Video Streaming in a DiffServ/IP Multicast Network

Roland Balmer, Torsten Braun, and Manuel Günter
Institute of Computer Science and Applied Mathematics
University of Bern
Neubrückstrasse 10
CH-3012 Bern
{balmer|braun|mguenter}@iam.unibe.ch

Abstract: *Video streaming services need Quality of Service (QoS) when broadcasted over the Internet. Differentiated Services are appropriate in large IP networks and managed by central QoS Brokers. A QoS Broker can not exactly set up the requested QoS without the information about the video stream. To exchange this information between the QoS Broker and the Service Provider a common protocol is required. The Broker Signalling Protocol has been developed to communicate between brokers. The flexible syntax of the protocol also allows to use it in a simpler case, i.e. between end systems and QoS Brokers.*

Keywords: *Broker Signalling Protocol, QoS, Multicast, Video Streaming*

1. Introduction

When a user pays for a video streaming service the transmission quality must be perfect. In the Internet, Quality of Service (QoS) support is required. The IETF developed Differentiated Services to support QoS in large IP networks. Inside the network routers may be controlled by QoS Brokers. Each QoS Broker is only responsible for the routers that are inside the domain of an Internet Service Provider (ISP). For QoS support outside of the own network, the QoS Broker must negotiate with the neighbour providers [7].

The only contact of a user to the Internet is his local ISP, which has to negotiate with other ISPs on behalf of the client. This requires that a QoS Broker handles two types of communication relations. First, it must be able to negotiate with the neighbour ISP and second, it must handle the request from the clients. Our approach is to handle the two cases with the same protocol.

The two cases can use a common core for the basic negotiation. They differ in the level of abstraction. The user has only knowledge of basic parameters about his traffic such as the bandwidth and whether the traffic is real-time. For the broker the task is more difficult. It needs to negotiate with other ISPs and needs more technical information, e.g. the end-to-end delay. The same protocol is used for both cases, but we need to adjust it to the special requirements.

In this paper a broker-to-broker protocol described in section 2 is adjusted to the new requirements of the streamed video scenario that is introduced in section 3. Section 4 presents the modifications and section 5 explains the mapping of the broker-to-broker protocol parameters to Differentiated Services. Section 6 concludes the paper.

2. Broker Signalling Protocol

The Broker Signalling Protocol (BSP) was developed as a protocol that can be used for the communication between two or more Internet Service Providers (ISPs). The goal is to establish Service Level Agreements (SLA) between the corresponding ISPs and provide the requested Quality of Service [4].

2.1. Basic BSP Specification

The initial goals of the BSP were as follows
1. It should be extensible and flexible, because the technical descriptions of a service vary for the existing Differentiated Services classes. It should also be easy to integrate new services.
2. The BSP must handle synchronization problems. The network state may change during negotiation.
3. The protocol must be secure. The negotiation must be authenticated and may be encrypted.

To guarantee the extensibility the basis of the BSP must be simple, but still allow the definition of arbitrary data types. The protocol uses a simple byte stream structure and is hierarchical. Each parameter is represented by an *object* that has a variable length and can contain a set of other objects. For such a hierarchy, start and end delimiter bytes are used to delimit the entities. Furthermore, an escape byte is needed to escape the delimiters that occur in the data.

The delimiters are:

Entity start:	}
Entity end:	{
Entity escape:	\

For the BSP we use two classes of entities. Figure 1 shows the relation between the basic and the derived classes.

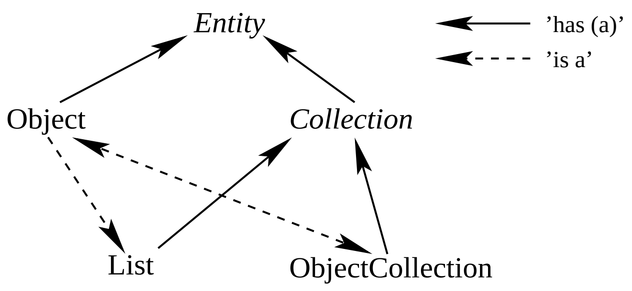

Figure 1: Protocol Entity Hierarchy.

