Distributing MPEG movies over the Internet using programmable networks

Dan He
COMPOSE group
INRIA/LaBRI
33402 Talence Cedex, France
Dan.He@inria.fr

Gilles Muller*
COMPOSE group
INRIA/LaBRI
33402 Talence Cedex, France
Gilles.Muller@inria.fr

Julia L. Lawall
DIKU, University of Copenhagen
Universitetsparken 1
2100 Copenhagen Ø, Denmark
julia@diku.dk

Abstract

Distributing video over the Internet is an increasingly important application. Nevertheless, the real-time and high bandwidth requirements of video make video distribution over today's Internet a challenge. Adaptive approaches can be used to respond to changes in bandwidth availability while limiting the effect of such changes on perceptual quality and resource consumption. Nevertheless, most existing adaptation mechanisms have limited scalability and do not effectively exploit the heterogeneity of the Internet.

In this paper, we describe the design and implementation of a MPEG video broadcasting service based on active networks. In an active network, routers can be programmed to make routing decisions based on local conditions. Because decisions are made locally, adaptation reacts rapidly to changing conditions and is unaffected by conditions elsewhere in the network. Programmability allows the adaptation policy to be tuned to the structure of the transmitted data, and to the properties of local clients. We use the PLAN-P domain-specific language for programming active routers; this language provides high-level abstractions and safety guarantees that allow complex protocols to be developed rapidly and reliably. Our experiments show that our approach to video distribution permits the decoding of up to 9 times as many frames in a heavily loaded network as distribution using standard routers.

1. Introduction

The significant bandwidth requirement of video makes large-scale video broadcasting problematic in today's Internet. Because bandwidth guarantees are not possible, the transmission process must adapt in real time to the available bandwidth. Such adaptation must take into account the heterogeneous nature of client connections to the Internet, to ensure that each client receives the video at an acceptable level of quality. Several approaches have been proposed to address this issue, differing in the component of the network architecture that controls adaptation.

- **End-to-end adaptation:** In these approaches, a server distributes a single encoding of the video over a unicast or multicast stream [3, 6, 7, 13, 14]. The server adapts the transmission in response to client feedback. This approach must deal with network latency, so that feedback information is not outdated when received by the server. This approach also does not scale well, since feedback from a large number of clients can affect the available bandwidth [7]. Finally, heterogeneity is not really addressed, since the degree of adaptation is always determined by the network segment that has the lowest available bandwidth.

- **Standard-router based adaptation:** In a standard network, adaptation can be provided within the network by combining the use of the standard multicast group management protocol (IGMP) with the encoding of the video information as a layered set of multicast streams, each representing a level of adaptation [13]. Clients join or leave a multicast group depending on the received video quality; a router forwards a multicast stream only if one of its clients is subscribed to the corresponding group. Adaptation relies on a distributed agreement; all clients on a given segment have to leave a group before the router stops forwarding the associated stream. As the number of clients increases, the time to reach this agreement also increases. Thus, the adaptation strategy may be unable to react quickly to changes in the network status.

- **Active-router based adaptation:** In *Active networks*, routers are active elements that can execute dynamically downloaded protocols [9, 16, 20]. The key feature of active networks is that routers can make local decisions. Therefore, an active protocol can observe

*Now at Ecole des Mines de Nantes (Gilles.Muller@emn.fr).

the local segment load and perform appropriate adaptation on a per segment basis. Because the adaptation decision is local to the router, the reaction time is almost instantaneous. This approach has been successfully applied to audio [18] and video [11] broadcast.

While the active network solution solves the scalability and heterogeneity problems, it raises other issues [16]: (i) *safety*, because distributed systems are complex and difficult to implement and debug; (ii) *security*, because network routers are shared resources; (iii) *portability*, because active protocols have to be executed on different router architectures; (iv) *performance*, because the execution of active protocols must not degrade the router bandwidth.

Existing active network implementations address these issues to some degree. In *platform specific* solutions [11], active protocols are developed in the C language at the kernel level. Although highly efficient, this approach requires highly skilled programmers, is error-prone, and is not portable. *Java-based* solutions [19, 20] are easier to program and ensure portability. Nevertheless, the efficiency of Java is still an issue. Also, the associated run-time system must check network-specific safety and security properties of active protocols. Finally, *Domain-Specific Language* (DSL) solutions provide high-level abstractions dedicated to the design of protocols. These abstractions can permit static verification of safety and security properties and simplify protocol development, to the point that a novice programmer can quickly develop realistic protocols. Two DSLs have been developed for active networks: PLAN [10] and PLAN-P [18]. While PLAN is interpreted, PLAN-P protocols are compiled by a JIT compiler that produces efficient kernel-mode code, showing that a DSL-based approach can achieve overall performance nearly comparable to that of using a protocol written in C [17].

This paper presents an experiment in designing and implementing an MPEG1 (or MPEG2) video broadcast application using active networks. MPEG reduces the bandwidth requirement of a movie by coding the movie as a sequence of interdependent frames. These dependencies, however, imply that the loss of a single frame can make other, correctly received frames undecodable. This situation can often occur in the context of a highly-loaded Internet, where router buffer overflow causes packets to be dropped randomly.

