Internet Engineering Task Force
INTERNET-DRAFT
draft-ietf-avt-issues-02.ps

Audio-Video Transport Working Group
H. Schulzrinne
AT&T Bell Laboratories
May 9, 1994
Expires: 04/01/94

Issues in Designing a Transport Protocol for Audio and Video Conferences and other Multiparticipant Real-Time Applications

Status of this Memo

This document is an Internet Draft. Internet Drafts are working documents of the Internet Engineering Task Force (IETF), its Areas, and its Working Groups. Note that other groups may also distribute working documents as Internet Drafts.

Internet Drafts are draft documents valid for a maximum of six months. Internet Drafts may be updated, replaced, or obsoleted by other documents at any time. It is not appropriate to use Internet Drafts as reference material or to cite them other than as a "working draft" or "work in progress."

Please check the I-D abstract listing contained in each Internet Draft directory to learn the current status of this or any other Internet Draft.

Distribution of this document is unlimited.

Abstract

This memorandum is a companion document to the current version of the RTP protocol specification draft-ietf-avt-rtp-*.{txt,ps}. It discusses protocol aspects of transporting real-time services (for example, voice or video) over packet-switched networks such as the Internet. It compares and evaluates design alternatives for a real-time transport protocol, providing rationales for the design decisions made for RTP. Also covered are issues of port assignment and multicast address allocation. An appendix provides a comprehensive glossary of terms related to multimedia conferencing.

This document is a product of the Audio-Video Transport working group within the Internet Engineering Task Force. Comments are solicited and should be addressed to the working group's mailing list at `rem-conf@es.net` and/or the author(s).

Contents

1 Introduction **4**

2 Goals **7**

3 Services **9**

3.1 Control and Data 11

3.2 Duplex or Simplex? 11

3.3 Framing 13

4 Version Identification **13**

4.1 Conference Identification 14

4.1.1 Demultiplexing 14

4.1.2 Aggregation 14

5 Media Encoding Identification **15**

5.0.3 Audio Encodings 16

5.0.4 Video Encodings 17

5.1 Playout Synchronization 17

5.1.1 Synchronization Methods 19

5.1.2 Detection of Synchronization Units 21

5.1.3 Interpretation of Synchronization Bit 23

6 Timing and Synchronization **24**

6.1 Timestamp Format 25

6.2 Timestamp Frequency 27

6.3 Synchronizing Sample Clock with System Clock 28

6.3.1 Synchronization Units, no Sender Adjustment 28

6.3.2 Sender Adjustment 29

6.3.3 Synchronization between Streams 29

6.3.4 Recommendation 30

7 Segmentation and Reassembly **30**

8 Source Identification **31**

8.1 Bridges, Translators and End Systems 31

8.2 Address Format Issues 33

8.3 Globally unique identifiers 34

8.4 Locally unique addresses 35

9 Energy Indication **36**

10 Error Control **37**

11 Security and Privacy **38**

11.1 Introduction 38

11.2 Confidentiality 39

11.3 Message Integrity and Authentication 40

11.4 Security for RTP vs. PEM 41

12 Quality of Service Control **42**

12.1 QOS Measures 42

12.2 Remote measurements 43

12.3 Monitoring by Third Party 44

13 The Use of Profiles **44**

14 Port Assignment **45**

15 Multicast Address Allocation **46**

15.1 Channel Sensing 47

15.2 Global Reservation Channel with Scoping 47

15.3 Local Reservation Channel 48

15.3.1 Hierarchical Allocation with Servers 48

15.3.2 Distributed Hierarchical Allocation 48

15.4 Restricting Scope by Limiting Time-to-Live 49

16 Security Considerations **49**

A Timestamp conversion **49**

B Glossary **53**

C Address of Author **61**

1 Introduction

This memorandum aims to provide a general introduction to some of the issues faced by end-to-end protocols for real-time services. It also provides a general background to some of the design decisions made in RTP [1]. The transport protocol for real-time applications (RTP) discussed in this memorandum aims to provide services commonly required by interactive multimedia conferences, such as playout synchronization, demultiplexing, media identification and active-party identification. However, RTP is not restricted to multimedia conferences; it is anticipated that other real-time services such as remote data acquisition and control may find its services of use.

In this context, a *conference* describes associations that are characterized by the participation of two or more agents, interacting in real time with one or more media of potentially different types. The agents are anticipated to be human, but may also be measurement devices, remote media servers, simulators and the like. Both two-party and multiple-party associations are to be

supported, where one or more agents can take active roles, i.e., generate data. Thus, applications not commonly considered a conference fall under this wider definition, for example, one-way media such as the network equivalent of closed-circuit television or radio, traditional two-party telephone conversations or real-time distributed simulations. Even though intended for real-time interactive applications, the use of RTP for the storage and transmission of recorded real-time data should be possible, with the understanding that the interpretation of some fields such as timestamps may be affected by this off-line mode of operation.

RTP uses the services of an end-to-end transport protocol such as UDP, TCP, OSI TP1 or TP4, ST-II or the like[1] . The services used are: end-to-end delivery, framing, demultiplexing and multicast. The underlying network is not assumed to be reliable and can be expected to lose, corrupt, arbitrarily delay and reorder packets. However, the use of RTP within quality-of-service (e.g., rate) controlled networks is anticipated to be of particular interest. Network layer support for multicasting is desirable, but not required. RTP is supported by a *real-time control protocol* (RTCP) in a relationship similar to that between IP and ICMP. However, RTP can be used, with reduced functionality, without a control protocol. The control protocol RTCP provides minimum functionality for maintaining conference state for one or more flows within a single transport association. RTCP is not guaranteed to be reliable; each participant simply sends the local information periodically to all other conference participants.

As an alternative, RTP could be used as a transport protocol layered directly on top of IP, potentially increasing performance and reducing header overhead. This may be attractive as the services provided by UDP, checksumming and demultiplexing, may not be needed for multicast real-time conferencing applications. This aspect remains for further study. The relationships between RTP and RTCP to other protocols of the Internet protocol suite are depicted in Fig. 1.

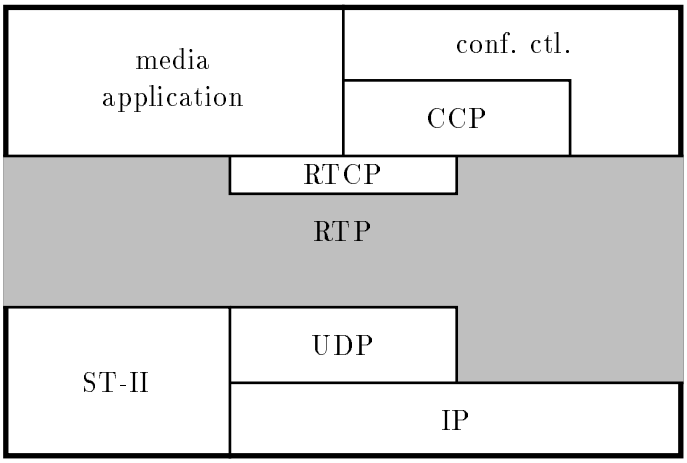

Figure 1: Embedding of RTP and RTCP in Internet protocol stack

[1] ST-II is not properly a transport protocol, as it is visible to intermediate nodes, but it provides services such as process demultiplexing commonly associated with transport protocols.

Conferences encompassing several media are managed by a (reliable) conference control protocol, whose definition is outside the scope of this note.

Within this working group, some common encoding rules and algorithms for media have been specified, keeping in mind that this aspect is largely independent of the remainder of the protocol. Without this specification, interoperability cannot be achieved. It is intended, however, to keep the two aspects as separate RFCs as changes in media encoding should be independent of the transport aspects. The encoding specification includes issues such as byte order for multi-byte samples, sample order for multi-channel audio, the format of state information for differential encodings, the segmentation of encoded video frames into packets, and the like.

When used for multimedia services, RTP sources will have to be able to convey the type of media encoding used to the receivers. The number of encodings potentially used is rather large, but a single application will likely restrict itself to a small subset of that. To allow the participants in conferences to unambiguously communicate to each other the current encoding, the working group is defining a set of encoding names to be registered with the Internet Assigned Numbers Authority (IANA). Also, short integers for a default mapping of common encodings are specified.

The issue of port assignment will be discussed in more detail in Section 14. It should be emphasized, however, that UDP port assignment does not imply that all underlying transport mechanisms share this or a similar port mechanism.

This memorandum aims to summarize some of the discussions held within the audio-video transport (AVT) working group chaired by Stephen Casner, but the opinions are the author's own. Where possible, references to previous work are included, but the author realizes that the attribution of ideas is far from complete. The memorandum builds on operational experience with Van Jacobson's and Steve McCanne's `vat` audio conferencing tool as well as implementation experience with the author's NEVOT network voice terminal. This note will frequently refer to NVP [2], the network voice protocol, a protocol used in two versions for early Internet wide-area packet voice experiments. CCITT has standardized as recommendations G.764 and G.765 a packet voice protocol stack for use in digital circuit multiplication equipment.

The name RTP was chosen to reflect the fact that audio and video conferences may not be the only applications employing its services, while the real-time nature of the protocol is important, setting it apart from other multimedia-transport mechanisms, such as the MIME multimedia mail effort [3].

The remainder of this memorandum is organized as follows. Section 2 summarizes the design goals of this real-time transport protocol. Then, Section 3 describes the services to be provided in more detail. Two appendices discuss the issues of port assignment and multicast address allocation, respectively. A glossary defines terms and acronyms, providing references for further detail. The actual protocol specification embodying the recommendation and conclusions of this report is contained in a separate document.

2 Goals

Design decisions should be measured against the following goals, not necessarily listed in order of importance:

content flexibility: While the primary applications that motivate the protocol design are conference voice and video, it should be anticipated that other applications may also find the services provided by the protocol useful. Some examples include distribution audio/video (for example, the "Radio Free Ethernet"application by Sun), distributed simulation and some forms of (loss-tolerant) remote data acquisition (for example, active badge systems [4, 5]). Note that it is possible that the same packet header field may be interpreted in different ways depending on the content (e.g., a synchronization bit may be used to indicate the beginning of a talkspurt for audio and the beginning of a frame for video). Also, new formats of established media, for example, high-quality multi-channel audio or combined audio and video sources, should be anticipated where possible.

extensible: Researchers and implementors within the Internet community are currently only beginning to explore real-time multimedia services such as video conferences. Thus, the RTP should be able to incorporate additional services as operational experience with the protocol accumulates and as applications not originally anticipated find its services useful. The same mechanisms should also allow experimental applications to exchange application-specific information without jeopardizing interoperability with other applications. Extensibility is also desirable as it will hopefully speed along the standardization effort, making the consequences of leaving out some group's favorite fixed header field less drastic.

It should be understood that extensibility and flexibility may conflict with the goals of bandwidth and processing efficiency.

independent of lower-layer protocols: RTP should make as few assumptions about the underlying transport protocol as possible. It should, for example, work reasonably well with UDP, TCP, ST-II, OSI TP, VMTP and experimental protocols, for example, protocols that support resource reservation and quality-of-service guarantees. Naturally, not all transport protocols are equally suited for real-time services; in particular, TCP may introduce unacceptable delays over anything but low-error-rate LANs. Also, protocols that deliver streams rather than packets needs additional framing services as discussed in Section 3.3.

It remains to be discussed whether RTP may use services provided by the lower-layer protocols for its own purposes (time stamps and sequence numbers, for example).

The goal of independence from lower-layer considerations also affects the issue of address representation. In particular, anything too closely tied to the current IP 4-byte addresses may face early obsolescence. It is to be anticipated, however, that experience gained will suggest a new protocol revision in any event by that time.

bridge-compatible: Operational experience has shown that RTP-level bridges are necessary and desirable for a number of reasons. First, it may be desirable to aggregate several media streams into a single stream and then retransmit it with possibly different encoding, packet

size or transport protocol. A packet "translator" that achieves multicasting by user-level copying may be needed where multicast tunnels or IP connectivity are unavailable or the end-systems are not multicast-capable.

bandwidth efficient: It is anticipated that the protocol will be used in networks with a wide range of bandwidths and with a variety of media encodings. Despite increasing bandwidths within the national backbone networks, bandwidth efficiency will continue to be important for transporting conferences across 56 kb links, office-to-home high-speed modem connections and international links. To minimize end-to-end delay and the effect of lost packets, packetization intervals have to be limited, which, in combination with efficient media encodings, leads to short packet sizes. Generally, packets containing 16 to 32 ms of speech are considered optimal [6–8]. For example, even with a 65 ms packetization interval, a 4800 b/s encoding produces 39 byte packets. Current Internet voice experiments use packets containing around 20 ms of audio, which translates into 160 bytes of audio information coded at 64 kb/s. Video packets are typically much longer, so that header overhead is less of a concern.