The first byte after the start delimiter distinguishes the classes, the content byte identifies the type. From the abstract base class Collection only the derived classes are used.

The simplest subclass is the List. The content byte is L. The next byte indicates the data type of the following data. For an ASCII stream it is A and numbers are represented with H for integer variables, that are converted to a hexadecimal number, and D (double) for real variables. The rest of the list represents the value.

The second class, which is derived from the Collection class, is the ObjectCollection. The content byte is C. The ObjectCollection holds a variable number of Objects.

The most important class is the Object. The content byte is O. Objects are used to define the parameters. It contains the name and the value. Consider the following example:

$\{Ox\{LH1\}\{LAname\}\{LAvalue\}\}$

The first byte after the content byte (x) indicates the status of the object. The two important flags are P for *please complete* and F for *fixed.* The next three entities are Collections. The first is an identifier for the object, the second is the name of the parameter and the third is the value of the object. The first two Collections are always Lists, the third can be a List or an ObjectCollection.

To use the BSP it is only required to define the parameters. The first implementation has been performed for inter-ISP SLA establishements. More than 30 objects have been defined for that purpose [3].

2.2. Message Exchange in BSP

Each BSP message consists of BSP objects. Figure 2 shows the messages exchange to set up a connection.

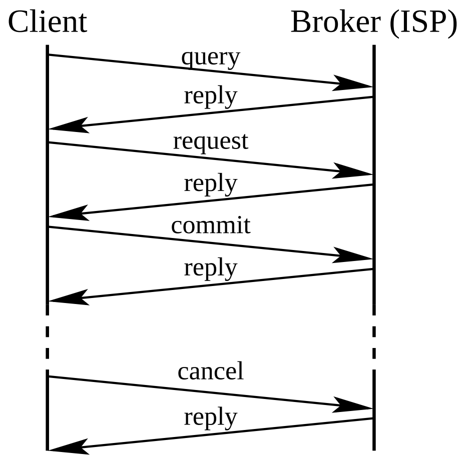

Figure 2: Timetable for message exchange.

Three steps are required to get the requested service. First, the client sends a *query* with the desired parameters. The broker checks if this *query* is possible. The result is a *reply* with the proposal of the broker. In this phase there can be more than one *query-reply* exchange. Ultimately, the *reply* contains an object with all needed parameters. The customer can re-use this object as a complete service request. With such a *request* the client reserves the resources. The following *commit* starts the service.

The use of the messages *request* and *commit* seems to produce overhead. The problem is, that the result of a *request* can have more than one valid *reply.* The *commit* message has the task to determine which *reply* is used.

In general, the service has an end-time and will stop automatically, but with the *cancel* message we can stop the service on demand.

With this definition of the BSP we can fulfil the three initial requirements:
1. Extensibility: For BSP extensions, we only need to define new parameters and adjust the interface. If a parameter is unknown to a BSP-API, the parameter will be ignored.
2. Synchronization: The structure of the BSP is useful to negotiate between two brokers. The parameters can be adjusted to any environment. Also a

two phase commitment *request-commit* is integrated to solve broker synchronization problems.
3. Security: Security support has been introduced on two levels. First, the signature object can be added to sign the message. In this case the authentication of the submitted message can be guaranteed. To ensure privacy the entire byte stream can be encrypted or can be sent over a secure connection.

3. A Streamed Video Scenario

An example of a client-broker communication is a streamed video scenario developed in the Stream-Com project [11]. The idea is to provide a pay per view service over public broadcast media. The basic idea is, that a client can select a movie or some live event transmission from a catalogue. The features of the service are predefined and contain among other things the time of transmission, the price, and the quality.

The research goal is to transmit a video stream over a broadcast medium. There are special requirements for this. First, because the service is not for free, a paying mechanism is needed. This is provided by the ticketing system [2] implemented by the Ticket Broker. Second, the copyright of the information must be protected. This is the task of the Content Provider. Third, the transport needs some Quality of Service (QoS) support. This is the task of the ISP and will be controlled by a QoS Broker.

