Network Working Group
Request for Comments: 2689
Category: Informational

C. Bormann
Universitaet Bremen TZI

September 1999

Providing Integrated Services over Low-bitrate Links

Status of this Memo

Copyright Notice

Abstract

This document describes an architecture for providing integrated services over low-bitrate links, such as modem lines, ISDN B-channels, and sub-T1 links. It covers only the lower parts of the Internet Multimedia Conferencing Architecture [1]; additional components required for application services such as Internet Telephony (e.g., a session initiation protocol) are outside the scope of this document. The main components of the architecture are: a real-time encapsulation format for asynchronous and synchronous low-bitrate links, a header compression architecture optimized for real-time flows, elements of negotiation protocols used between routers (or between hosts and routers), and announcement protocols used by applications to allow this negotiation to take place.

1. Introduction

As an extension to the "best-effort" services the Internet is well-known for, additional types of services ("integrated services") that support the transport of real-time multimedia information are being developed for, and deployed in the Internet. Important elements of this development are:

- parameters for forwarding mechanisms that are appropriate for real-time information [11, 12],
- a setup protocol that allows establishing special forwarding treatment for real-time information flows (RSVP [4]),
- a transport protocol for real-time information (RTP/RTCP [6]).

In addition to these elements at the network and transport levels of the Internet Multimedia Conferencing Architecture [1], further components are required to define application services such as Internet Telephony, e.g., protocols for session initiation and control. These components are outside the scope of this document.

Up to now, the newly developed services could not (or only very inefficiently) be used over forwarding paths that include low-bitrate links such as 14.4, 33.6, and 56 kbit/s modems, 56 and 64 kbit/s ISDN B-channels, or even sub-T1 links. The encapsulation formats used on these links are not appropriate for the simultaneous transport of arbitrary data and real-time information that has to meet stringent delay requirements. Transmission of a 1500 byte packet on a 28.8 kbit/s modem link makes this link unavailable for the transmission of real-time information for about 400 ms. This adds a worst-case delay that causes real-time applications to operate with round-trip delays on the order of at least a second -- unacceptable for real-time conversation. In addition, the header overhead associated with the protocol stacks used is prohibitive on low-bitrate links, where compression down to a few dozen bytes per real-time information packet is often desirable. E.g., the overhead of at least 44 (4+20+8+12) bytes for HDLC/PPP, IP, UDP, and RTP completely overshadows typical audio payloads such as the 19.75 bytes needed for a G.723.1 ACELP audio frame -- a 14.4 kbit/s link is completely consumed by this header overhead alone at 40 real-time frames per second total (i.e., at 25 ms packetization delay for one stream or 50 ms for two streams, with no space left for data, yet). While the header overhead can be reduced by combining several real-time information frames into one packet, this increases the delay incurred while filling that packet and further detracts from the goal of real-time transfer of multi-media information over the Internet.

This document describes an approach for addressing these problems. The main components of the architecture are:

- a real-time encapsulation format for asynchronous and synchronous low-bitrate links,

- a header compression architecture optimized for real-time flows,

- elements of negotiation protocols used between routers (or between hosts and routers), and

- announcement protocols used by applications to allow this negotiation to take place.

2. Design Considerations

The main design goal for an architecture that addresses real-time multimedia flows over low-bitrate links is that of minimizing the end-to-end delay. More specifically, the worst case delay (after removing possible outliers, which are equivalent to packet losses from an application point of view) is what determines the playout points selected by the applications and thus the delay actually perceived by the user.

In addition, any such architecture should obviously undertake every attempt to maximize the bandwidth actually available to media data; overheads must be minimized.

An important component of the integrated services architecture is the provision of reservations for real-time flows. One of the problems that systems on low-bitrate links (routers or hosts) face when performing admission control for such reservations is that they must translate the bandwidth requested in the reservation to the one actually consumed on the link. Methods such as data compression and/or header compression can reduce the requirements on the link, but admission control can only make use of the reduced requirements in its calculations if it has enough information about the data stream to know how effective the compression will be. One goal of the architecture therefore is to provide the integrated services admission control with this information. A beneficial side effect may be to allow the systems to perform better compression than would be possible without this information. This may make it worthwhile to provide this information even when it is not intended to make a reservation for a real-time flow.

3. The Need for a Concerted Approach

Many technical approaches come to mind for addressing these problems, in particular a new form of low-delay encapsulation to address delay and header compression methods to address overhead. This section shows that these techniques should be combined to solve the problem.

