and texture are re-assembled, assigned to different priority classes, and packetized into different classes of network packets provided by DiffServ. Taking advantage of the enhanced communication stack, our scheme distinguishes not only different kinds of frames, but also different types of information within the same frame. Readers can refer [15] to obtain details of our new transmission scheme. Besides our scalable transmission approach, for the sake of comparison we also implemented the traditional approach in which the bitstream is packetized with no information re-organization/prioritization and all packets have a fixed size (600bytes). Figure 6 and 7 show the comparison results example of our approach and the traditional approach for Bream. It can be seen that our proposed transmission approach based on the enhanced communication is much better than the traditional one.

6. Conclusion

In this paper we presented a new enhanced communication approach at the end system to support distributed multimedia applications. Two new mechanisms were highlighted in our approach: enhanced communication stack to support differentiation within one flow, and the network-awareness provision for applications at the end system. Our approach improves the capability of both error resilience and flexible rate control for transmitting compressed multimedia bitstreams, particularly those using scalable coding technologies. We have implemented a prototype and used an experimental testbed to study the effect of our approach. Experimental results demonstrate that our approach can be an effective means for packet classification and scheduling for multimedia transmission.

7. References

- [1]S. Shenker, R. Braden, D. Clark, "Integrated services in the Internet architecture: an overview", RFC 1633, June 1994.
- [2]McCanne, "Receiver-driven layered multicast", SIGCOMM Symposium on Communications Architectures and Protocols, Palo Alto, California, August 1996.
- [3]D. Hoffman and M. Speer, "Hierarchical video distribution over internet-style networks", ICIP'96, Lausanne, Switzerland, September 1996.
- [4]X. Li, M.H. Ammar, and S. Paul, "Layered video multicast with retransmission (LVMR): Evaluation of error recovery", Proceedings of NOSSDAV, May 1997.
- [5]R. Braden, L. Zhang, and S. Berson, et al, "Resource Reservation Protocol - Version 1 Functional Specification", RFC 2205, September 1997.
- [6]J. Wroclawski, "The Use of RSVP with IETF Integrated Services", RFC 2210, September 1997.
- [7]Richard P. Draves, Allison Mankin, Brian D. Zill, "Implementing IPv6 for Windows NT", Proceedings of the 2nd USENIX Windows NT Symposium, Seattle, WA, Aug. 3-4, 1998.
- [8]D. Sisalem, "QoS Control using Adaptive Layered data transmission", IEEE International Conference on Multimedia Computing and Systems, 1998.
- [9]K. Nichols, S.Blake, F. Baker, D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", RFC2474, December 1998.
- [10]S. Blake, M. Carlson, and E. Davies, et al, "An Architecture for Differentiated Services", RFC 2475, December 1998.
- [11]"MPEG-4 video verification model version 13.0", ISO/IEC JTC1/SC29/WG11/N2687, March 1999.
- [12]B. Carpenter, D. Kandlur, "Diversifying Internet Delivery", IEEE Spectrum, Nov. 1999.
- [13]M. Hemy, U. Hengartner, P. Steenkiste, T. Gross, "MPEG System Streams in Best-Effort Networks", Packet Video'99, New York.
- [14]S. Li, F. Wu, Y. Zhang, "Study of a new approach to improve FGS video coding efficiency", ISO/IEC JTC1/SC29/WG11/m5583, December 1999, Maui.
- [15]H. Shao, W. Zhu and Y. Zhang, "User-aware Object-based video multicast over the Internet", 10th International Workshop for Packet Video, May, 2000, Sardinia, Italy.