For UDP multicast (without counting the overhead of source routing as currently used in tunnels or a separate IP encapsulation as planned), IPv4 incurs 20 bytes and UDP an additional 8 bytes of header overhead, to which datalink layer headers of at least 4 bytes must be added. With RTP header lengths between 4 and 8 bytes, the total overhead amounts to between 36 and 40 (or more) bytes per audio or video packet. For 160-byte audio packets, the overhead of 8-byte RTP headers together with UDP, IP and PPP (as an example of a datalink protocol) headers is 25%. For low bitrate coding, packet headers can easily double the necessary bit rate.

Thus, it appears that any fixed headers beyond eight bytes would have to make a significant contribution to the protocol's capabilities as such long headers could stand in the way of running RTP applications over low-speed links. The current fixed header lengths for NVP and `vat` are 4 and 8 bytes, respectively. It is interesting to note that G.764 has a *total* header overhead, including the LAPD data link layer, of only 8 bytes, as the voice transport is considered a network-layer protocol. The overhead is split evenly between layers 2 and 3.

Bandwidth efficiency can be achieved by transporting non-essential or slowly changing protocol state in optional fields or in a separate low-bandwidth control protocol. Also, header compression [9] may be used.

international: Even now, audio and video conferencing tools are used far beyond the North American continent. It would seem appropriate to give considerations to internationalization concerns, for example to allow for the European A-law audio companding and non-US-ASCII character sets in textual data such as site identification.

processing efficient: With arrival rates of on the order of 40 to 50 packets per second for a single voice or video source, per-packet processing overhead may become a concern, particularly if the protocol is to be implemented on other than high-end workstations. Multiplication and division operations should be avoided where possible and fields should be aligned to their natural size, i.e., an n-byte integer is aligned on an n-byte multiple, where possible.

implementable now: Given the anticipated lifetime and experimental nature of the protocol, it must be implementable with current hardware and operating systems. That does not preclude

that hardware and operating systems geared towards real-time services may improve the performance or capabilities of the protocol, e.g., allow better intermedia synchronization.

3 Services

The services that may be provided by RTP are summarized below. Note that not all services have to be offered. Services anticipated to be optional are marked with an asterisk.

- framing (*)
- demultiplexing by media source
- demultiplexing by channel
- demultiplexing by synchronization source
- determination of media encoding
- playout synchronization between a source and a set of destinations
- error detection (*)
- encryption (*)
- quality-of-service monitoring (*)

In the following sections, we will discuss how these services are reflected in the proposed packet header. Information to be conveyed can be roughly divided into information that changes with every data packet and other information that stays constant for longer time periods, in particular, information that is needed when initially establishing communication or tearing down an assocation between communicating applications. State information that does not change with every packet can be carried in several different ways:

as a fixed part of the data header: This method is easiest to decode and ensures state synchronization between sender and receiver(s), but can be bandwidth inefficient or restrict the amount of state information to be conveyed.

as a header option: The information is only carried when needed. It requires more processing by the sending and receiving application. If contained in every packet, it is also less bandwidth-efficient than the first method.

within separate control packets: This approach is roughly equivalent to header options in terms of processing and bandwidth efficiency. Some means of identifying when a particular option takes effect within the data stream may have to be provided.

within a multicast conference announcement: Instead of residing at a well-known conference server, information about on-going or upcoming conferences may be multicast to a well-known multicast address.

within conference control: The state information is conveyed when the conference is established or when the information changes. As for RTCP packets, a synchronization mechanism between data and control may be required for certain information.

through a conference directory: This is a variant of the conference control mechanism, with a (distributed) directory at a well-known (multicast) address maintaining state information about on-going or scheduled conferences. Changing state information during a conference is probably more difficult than with conference control as participants need to be told to look at the directory for changed information. Thus, a directory is probably best suited to hold information that will persist through the life of the conference, for example, its multicast group, list of media encodings, title and organizer.

The first two methods are examples of in-band signaling, the others of out-of-band signaling.

Options can be encoded in a number of ways, resulting in different tradeoffs between flexibility, processing overhead and space requirements. In general, options consists of a type field, possibly a length field, and the actual option value. The length field can be omitted if the length is implied by the option type. Implied-length options save space, but require special treatment while processing. While options with explicit length that are added in later protocol versions are backwards-compatible (the receiver can just skip them), implied-length options cannot be added without modifying all receivers, unless they are marked as such and all have a known length. As an example, IP defines two implied-length options, no-op and end-of-option, both with a length of one octet. Both CLNP and IP follow the type-length-data model, with different substructure of the type field.

For indicating the extent of options, a number of alternatives have been suggested.

option length: The fixed header contains a field containing the length of the options, as used for IP. This makes skipping over options easy, but consumes precious header space.

end-of-options bit: Each option contains a special bit that is set only for the last option in the list. In addition, the fixed header contains a flag indicating that options are present. This conserves space in the fixed header, at the expense of reducing usable space within options, e.g., reducing the number of possible option types or the maximum option length. It also makes skipping options somewhat more processing-intensive, particulary if some options have implied lengths and others have explicit lengths. Skipping through the options list can be accelerated slightly by starting options with a length field.

end-of-options option: A special option type indicates the end of the option list, with a bit in the fixed header indicating the presence of options. The properties of this approach are similar to the previous one, except that it can be expected to take up more header space.

options directory: An options-present bit in the fixed header indicates the presence of an options directory. The options directory in turn contains a length field for the options list and possibly bits indicating the presence of certain options or option classes. The option length makes skipping options fast, while the presence bits allow a quick decision whether the options list should be scanned for relevant options. If all options have a known, fixed length, the bit mask can be used to directly access certain options, without having to traverse parts of the options list. The drawback is increased header space and the necessity to create the directory. If options are explicitly coded in the bit mask, the type, number and numbering of options is restricted. This approach is used by PIP [10].

3.1 Control and Data

Like most protocols at all layers of a protocol stack, a real-time protocol needs to convey higher-layer data as well as more infrequent control information.

separate lower-level streams replace control without affecting data operating system : number of sockets vs. ability to schedule execution

inband fate-sharing: "they all go together" when options take effect

Firewalls

3.2 Duplex or Simplex?

In terms of information flow, protocols can be roughly divided into three categories:

1. For one instance of a protocol, packets travel only in one direction; i.e., the receiver has no way to directly influence the sender. UDP is an example of such a protocol.
2. While data only travels in one direction, the receiver can send back control packets, for example, to accept or reject a connection, or request retransmission. ST-II in its standard simplex mode is an example; TCP is symmetric (see next item), but during a file transfer, it typically operates in this mode, where one side sends data and the receiver of the data returns acknowledgements.
3. The protocol is fully symmetric during the data transfer phase, with user data and control information travelling in both directions. TCP is a symmetric protocol.

Note that bidirectional data flow can usually be simulated by two or more one-directional data flows in opposite directions, however, if the data sinks need to transmit control information to the source, a decoupled stream in the reverse direction will not do without additional machinery to bridge the gap between the two protocol state machines.

For most of the anticipated applications for a real-time transport protocol, one-directional data flow appears sufficient. Also, in general, bidirectional flows may be difficult to maintain in one-to-many settings commonly found in conferences. Real-time requirements combined with network latency make achieving reliability through retransmission difficult, eliminating another reason for a bidirectional communication channel. Thus, we will focus only on control flow from the receiver of a data flow to its sender. For brevity, we will refer to packets of this control flow as *reverse control* packets.

There are at least two areas within multimedia conferences where a receiver needs to communicate control information back to the source. First, the sender may want or need to know how well the transmission is proceding, as traditional feedback through acknowledgements is missing (and usually infeasible due to acknowledgment implosion). Secondly, the receiver should be able to request a selective update of its state, for example, to obtain missing image blocks after joining an on-going conference. Note that for both uses, unicast rather than multicast is appropriate.

Three approaches allowing the sender to distinguish reverse control packets from data packets are compared here:

sender port equals reverse port, marked packet: The same port number is used both for data and return control messages. Packets then have to be marked to allow distinguishing the two. Either the presence of certain options would indicate a reverse control packet, or the options themselves would be interpreted as reverse control information, with the rest of the packet treated as regular data. The latter approach appears to be the most flexible and symmetric, and is similar in spirit to transport protocols with piggy-backed acknowledgements as in TCP. Also, since several conferences with different multicast addresses may be using the same port number, the receiver has to include the multicast address in its reverse control messages. As a final identification, the control packets have to bear the flow identifier they belong to. The scheme has the grave disadvantage that every application on a host has to receive the reverse control messages and decide whether it involves a flow it is responsible for.

single reverse port: Reverse control packets for all flows use a single port that differs from the data port. Since the type of the packet (control vs. data) is identified by the port number, only the multicast address and flow number still needs to be included, without a need for a distinguishing packet format. Adding a port means that port negotiation is somewhat more complicated; also, as in the first scheme, the application still has to demultiplex incoming control messages.

different reverse port for each flow: This method requires that each source makes it known to all receivers on which port it wishes to receive reverse control messages. Demultiplexing based on flow and multicast address is no longer necessary. However, each participant sending data and expecting return control messages has to communicate the port number to all other participants. Since the reverse control port number should remain constant throughout the conference (except after application restarts), a periodic dissemination of that information is sufficient. Distributing the port information has the advantage that it gives applications the flexibility to designate only certain flows as potential recipients of reverse control information.

Unfortunately, the delay in acquiring the reverse control port number when joining an ongoing conference may make one of the more interesting uses of a reverse control channel difficult to implement, namely the request by a new arrival to the sender to transmit the complete current state (e.g., image) rather than changes only.

3.3 Framing

To satisfy the goal of transport independence, we cannot assume that the lower layer provides framing. (Consider TCP as an example; it would probably not be used for real-time applications except possibly on a local network, but it may be useful in distributing recorded audio or video segments.) It may also be desirable to pack several RTPDUs into a single TPDU.

The obvious solution is to provide for an optional message length prefixed to the actual packet. If the underlying protocol does not message delineation, both sender and receiver would know to use the message length. If used to carry multiple RTPDUs, all participants would have to arrive at a mutual agreement as to its use. A 16-bit field should cover most needs, but appears to break the 4-byte alignment for the rest of the header. However, an application would read the message length first and then copy the appropriate number of bytes into a buffer, suitably aligned.

4 Version Identification

Humility suggests that we anticipate that we may not get the first iteration of the protocol right. In order to avoid "flag days" where everybody shifts to a new protocol, a version identifier could ensure continued interoperability. Alternatively, a new port could be used, as long as only one port (or at most a few ports) is used for all media. The difficulty in interworking between the current `vat` and NVP protocols further affirms the desirability of a version identifier. However, the version identifier can be anticipated to be the most static of all proposed header fields. Since the length of the header and the location and meaning of the option length field may be affected by a version change, encoding the version within an optional field is not feasible.

Putting the version number into the control protocol packets would make RTCP mandatory and would make rapid scanning of conferences significantly more difficult.

`vat` currently offers a 2-bit version field, while this capability is missing from NVP. Given the low bit usage and their utility in other contexts (IP, ST-II), it may be prudent to include a version identifier. To be useful, any version field must be placed at the very beginning of the header. Assigning an initial version value of one to RTP allows interoperability with the current `vat` protocol.

4.1 Conference Identification

A conference identifier (conference ID) could serve two mutually exclusive functions: providing another level of demultiplexing or a means of logically aggregating flows with different network addresses and port numbers. `vat` specifies a 16-bit conference identifier.

4.1.1 Demultiplexing

Demultiplexing by RTP allows one association characterized by destination address and port number to carry several distinct conferences. However, this appears to be necessary only if the number of conferences exceeds the demultiplexing capability available through (multicast) addresses and port numbers.