We have implemented an approach to adaptation that avoids random packet dropping by preventing segment overloading. In our approach, the router measures the available bandwidth in the local segment. If this bandwidth is not sufficient for the MPEG stream, the router drops less important frames. The precise strategy for the degradation of the MPEG stream is based on precomputed information about the stream's bandwidth requirement pattern.

To implement our approach, we have designed an improved version of the PLAN-P DSL [17]. The syntax of the original PLAN-P DSL was based on Standard ML and was unnatural for many system and network programmers. The precise subset of SML chosen was also insufficiently expressive, leading to a somewhat contorted programming style. The syntax of the new version is based on C and solves these problems. In some cases, the ability to express protocols in an imperative style also improves performance.

The contributions of this paper are as follows:

- We describe a complete architecture for implementing a video broadcasting service based on an active network architecture.
- We have implemented an improved version of the PLAN-P DSL based on the C syntax. This new language offers the same safety and efficiency advantages as the previous SML-based language, but is more natural for systems programmers.
- We show that video adaptation can be implemented efficiently in active networks. When used with a dedicated PLAN-P OS, the total treatment time within the router is 20μs, of which only 4μs are used by the PLAN-P program. This degree of performance is sufficient to allow the simultaneous treatment of up to 384 movies, encoded at a rate of 1.5Mb/s each.
- We have deployed an experimentation platform, including wireless IEEE 802.11 Ethernet. Using this platform, we have measured the effect of various background loads on movie broadcast. Our measurements show that up to 9 times more frames can be successfully decoded by clients than with standard routing.

Thanks to the active-network based architecture, our service offers the following advantages:

- Active routers manage client subscriptions to movies in a hierarchical manner. Video traffic is thus only forwarded to the parts of the network where there is at least one client.
- Because adaptation is controlled by local measurement of the network load, response to changing conditions can be almost instantaneous. Concretely, before forwarding a packet, the adaptation policy checks that the currently measured bandwidth allows successful delivery of the packet.
- Adaptation in response to limited bandwidth availability in some parts of the network has no effect on the quality of the video in other parts of the network.
- The adaptation strategy can easily be optimized to the specific structure of a given MPEG video.

The rest of this paper is organized as follows. Section 2 presents the architecture of the system, and describes the adaptation strategy. Section 3 describes the PLAN-P language and run-time system. Section 4 describes the implementation of our system, including the server, the clients, and the adaptation strategy in the active routers. Section 5 analyzes the performance of our approach, both in terms of its effect on the quality of the movie received by the client and in terms of the execution cost of the protocol in the active router. Section 6 describes related work. Section 7 concludes and presents future work.

2. Service architecture

Our goal in the design of a MPEG video broadcast service is to follow the best-effort principle of Internet. For that we need to adapt the quality of the video stream to the available bandwidth. Our solution relies on the use of *active routers*, which are programmable entities spread in the Internet that process particular types of traffic in a more appropriate manner than standard routers. In the rest of this section, we first present the basic design decisions for the service, and then describe the adaptation policy.

2.1. Active network architecture

We propose a network architecture containing both active and standard (non-extensible) routers. Active routers are located in critical places of the network. For video distribution, these routers are to be accessed by registered video service providers, who download appropriate adaptation policy protocols. Protocols are written in the PLAN-P language, and are checked and JIT compiled in the active routers [17, 18]. This approach provides both safety and efficiency.

Because we assume a mixture of active and standard routers, data must be sent through the network using either TCP or UDP. TCP is reliable, because lost packets are retransmitted. Nevertheless, in the case of streaming video, resent packets may arrive too late to be correctly displayed. We thus use the unreliable UDP, which does not retransmit lost packets, and instead try to prevent congestion.

We use a server that transmits the video at a constant bit rate, to avoid burstiness. Within the network, we use active routers to adapt the video stream on a per segment basis to the amount of traffic that a given segment can correctly deliver at the current time. Active routers also manage client subscriptions. An active router can thus further reduce network traffic by only sending a video stream to the parts of the network in which there are subscribed clients.

2.2. Adaptation policy

The goal of the adaptation policy is to reduce the bandwidth of the MPEG stream so as to avoid segment congestion. We perform adaptation by frame dropping. Since MPEG relies on a coding of images into interdependent frames, randomly dropping one frame may prevent the decoding of several neighboring frames, thus seriously degrading the video quality. Thus, the frames to drop must be carefully selected.

There are three kinds of frames in a MPEG stream: I, P, and B frames. Only an I frame contains a complete image and can be decoded independently. Decoding a P frame relies on the result of decoding the previous I or P frame. Decoding a B frame relies on the decoding of the nearest previous and successive I or P frames. An I frame and all of the frames that follow it, up to the next I frame, can be considered as a single unit, known as a *group of pictures* (GOP). These frame interdependencies imply that the highest priority should be given to the transmission of I frames. Because a B frame does not influence the decoding of any other frame, transmission of B frames has the lowest priority.