3.1. Real-Time Encapsulation

The purpose of defining a real-time link-layer encapsulation protocol is to be able to introduce newly arrived real-time packets into the link-layer data stream without having to wait for the currently transmitted (possibly large) packet to end. Obviously, a real-time encapsulation must be part of any complete solution as the problem of delays induced by large frames on the link can only be solved on this layer.

To be able to switch to a real-time packet quickly in an interface driver, it is first necessary to identify packets that belong to real-time flows. This can be done using a heuristic approach (e.g., favor the transmission of highly periodic flows of small packets transported in IP/UDP, or use the IP precedence fields in a specific way defined within an organization). Preferably, one also could make use of a protocol defined for identifying flows that require special treatment, i.e. RSVP. Of the two service types defined for use with RSVP now, the guaranteed service will only be available in certain environments; for this and various other reasons, the service type chosen for many adaptive audio/video applications will most likely be the controlled-load service. Controlled-load does not provide control parameters for target delay; thus it does not unambiguously identify those packet streams that would benefit most from being transported in a real-time encapsulation format. This calls for a way to provide additional parameters in integrated services flow setup protocols to control the real-time encapsulation.

Real-time encapsulation is not sufficient on its own, however: Even if the relevant flows can be appropriately identified for real-time treatment, most applications simply cannot operate properly on low-bitrate links with the header overhead implied by the combination of HDLC/PPP, IP, UDP, and RTP, i.e. they absolutely require header compression.

3.2. Header Compression

Header compression can be performed in a variety of elements and at a variety of levels in the protocol architecture. As many vendors of Internet Telephony products for PCs ship applications, the approach that is most obvious to them is to reduce overhead by performing header compression at the application level, i.e. above transport protocols such as UDP (or actually by using a non-standard, efficiently coded header in the first place).

Generally, header compression operates by installing state at both ends of a path that allows the receiving end to reconstruct information omitted at the sending end. Many good techniques for header compression (RFC 1144, [2]) operate on the assumption that the path will not reorder the frames generated. This assumption does not hold for end-to-end compression; therefore additional overhead is required for resequencing state changes and for compressed packets making use of these state changes.

Assume that a very good application level header compression solution for RTP flows could be able to save 11 out of the 12 bytes of an RTP header [3]. Even this perfect solution only reduces the total header overhead by 1/4. It would have to be deployed in all applications,

even those that operate on systems that are attached to higher-bitrate links.

Because of this limited effectiveness, the AVT group that is responsible for RTP within the IETF has decided to not further pursue application level header compression.

For router and IP stack vendors, the obvious approach is to define header compression that can be negotiated between peer routers.

Advanced header compression techniques now being defined in the IETF [2] certainly can relieve the link from significant parts of the IP/UDP overhead (i.e., most of 28 of the 44 bytes mentioned above).

One of the design principles of the new IP header compression developed in conjunction with IPv6 is that it stops at layers the semantics of which cannot be inferred from information in lower layer (outer) headers. Therefore, this header compression technique alone cannot compress the data that is contained within UDP packets.

Any additional header compression technique runs into a problem: If it assumes specific application semantics (i.e., those of RTP and a payload data format) based on heuristics, it runs the risk of being triggered falsely and (e.g. in case of packet loss) reconstructing packets that are catastrophically incorrect for the application actually being used. A header compression technique that can be operated based on heuristics but does not cause incorrect decompression even if the heuristics failed is described in [7]; a companion document describes the mapping of this technique to PPP [10].

With all of these techniques, the total IP/UDP/RTP header overhead for an audio stream can be reduced to two bytes per packet. This technology need only be deployed at bottleneck links; high-speed links can transfer the real-time streams without routers or switches expending CPU cycles to perform header compression.

4. Principles of Real-Time Encapsulation for Low-Bitrate Links

The main design goal for a real-time encapsulation is to minimize the delay incurred by real-time packets that become available for sending while a long data packet is being sent. To achieve this, the encapsulation must be able to either abort or suspend the transfer of the long data packet. As an additional goal is to minimize the overhead required for the transmission of packets from periodic flows, this strongly argues for being able to suspend a packet, i.e. segment it into parts between which the real-time packets can be transferred.