Efficiency arguments suggest that combining several conferences or media within a single multicast group is not desirable. Combining several conferences or media within a single multicast address reduces the bandwidth efficiency afforded by multicasting if the sets of destinations are different. Also, applications that are not interested in a particular conference or capable of dealing with particular medium are still forced to handle the packets delivered for that conference or medium. Consider as an example two separate applications, one for audio, one for video. If both share the same multicast address and port, being differentiated only by the conference identifier, the operating system has to copy each incoming audio and video packet into two application buffers and perform a context switch to both applications, only to have one immediately discard the incoming packet.

Given that application-layer demultiplexing has strong negative efficiency implications and given that multicast addresses are not an extremely scarce commodity, there seems to be no reason to burden every application with maintaining and checking conference identifiers for the purpose of demultiplexing. However, if this protocol is to be used as a transport protocol, demultiplexing capability is required.

It is also not recommended to use a conference identifier to distinguish between different encodings, as it would be difficult for the application to decide whether a new conference identifier means that a new conference has arrived or simply all participants should be moved to the new conference with a different encoding. Since the encoding may change for some but not all participants, we could find ourselves breaking a single logical conference into several pieces, with a fairly elaborate control mechanism to decide which conferences logically belong together.

4.1.2 Aggregation

Particularly within a network with a wide range of capacities, differing multicast groups for each media component of a conference allows to tailor the media distribution to the network bandwidths and end-system capabilities. It appears useful, however, to have a means of identifying groups that logically belong together, for example for purposes of time synchronization.

A conference identifier used in this manner would have to be globally unique. It appears that such logical connections would better be identified as part of the higher-layer control protocol by identifying all multicast addresses belonging to the same logical conference, thereby avoiding the assignment of globally unique identifiers.

5 Media Encoding Identification

Most protocols contain an indication of the next protocol layer. Examples include the next-protocol field in IP or port numbers in UDP and TCP. Datagram protocols need to include the next-protocol indication in every packet, while connection-oriented protocols such as ATM may establish this as part of the connection setup (e.g., AAL type for ATM).

For a real-time protocol such as RTP that deals with media, the equivalent of the next protocol is the type of media (audio, video, ...) and its encoding. We define encoding as the set of attributes that describe how the media content is translated into digital information and possibly compressed. Just like other next-protocol indicators, the type of media can be expected to remain constant through the lifetime of a conference, say. However, the media encoding may change for any number of reasons, for example, receivers joining a conference may be less capable than the current participants and thus require a different encoding or the network conditions may call for an encoding with a reduced bit rate or increased loss tolerance. Sometimes the type of material conveyed may suggest a different encoding, e.g., switching from commentary to music.

In general, the number of distinct encodings should be kept as small as possible to increase the chance that applications can interoperate. A new standard encoding should only be recognized if it significantly enhances the range of media quality or the types of networks conferences can be conducted over. The unnecessary proliferation of encodings can be reduced by making reference implementations of standard encoders and decoders widely available.

It should be noted that encodings may not be enumerable as easily as, say, transport protocols. A particular family of related encoding methods may be described by a set of parameters, as discussed below in the sections on audio and video encoding. Thus, in the most general form, each packet would contain a complete description of the encoding, e.g., the number of audio channels, the sampling frequency and the compression method used. To save header space, an index into a table can be transmitted instead for all or some of these elements. (It would certainly be inappropriate to carry the full name of a compression algorithm, in particular since a canonical name needs to be agreed upon in any event.)

As with other information, the encoding can be transmitted as part of regular data packets or out-of-band, i.e., through a separate lower-layer network association. Out-of-band means provide greater flexibility in how to send the information, but complicate the receiver and delay the ability of receivers to join on-going conferences, for example. Also, as discussed earlier, the encoding may change during a session. If the information necessary for the decoder is conveyed out-of-band, some means of indicating when the change is effective needs to be incorporated. The indication that the

encoding is about to change must reach all receivers *reliably* before the first packet employing the new encoding. (Misinterpreting the content of a packet as a different encoding is likely to be quite noticeable and may persist through several packets if prediction is used.) Each receiver needs to track pending changes of encodings and check for every incoming packet whether an encoding change is to take effect with this packet. To allow scanning of conferences or broadcast media, it is important that the media encoding of a newly acquired stream can be determined quickly.

We now discuss briefly methods that can be used to convey encoding information if packets contain an index into a table of encodings rather than a full description. If a conference, say, is created and announced through a directory service, it is easy to include a globally consistent table mapping indices into encoding descriptions. However, this static approach makes it impossible to add new encodings during a conference, say, without restarting all applications.

XXXXXX

At the other extreme, we can have each site announce its own mappings, requiring receivers to keep a per-site table filled by these announcements.

RTP chooses a combination of the two approaches

global: Here, the media identifier is an index into a global table of encodings. A global list reduces the need for out-of-band information. Transmitting the parameters associated with an encoding may be difficult, however, if it has to be done within the header space constraints of per-packet signaling.

To make detecting coder mismatches easier, encodings for all media should be drawn from the same numbering space. To facilitate experimentation with new encodings, a part of any global encoding numbering space should be set aside for experimental encodings, with numbers agreed upon within the community experimenting with the encoding, with no network-wide guarantee of uniqueness.

5.0.3 Audio Encodings

Audio data is commonly characterized by three independent descriptors: encoding (the translation of one or more audio samples into a channel symbol), the number of channels (mono, stereo, ...) and the sampling rate.

Theoretically, sampling rate and encoding are (largely) independent. We could, for example, apply mu-law encoding to any sampling rate even though it is traditionally used with a rate of 8,000 Hz. In practical terms, it may be desirable to limit the combinations of encoding and sampling rate to the values the encoding was designed for.[2] Channel counts between 1 and 6 should be sufficient even for surround sound.

[2]Given the wide availability of mu-law encoding and its low overhead, using it with a sampling rate of 16,000 or 32,000 Hz might be quite appropriate for high-quality audio conferences, even though there are other encodings, such as G.722, specifically designed for such applications. Note that the signal-to-noise ratio of mu-law encoding is

The audio encodings listed in Table 1 appear particularly interesting, even though the list is by no means exhaustive and does not include some experimental encodings currently in use, for example a non-standard form of LPC. The bit rate is shown per channel. k samples/s, b/sample and kb/s denote kilosamples per second, bits per sample and kilobits per second, respectively. If sampling rates are to be specified separately, the values of 8, 16, 32, 44.1, and 48 kHz suggest themselves, even though other values (11.025 and 22.05 kHz) are supported on some workstations (the Silicon Graphics audio hardware and the Apple Macintosh, for example). Clearly, little is to be gained by allowing arbitrary sampling rates, as conversion particularly between rates not related by simple fractions is quite cumbersome and processing-intensive [11].

Org.	Name	k samples/s	b/sample	kb/s	description
CCITT	G.711	8.0	8	64	mu-law PCM
CCITT	G.711	8.0	8	64	*A*-law PCM
CCITT	G.721	8.0	4	32	ADPCM
Intel	DVI	8.0	4	32	APDCM
CCITT	G.723	8.0	3	24	ADPCM
CCITT	G.726				ADPCM
CCITT	G.727				ADPCM
NIST/GSA	FS 1015	8.0		2.4	LPC-10E
NIST/GSA	FS 1016	8.0		4.8	CELP
NADC	IS-54	8.0		7.95	N. American Digital Cellular, VSELP
CCITT	G.728	8.0		16	LD-CELP
GSM		8.0		13	RPE-LTP
CCITT	G.722	8.0		64	7 kHz, SB-ADPCM
ISO	3-11172			256	MPEG audio
		32.0	16	512	DAT
		44.1	16	705.6	CD, DAT playback
		48.0	16	786	DAT record

Table 1: Standardized and common audio encodings

5.0.4 Video Encodings

Common video encodings are listed in Table 2. Encodings with tunable rate can be configured for different rates, but produce a fixed-rate stream. The average bit rate produced by variable-rate codecs depends on the source material.

5.1 Playout Synchronization

A major purpose of a real-time protocol is to provide the *support* for various forms of synchronization, without necessarily performing the synchronization itself. We can distinguish three kinds of

about 38 dB, equivalent to an AM receiver. The "telephone quality" associated with G.711 is due primarily to the limitation in frequency response to the 200 to 3500 Hz range.

Org.	name	rate	remarks
CCITT	JPEG	tunable	
CCITT	MPEG	variable, tunable	
CCITT	H.261	tunable, $p \times 64$ kb/s	
Bolter		variable, tunable	
PictureTel		??	
Cornell U.	CU-SeeMe	variable	
Xerox Parc	nv	variable, tunable	
BBN	DVC	variable, tunable	block differences

Table 2: Common video encodings

synchronization:

playout synchronization: The receiver plays out the medium a fixed time after it was generated at the source (end-to-end delay). This end-to-end delay may vary from synchronization unit to synchronization unit. In other words, playout synchronization assures that a constant rate source at the sender again becomes a constant rate source at the receiver, despite delay jitter in the network.

intra-media synchronization: All receivers play the same segment of a medium at the same time. Intra-media synchronization may be needed during simulations and wargaming.

inter-media synchronization: The timing relationship between several media sources carried as independent network streams is reconstructed at the receiver. The primary example is the synchronization between audio and video (lip-sync). Note that different receivers may experience different delays between the media generation time and their playout time.

See [12] for a general discussion and a service model.

Playout synchronization is required for most media, while intra-media and inter-media synchronization may or may not be implemented. In connection with playout synchronization, we can group packets into playout units, a number of which in turn form a synchronization unit. More specifically, we define:

synchronization unit: A synchronization unit consists of one or more playout units (see below) that, as a group, share a common fixed delay between generation and playout of each part of the group. The delay may change at the beginning of such a synchronization unit. The most common synchronization units are talkspurts for voice and frames for video transmission.

playout unit: A playout unit is a group of packets sharing a common timestamp. (Naturally, packets whose timestamps are identical due to timestamp wrap-around are not considered part of the same playout unit.) For voice, the playout unit would typically be a single voice

segment, while for video a video frame could be broken down into subframes, each consisting of packets sharing the same timestamp and ordered by some form of sequence number.

Two concepts related to synchronization and playout units are *absolute* and *relative* timing. Absolute timing maintains a fixed timing relationship between sender and receiver, while relative timing ensures that the spacing between packets at the sender is the same as that at the receiver, measured in terms of the sampling clock. Playout units within the synchronization unit maintain relative timing with respect to each other; absolute timing is undesirable if the receiver clock runs at a (slightly) different rate than the sender clock.

Most proposed synchronization methods require a timestamp. The timestamp has to have a sufficient range that wrap-arounds are infrequent. It is desirable that the range exceeds the maximum expected inactive (e.g., silence) period. Otherwise, if the silence period lasts a full timestamp range, the first packet of the next talkspurt would have a timestamp one larger than the last packet of the current talkspurt. In that case, the new talkspurt could not be readily discerned if the difference in increment between timestamps and sequence numbers is used to detect a new talkspurt.

A timestamp may be useful not only at the transport, but also at the network layer, for example, for scheduling packets based on urgency. The playout timestamp would be appropriate for such a scheduling timestamp, as it would better reflect urgency than a network-level departure timestamp. Thus, it may make sense to use a network-level timestamp such as the one provided by ST-II at the transport layer.

5.1.1 Synchronization Methods

The necessary header components are determined to some extent by the method of synchronizing sender and receivers. In this section, we formally describe some of the popular approaches, building on the exposition and terminology of Montgomery [13].

We define a number of variables describing the synchronization process. In general, the subscript n represents the nth packet in a synchronization unit, $n = 1, 2, \ldots$. Let a_n, d_n, p_n and t_n be the arrival time, variable delay, playout time and generation time of the nth packet, respectively. Let τ denote the fixed delay from sender to receiver, due to propagation delays and clock differences. Finally, $d_{\max}$ describes the *estimated* maximum variable delay within the network. The estimate is typically chosen in such a way that only a very small fraction (on the order of 1%) of packets take more than $\tau + d_{\max}$ time units. For best performance under changing network load conditions, the estimate should be refined based on the actual delays experienced. The variable delay in a network consists of queueing and media access delays, while propagation and processing delays make up the fixed delay. Additional end-to-end fixed delay is unavoidably introduced by packetization; the non-real-time nature of most operating systems adds a variable delay both at the transmitting and receiving end. All variables are expressed in clock ticks of a clock with the same frequency, unless otherwise noted. (For simplicity, the examples ignore that the sender and receiver clocks may not run at exactly the same speed. It will be shown later that the algorithm continues to work.) Clock offsets will appear as constant network delays and are thus accomodated. The relationship between

the variables is depicted in Fig. 2. The arrows in the figure indicate the transmission of the packet across the network, occurring after the packetization delay. The packet with sequence number 5 misses the playout deadline and, depending on the algorithm used by the receiver, is either dropped or treated as the beginning of a new talkspurt.