The strategy for dropping frames depends on the per second bandwidth requirements B_i, B_p, and B_b of I, P, and B frames, respectively, as well as the available bandwidth B_a. Let us consider a frequently used GOP structure: IBBPBBPBBPBBPBB. According to this structure and the relative priority of the types of MPEG frames, we use the following strategy:

- I frames: The protocol never drops an I frame.
- P frames: Once a P frame has been dropped, the rest of the GOP cannot be decoded. Thus, P frames are only slightly less important than I frames. The protocol only drops a P frame if a previous P frame in the current GOP has been dropped or the available bandwidth is less than the bandwidth requirement of P frames:

$$B_p > B_a$$

- B frames: If the protocol has dropped a P frame in the current GOP, the B frame is dropped. Otherwise, reflecting the low priority of B frames, we only transmit such a frame if the available bandwidth allows both P and I frames to be transmitted as well. Thus, we drop the B frame if the following condition holds:

$$B_p + B_i + B_b > B_a$$

The structure of the PLAN-P program presented in Section 4 to implement this strategy is relatively independent of these equations. Thus, the implementation can be easily modified to account for other GOP structures, and the new protocol can be dynamically downloaded into the active routers.

We have described the adaptation policy in terms of the treatment of complete frames. Nevertheless, in practice, a frame is too large to be transmitted as a single packet, and must be broken up into a sequence of packets by the server. The implementation described in Section 4 thus extends the above treatment of frames to the constituent packets.

3. Overview of PLAN-P

PLAN-P is a domain-specific language for implementing network protocols in active networks. The language is derived from the PLAN language for active networks [9], but has a significantly different semantics [17]. In this section, we present a new version of the PLAN-P language, having a syntax based on that of C.

The main element of application-specific processing in PLAN-P is a *protocol*, which defines a set of *stream* functions that control transmission through the router. Separate streams can be defined for network traffic with distinct characteristics, *e.g.* TCP packets, UDP packets, etc. Named streams can be defined using the IP option mechanism. Our protocol uses two streams: one that manages the installation of new movies and client subscriptions, and another than controls the video distribution. The latter is defined below:

```
typedef struct {
  ip ipheader;
  udp udpheader;
  int ver;
  int seq_no;
  int pictype;
  int frame_no;
  int payload_len;
  blob payload;
} packet;

stream network(packet p) {
 try {
  movie_data m =
   find(movie_table,hashkey(ipDst(p->ipheader)));
  adaptiveForward(m,p);
 }
 handle NoEntry => OnRemote(network, p);
}
```

The name `network` and the parameter type `packet` together imply that this stream is applied to any UDP packet containing at least the five integers required by the definition of the `packet` structure. The stream uses the primitive `find()` to determine whether the destination IP address of the incoming packet is the multicast IP address of a known movie, as recorded in the hash table `movie_table`. If so, the function `adaptiveForward()`, which is defined in the PLAN-P protocol, is invoked on the information `m` about the properties of the movie and the current packet `p` to determine whether the packet should be forwarded. If the destination address is not that of a known movie, `find()` raises a `NoEntry` exception, causing the packet to be forwarded normally.

The PLAN-P run-time system provides a permanent state that persists across the treatment of successive packets. This state allows the treatment of the current packet to be determined by information derived from the treatment of previous packets. For adaptive video, the permanent state contains two hash tables, declared as follows:

```
initstate {
 dropped_info hash_table interface_table =
  mkTable(25);
 movie_data hash_table movie_table =
  mkTable(25);
}
```

The `dropped_info` and `movie_data` types are defined by the protocol. The `interface_table` hash table records information about dropped packets at each interface, and is used to determine whether a given packet will be decodable if sent at a particular interface. The `movie_data` hash table records the dropping threshold for P and B packets for each movie.

The PLAN-P language provides network-specific types and library functions that facilitate the development of correct protocols. The types `ip` and `udp` are the types of IP and UDP packet headers, respectively. The primitive `OnRemote()` forwards a packet using the standard routing mechanism of the router. The primitive `OnNeighbor()` forwards a packet at a specific interface, allowing the PLAN-P program to take control of the routing path. The type `interface` is the type of a router interface. The primitive `thisInterface()` returns the interface at which the current packet was received, and the primitive `getInterfaces()` returns a list of all of the interfaces available on the router. The primitive `InterfaceLoad()` returns the current load at a specified interface, expressed in bits per second. PLAN-P also provides primitives for general programming, including `insert()`, `find()`, `remove()`, and `hashKey()` for manipulating hash tables, and `new()` for creating new data structures.

There are several restrictions on the syntax of the language that facilitate safety checks. The only looping construct is a `for` loop, which maps over the elements of a list. Because circular lists cannot be created, a `for` loop cannot loop infinitely. Thus, termination is guaranteed within the execution of a PLAN-P protocol at a router. The simple control flow of PLAN-P programs also facilitates checking that packets are not undesirably duplicated or discarded [17]. Finally, PLAN-P includes a static type checker that is aware of network-specific types, and requires that all variables are completely initialized (*e.g.* there is no "null" value in the PLAN-P language), for further safety.

Many of the features described above were found in the previous version of PLAN-P based on the syntax of ML [17, 18]. Nevertheless, the previous syntax was somewhat limited in ways that affected the ability to quickly develop correct protocols. Only tuples accessed by generic selector functions (`#1`, `#2`, etc.) were provided, rather than user-defined, named record structures. Furthermore, the global protocol state, which is heavily used in a typical PLAN-P protocol, was made accessible only by a local parameter of each stream, and thus had to be frequently passed between function calls. These problems have been resolved in the C-like syntax, using the `typedef` and `initstate` declarations illustrated above. These features have paid off in terms of both the initial development time and maintainability of protocols. Our original protocol for adaptive video treated only a specific movie. Extending this protocol to manage the installation of new movies and to manage client subscriptions required about one day for a programmer familiar with PLAN-P, but having no specialized knowledge of network or kernel programming.