4.1. Using existing IP fragmentation

Transmitting only part of a packet, to allow higher-priority traffic to intervene and then resuming its transmission later on, is a kind of fragmentation. Fragmentation is an existing functionality of the IP layer: An IPv4 header already contains fields that allow a large IP datagram to be fragmented into small parts. A sender's "real-time PPP" implementation might simply indicate a small MTU to its IP stack and thus cause all larger datagrams to be fragmented down to a size that allows the access delay goals to be met (this assumes that the IP stack is able to priority-tag fragments, or that the PPP implementation is able to correlate the fragments to the initial one that carries the information relevant for prioritizing, or that only initial fragments can be high-priority). (Also, a PPP implementation can negotiate down the MTU of its peer, causing the peer to fragment to a small size, which might be considered a crude form of negotiating an access delay goal with the peer system -- if that system supports priority queueing at the fragment level.)

Unfortunately, a full, 20 byte IP header is needed for each fragment (larger when IP options are used). This limits the minimum size of fragments that can be used without too much overhead. (Also, the size of non-final fragments must be a multiple of 8 bytes, further limiting the choice.) With path MTU discovery, IP level fragmentation causes TCP implementations to use small MSSs -- this further increases the per-packet overhead to 40 bytes per fragment.

In any case, fragmentation at the IP level persists on the path further down to the datagram receiver, increasing the transmission overheads and router load throughout the network. With its high overhead and the adverse effect on the Internet, IP level fragmentation can only be a stop-gap mechanism when no other fragmentation protocol is available in the peer implementation.

4.2. Link-Layer Mechanisms

Cell-oriented multiplexing techniques such as ATM that introduce regular points where cells from a different packet can be interpolated are too inefficient for low-bitrate links; also, they are not supported by chips used to support the link layer in low-bitrate routers and host interfaces.

Instead, the real-time encapsulation should as far as possible make use of the capabilities of the chips that have been deployed. On synchronous lines, these chips support HDLC framing; on asynchronous lines, an asynchronous variant of HDLC that usually is implemented in software is being used. Both variants of HDLC provide a delimiting mechanism to indicate the end of a frame over the link. The obvious solution to the segmentation problem is to combine this mechanism with an indication of whether the delimiter terminates or suspends the current packet.

This indication could be in an octet appended to each frame information field; however, seven out of eight bits of the octet would be wasted. Instead, the bit could be carried at the start of the next frame in conjunction with multiplexing information (PPP protocol identifier etc.) that will be required here anyway. Since the real-time flows will in general be periodic, this multiplexing information could convey (part of) the compressed form of the header for the packet. If packets from the real-time flow generally are of constant length (or have a defined maximum length that is often used), the continuation of the suspended packet could be immediately attached to it, without expending a further frame delimiter, i.e., the interpolation of the real-time packet would then have zero overhead. Since packets from low-delay real-time flows generally will not require the ability to be further suspended, the continuation bit could be reserved for the non-real-time packet stream.

One real-time encapsulation format with these (and other) functions is described in ITU-T H.223 [13], the multiplex used by the H.324 modem-based videophone standard [14]. It was investigated whether compatibility could be achieved with this specification, which will be used in future videophone-enabled (H.324 capable) modems. However, since the multiplexing capabilities of H.223 are limited to 15 schedules (definitions of sequences of packet types that can be identified in a multiplex header), for general Internet usage a superset or a more general encapsulation would have been required. Also, a PPP-style negotiation protocol was needed instead of using (and necessarily extending) ITU-T H.245 [15] for setting the parameters of the multiplex. In the PPP context, the interactions with the encapsulations for data compression and link layer encryption needed to be defined (including operation in the presence of padding). But most important, H.223 requires synchronous HDLC chips that can be configured to send frames without an attached CRC, which is not possible with all chips deployed in commercially available routers; so complete compatibility was unachievable.

Instead of adopting H.223, it was decided to pursue an approach that is oriented towards compatibility both with existing hardware and existing software (in particular PPP) implementations. The next subsection groups these implementations according to their capabilities.

4.3. Implementation models

This section introduces a number of terms for types of implementations that are likely to emerge. It is important to have these different implementation models in mind as there is no single approach that fits all models best.

4.3.1. Sender types

There are two fundamental approaches to real-time transmission on low-bitrate links:

Sender type 1
: The PPP real-time framing implementation is able to control the transmission of each byte being transmitted with some known, bounded delay (e.g., due to FIFOs). For example, this is generally true of PC host implementations, which directly access serial interface chips byte by byte or by filling a very small FIFO. For type 1 senders, a suspend/resume type approach will be typically used: When a long frame is to be sent, the attempt is to send it undivided; only if higher priority packets come up during the transmission will the lower-priority long frame be suspended and later resumed. This approach allows the minimum variation in access delay for high-priority packets; also, fragmentation overhead is only incurred when actually needed.