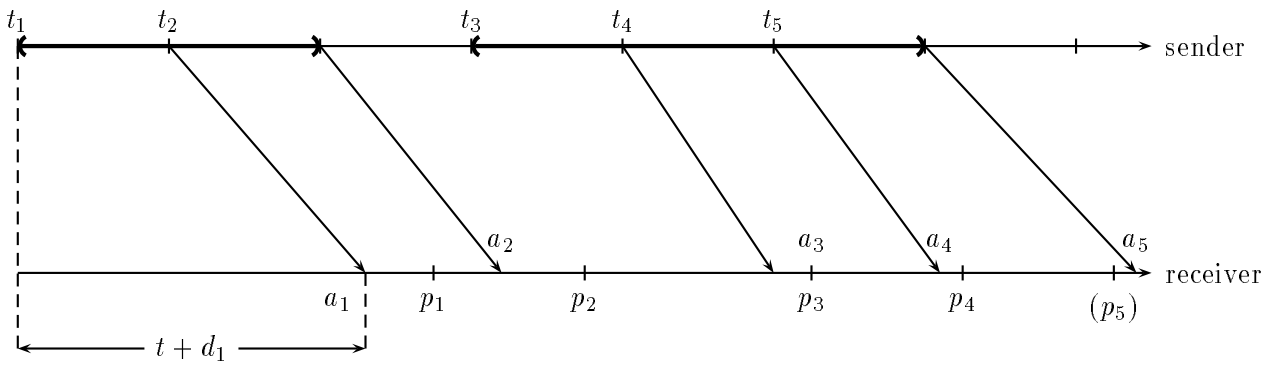

Figure 2: Playout Synchronization Variables

Given the above definitions, the relationship

$$a_n = t_n + d_n + \tau \tag{1}$$

holds for every packet. For brevity, we also define l_n as the "laxity" of packet n, i.e., the time $p_n - a_n$ between arrival and playout. Note that it may be difficult to measure a_n with resolution below a packetization interval, particularly if the measurement is to be in units related to the playback process (e.g., samples). All synchronization methods differ only in how much they delay the first packet of a synchronization unit. All packets within a synchronization unit are played out based on the position of the first packet:

$$p_n = p_{n-1} + (t_n - t_{n-1}) \text{ for } n > 1 \tag{2}$$

or

$$p_n = t_n + (p_1 - t_1) \tag{3}$$

The second equation is preferable since only a single value, $p_1 - t_1$, needs to be stored. If p and t are values of clocks with different frequencies, related by a factor c, the following rearrangement is necessary:

$$p_n = p_1 + c(t_n - t_1)$$

Three synchronization methods are of interest. We describe below how they compute the playout time for the first packet in a synchronization unit and what measurement is used to update the delay estimate $d_{\max}$.

blind delay: This method assumes that the first packet in a talkspurt experiences only the fixed delay, so that the full $d_{\max}$ has to be added to allow for other packets within the talkspurt experiencing more delay.

$$p_1 = a_1 + d_{\max}. \tag{4}$$

The estimate for the variable delay is derived from measurements of the laxity l_n, so that the new estimate after n packets is computed $d_{\max,n} = f(l_1, \ldots, l_n)$, where the function $f(\cdot)$ is a suitably chosen smoothing function. Note that blind delay does not require timestamps to determine p_1, only an indication of the beginning of a synchronization unit. Timestamps may be required to compute p_n, however, unless $t_n - t_{n-1}$ is a known constant.

absolute timing: If the packet carries a timestamp measured in time units known to the receiver, we can improve our determination of the playout point:

$$p_1 = t_1 + \tau + d_{\max}.$$

This is, clearly, the best that can be accomplished. Here, instead of estimating $d_{\max}$, we estimate $\tau + d_{\max}$ as some function of $p_n - t_n$. For this computation, it does not matter whether p and t are measured with clocks sharing a common starting point, but they have to be convertible into each other if their clock frequencies should differ.

added variable delay: Each node adds the variable delay experienced within it to a delay accumulator within the packet, yielding d_n.

$$p_1 = a_1 - d_1 + d_{\max}$$

From Eq. 1, it is readily apparent that absolute delay and added variable delay yield the same playout time. The estimate for $d_{\max}$ is based on the measurements for d. Given a clock with suitably high resolution, these estimates can be better than those based on the difference between a and p; however, it requires that all routers can recognize RTP packets. Also, determining the residence time within a router may not be feasible.

For absolute timing, we need to compute τ, which is reasonably estimated as the minimum delay seen during a synchronization period.

In summary, absolute timing is to be preferred due to its lower delays compared to blind delay, while synchronization using added variable delays is currently not feasible within the Internet (it is, however, used for G.764).

5.1.2 Detection of Synchronization Units

The receiver must have a way of readily detecting the beginning of a synchronization unit, as the playout scheduling of the first packet in a synchronization unit differs from that in the remainder of the unit. This detection has to work reliably even with packet reordering; for example, reordering at the beginning of a talkspurt is particularly likely since common silence detection algorithms send a group of stored packets at the beginning of the talkspurt to prevent front clipping.

Two basic methods have been proposed:

timestamp and sequence number: The sequence number increases by one with each packet transmitted, while the timestamp reflects the total time covered, measured in some appropriate unit. A packet is declared to start a new synchronization unit if (a) it has the highest timestamp and sequence number seen so far (within this wraparound cycle) and (b) the difference in timestamp values (converted into a packet count) between this and the previous packet is greater than the difference in sequence number between those two packets.

This approach has the disadvantage that it may lead to erroneous packet scheduling with blind delay if packets are reordered. An example is shown in Table 3. In the example, the playout delay is set at 50 time units for blind timing and 550 time units for absolute timing. The packet intergeneration time is 20 time units.

		blind timing				absolute timing	
		no reordering		with reordering			
seq.	timestamp	arrival	playout	arrival	playout	arrival	playout
200	1020	1520	1570	1520	1570	1520	1570
201	1040	1530	1590	1530	1590	1530	1590
202	1220	1720	1770	1725	1750	1725	1770
203	1240	1725	1790	1720	1770	1720	1790
204	1260	1792	1810	1791	1790	1791	1810

Table 3: Example where out-of-order arrival leads to packet loss for blind timing

More significantly, detecting synchronization units requires that the playout mechanism can translate timestamp differences into packet counts, so that it can compare timestamp and sequence number differences. If the timespan "covered" by a packet changes with the encoding or even varies for each packet, this may be cumbersome. NVP provides the timestamp/sequence number combination for detecting talkspurts. The following method avoids these drawbacks, at the cost of one additional header bit.

synchronization bit: The beginning of a synchronization unit is indicated by setting a synchronization bit within the header. The receiver, however, can only use this information if no later packet has already been processed. Thus, packet reordering at the beginning of a talkspurt leads to missing opportunities for delay adjustment. With the synchronization bit, a sequence number is not necessary to detect the beginning of a synchronization unit, but a sequence number remains useful for detecting packet loss and ordering packets bearing the same timestamp. With just a timestamp, it is impossible for the receiver to get an accurate count of the number of packets that it should have received. While gaps within a talkspurt give some indication of packet loss, the receiver cannot tell what part of the tail of a talkspurt has been transmitted. (Example: consider the talkspurts with time stamps 100, 101, 102, 110, 111. Packets with timestamp 100 and 110 have the synchronization bit set. The receiver has no way of knowing whether it was supposed to have received two talkspurts with a total of five packets, or two or more talkspurts with up to 12 packets.) The synchronization bit

is used by `vat`, without a sequence number. It is also contained in the original version of NVP [14]. A special sequence number, as used by G.764, is equivalent.

5.1.3 Interpretation of Synchronization Bit

Two possibilities for implementing synchronization bits are discussed here. Note that they can co-exist within the same protocol.

start of synchronization unit: The first packet in a synchronization unit is marked with a set synchronization bit. With this use of the synchronization bit, the receiver detects the beginning of a synchronization unit with the following simple algorithm:

```
if synchronization bit = 1
   and current sequence number > maximum sequence number seen so far
then
  this packet starts a new synchronization unit

if current sequence number > maximum sequence number
then
  maximum sequence number := current sequence number
endif
```

Comparisons and arithmetic operations are modulo the sequence number range.

end of synchronization unit: The last packet in a synchronization unit is marked. As pointed out elsewhere, this information may be useful for initiating appropriate fill-in during silence periods and to start processing a completed video frame. If a voice silence detector uses no hangover, it may have difficulty deciding which is the last packet in a talkspurt until it judges the first packet to contain no speech. The detection of a new synchronization unit by the receiver is only slightly more complicated than with the previous method:

```
if sync_flag then
  if sequence number >= sync_seq then
    sync_flag := FALSE
  endif
  if sequence number = sync_seq then
    signal beginning of synchronization unit
  endif
endif

if synchronization bit = 1 then
  sync_seq  := sequence number + 1
  sync_flag := TRUE
endif
```

By changing the equal sign in the second comparison to 'if sequence number > sync_seq', a new synchronization unit is detected even if packets at the beginning of the synchronization unit are reordered. As reordering at the beginning of a synchronization unit is particularly likely, for example when transmitting the packets preceding the beginning of a talkspurt, this should significantly reduce the number of missed talkspurt beginnings. G.764 implements the inverse of a end-of-synchronization unit bit as a so-called "more" bit which is set to one for all but the last packet within a talkspurt.

6 Timing and Synchronization

Providing the necessary information for timing recovery and synchronization is central to the role of any protocol carrying real-time information. On initial inspection, it appears as all that is required is a timestamp. However, the notion of a timestamp becomes more complicated and involves a set of trade-offs on closer inspection. This section will investigate the choices offered to the protocol designer.

We define *system clock* as the clock that is used to display wallclock (real) time, which is returned, for example, by the `gettimeofday()` system call. Often, this clock is synchronized by a protocol such as NTP so that its notion of time is close (say, within a few tens of milliseconds) to universal time. Its user-visible resolution can range from a microsecond to above ten milliseconds.

The *sample clock*, on the other hand, drives the acquisition of the real-time data to be transported. For telephone-quality audio, for example, the sample clock will tick at 8000 Hz. Its value is usually not directly accessible to a program, but can only be deduced from the number of samples obtained from the input device. The sample clock on workstations is commonly derived from a simple crystal oscillator with a frequency accuracy not much better than 10^{-4} (50 to 100 ppm), with significant drift over time and temperature. However, other sample clocks are extremely accurate; in particular, the reference clocks of synchronous networks such as the AT&T long-distance networks or the clocks used to generate the sync signal in NTSC color television production may be accurate to 10^{-9} or better. Since we are concerned with workstations and consumer products, we assume that the sample clock ticks at a rough approximation to its nomimal frequency, with no obvious relation to the system clock.

The timestamp mechanism has to address three problems:

1. the system clock is often slaved to a universal notion of time, the sample clock rarely is
2. communicating hosts have slightly different notions of system time
3. system clock and sample clock cannot be correlated exactly

6.1 Timestamp Format

The timestamp has to satisfy the requirements listed below. We define *native* clock frequency as the frequency used internally by the particular payload, e.g., the sample rate of 8000 for telephone-quality audio.

Conversion: Since much of the handling of timestamps is media-specific, the timestamp should either be in the format most appropriate for the media (e.g., samples for audio) or easily and without off-by-one error convertible into the appropriate native clock frequency.

Synchronization: The timestamp, possibly augmented by other information, should contain enough information for inter-media and intra-media synchronization.

Range: The range of the timestamp must be large enough so that there is no ambiguity at the receiver, particularly after loosing a number of packets.

Live and recordings: The timestamp should work with both live transmission and playbacks of earlier recordings, with live transmission and playback of different sources possibly occurring at the same time.