PLAN-P has been implemented for a Sparc running Solaris, and for a PowerPC running a dedicated OS kernel. On Sparc, the PLAN-P run-time system runs as a loadable kernel module. This module provides both an interpreter, and a JIT compiler generated automatically from the interpreter using partial evaluation [17]. We are currently working with France Telecom Research on the design of a dedicated PLAN-P OS for a PowerPC-based machine. Preliminary results show that the complete system image requires 601Kb of memory [8]. The performance of the adaptation policy for both implementations is given in Section 5.

4. Service Implementation

We now describe the implementation of our approach, including the implementation of the server, the implementation of the adaptation process using PLAN-P in the routers, and the implementation of the clients.

4.1. Streaming server

In our architecture, the streaming server provides two functionalities: preparing a MPEG movie for transmission and serving client requests by sending a movie.

A design choice of our adaptive strategy is that the video server sends a MPEG movie at a constant bit rate. MPEG frames may exceed the maximum size of an Ethernet packet. If frames are fragmented within the network, the server cannot control the delay between the individual packets. Thus, the server splits each frame into smaller packets, and computes the delay between two consecutive packets according to the streaming rate of the movie (chosen at encoding time) and the actual sizes of the packets. Depending on the CPU power and memory bandwidth of the server, the formatting can be done while transmitting the video, or in advance, when installing the movie in the server.

4.2. Active routers

Each active router both handles control messages from the server and the client, and adapts the transmission of the video stream. The complete active-router protocol is 176 lines of PLAN-P code, of which 65 are only related to server and client management and the remainder implement the video transmission process.

Server and client management

Two kinds of initialization take place in the active router: a video service provider can install a movie, and a client can request to receive a (possibly on-going) movie.

To install a movie, the video service provider gives the multicast IP address that will be used for transmission of the movie and the average bandwidth requirements for I, P, and B frames (these values can be an approximation, in the case of live video). The latter information is used to calculate the thresholds for dropping these frames, as described in Section 2.2. The IP address and the associated thresholds are stored in the hash table `movie_table`.

Currently, we only allow clients to join a movie; existing clients are assumed to remain interested in a movie until its end. Thus, the only information that is needed about clients is whether there is a client for a particular movie at a particular interface. Accordingly, the arrival of the first client for a particular movie at an interface is recorded by initializing the entry in the hash table `interface_table`. This table is subsequently used to record information about the adaptation process at each interface, as described below.

Video transmission

As outlined in Section 2.2, the protocol uses two criteria to decide whether to drop a packet. First, a packet is dropped if part of some frame on which decoding of the packet depends has already been dropped. Second, a packet is dropped if our heuristics suggest that there is not enough available bandwidth to transmit both the entire current frame and other incoming frames with higher priority.

Information about previously dropped frames at each interface is recorded in the hash table `interface_table`. Each entry in this hash table is a pair of a frame type and a frame number; this information is sufficient to determine whether a P frame in the current GOP has been dropped, or whether a previous packet of the current frame has been dropped. The hash table is updated using the function `update_table()`. The entry `(0,0)` represents the case where no packet has yet been dropped in the current GOP.

Available bandwidth is determined using the function `available_bandwidth()`, defined as follows:

```
int available_bandwidth(interface i) {
 return 2000000 - InterfaceLoad(i);
}
```

This function obtains the load at a given interface using the PLAN-P operator `InterfaceLoad()`. For simplicity, we assume that all interfaces of the active router have the same outgoing bandwidth (2Mb/s).

Adaptation is performed individually at each interface at which there is a subscriber to the movie. The function `adaptiveForward` determines whether the current packet is part of an I, P, or B frame, and invokes the function `I_forward`, `P_forward`, or `B_forward`, respectively, which implement the adaptive strategy.

A packet that is part of an I frame is always forwarded if any clients are present at the interface:

```
void I_forward(int index, interface i, movie p) {
 try {
  find(interface_table, hashKey(index));
  update_table(index,0,0);
  OnNeighbor(network, i, p);
 } handle NoEntry => {} -- no client
}
```

The parameter `index` represents the pair of the current movie and the current interface, and is used as a key into the `interface_table` hash table. If the call to `find` succeeds, there is a client at the current interface, and we forward the packet using `OnNeighbor()`. Because the forwarding of an I frame implies that subsequent frames are decodable, the hash table entry corresponding to the current interface is cleared. If the call to `find` raises a `NoEntry` exception, there is no client for the movie at the current interface, and the packet is not forwarded.