Figure 3 shows the basic scenario, with the three different commercial partners (Content Provider, Ticket Broker and QoS Broker) and a client. The

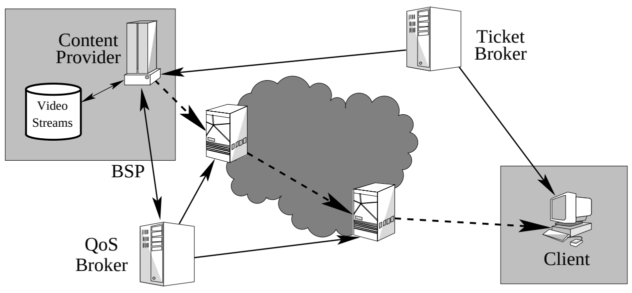

Figure 3: Base scenario for StreamCom.

three entities, Content Provider, Ticket Broker and QoS Broker are independent from each other and can be parts of different networks.

The communication between the Ticket Broker and the Content Provider does not require special network resources and will not be considered in this context. In the following, two other issue are of interest: First, the characteristics of the video stream must be known. These characteristics will be used to specify the communication requirements. The second issue, the connection between the Content Server and the QoS Broker will be discussed in the next section.

3.1. Video Stream Transport

Figure 4 shows a simple example of a video stream transmission [6] [5]. To protect the copyright of the

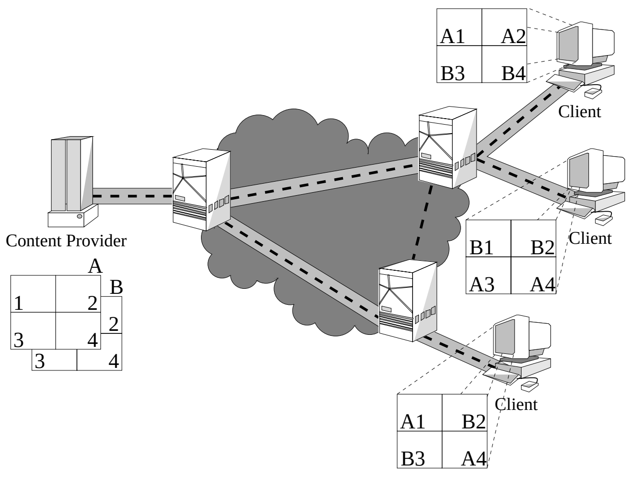

Figure 4: Transport of video streams.

video stream owner each picture is divided into p frames, in the example p is set to 4. Each frame is water-marked and encrypted with a unique key. The client receives a set of frames that it must recombine to obtain the complete video stream. To ensure that each client receives a unique copy of the video stream, the same frame is sent f times, in the example f is set to 2. With this method we can create f^p different video streams. In the example we can create $f^p = 2^4 = 16$ different video streams. For 16 individual video streams only the double bandwidth is required. If only the relevant parts of the picture would be duplicated and water-marked this factor can further be reduced.

This pre-processing makes only sense, when the video stream is sent over a broadcast medium like IP multicast or satellite. In this case the number of different streams is the number of the parallel pictures p multiplied with the number of frames f per picture plus one stream for bulk part of the picture $(f*p+1)$ or in the example $f*p+1 = 2*4+1 = 9$. That means that for 16 different video streams only 9 connections are established.

For traffic over an IP network, IP multicast [8] will be used. Each client receives his set of frames and must join the corresponding multicast groups.

3.2. Resources Reservation for Video Streams

For all streams from the Content Provider to the clients resources must be reserved. Otherwise, the quality of the incoming video on the client side would be unacceptable. Because the quality is part of the service, the Content Provider is responsible for requesting QoS support.

Monitoring and allocation of the resources is the task of the QoS Broker. The QoS Broker controls all routers within its domain. All these routers provide Differentiated Services [1]. The border routers have the task to check and classify the traffic whether matching with the predefined parameters.

The QoS Broker uses an internal management system to control each router. Over this interface, the QoS Broker can change the routers configurations, but can also obtain network status information [10].

The QoS Broker receives the information about the video streams directly from the Content Provider, which is also the only instance that knows the clients. The communication has been realized with an adapted BSP implementation.