Change in format: The receiver should stay synchronized if a sender changes encoding to a different sample rate, even if packets during the transition are lost.

In addition, it is desirable that applications such as quality-of-service monitoring and media recording tools can work without detailed knowledge of the media carried and without a lot of out-of-band side information. The latter is important since these tools are likely to be participating quietly in, say, a multicast conference, without being part of the session setup, and thus may not have access to all the higher-layer parameters exchanged. Providing a table of mappings from default formats to clock-frequencies is workable, however.

We can characterize timestamps by their clock frequency and their clock origin. Since the two are subtly related, we look at some possible combinations:

fixed frequency, fixed origin: The clock ticks at a fixed frequency starting at some well-known origin in real-time. Thus, given the approximate system time and the timestamp, the receiver can deduce the time of generation. Appendix A shows that and how this timestamp can be converted to and from any other clock frequency lower than the timestamp clock frequency without introducing errors. In other words, the sender can keep a timestamp in some media-appropriate manner and convert it to the fixed-frequency timestamp, with truncation as necessary. The receiver can reverse that conversion and arrive at the same value that the sender started out with. However, this only works if the receiver can reconstruct the full, untruncated timestamp, e.g., through common knowledge of the approximate universal (wall-clock) time.

Because of the reconstruction requirement, a playback application has to translate the timestamps to the sending time by adding (modulo the timestamp width) an appropriate offset.

This approach has the advantage that tools can be completely oblivious to the particular media and its native clock frequency. For example, there can be a single recorder and playback tool for all media, with no set-up required. Also, these tools can ignore packets declaring format changes (the FMT packet in RTP).

This approach has the disadvantage that conversion to and from the native frequency to the fixed clock frequency requires some care and a few floating-point operations per packet.

native frequency, random origin: The sender picks some random starting value for the timestamp and increments it appropriately. This slightly simplifies the code for senders and playback applications. It also removes one predictable element from the header, which might be helpful for preventing known-plaintext attacks on encrypted data. However, clock conversion may be problematic unless the sample clock can be related to system time by some other information, such as extra packets containing a mapping between the two.

If the receiver performs sample rate conversion, the use of native frequency does not avoid clock conversion. Sample rate conversion is particularly common with audio, where the playout buffer would be measured in units of samples at the playout sample frequency, while different senders may send audio at different sample rates. Thus, conversion at playout time would not be an option.

We illustrate this case through an example of a sender with a sample clock of 44100 Hz and a receiver clock of 8000 Hz. Simply dividing the incoming 32-bit timestamp by 44100/8000 (5.5125) will leave the resulting timestamp space wrapping around at 779133989. Thus, with this simple conversion of the timestamp, delays between the playout point and the packet timestamp cannot be readily computed. (These delay computations are needed to get the absolute delay and delay variance.) For computing the insertion point, the slightly less convenient difference-and-add method of Equation 2 has to be used rather than Equation 3.

native frequency, fixed origin: This approach avoids unnecessary conversions in receivers, but makes it more difficult to write universal tools. Tools also have to track packets on a per-site basis that announce format changes, as they may imply clock frequency changes.

The sender computes the low-order bits of the native clock once on starting up the stream. This computation is rather trivial:

```
t = fmod((time(0) + 2208988800) * c, 4294967296.);
```

Here, c is the floating-point conversion factor from seconds to the native sample clock frequency. The offset converts the Unix time origin to the NTP time origin. The modulus is 2**32. When acquiring data, the sample clock t increments without regard to any notion of system time, e.g., one tick for every audio sample. The receiver can, in most cases, ignore the fact that the clock has a known origin. If it is convenient to convert the timestamp clock frequency, this can be done accurately (in all but pathological cases).

As noted for the corresponding fixed-frequency case, the fixed origin requires that playback applications modify the timestamp.

fixed frequency, random origin: For the reasons discussed, this approach fails when converting clock values to the native clock frequency, unless timestamp values can be related by some other means to system time.

6.2 Timestamp Frequency

We briefly survey some existing timestamp formats. All of them ignore the difference between sample time and system time, or, rather, assume that the two run at the same accurate rate.

SMPTE time code: SMPTE (commonly pronounced sehm-tee) stands for the Society of Motion Picture and Television Engineers, but has come to denote a particular way of describing frame-based video timing. The SMPTE time code is an international standard [15]. It consists of hours, minutes, seconds and frames since the start of the stream, video clip, etc. For NTSC, the frame rate is 29.97 frames per second. This is handled by dropping the first frame index of every minute, except every tenth minute.

SMPTE time codes are also used by the MPEG I encoding [16, p. 131], with a one-bit drop flag, five bits for hours, six bits for minutes, 1 'fixed' bit, six bits for seconds and a six-bit frame number. (Valid frame numbers range from 0 to 59).

MPEG: The MPEG I encoding uses a 33 bit clock with a resolution of 90 kHz [16] as the system clock reference and for presentation time stamps. The frequency was chosen based on the divisibility by the nominal video picture rates of 24 Hz, 25 Hz, 29.97 Hz and 30 Hz [16, p.42]. The frequency would also fit nicely with the 20 ms audio packetization interval. The length of 33 bit is clearly inappropriate, however, for software implementations. 32 bit timestamps still cover more than half a day and thus can be readily extended to full unique timestamps or 33 bits if needed. "There is no requirement that the PTS value be initialized to any particular value at the start of the stream." [16, p. 42].

QuickTime: In the movie resource header, the time scale relates the movie's internal time coordinates to seconds. The time scale, measured in ticks per second, is expressed as a 32-bit integer [17, p. 4-10]. Audio sample rates are expressed as 32-bit binary fixed point numbers with a 16-bit fractional part.

NVP timestamp: NVP uses 10-bit per-packet timestamps without fixed time origin. This approach very efficient in terms of processing and bit-use, but cannot be used without out-of-band information if the time interval of media "covered" by a packet varies from packet to packet. This occurs for example with variable-rate encoders or if the packetization interval is changed during a conference. Note that there is no *inherent* necessity that all participants within a conference use the same packetization interval. Local implementation considerations such as available clocks may suggest different intervals. As another example, consider a conference with audience feedback. For the lecture audio, a long packetization interval may be desirable to better amortize packet headers, while for side chats, delays are more important, thus suggesting a shorter packetization interval. The NVP timestamp wraps around after only 20.5 seconds for 20 ms audio packets.

6.3 Synchronizing Sample Clock with System Clock

Due to the drift and skew of the sample clock, system clock and sample clock are likely to diverge with time. Also, all sample clocks within the group of hosts participating in a real-time association are likely to differ in rate. For simplicity, we will discuss only with synchronizing several incoming streams at a single site (many-to-one) so that all media generated at some time t will be delivered to the end application at some time $t + D$, despite different network delays and clock drift. for a broader approach, including the one-to-many case, see [18]. Maintaining synchronization across receivers will require additional coordination messages.

The difficulty of compensating for the difference in speed between sample clocks and system clocks depends on the type of data carried. We can distinguish continuous media and media with synchronization units. Continuous audio, without talkspurts, such as music is one prominent example of continuous media. Video and voice (with silence detection) can be broken up into either periodic or irregular synchronization units, namely frames and talkspurts.

The adjustments described here apply both to compensating for network delay variations and clock skew. For continuous media, adjustments at the receiver can be done by changing the sample clock frequency, or inserting/dropping individual samples. Changing the sample clock frequency is usually difficult, although there are some devices with continuously adjustable nominal sampling rate. Adjustment of the sample clock is also difficult if several streams are mixed, each with their own slightly different sample clock frequency. Adding and dropping samples may be perceptible; however, casual experiments indicate that dropping on the order of ten samples every few seconds is barely noticeable. Larger adjustments can be made if appropriate filtering is employed. This adjustment is best performed periodically, in effect inserting artificial synchronization units.

For media consisting of synchronization units, adjustments can be hidden in the pauses between synchronization units, as long as the adjustment does not completely obliterate the pause.

We consider the effect of skew between sender and receiver sample clocks and their system clocks in a range of scenarios. In our examples, we let our system and sample clock tick at the same rate; all system clocks are perfectly synchronized. The network delay is assumed to be 200 units. For media with synchronization units, we use the designation talkspurt for simplicity. In our examples, the sender sample clock is fast, that is, its frequency is 10% above the nominal rate. The receiver sample clock runs at its nominal rate. We use absolute timing, where it matters.

6.3.1 Synchronization Units, no Sender Adjustment

This is the most simple case. The sender does not adjust the timestamp to system time; synchronization units allow compensation without being objectionable. At time 0, the first talkspurt is sent, which arrives at 200. It will get played back immediately at time 200. The next talkspurt starts at time 1000, but sample clock 1100. It arrives at 1200. If we blindly add 200 to arrive at the playout delay, it would get played 100 units too late. However, the algorithm measuring network delay variation will have observed that the apparent network delay, measured according

to the timestamps, is now only 100 and will play out the packet at the correct sample time 1200. Thus, clock drift is automatically compensated for. Note that the receiver has no need to know about the system sending time; any system clock difference between receivers is immaterial.

6.3.2 Sender Adjustment

Sender adjustment by itself is not sufficient since the receiver's sample clock is likely not in complete agreement with system time, so that the receiver has to perform additional adjustment. Also, it may well be that sender and receiver sample clocks are both either fast or slow, so that less compensation is required by just having the receiver take care of the difference.

6.3.3 Synchronization between Streams

As pointed out in [18] and elsewhere, many-to-one synchronization, that is, synchronizing several sources at a single receiver, only requires agreeing on the maximum playout compensation. For conciseness of explanation, we consider two media, say audio and video, and assume that sample clock and system clock have the same frequency. We assume that the receiver transmits the actual system time when the first packet in a synchronization unit was generated, either through the timestamp itself or some other side information. In detail, the steps are:

1. Compute the playout delay in sample clock units, that is, subtract the insertion point of the beginning of the synchronization unit from the current playout pointer. Convert to system clock units.
2. Compare playout delays of all streams to be synchronized.
3. Compute the actual playout delay in system time units based on the system time information provided with each stream and the system time of playout. In practice, this means that the playout delay in sample clock is computed as the difference between the insertion point for the beginning of the synchronization unit and the current playout pointer. This value is then added to the generation time. This does yield a slight inaccuracy if the playout sample clock is fast or slow with respect to the system clock, but with a playout delay of one second, the inaccuracy is likely less than 0.1 ms[3] and thus negligible.
4. determine the maximum playout delay among all streams, measured in system time units
5. each stream

This scheme only uses system time for synchronization and works even if none of the four sample clocks (audio and video at sender and receiver) are synchronized to each other or global system time.

[3] using a standard, low-cost crystal oscillator as a guide

6.3.4 Recommendation

Any choice of timestamp must be a compromise between several competing design objectives, in particular, simplicity of media-specific applications such as conferencing tools and media-independent applications such as record and playback applications. The skew between system and sample clock requires information allowing the receiver to align the two if needed for inter-media synchronization. The general principle of not having the sender do something that the receiver has to do anyway argues for not closely aligning the sample clock with system clock at the sender. Since clock frequency conversion does impose some, if moderate complexity on senders and receivers, using native timestamp frequency appears preferable. It should be noted, however, that timestamp conversion will still be necessary for some applications, in particular, where receivers and senders may operate at different (audio) sample rates. This, and the capability to easily define profiles with a fixed clock frequency, argues for maintaining the capability for doing accurate sample clock conversion by logically starting the sample count at a globally known system time. In summary, the combination of a timestamp based on the native clock, started at a known point in time, offers sufficient flexibility and ease of implementation.

7 Segmentation and Reassembly

For high-bandwidth video, a single frame may not fit into the maximum transport unit (MTU). Thus, some form of frame sequence number is needed. If possible, the same sequence number should be used for synchronization and fragmentation. Six possibilities suggest themselves:

overload the timestamp: No sequence number is used. Within a frame, the timestamp has no meaning. Since it is used for synchronization only when the synchronization bit is set, the other timestamps can just increase by one for each packet. However, as soon as the first frame gets lost or reordered, determining positions and timing becomes difficult or impossible.

packet count: The sequence number is incremented for every packet, without regard to frame boundaries. If a frame consists of a variable number of packets, it may not be clear what position the packet occupies within the frame if packets are lost or reordered. Continuous sequence numbers make it possible to determine if all packets for a particular frame have arrived, but only after the first packet of the next frame, distinguished by a new timestamp, has arrived.

packet count within a frame: The sequence number is reset to zero at the beginning of each frame. This approach has properties complementary to continuous sequence numbers.

packet count and first-packet sequence number: Packets use a continuously incrementing sequence number plus an option field in every packet indicating the initial sequence number within the playout unit[4] . Carrying both a continuous and packet-within-frame count achieves the same effect.