A packet that is part of a P frame is dropped if the hash table `interface_table` indicates that the protocol has dropped a P packet in the current GOP at this interface, thus rendering the current P packet undecodable. The packet is also dropped if insufficient bandwidth is available, as determined by comparing the available bandwidth to the threshold, `threshold`, extracted from the data recorded about the movie:

```
void P_forward(int threshold, int index,
               interface i, movie p) {
 try {
  dropped_info dropped =
   find(interface_table, hashKey(index));
  bool undecodable =
   (dropped->frame_type == P_frame);
  if (!undecodable) {
   if (available_bandwidth(i) < threshold)
    update_table(index,P_frame,p->frame_no);
   else OnNeighbor(network, i, p);
  }
 } handle NoEntry => {} -- no client
}
```

If a P packet is dropped because the frame is undecodable, no further treatment is needed; the record of the previously dropped P frame causes all subsequent frames in the current GOP to be dropped. If the packet is dropped because insufficient bandwidth is available, the hash table is updated accordingly. Otherwise, the packet is transmitted at the interface using the primitive `OnNeighbor()`.

The treatment of a B packet is similar to that of a P packet. Here, however, the frame is also considered to be undecodable if a packet with the same frame number, *i.e.*, a previous fragment of the current B frame, has been dropped.

4.3. Clients

A client requests to join a movie by communicating with the server via a signaling channel. This request message is intercepted by active routers as a subscription signal for a movie. When the client joins the requested movie, it forks a receiving thread and launches a front-end player.[1] The receiving thread collects the packets corresponding to a single frame, and then sends the frame to the front-end player via a FIFO pipeline. When a frame is missing, the receiving thread simply sends the previous frame, to preserve the timing of the video.

5. Experiments

In this section, we evaluate the performance of the adaptation policy and its impact on MPEG multicast distribution. We compare our active adaptive policy with a non-active policy in terms of the decodable frames received in the client and the bandwidth wasted due to the transmission of undecodable frames.

5.1. Experimental Setting

The experiment is carried out in the Wireless LAN environment shown in Figure 1. The streaming server is a Sun 333Mhz Ultra Sparc IIi workstation with 384Mb RAM. The active router is a Sun 167Mhz Ultra Sparc workstation with 320Mb RAM. The client is a 700Mhz Pentium III laptop with 320Mb RAM, and is connected to the network via a Lucent 802.11 wireless Ethernet card. Only one wireless access point is included; more could be added to permit further experiments involving mobility. In order to produce congestion in the active router, we set the wireless access points with the maximum bandwidth of 2Mb/s. The experimental testbed also contains an additional load traffic generator and a client for this load traffic.

We use an excerpt of *Star Wars* as our video data.[2] This video has been widely used in the research community and

[1] We use MTVP: http://www.mpegtv.com/.

[2] http://www.wroclaw.tpsa.pl/users/starwars/mpegnew/astero1.mpg

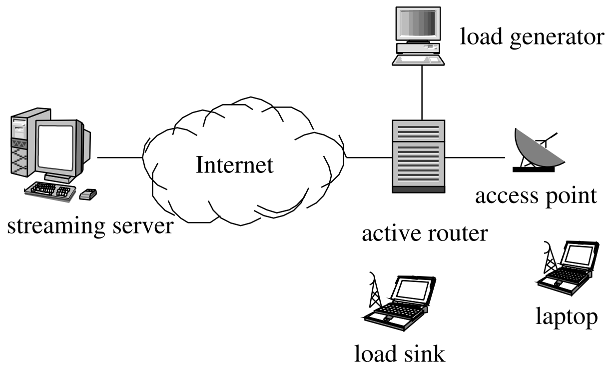

Figure 1. Experimental testbed

includes scenes with a variety of different properties, implying varying degrees of compression. The experimental data consists of 748 frames to be transmitted at 30 frames per second, and thus lasts about 25 seconds. The average bandwidth requirement and size of each frame type are shown in Figure 2. The thresholds for dropping P and B frames are 100kb/s and 300kb/s, respectively.

Frame	Ave. BW (kb/s)	Ave. size (kb)	Total frames
GOP	400	71.4	748
I	91	45.7	50
P	137	17.1	200
B	172	8.6	498

Figure 2. Profile of the Star Wars excerpt

5.2. Received video quality

These experiments measure the impact of our adaptation policy on MPEG video transmission. We add a constant background traffic load on the segment to which video is forwarded, thus potentially causing frames to be dropped by the wireless access point. We then measure the effect of this background traffic load on the percentage of frames that can be decoded by the client, to assess both the effect on the quality of the movie and the amount of wasted bandwidth due to the transmission of undecodable frames.

Decodable frame ratio

We first compare the number of frames decodable by the client to the number of frames in the movie. This ratio suggests the perceived quality of the movie. The adaptation policy never drops a packet of an I frame, and thus at least the beginning of each GOP is decodable. In contrast, a router buffer overflow can cause any kind of packet to be dropped, including packets of I frames.

These intuitions are reflected in the measurements shown in Figure 3. When we add a load of 1650kb/s, the available bandwidth measured by the PLAN-P protocol is typically above 300kb/s. A few B frames are dropped in the protocol, but most GOPs are transmitted intact. When we add a load of 1900kb/s, the available bandwidth measured by the PLAN-P protocol is typically below 100kb/s. Thus, for almost all GOPs, the protocol transmits only I frames. Nevertheless, because such frames are independently decodable, use of the adaptive policy allows 9 times as many frames to be decoded in the client than with standard routing.