4. BSP in a Streamed Video Scenario

The previous section described the streamed video scenario from the view of the produced traffic. From this description the following requirements can be extracted:

1. Between the Content Provider and the ISP a long-term contract can be established. Also, the Content Provider will produce a continuous load on the network.
2. The collaboration between the Content Provider and QoS Broker is over a long period. Therefore, it is sufficient to establish a secure connection once per session.
3. The parameters such as bandwidth, video stream characteristics and transmission time are defined in advance.

The first requirement simplifies the communication between the Content Provider and QoS Broker. A long term contract does not need to be negotiated via BSP. Information about the mode of payment or more information about the Content Provider is exchanged in advance. There is no need to negotiate it for each session or single video stream. This reduces the number of BSP objects we need to define. Also, the Content Provider will produce a continuous amount of traffic. The network of the ISP must support this traffic, otherwise it must increase the capacity or terminate the contract with the Content Provider. The ISP should minimize over-provisioning.

The second requirement reduces the number of objects per message. The first step of the Content Provider is to connect to the QoS Broker and to establish a secure connection. To do this the Content Provider can use key parameters that are negotiated in advance. Because the predefined parameters must be updated from time to time, the corresponding objects are needed for the protocol.

The third requirement reduces the number of messages. The Content Provider knows the characteristics of the video stream and requests the minimum of the resources needed. Otherwise, if the requested amount of resources would be larger the benefits of the service would be reduced. If it would be smaller the quality of the video stream would be worse and the conditions of the video service would be broken. For the Content Provider it is sufficient to receive a *yes* or *no* for a resource request.

With these conditions the description for the BSP in the video stream scenario can be divided into three different groups: Global and long term objects, video stream description, and the messages.

4.1. Global and Long Term Objects

The global and long term objects, which are used in the streamed video scenario, are listed in Table 1. These objects are not often used in messages, but

`customer`	`customer_key`
`provider`	`provider_key`
	`encrypt_key`

Table 1: Global and long term BSP objects.

they are needed to open and use the communication. They have the following meaning:

customer: The IP address to describe the customer. In this case it is the Content Provider.

provider: The IP address to describe the provider. In this case it is the QoS Broker.

costumer_key: The key that the costumer uses to sign the message.

provider_key: The key that the provider uses to sign the message.

encrypt_key: The key that is used to encrypt the messages.

From these five objects, only the first one is used regularly. The Content Provider uses the *customer* object when it sets up the connection to the QoS Broker. After that it is no longer needed, because the connection has been established.

The four others object are used to define the connection between the Content Provider and the QoS Broker. When transmitted in a BSP message, they are used to redefine the parameters for the connection.

The three keys are used for security. The *provider_key* is used by the QoS Broker to sign a message. The signature is created over the whole message and the key. It is then added to the message. In the same way the Content Provider uses *costumer_key* to sign the messages. The *encrypt_key* is used to encrypt the messages.

The fourth object, the *provider* object, is usually not used in a message. The only use of this object is for the infrequent case that the QoS Broker will change.

4.2. Objects Defining the Video Stream

The idea behind the service description is that the user does not need the full technical background. A normal user of the protocol must be able to describe the traffic on a higher level and must be able to describe the most important parameters. But it is not the goal that the user of the service can define all existing parameters that are used inside a DiffServ router. Therefore, the BSP objects support some levels of abstraction. Table 2 lists the existing objects.

src_addr	bandwidth	start
src_port	ex_bandwidth	end
dest_addr	realtime	start_offset
dest_port	loss	end_offset
protoid		
flowid		
status		

Table 2: BSP objects to define a video stream.

The meaning of these objects are:
src_addr, src_port: define the source of the video stream, indicating the IP address and the corresponding port.
dest_addr, dest_port: define the destination of the video stream.
protoid: define the higher layer protocol that is used to transport the video stream.
flowid, status: These objects are used to identify the video stream and to transmit the status.
bandwidth, ex_bandwidth: define the average bandwidth and the possible peak rate of the video stream.
realtime: A flag to define whether it is a real-time data stream.
loss: A flag to define whether the data stream is sensitive to loss of packets.
start, end: Explicit start and end time.
start_offset, end_offset: Relative start and end time, in relation to the time of the request or for end_offset in relation to the start time.