[4] suggested by Steve Casner

packet count with last-packet sequence number: Packets carry a continuous sequence number plus an option in every packet indicating the last sequence number within the playout unit. This has the advantage that the receiver can readily detect when the last packet for a playout unit has been received. The transmitter may not know, however, at the beginning of a playout unit how many packets it will comprise. Also, the position within the playout unit is more difficult to determine if the initial packet and the previous frame is lost.

packet count and frame count: The sequence number counts packets, without regard to frame boundaries. A separate counter increments with each frame. Detecting the end of a frame is delayed until the first packet belonging to the next frame. Also, the frame count cannot help to determe the position of the packet within a frame.

It could be argued that encoding-specific location information should be contained within the media part, as it will likely vary in format and use from one media to the next. Thus, frame count, the sequence number of the last or first packet in a frame etc. belong into the media-specific header.

The size of the sequence number field should be large enough to allow unambiguous counting of expected vs. received packets. A 16-bit sequence number would wrap around every 20 minutes for a 20 ms packetization interval. Using 16 bits may also simplify modulo arithmetic.

8 Source Identification

8.1 Bridges, Translators and End Systems

It is necessary to be able to identify the origin of the real-time data in terms meaningful to the application. First, this is required to demultiplex sites (or sources) within the same conference. Secondly, it allows an indication of the currently active source.

Currently, NVP makes no explicit provisions for this, assuming that the network source address can be used. This may fail if intermediate agents intervene between the content source and final destination. Consider the example in Fig. 3. An RTP-level *bridge* is defined as an entity that transforms either the RTP header or the RTP media data or both. Such a bridge could for example merge two successive packets for increased transport efficiency or, probably the most common case, translate media encodings for each stream, say from PCM to LPC (called transcoding). A *synchronizing bridge* is defined here as a bridge that recreates a synchronous media stream, possibly after mixing several sources. An application that mixes all incoming streams for a particular conference, recreates a synchronous audio stream and then forwards it to a set of receivers is an example of a synchronizing bridge. A synchronizing bridge could be built from two end system applications, with the first application feeding the media output to the media input of the second application and vice versa.

In figure 3, the bridges are used to translate audio encodings, from PCM and ADPCM to LPC. The bridge could be either synchronizing or not. Note that a resynchronizing bridge is only necessary

if audio packets depend on their predecessors and thus cannot be transcoded independently. It may be advantageous if the packetization interval can be increased. Also, for low speed links that are barely able to handle one active source at a time, mixing at the bridge avoids excessive queueing delays when several sources are active at the same time. A synchronizing bridge has the disadvantage that it always increases the end-to-end delay.

We define *translators* as transport-level entities that translate between transport protocols, but leave the RTP protocol unit untouched. In the figure, the translator connects a multicast group to a group of hosts that are not multicast capable by performing transport-level replication.

We define an *end system* as an entity that receives and generates media content, but does not forward it.

We define three types of sources: the *content source* is the actual origins of the media, e.g., the talker in an audiocast; a *synchronization source* is the combination of several content sources with its own timing; *network source* is the network-level origin as seen by the end system receiving the media.

The end system has to synchronize its playout with the synchronization source, indicate the active party according to the content source and return media to the network source. If an end system receives media through a resynchronizing bridge, the end system will see the bridge as the network and synchronization source, but the content sources should not be affected. The translator does not affect the media or synchronization sources, but the translator becomes the network source. (Note that having the translator change the IP source address is not possible since the end systems need to be able to return their media to the translator.) In the (common) case where no bridge or translator intercepts packets between sender and receiver, content, synchronization and network source are identical. If there are several bridges or translators between sender and receiver, only the last one is visible to the receiver.

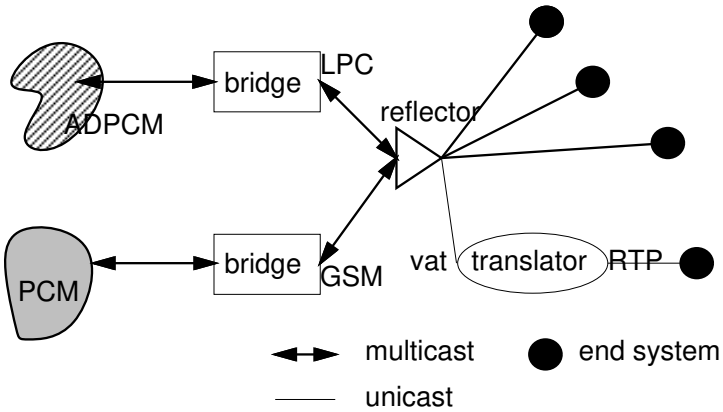

Figure 3: Bridge topology

`vat` audio packets include a variable-length list of at most 64 4-byte identifiers containing all content sources of the packet. However, there is no convenient way to distinguish the synchronization source from the network source. The end system needs to be able to distinguish synchronization sources

because jitter computation and playout delay differ for each synchronization source.

8.2 Address Format Issues

The limitation to four bytes of addressing information may not be desirable for a number of reasons. Currently, it is used to hold an IP address. This works as long as four bytes are sufficient to hold an identifier that is unique throughout the conference and as long as there is only one media source per IP address. The latter assumption tends to be true for many current workstations, but it is easy to imagine scenarios where it might not be, e.g., a system could hold a number of audio cards, could have several audio channels (Silicon Graphics systems, for example) or could serve as a multi-line telephone interface.[5]

The combination of IP address and source port can identify multiple sources per site if each content source uses a different source port. For a small number of sources, it appears feasible, if inelegant, to allocate ports just to distinguish sources. In the PBX example a single output port would appear to be the appropriate method for sending all incoming calls across the network. The mechanisms for allocating unique file names could also be used. The difficult part will be to convince all applications to draw from the same numbering space.

For efficiency in the common case of one source per workstation, the convention (used in `vat`) of using the network source address, possibly combined with the user id or source port, as media and synchronization source should be maintained.

There are several possible approaches to naming sources. We compare here two examples representing naming through globally unique network addresses and through a concatenation of locally unique identifiers.

The receiver needs to be able to uniquely identify the content source so that speaker indication and labeling work. For playout synchronization, the synchronization source needs to be determined. The identification mechanism has to continue to work even if the path between sender and receiver contains multiple bridges and translators.

Also, in the common case of no bridges or translators, the only information available at the receiver is the network address and source port. This can cause difficulties if there is more than one participant per host in a certain conference. If this can occur, it is necessary that the application opens two sockets, one for listening bound to the conference port number and one for sending, bound to some locally unique port. That randomly chosen port should also be used for reverse application data, i.e., requests from the receiver back to the content source. Only the listening socket needs to be a member of the IP multicast group. If an application multiplexes several locally generated sources, e.g., an interface to an audio bridge, it should follow the rules for bridges, that is, insert content source information.

[5] If we are willing to forego the identification with a site, we could have a multiple-audio channel site pick unused IP addresses from the local network and associate it with the second and following audio ports.

8.3 Globally unique identifiers

Sources are identified by their network address and the source port number. The source port number rather than some other integer has to be chosen for the common case that RTP packets contain no SSRC or CSRC options. Since the SDES option contains an address, it has to be the network address plus source port, no other information being available to the receiver for matching. (The SDES address is not strictly needed unless a bridge with mixing is involved, but carrying it keeps the receiver from having to distinguish those cases.) Since tying a protocol too closely to one particular network protocol is considered a bad idea (witness the difficulty of adopting parts of FTP for non-IP protocols), the address should probably have the form of a type-lenght-value field. To avoid having to manage yet another name space, it appears possible to re-use the Ethertype values, as all commonly used protocols with their own address space appear to have been assigned such a value. Other alternatives, such as using the BSD Unix `AF_` constants suffer from the drawback that there does not appear to be a universally agreed-upon numbering. NSAPs can contain other addresses, but not every address format (such as IP) has an NSAP representation. The receiver application does not need to interpret the addresses themselves; it treats address format identifier (e.g., the Ethertype field) and address as a globally unique byte string. We have to assure a single host does not use two network addresses, one for transmission and a different one in the SDES option.

The rules for adding CSRC and SSRC options are simple:

end system: End systems do not insert CSRC or SSRC options. The receiver remembers the CSRC address for each site; if none is explicitly specified, the SSRC address is used. If that is also missing, the network address is used. SDES options are matched to this content source address.

bridge: A bridge adds the network source address of all sources contributing to a particular outgoing packet as CSRC options. A bridge that receives a packet containing CSRC options may decide to copy those CSRC options into an outgoing packet that contains data from that bridge.

translator: The translator checks whether the packet already contains a SSRC (inserted by an earlier translator). If so, no action is required. Otherwise, the translator inserts an SSRC containing the network address of the host from which the packet was received.

The SSRC option is set only by the translator, unless the packet already bears such an option.

Globally unique identifiers based on network addresses have the advantage that they simplify debugging, for example, allowing to determine which bridge processed a message, even after the packet has passed through a translator.

8.4 Locally unique addresses

In this scheme, the SSRC, CSRC and SDES options contain locally unique identifiers of some length. For lengths of at least four bytes, it is sufficient to have the application pick one at random, without local coordination, with sufficiently low probability of collision within a single host. The receiver creates a globally unique identifier by concatenating the network address and one or more random identifiers. The synchronization source is identified by the concatenation of the SSRC identifier and the network address. Only translators are allowed to set the SSRC option. If a translator receives an RTP packet which already contains an SSRC option, as can occur if a packet traverses several translators, the translator has to choose a new set of values, mapping packets with the same network source, but different incoming SSRC value into different outgoing SSRC values. Note that the SSRC constitute a label-swapping scheme similar to that used for ATM networks, except that the assocation setup is implicit. If a translator loses state (say, after rebooting), the mapping is simply reestablished as packets arrive from end systems or other translators. Until the receivers timeout, a single source may appear twice and there may be a temporary confusion of sources and their descriptors.

The rules are:

end system: An end system never inserts CSRC options and typically does not insert an SSRC option. An end system application may insert an SSRC option if it originates more than one stream for a single conference through a single network and transport address, e.g., a single UDP port. The SDES option contains a zero for the identifier, indicating that the receiver is to much on network address only. The receiver determines the synchronization source as the concatenation of network source and synchronization source.

bridge: A bridge assigns each source its own CSRC identifier (non-zero), which is then used also in the SDES option.

translator: The translator maintains a list of all incoming sources, with their network and SSRC, if present. Sources without SSRC are assigned an SSRC equal to zero. Each of these sources is assigned a new local identifier, which is then inserted into the SSRC option.

Local identifiers have advantages: the length of the identifiers within the packet are significantly shorter (four to six vs. at least ten bytes with padding); comparison of content and synchronization source are quicker (integer comparison vs. variable-length string comparison). The identifiers are meaningless for debugging. In particular, it is not easy for the receiver sitting behind a translator and a bridge to determine where a bridge is located, unless the bridge identifies itself periodically, possibly with another SDES-like option containing the actual network address.

The major drawbacks appear to be the additional translator complexity: translators needs to maintain a mapping from incoming network/SSRC to outgoing SSRC.

Note that using IP addresses as "random" local identifiers is not workable if there is any possibility that two sources participating in the same conference can coexist on the same host.

A somewhat contrived scenaria is shown in Fig. 4.