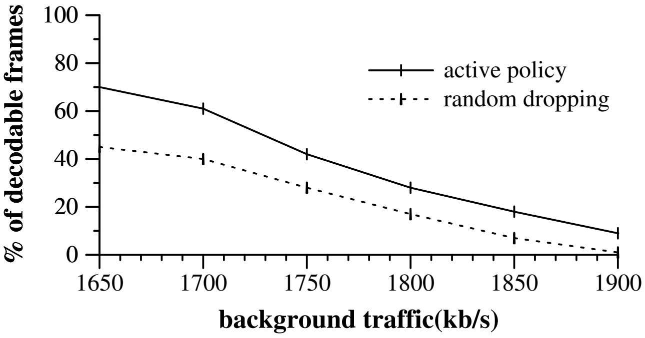

Figure 3. Decodable frame ratio

Bandwidth waste

Figure 4 shows the number of undecodable frames that are received by the client, with and without the adaptation policy, given an added load of 1750kb/s. Such frames represent wasted bandwidth.

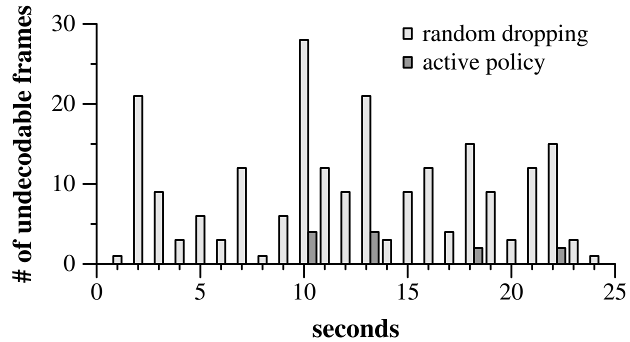

Figure 4. Undecodable frames received by the client (1750kb/s added load)

The adaptation policy attempts to predict when frames might be dropped at the outgoing link, and selects only the most important frames for transmission in this case. This policy generally avoids transmitting frames that are rendered undecodable by subsequent random packet dropping. For a background load of 1750kb/s, very few of the frames received by the client are undecodable.

When there is no adaptation, many of the frames received by the client are undecodable. At 10 seconds into the movie packets of I frames are lost, and both GOPs that arrive at this time are thus consequently undecodable. Because most of the packets of these GOPs are nevertheless transmitted, significant bandwidth is wasted.

5.3. Packet processing time

The use of PLAN-P programs in active routers can potentially affect the scalability of the approach to video distribution and imply the need to use high performance processors in routers. We first show that because of the JIT compilation of PLAN-P protocols, the processing time within the protocol for each packet is quite small. We then assess scalability, by combining the cost of the treatment of a packet within PLAN-P with the cost incurred by a dedicated PLAN-P OS for the PowerPC to transfer packet data between router interfaces and the PLAN-P protocol.

Figure 5 shows the average cost of the treatment of a single packet by the PLAN-P protocol in the case where the router has one outgoing interface, at which there is at least one client. We test the protocol presented here (referred to as CPLAN-P), as well as a comparable protocol written in the original SML-like language (referred to as SMLPLAN-P). As test data, we use the first 500 packets of the Star Wars movie excerpt. We consider three possible loads at the outgoing interface: a load below 1700kb/s, in which case any type of packet is transmitted; a load between 1700kb/s and 1900kb/s, in which case only an I or P packet is transmitted; and a load between 1900kb/s and 2000kb/s, in which case only an I packet is transmitted. Measurements were taken on the 167MHz UltraSPARC-II used as a router in our experimental testbed by using the function `gethrtime`, and on a 350MHz PowerPC G4 by using the number of cycles recorded in the Performance Counter register. In all cases, the processing times are roughly comparable across the various loads, the differences reflecting the varying costs of forwarding packets (highest below 1700kb/s) and updating hash table entries (highest between 1700kb/s and 1900kb/s). The CPLAN-P implementation is consistently between 1.3 and 1.5 times faster than its SMLPLAN-P counterpart.

Because the adaptive policy is applied at every interface, the execution time of the protocol increases linearly in the number of interfaces. Figure 6 shows the average incremental cost, as the number of outgoing interfaces increases. The first three groups of bars show the extra cost per interface in

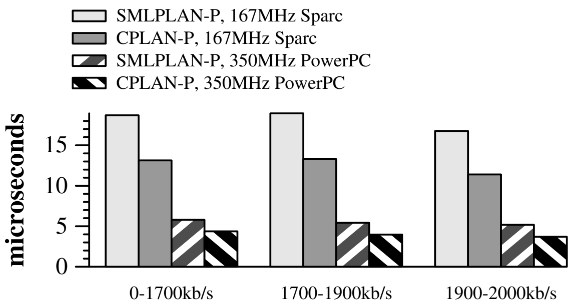

Figure 5. Processing time for a single packet and a single outgoing interface

the case where there is a client at every outgoing interface. The fourth group of bars shows the extra cost per interface for the case where there is no client at any interface. Because there is no adaptation, the cost is not affected by the load, and merely reflects the cost of the failure of a hash table lookup (*e.g.* see the definition of `I_forward` in Section 4.2).