Sender type 2
: With type 2 senders, the interface between the PPP real-time framing implementation and the transmission hardware is not in terms of streams of bytes, but in terms of frames, e.g., in the form of multiple (prioritized) send queues directly supported by hardware. This is often true of router systems for synchronous links, in particular those that have to support a large number of low-bitrate links. As type 2 senders have no way to suspend a frame once it has been handed down for transmission, they typically will use a queues-of-fragments approach, where long packets are always split into units that are small enough to maintain the access delay goals for higher-priority traffic. There is a trade-off between the variation in access delay resulting from a large fragment size and the overhead that is incurred for every long packet by choosing a small fragment size.

4.3.2. Receiver types

Although the actual work of formulating transmission streams for real-time applications is performed at the sender, the ability of the receiver to immediately make use of the information received depends on its characteristics:

Receiver type 1
: Type 1 receivers have full control over the stream of bytes received within PPP frames, i.e., bytes received are available immediately to the PPP real-time framing implementation (with some known, bounded delay e.g. due to FIFOs etc.).

Receiver type 2
: With type 2 receivers, the PPP real-time framing implementation only gets hold of a frame when it has been received completely, i.e., the final flag has been processed (typically by some HDLC chip that directly fills a memory buffer).

4.4. Conclusion

As a result of the diversity in capabilities of current implementations, there are now two specifications for real-time encapsulation: One, the multi-class extension to the PPP multi-link protocol, is providing the solution for the queues-of-fragments approach by extending the single-stream PPP multi-link protocol by multiple classes [8]. The other encapsulation, PPP in a real-time oriented HDLC-like framing, builds on this specification end extends it by a way to dynamically delimit multiple fragments within one HDLC frame [9], providing the solution for the suspend/resume type approach.

5. Principles of Header Compression for Real-Time Flows

A good baseline for a discussion about header compression is in the new IP header compression specification that was designed in conjunction with the development of IPv6 [2]. The techniques used there can reduce the 28 bytes of IPv4/UDP header to about 6 bytes (depending on the number of concurrent streams); with the remaining 4 bytes of HDLC/PPP overhead and 12 bytes for RTP the total header overhead can be about halved but still exceeds the size of a G.723.1 ACELP frame. Note that, in contrast to IP header compression, the environment discussed here assumes the existence of a full-duplex PPP link and thus can rely on negotiation where IP header compression requires repeated transmission of the same information. (The use of the architecture of the present document with link layer multicasting has not yet been examined.)

Additional design effort was required for RTP header compression. Applying the concepts of IP header compression, of the (at least) 12 bytes in an RTP header, 7 bytes (timestamp, sequence, and marker bit) would qualify as RANDOM; DELTA encoding cannot generally be used without further information since the lower layer header does not unambiguously identify the semantics and there is no TCP checksum that can be relied on to detect incorrect decompression. Only a more semantics-oriented approach can provide better compression (just as RFC 1144 can provide very good compression of TCP headers by making use of semantic knowledge of TCP and its checksumming method).

For RTP packets, differential encoding of the sequence number and timestamps is an efficient approach for certain cases of payload data formats. E.g., speech flows generally have sequence numbers and timestamp fields that increase by 1 and by the frame size in timestamp units, resp.; the CRTP (compressed RTP) specification makes use of this relationship by encoding these fields only when the second order difference is non-zero [7].

6. Announcement Protocols Used by Applications

As argued, the compressor can operate best if it can make use of information that clearly identifies real-time streams and provides information about the payload data format in use.

If these systems are routers, this consent must be installed as router state; if these systems are hosts, it must be known to their networking kernels. Sources of real-time information flows are already describing characteristics of these flows to their kernels and to the routers in the form of TSpecs in RSVP PATH messages [4]. Since these messages make use of the router alert option, they are seen by all routers on the path; path state about the packet stream is normally installed at each of these routers that implement RSVP. Additional RSVP objects could be defined that are included in PATH messages by those applications that desire good performance over low-bitrate links; these objects would be coded to be ignored by routers that are not interested in them (class number 11bbbbbb as defined in [4], section 3.10).

Note that the path state is available in the routers even when no reservation is made; this allows informed compression of best-effort traffic. It is not quite clear, though, how path state could be torn down quickly when a source ceases to transmit.