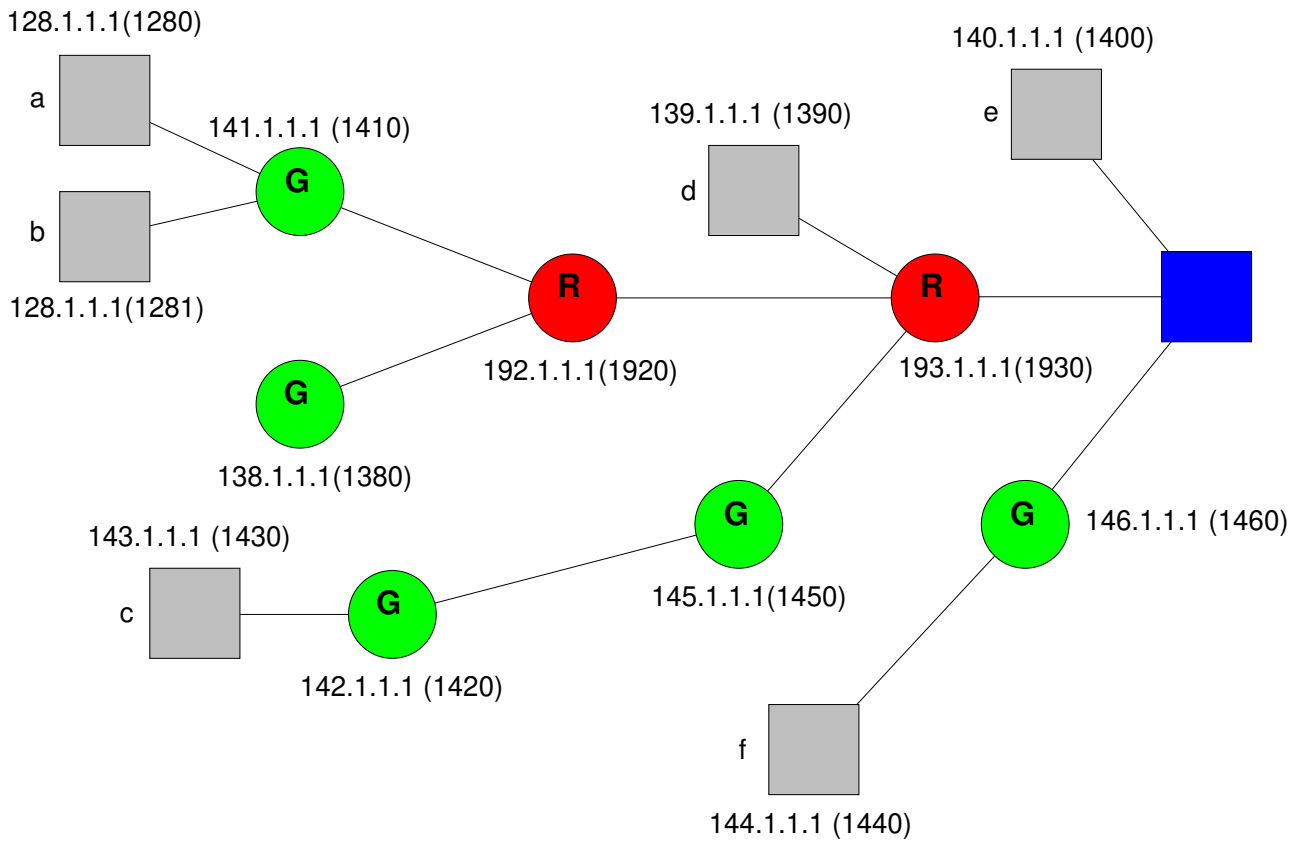

Figure 4: Complicated topology with translators (R) and bridges (G)

9 Energy Indication

G.764 contains a 4-bit noise energy field, which encodes the white noise energy to be played by the receiver in the silences between talkspurts. Playing silence periods as white noise reduces the noise-pumping where the background noise audible during the talkspurt is audibly absent at the receiver during silence periods. Substituting white noise for silence periods at the receiver is not recommended for multi-party conferences, as the summed background noise from all silent parties would be distractive. Determining the proper noise level appears to be difficult. It is suggested that the receiver simply takes the energy of the last packet received before the beginning of a silence period as an indication of the background noise. With this mechanism, an explicit indication in the packet header is not required.

10 Error Control

In principle, the receiver has four choices in handling packets with bit errors [19]:

no checking: the receiver provides no indication whether a data packet contains bit errors, either because a checksum is not present or is not checked.

discard: the receiver discards errored packets, with no indication to the application.

receive: the receiver delivers and flags errored packets to the application.

correct: the receiver drops errored packets and requests retransmission.

It remains to be decided whether the header, the whole packet or neither should be protected by checksums. NVP protects its header only, while G.764 has a single 16-bit check sequence covering both datalink and packet voice header. However, if UDP is used as the transport protocol, a checksum over the whole packet is already computed by the receiver. (Checksumming for UDP can typically be disabled by the sending or receiving host, but usually not on a per-port basis.) ST-II does not compute checksums for its payload. Many data link protocols already discard packets with bit errors, so that packets are rarely rejected due to higher-layer checksums.

Bit errors within the data part may be easier to tolerate than a lost packet, particularly since some media encoding formats may provide built-in error correction. The impact of bit errors within the header can vary; for example, errors within the timestamp may cause the audio packet to be played out at the wrong time, probably much more noticeable than discarding the packet. Other noticeable effects are caused by a wrong flow or encoding identifier. If a separate checksum is desired for the cases where the underlying protocols do not already provide one, it should be optional. Once optional, it would be easy to define several checksum options, covering just the header, the header plus a certain part of the body or the whole packet.

A checksum can also be used to detect whether the receiver has the correct decryption key, avoiding noise or (worse) denial-of-service attacks. For that application, the checksum should be computed across the whole packet, before encrypting the content. Alternatively, a well-known signature could be added to the packet and included in the encryption, as long as known plaintext does not weaken the encryption security.

Embedding a checksum as an option may lead to undiscovered errors if the the presence of the checksum is masked by errors. This can occur in a number of ways, for example by an altered option type field, a final-option bit erroneously set in options prior to the checksum option or an erroneous field length field. Thus, it may be preferable to prefix the RTP packet with a checksum as part of the specification of running RTP over some network or transport protocol. To avoid the overhead of including a checksum even in the common case where it is not needed, it might be appropriate to distinguish two RTP protocol variations through the next-protocol value in the lower-layer protocol header; the first would include a checksum, the second would not. The checksum

itself offers a number of encoding possibilities[6] :

- have two 16-bit checksums, one covering the header, the other the data part
- combine a 16-bit checksum with a byte count indicating its coverage, thus allowing either a header-only or a header-plus-data checksum

The latter has the advantage that the checksum can be computed without determining the header length.

The error detection performance and computational cost of some common 16-bit checksumming algorithms are summarized in Table 4. The implementations were drawn from [20] and compiled on a SPARC IPX using the Sun ANSI C compiler with optimization. The checksum computation was repeated 100 times; thus, due to data cache effects, the execution times shown are probably better than would be measured in an actual application. The relative performance, however, should be similar. Among the algorithms, the CRC has the strongest error detection properties, particularly for burst errors, while the remaining algorithms are roughly equivalent [20]. The Fletcher algorithm with modulo 255 (shown here) has the peculiar property that a transformation of a byte from 0 to 255 remains undetected. CRC, the IP checksum and Fletcher's algorithm cannot detect spurious zeroes at the end of a variable-length message [21]. The non-CRC checksums have the advantage that they can be updated incrementally if only a few bytes have changed. The latter property is important for translators that insert synchronization source indicators.

algorithm	ms
IP checksum	0.093
Fletcher's algorthm, optimized [21]	0.192
CRC CCITT	0.310
Fletcher's algorithm, non-optimized [22]	2.044

Table 4: Execution time of common 16-bit checksumming algorithms, for a 1024-byte packet, in milliseconds

11 Security and Privacy

11.1 Introduction

The discussions in this sections are based on the work of the privacy enhanced mail (PEM) working group within the Internet Engineering Task Force, as documented in [23,24] and related documents. The reader is referred to RFC 1113 [23] or its successors for terminology. Also relevant is the work

[6] suggested by S. Casner

on security for SNMP Version 2. We discuss here how the following security-related services may be implemented for packet voice and video:

Confidentiality: Measures that ensure that only the intended receiver(s) can decode the received audio/video data; for others, the data contains no useful information.

Authentication: Measures that allow the receiver(s) to ascertain the identity of the sender of data or to verify that the claimed originator is indeed the originator of the data.

Message integrity: Measures that allow the receiver(s) to detect whether the received data has been altered.

As for PEM [23], the following privacy-related concerns are not addressed at this time:

- access control
- traffic flow confidentiality
- routing control
- assurance of data receipt and non-deniability of receipt
- duplicate detection, replay prevention, or other stream-oriented services

These services either require connection-oriented services or support from the lower layers that is currently unavailable. A reasonable goal is to provide privacy at least equivalent to that provided by the public telephone system (except for traffic flow confidentiality).

As for privacy-enhanced mail, the sender determines which privacy enhancements are to be performed for a particular part of a data transmission. Therefore, mechanisms should be provided that allow the sender to determine whether the desired recipients are equipped to process any privacy-enhancements. This is functionally similar to the negotiation of, say, media encodings and should probably be handled by similar mechanisms. It is anticipated that privacy-enhanced mail will be used in the absence of or in addition to session establishment protocols and agents to distributed keys or negotiate the enhancements to be used during a conference.

11.2 Confidentiality

Only data encryption can provide confidentiality as long as intruders can monitor the channel. It is desirable to specify an encryption algorithm and provide implementations without export restrictions. Although DES is widely available outside the United States, its use within software in both source and binary form remains difficult.

We have the choice of either encrypting and/or authenticating the whole packet or only the options and payload. Encrypting the fixed header denies the intruder knowledge about some conference details (such as timing and format) and protects against replay attacks. Encrypting the fixed header also allows some heuristic detection of key mismatches, as the version identifier, timestamp and other header information are somewhat predictable. However, header encryption makes packet traces and debugging by external programs difficult. Also, since translators may need to inspect and modify the header, but do not have access to the sender's key, at least part of the header needs to remain unencrypted, with the ability for the receiver to discern which part has been encrypted. Given these complications and the uncertain benefits of header encryption, it appears appropriate to limit encryption to the options and payload part only.

In public key cryptography, the sender uses the receiver's public key for encryption. Public key cryptography does not work for true multicast systems since the public encoding key for every recipient differs, but it may be appropriate when used in two-party conversations or application-level multicast. In that case, mechanisms similar to privacy enhanced mail will probably be appropriate. Key distribution for symmetric-key encryption such as DES is beyond the scope of this recommendation, but the services of privacy enhanced mail [23,25] may be appropriate.

For one-way applications, it may desirable to prohibit listeners from interrupting the broadcast. (After all, since live lectures on campus get disrupted fairly often, there is reason to fear that a sufficiently controversial lecture carried on the Internet could suffer a similar fate.) Again, asymmetric encryption can be used. Here, the decryption key is made available to all receivers, while the encryption key is known only to the legitimate sender. Current public-key algorithms are probably too computationally intensive for all but low-bit-rate voice. In most cases, filtering based on sources will be sufficient.

11.3 Message Integrity and Authentication

The usual message digest methods are applicable if only the integrity of the message is to be protected against tampering. Again, services similar to that of privacy-enhanced mail [26] may be appropriate. The MD5 message digest [27] appears suitable. It translates any size message into a 128-bit (16-byte) signature. On a SPARCstation IPX (Sun 4/50), the computation of a signature for a 180-byte audio packet takes approximately 0.378 ms[7] Defining the signature to apply to all data beginning at the signature option allows operation when translators change headers. The receiver has to be able to locate the public key of the claimed sender. This poses two problems: first, a way of identifying the sender unambiguously needs to be found. The current methods of identification, such as the SMTP (e-mail) address, are not unambiguous. Use of a distinguished name as described in RFC 1255 [28] is suggested.

The authentication process is described in RFC 1422 [25]:

[7] The processing rates for Sun 4/50 (40 MHz clock) and SPARCstation 10's (36 MHz clock) are 0.95 and 2.2 MB/s, respectively, measured for a single 1000-byte block. Note that timing the repeated application of the algorithm for the same block of data gives optimistic results since the data then resides in the cache.

> In order to provide message integrity and data origin authentication, the originator generates a message integrity code (MIC), signs (encrypts) the MIC using the private component of his public-key pair, and includes the resulting value in the message header in the MIC-Info field. The certificate of the originator is (optionally) included in the header in the Certificate field as described in RFC 1421. This is done in order to facilitate validation in the absence of ubiquitous directory services. Upon receipt of a privacy enhanced message, a recipient validates the originator's certificate (using the IPRA public component as the root of a certification path), checks to ensure that it has not been revoked, extracts the public component from the certificate, and uses that value to recover (decrypt) the MIC. The recovered MIC is compared against the locally calculated MIC to verify the integrity and data origin authenticity of the message.