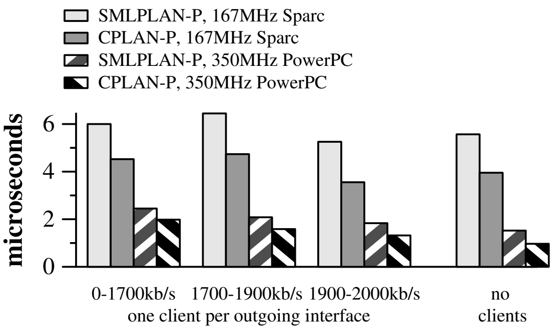

Figure 6. Average incremental cost per additional interface

Based on these figures, we calculate the number of movies that can be processed per second, when using a dedicated PLAN-P OS for the PowerPC [8]. A typical encoding rate used for television is 1.5Mb/s. In our architecture, the server divides frames into packets of at most 1440 bytes (11.52kb) of video data. This strategy implies that the active router receives about 130 packets per second for a single movie. The dedicated OS requires 8μs to transfer data from an interface to PLAN-P or vice versa. Based on Figure 5, the average cost within the PLAN-P protocol is 4μs on the PowerPC, when there is one incoming and one outgoing interface and using the CPLAN-P protocol. The total treatment cost of a single packet in the router is thus 20μs, or 2600μs per movie per second. This rate allows the treatment of up to 384 movies per second. For one incoming interface and seven outgoing interfaces, the average cost per packet within the protocol is 14μs while the OS cost is 64μs. This performance allows the treatment of up to 98 movies movies per second in the router.

6. Related work

Our design relates to many studies in areas such as adaptive video transmission, video coding and transcoding, and active networks.

Adaptive video transmission Most research in adaptive video transmission relies on feedback information sent from the receiver(s) to the server [3, 4, 5, 7, 13, 14]. While these approaches perform well for a limited number of clients, they do not scale well and do not adequately support variations in bandwidth across the network.

Video coding methods Another approach to improve the transmission of a video stream is to use a coding strategy other than MPEG, such as wavelets [6] or Forward error correction coding [14]. Nevertheless, because network conditions change frequently, a coding strategy based on information collected at one point in time might not work well throughout a video transmission.

Video transcoding To deal with heterogeneous networks, transcoding can be used at proxies in gateways within the network to reduce the bandwidth requirement of the video [1, 2]. Still transcoding increases the end-to-end delay, because transcoding requires partial or complete decoding of received frames followed by a CPU intensive reencoding process.

Video caching Another approach that can deal with heterogeneous networks is multimedia proxy caching, which adapts to bandwidth changes by prefetching frames that are expected to be frequently accessed by clients. The benefits of this caching depend on both the structure of the video and on local network traffic patterns. Thus, a fixed caching strategy might not work well for all video streams [15].

Active networks By enabling a router to take decisions, active networks allow new solutions to the problem of implementing large-scale Internet services. Still, while active networks have been studied since 1995, few realistic applications have been described. Success has, however, been achieved in implementing reliable message multicast [12] and adaptive audio and video broadcast [11, 18].

The video broadcast service described by Keller *et al.* [11] is probably the work most closely related to ours. They, however, use a wavelet-based encoding of the video. More significantly, their active network is platform-specific and programming must be done in C at kernel level. Therefore, programming protocols is error-prone and requires highly-skilled engineers. Also, the lack of portability of this solution complicates protocol testing and deployment.

7 Conclusion and Future Work

In this paper, we have presented the design and implementation of a video broadcast service based on active network routers. Thanks to the ability to deploy service-specific functions in the network, we have been able to develop efficient adaptation and management functions while keeping the implementation simple. The PLAN-P language thus provides a promising solution to permit service providers and router vendors to rapidly and safely deploy new network protocols.

We have evaluated our approach on an active network platform composed of several Ethernet segments, including wireless Ethernet. For this platform, the adaptation strategy described improves the quality of the video delivered to the mobile client by allowing up to 9 times more frames to be properly decoded than with no adaptation.

Our experiments show that the PLAN-P protocol has low cost and scales well to normal usage contexts. Using a dedicated PLAN-P OS developed in previous work [8], the performance of our approach on a 350MHz PowerPC G4 allows the simultaneous treatment of up to 384 movies, each having a bandwidth requirement of 1.5Mb/s. Better performance could be achieved using hardware specialized for active networks, such as the Intel IXP1200. We have also found that the new C-like syntax further simplifies programming and maintenance of network protocols, as compared to the previous syntax based on SML. Protocols written in the new syntax are often more efficient as well; the protocol described here is between 1.3 and 1.5 times faster than its counterpart in the old syntax.

Our work opens several perspectives.

- In this paper, client management by the active routers is limited to allowing new clients to join the video broadcast. This strategy can be extended to allow clients to explicitly leave the video broadcast as well, and to reroute video traffic when a mobile client moves from one access point to another.

- We have considered adaptation of MPEG video via packet dropping. PLAN-P could also be used to implement adaptation strategies that are appropriate to other video formats, including transcoding to reduce the bandwidth requirement.

- The adaptation decisions taken by the protocol we have presented rely only on measurement of the locally available bandwidth. To provide a more complete picture of network conditions and further reduce the transmission of packets that are ultimately dropped, an active router could communicate load information to the active router from which it receives a video stream, thus adding a limited form of feedback in the network.