7. Elements of Hop-By-Hop Negotiation Protocols

The IP header compression specification attempts to account for simplex and multicast links by providing information about the compressed streams only in the forward direction. E.g., a full IP/UDP header must be sent after F_MAX_TIME (currently 3 seconds), which is a negligible total overhead (e.g. one full header every 150 G.723.1 packets), but must be considered carefully in scheduling the real-time transmissions. Both simplex and multicast links are not prevailing in the low-bitrate environment (although multicast functionality may become more important with wireless systems); in this document, we therefore assume full-duplex capability.

As compression techniques will improve, a negotiation between the two peers on the link would provide the best flexibility in implementation complexity and potential for extensibility. The peer routers/hosts can decide which real-time packet streams are to be compressed, which header fields are not to be sent at all, which multiplexing information should be used on the link, and how the remaining header fields should be encoded. PPP, a well-tried suite of negotiation protocols, is already used on most of the low-bitrate links and seems to provide the obvious approach. Cooperation from PPP is also needed to negotiate the use of real-time encapsulations between systems that are not configured to automatically do so. Therefore, PPP options that can be negotiated at the link setup (LCP) phase are included in [8], [9], and [10].

8. Security Considerations

Header compression protocols that make use of assumptions about application protocols need to be carefully analyzed whether it is possible to subvert other applications by maliciously or inadvertently enabling their use.

It is generally not possible to do significant hop-by-hop header compression on encrypted streams. With certain security policies, it may be possible to run an encrypted tunnel to a network access server that does header compression on the decapsulated packets and sends them over an encrypted link encapsulation; see also the short mention of interactions between real-time encapsulation and encryption in section 4 above. If the security requirements permit, a special RTP payload data format that encrypts only the data may preferably be used.

9. References

[1] Handley, M., Crowcroft, J., Bormann, C. and J. Ott, "The Internet Multimedia Conferencing Architecture", Work in Progress.

[2] Degermark, M., Nordgren, B. and S. Pink, "IP Header Compression", RFC 2507, February 1999.

[3] Scott Petrack, Ed Ellesson, "Framework for C/RTP: Compressed RTP Using Adaptive Differential Header Compression", contribution to the mailing list rem-conf@es.net, February 1996.

[4] Braden, R., Zhang, L., Berson, S., Herzog, S. and S. Jamin, "Resource ReSerVation Protocol (RSVP) -- Version 1 Functional Specification", RFC 2205, September 1997.

[5] Sklower, K., Lloyd, B., McGregor, G., Carr, D. and T. Coradetti, "The PPP Multilink Protocol (MP)", RFC 1990, August 1996.

[6] Schulzrinne, H., Casner, S., Frederick, R. and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", RFC 1889, January 1996.

[7] Casner, S. and V. Jacobson, "Compressing IP/UDP/RTP Headers for Low-Speed Serial Links", RFC 2508, February 1999.

[8] Bormann, C., "The Multi-Class Extension to Multi-Link PPP", RFC 2686, September 1999.

[9] Bormann, C., "PPP in a Real-time Oriented HDLC-like Framing", RFC 2687, September 1999.

[10] Engan, M., Casner, S. and C. Bormann, "IP Header Compression over PPP", RFC 2509, February 1999.

[11] Wroclawski, J., "Specification of the Controlled-Load Network Element Service", RFC 2211, September 1997.

[12] Shenker, S., Partridge, C. and R. Guerin. "Specification of Guaranteed Quality of Service", RFC 2212, September 1997.

[13] ITU-T Recommendation H.223, "Multiplexing protocol for low bit rate multimedia communication", International Telecommunication Union, Telecommunication Standardization Sector (ITU-T), March 1996.

[14] ITU-T Recommendation H.324, "Terminal for low bit rate multimedia communication", International Telecommunication Union, Telecommunication Standardization Sector (ITU-T), March 1996.

[15] ITU-T Recommendation H.245, "Control protocol for multimedia communication", International Telecommunication Union, Telecommunication Standardization Sector (ITU-T), March 1996.

10. Author's Address

Carsten Bormann
Universitaet Bremen FB3 TZI
Postfach 330440
D-28334 Bremen, GERMANY

Phone: +49.421.218-7024
Fax: +49.421.218-7000
EMail: cabo@tzi.org

Acknowledgements

Much of the early discussion that led to this document was done with Scott Petrack and Cary Fitzgerald. Steve Casner, Mikael Degermark, Steve Jackowski, Dave Oran, the other members of the ISSLL subgroup on low bitrate links (ISSLOW), and in particular the ISSLL WG co-chairs Eric Crawley and John Wroclawski have helped in making this architecture a reality.

Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.