All these objects are used to define the video stream. The first four objects define the IP stream, for which the resource reservation is needed. The *protoid* object is used by the QoS Broker to adjust the flow description.

The next four objects describe the characteristics of the video stream. The two bandwidth objects are used to specify the bandwidth, and the peak rate of the traffic. The *realtime* flag indicates to the QoS Broker that the traffic is vulnerable to delays and the *loss* flags indicate that no packet should be lost.

The last four objects are used to define the time when the service should start. *Start* and *end* define the exact duration of the service. The two others define the time of the service relative to the actual time. The second possibility is intended for short duration services.

With all the described objects we can use the BSP to request and initiate a resource reservation for all video streams. But there is also a way to simplify the communication, which is described in the next section.

4.3. Messages in the Video Streaming Scenario

Because the traffic characteristics of the video stream should not be changed the Content Provider has only to know whether the service is possible or not. For the Content Provider it does not make sense to change the parameters of the video stream, because this would result in a lower quality for the entire service and this should be avoided. Table 3 lists the messages that are used for video stream scenario.

query	reply
request	error
cancel	
setup	

Table 3: The messages in the video stream scenario.

There is one new message in addition to the Broker-Broker case. The *setup* message is used to initiate the exchange of new parameters for the connection. With this message the global and long term parameters can be upgraded. Normally this is only done by the QoS Broker.

The other difference between a broker-broker communication and the video stream scenario is the lack of a *commit* message. The function of the *request-commit* pair in the broker-broker case, is replaced by the *request* message only. This can be done, because the Content Provider does not negotiate the parameters for the resource reservations and negotiation is bilateral. If a *query* is successful, then the *request* is possible and the reservation can be initiated. An other reason is that the start and end times for the service are fixed, because the service has a defined time when the video stream is transmitted. Therefore, the negotiation can be done in advance.

The other messages have the same meaning in both cases. The *query* is always used to check the availability of resources. *Cancel* can always be used to terminate a resource reservation for a flow. That might happen if nobody receives the particular video stream any longer. *Reply* is used to mark the response and *error* indicates that an error has occurred.

With these messages, it is possible to set up the resource reservation without an extensive communication.

4.4. BSP-API implementation

The BSP-API has been implemented as a prototype. For our video stream scenario a broker-broker communication is not needed yet. The task of the API is to translate the incoming queries and requests to a format that the QoS Broker can use. To do the translation, the API uses an internal parser. To add new services to the API the parser must be adjusted.

5. Mapping from BSP to DiffServ

In BSP the description of a flow is done with a set of parameters. These parameters do not request a special service, but describe the characteristics of the traffic. This allows the ISPs to support it in there own way. In a DiffServ network there will be two different service classes: assured forwarding (AF) and expedited forwarding (EF).

With the parameters from the BSP-API the QoS Broker must assign the flow to one of these DiffServ classes. From all the parameters describing a flow (Table 2) only *bandwidth, ex_bandwidth, realtime, loss* and *protoid* have an impact on the nature of the service. *Bandwidth* and *ex_bandwidth* can be translated directly to the correct configuration parameters. The translation of the others need more effort. Table 4 gives an overview of the possible sets

realtime	*loss*	*protoid*	
0	0	UDP/TCP	(1)
1	0	TCP	(2)
1	0	UDP	(3)
0	1	UDP	(4)
1	1	UDP	(5)

Table 4: The messages in the video stream scenario.

of parameters and the following list provides the corresponding interpretations:

1. If neither *realtime* nor *loss* is set, the flow can be handled by a low-priority assured forwarding service class. In this case the choice of the protocol does not matter.
2. A real-time flow with TCP depends on low delay and low jitter. Low delay can be achieved by short queues, but they must be large enough to buffer traffic bursts. Due to conflicts of an EF traffic shaper at the ingress router [9] with the TCP congestion control mechanism this flow has to be mapped to a specially configured AF class.
3. A real-time UDP flow can be handled perfectly with AF. AF can provide good delay and jitter values even for irregular flows with large bursts. The drawback of this PHB is a small chance of packet loss.
4. A loss-sensitive UDP flow should be transported with EF. The excess bandwidth limitation has to be set very carefully to prevent packet loss during bursts.
5. A real-time and loss-sensitive UDP flow should also be transported with EF. This flow is regulated in advance, so that the bandwidth will not exceed the negotiated limit. Again burst protection is a critical issue.