- The protocol we have presented distinguishes only between frame types. A further refinement would be to give more priority to P frames near the beginning of the GOP, on which the decoding of a large part of the GOP relies. For even more fine-grained control, the server could encode adaptation information in the header of each packet.

Availability

More information about PLAN-P is available at `http://compose.labri.fr/prototypes/cplanp`, including the implementation and some examples.

Acknowledgments

The authors would like to acknowledge others in the Compose group who gave us helpful comments on this work. This work was supported in part by the RNRT PHENIX project.

References

[1] E. Amir, S. McCanne, and R. Katz. An active service framework and its application to real-time multimedia transcoding. In *SIGCOMM Symposium on Communications Architectures and Protocols*, Vancouver, BC, Canada, Sept. 1998. ACM Press.

[2] E. Amir, S. McCanne, and H. Zhang. An application level video gateway. In *Proceedings of the Third ACM International Conference on Multimedia '95*, pages 255–265, San Francisco, CA, 1995.

[3] J.-C. Bolot, T. Turletti, and I. Wakeman. Scalable feedback control for multicast video distribution in the internet. In *SIGCOMM Symposium on Communications Architectures and Protocols*, pages 58–67, London, England, Aug. 1994. ACM/IEEE.

[4] S. Cen, C. Pu, and J. Walpole. Flow and congestion control for internet streaming applications. In *Proceedings Multimedia Computing and Networking 1998 (MMCN98)*, 1998.

[5] C. Cowan, S. Cen, J. Walpole, and C. Pu. Adaptive methods for distributed video presentation. In *Computing Surveys Symposium on Multimedia*, volume 27(4), pages 580–583, Dec. 1995.

[6] G. Davis. A wavelet-based analysis of fractal image compression. *IEEE Transactions on Image Processing*, 7(2):141–154, 1998.

[7] D. DeLucia and K. Obraczka. A multicast congestion control mechanism using representatives. In *Proceedings of the IEEE Infocom'97*, pages 463–470, Kobe, Japan, Apr. 1997.

[8] J. Fassino, J. Stefani, J. Lawall, and G. Muller. THINK: a software framework for component-based operating system kernels. In *USENIX 2002 Annual Technical Conference*, Monterey, CA, June 2002. To appear.

[9] M. Hicks, P. Kakkar, J. Moore, C. Gunter, and S. Nettles. PLAN: A Packet Language for Active Networks. In *Proceedings of the Third ACM SIGPLAN International Conference on Functional Programming Languages*, volume 34(1) of *ACM SIGPLAN Notices*, pages 86–93. ACM, June 1998.

[10] M. Hicks, J. Moore, D. Alexander, C. Gunter, and S. Nettles. PLANet: An active internetwork. In *Proceedings of the Eighteenth IEEE Computer and Communication Society INFOCOM Conference*, pages 1124–1133. IEEE, 1999.

[11] R. Keller, S. Choi, D. Decasper, M. Dasen, G. Fankhauser, and B. Plattner. An active router architecture for multicast video distribution. In *Proceedings of IEEE INFOCOM 2000*, pages 1137–1146, Tel Aviv, Israel, Mar. 2000.

[12] H. Lehman, S. Garland, and D. Tennenhouse. Active reliable multicast. In *Conference on Computer Communications (INFOCOM'98)*, pages 581–589, San Fransisco, CA, USA, Apr. 1998. IEEE.

[13] S. McCanne, V. Jacobson, and M. Vetterli. Receiver-driven layered multicast. In *SIGCOMM Symposium on Communications Architectures and Protocols*, pages 117–130, Stanford University, CA, Aug. 1996.

[14] K. Park and W. Wang. AFEC: An adaptive forward error correction protocol for end-to-end transport of real-time traffic. In *Proceedings of the IEEE International Conference on Computer Communications and Networks*, pages 196–205, Lafayette, LA, Oct. 1998.

[15] R. Rejaie and J. Kangasharju. Mocha: A quality adaptive multimedia proxy cache for internet streaming. In *Proceedings of the International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'01)*, pages 3–10, Port Jefferson, NY, June 2001. ACM Press.

[16] D. Tennenhouse, J. Smith, W. Sincoskie, D. Wetherall, and G. Minden. A survey of active network research. *IEEE Communications Magazine*, 35(1):80–86, Jan. 1997.

[17] S. Thibault, C. Consel, and G. Muller. Safe and efficient active network programming. In *17th IEEE Symposium on Reliable Distributed Systems*, pages 135–143, West Lafayette, Indiana, Oct. 1998.

[18] S. Thibault, J. Marant, and G. Muller. Adapting distributed applications using extensible networks. In *Proceedings of the 19th International Conference on Distributed Computing Systems*, pages 234–243, Austin, Texas, May 1999. IEEE Computer Society Press.

[19] D. Wetherall. Active network vision and reality: lessons from a capsule-based system. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles*, Kiawah Island, SC, Dec. 1999.

[20] D. Wetherall, J. Guttag, and D. Tennenhouse. ANTS: A toolkit for building and dynamically deploying network protocols. In *IEEE OPENARCH*, Apr. 1998.