In addition it has to be emphasized that the loss-sensitive flag does not make sense for TCP flows because TCP automatically retransmits lost packets. Therefore, this flag could be used to indicate other TCP flow properties.

6. Conclusion

The presented Broker Signalling Protocol is very flexible. Initially, it has been designed to negotiate the Service Level Agreements between network providers. The negotiation between two ISPs is more expensive than the communication between a normal user and his ISP. For this application the definition in

[3] is too heavy and it creates communication overhead. But the protocol is designed to be adjusted to the requirements of new services. The streamed video scenario is such a new service that needs existing objects but also defines new objects.

There are two ways of implementation. First, the two possible communication types are supported via two different APIs. The broker-broker communication is separated from the client-broker communication. But the advantage of BSP for the QoS Broker is that the two communication types can be handled over the same interface. The new objects need only to be added to the existing BSP-API and only the parser needs to be adjusted. This process can be done for each new service. In most cases only an identifier for the new service must be added and the BSP-API will translate the information in the corresponding format for the QoS Broker.

One limitation in the basic scenario still exists. Each multicast group member is treated in the same manner. There is no difference between clients that pay or do not pay for the service. Both receive the same resources, but the data received by the non paying users are useless since they are unable to decrypt it. In a future stage the allocation of resources will be restricted to the paying clients only.

7. Acknowledgements

This work has been performed in the frameworks of the project Commercialization of Streamed Information - StreamCom (SPP ICS 2000 Project 5003-057755) and the project Advanced Network and Agent Infrastructure for the Support of Federations of Workflow Trading Systems - ANAISOFT (SPP ICS EICom Projekt 5003-057753) which are funded by the Swiss National Science Foundation (SNF).

8. References

[1] Yoram Bernet, Andrew Smith, Steven Blake, and Dan Grossmann. A conceptual model for DiffServ routers, March 2000. Internet Draft.

[2] Levente Buttyan and Naouel Ben Salem. A Payment Scheme for Broadcast Multimedia Streams, July 2001. 6th IEEE Symposium on Computers and Communications, Hammamet, Tunisia.

[3] Emmanuel Granges. Bandwidth Broker for Differentiated Services. Diploma Thesis, IAM, University of Berne, March 2000.

[4] Manuel Günter and Torsten Braun. Evaluation of Bandwidth Broker Signaling. In *Proceedings of the International Conference on Network Protocols ICNP'99, IEEE Computer Society*, pages 145–152, November 1999. ISBN 0-7695-0412-4.

[5] Dimitri Konstantas and Dimitris Thanos. COiN-Video: A Model for the Dissemination of Copyrighted Video Streams Over Open Networks, September 2000. Object System Group Technical Report 2000.

[6] Dimitri Konstantas and Dimitris Thanos. Commercial Dissemination of Video over Open Networks: Issues and Approaches, September 2000. Object System Group Technical Report 2000.

[7] British Columbia Institute of Technologie. Bandwidth Broker high level design, 1999. http://www.merit.edu/working.groups/i2-qbone-bb/.

[8] Sanjoy Paul. *Multicasting on the Internet and its Applications.* Kluwer Academic Publishers, Feburary 1998.

[9] Günter Stattenberger and Torsten Braun. Performance Evaluation of a Linux DiffServ Implementation, April 2001. submitted for publication.

[10] Günter Stattenberger, Torsten Braun, and Marcus Brunner. A Platform-Independent API for Quality of Service Management, June 2001. IEEE Workshop on High Performance Switching and Routing, Dallas, USA.

[11] StreamCom. http://cui.unige.ch/OSG/projects/streamCom/, 2000.