For audio/video applications with loose control, the certificate could be carried periodically to allow new listeners to obtain it and to achieve a measure of reliability.

Symmetric key methods such as DES can also be used. Here, the key is simply prefixed to the message when computing the message digest (MIC), but not transmitted. The receiver has to obtain the sender's key through a secure channel, e.g., a PEM message. The method has the advantage that no cryptography is involved, thus alleviating export-control concerns. It is used for SNMP Version 2 authentication.

11.4 Security for RTP vs. PEM

It is the author's opinion that RTP should aim to reuse as much of the PEM technology and syntax as possible, unless there are strong reasons in the nature of real-time traffic to deviate. This has the advantage that terminology, implementation experience, certificate mechanisms and possibly code can be reused. Also, since it is hoped that RTP finds use in a range of applications, a broad spectrum of security mechanisms should be provided, not necessarily limited by what is appropriate for large-distribution audio and video conferences.

It should be noted that connection-oriented security architectures are probably unsuitable for RTP applications as they rely on reliable stream transmission and an explicit setup phase with typically only a single sender and receiver.

There are a number of differences between the security requirements of PEM and RTP that should be kept in mind:

Transparency: Unlike electronic mail, it is safe to assume that the channel will carry 8 bit data unaltered. Thus, a conversion to a canonical form or encoding binary data into a 64-element subset as done for PEM is not required.

Time: As outlined at the beginning of this document, processing speed and packet overhead have to be major considerations, much more so than with store-and-forward electronic mail. Message

digest algorithms and DES can be implemented sufficiently fast even in software to be used for voice and possibly for low-bit rate video. Even for short signatures, RSA encryption is fairly slow.

Note that the ASN.1/BER encoding of asymmetrically-encrypted MICs and certificates adds no significant processing load. For the MICs, the ASN.1 algorithm yields only additional constant bytes which a paranoid program can check, but does not need to decode. Certificates are carried much more infrequently and are relatively simple structures. It would seem unnecessary to supply a complete ASN.1/BER parser for any of the datastructures.

Space: Encryption algorithm require a minimum data input equal to their keylength. Thus, for the suggested key length for RSA encryption of 508 to 1024 bits, the 16-byte message digest expands to a 53 to 128 byte MIC. This is clearly rather burdensome for short audio packets. Applying a single message digest to several packets seems possible if the packet loss rates are sufficiently low, even though it does introduce minor security risks in the case where the receiver is forced to decide between accepting as authentic an incomplete sequence of packets or rejecting the whole sequence. Note that it would not be necessary to wait with playback until a complete authenticated block has been received; in general, a warning that authentication has failed would be sufficient for human users. The application should also issue a warning if no complete block could be authenticated for several blocks, as that might indicate that an impostor was feigning the presence of MIC-protected data by strategically dropping packets.

The initialization vector for DES in cipher block mode adds another eight bytes.

Scale: The symmetric key authentication algorithm used by PEM does not scale well for a large number of receivers as the message has to contain a separate MIC for each receiver, encrypted with the key for that particular sender-receiver pair. If we forgo the ability to authenticate an individual user, a single session key shared by all participants can thwart impostors from outside the group holding the shared secret.

12 Quality of Service Control

Because real-time services cannot afford retransmissions, they are directly affected by packet loss and delays. Delay jitter and packet loss, for example, provide a good indication of network congestion and may suggest switching to a lower-bandwidth coding. To aid in fault isolation and performance monitoring, quality-of-service (QOS) measurement support is useful. QOS of service monitoring is useful for the receiver of real-time data, the sender of that data and possibly a third-party monitor, e.g., the network provider, that is itself not part of the real-time data distribution.

12.1 QOS Measures

For real-time services, a number of QOS measures are of interest, roughly in order of importance:

- packet loss
- packet delay variation (variance, minimum/maximum)
- relative clock drift (delay between sender and receiver timestamp)

In the following, the terms receiver and sender pertain to the real-time data, not any returned QOS data. If the receiver is to measure packet loss, an indication of the number of packets actually transmitted is required. If the receiver itself does not need to compute packet loss percentages, it is sufficient for the receiver to indicate to the sender the number of packets received and the range timestamps covered, thus avoiding the need for sequence numbers. Translation into loss at the sender is somewhat complicated, however, unless restrictions on permissible timestamps (e.g., those starting a synchronization unit) are enforced. If sequence numbers are available, the receiver has to track the number of times that the sequence number has wrapped around, even in the face of packet reordering. If c denotes the cycle count, M the sequence number modulus and s_n the sequence number of the n received packet, where s_n is not necessarily larger than s_{n-1}, we can write:

$$\begin{array}{ll} c_n = c_{n-1} + 1 & \text{for } -M < s_n - s_{n-1} < -M/2 \\ c_n = c_{n-1} - 1 & \text{for } M/2 < s_n - s_{n-1} < M \\ c_n = c_{n-1} & \text{otherwise} \end{array}$$

For example, the sequence number sequence $65534, 2, 65535, 1, 3, 5, 4$ would yield the cycle number sequence $0, 1, 0, 1, 1, 1, 1$ for $M = 65536$, i.e., 16-bit sequence numbers. The total number of expected packets is then computed simply as $s_n + M * c_n - s_0 + 1$, where the first received packet has index 0.

The user of the measurements should also have some indication as to the time period they cover so that the degree of confidence in these statistical meassurements can be established.

12.2 Remote measurements

It may be desirable for the sender, interested multicast group members or a non-group member (third party) to have automatic access to quality-of-service measurements. In particular, it is necessary for the sender to gather a number of reception reports from different parts of the Internet to "triangulate" where packets get lost or delayed.

Two modes of operation can be distinguished: monitor-driven or receiver-driven. In the monitor-driven case, a site interested in QOS data for a particular sender contacts the receiver through a back channel and requests a reception report. Alternatively, each site can send reception reports to a monitoring multicast group or as session data, along with the "regular station identification" to the same multicast group used for data. The first approach requires the most implementation effort, but produces the least amount of data. The other two approaches have complementary properties.

In most cases, sender-specific quality of service information is more useful for tracking network problems than aggregrate data for all senders. Since a site cannot transmit reception reports for all senders it has ever heard from, some selection mechanism is needed, such as most-recently-heard or cycling through sites.

Source identification poses some difficulties since the network address seen by the receiver may not be meaningful to other members of the multicast group, e.g., after IP-SIP address translation. On the other hand, network addresses are easier to correlate with other network-level tools such as those used for Mbone mapping.

minimum and maximum difference between departure and arrival timestamp. This has the advantage that the fixed delay can also be estimated if sender and receiver clocks are known to be synchronized. Unfortunately, delay extrema are noisy measurement that give only limited indication of the delay variability. The receiver could also return the playout delay value it uses, although for absolute timing, that again depends on the clock differential, as well as on the particular delay estimation algorithm employed by the receiver. In summary, a minimal set of useful measurements appears to be the expected and received packet count, combined with the minimum and maximum timestamp difference.

12.3 Monitoring by Third Party

Except for delay estimates based on sequence number ranges, the above section applies for this case as well.

13 The Use of Profiles

RTP is intended to be a rather 'thin' protocol, partially because it aims to serve a wide variety of real-time services. The RTP specification intentionally leaves a number of issues open for other documents (profiles), which in turn have the goal of making it easy to build interoperable applications for a particular application domain, for example, audio and video conferences.

Some of the issues that a profile should address include:

- the interpretation of the 'content' field with the CDESC option
- the structure of the content-specific part at the end of the CDESC option
- the mechanism by which applications learn about and define the mapping between the 'content' field in the RTP fixed header and its meaning
- the use of the optional framing field prefixed to RTP packets (not used, used only if underlying transport protocol does not provide framing, used by some negotiation mechanism, always used)

- any RTP-over-x issues, that is, definitions needed to allow RTP to use a particular underlying protocol
- content-specific RTP, RTCP or reverse control options
- port assignments for data and reverse control

14 Port Assignment

Since it is anticipated that UDP and similar port-oriented protocols will play a major role in carrying RTP traffic, the issue of port assignment needs to be addressed. The way ports are assigned mainly affects how applications can extract the packets destined for them. For each medium, there also needs to be a mechanism for distinguishing data from control packets.

For unicast UDP, only the port number is available for demultiplexing. Thus, each media will need a separate port number pair unless a separate demultiplexing agent is used. However, for one-to-one connections, dynamically negotiating a port number is easy. If several UDP streams are used to provide multicast by transport-level replication, the port number issue becomes somewhat more difficult. For ST-II, a common port number has to be agreed upon by all participants, which may be difficult particularly if a new site wants to join an on-going connection, but is already using the port number in a different connection.

For UDP multicast, an application can select to receive only packets with a particular port number and multicast address by binding to the appropriate multicast address[8] . Thus, for UDP multicast, there is no need to distinguish media by port numbers, as each medium could have its designated and unique multicast group. Any dynamic port allocation mechanism would fail for large, dynamic multicast groups, but might be appropriate for small conferences and two-party conversations.

Data and control packets for a single medium can either share a single port or use two different port numbers. (Currently, two adjacent port numbers, 3456 and 3457, are used.) A single port for data and control simplifies the receiver code and translators and, less important, conserves port numbers. With the proliferation of firewalls, limiting the number of ports has assumed additional importance. Sharing a single port requires some other means of identifying control packets, for example as a special encoding code. Alternatively, all control data could be carried as options within data packets, akin to the NVP protocol options. Since control messages are also transmitted if no actual medium data is available, header content of packets without media data needs to be determined. With the use of a synchronization bit, the issue of how sequence numbers and timestamps are to be treated for these packets is less critical. It is suggested to use a zero timestamp and to increment the sequence number normally. Due to the low bandwidth requirements of typical control information, the issue of accomodating control information in any bandwidth reservation scheme should be manageable. The penalty paid is the eight-byte overhead of the RTP header for control packets that do not require time stamps, encoding and sequence number information.

[8] This extension to the original multicast socket semantics is currently in the process of being deployed.

Using a single RTCP stream for several media may be advantageous to avoid duplicating, for example, the same identification information for voice, video and whiteboard streams. This works only if there is one multicast group that *all* members of a conference subscribe to. Given the relatively low frequency of control messages, the coordination effort between applications and the necessity to designate control messages for a particular medium are probably reasons enough to have each application send control messages to the same multicast group as the data.

In conclusion, for multicast UDP, one assigned port number, for both data and control, seems to offer the most advantages, although the data/control split may offer some bandwidth savings.

15 Multicast Address Allocation

A fixed, permanent allocation of network multicast addresses to invidual conferences by some naming authority such as the Internet Assigned Numbers Authority is clearly not feasible, since the lifetime of conferences is unknown, the potential number of conferences is rather large and the available number space limited to about 2^{28}, of which 2^{16} have been set aside for dynamic allocation by conferences.

The alternative to permanent allocation is a *dynamic allocation*, where an initiator of a multicast application obtains an unused multicast address in some manner (discussed below). The address is then made available again, either implicitly or explicitly, as the application terminates.

The address allocation may or may not be handled by the same mechanism that provides conference naming and discovery services. Separating the two has the advantage that dynamic (multicast) address allocation may be useful to applications other than conferencing. Also, different mechanisms (for example, periodic announcements vs. servers) may be appropriate for each.

We can distinguish two methods of multicast address assignment:

function-based: all applications of a certain type share a common, global address space. Currently, a reservation of a 16-bit address space for conferences is one example. The advantage of this scheme is that directory functions and allocation can be readily combined, as is done in the `sd` tool by Van Jacobson. A single namespace spanning the globe makes it necessary to restrict the scope of addresses so that allocation does not require knowing about and distributing information about the existence of all global conferences.

hierarchical: Based on the location of the initiator, only a subset of addresses are available. This limits the number of hosts that could be involved in resolving collisions, but, like most hierarchical assignment, leads to sparse allocation. Allocation is independent of the function the address is used for.

Clearly, combinations are possible, for example, each local namespace could be functionally divided if sufficiently large. With the current allocation of 2^{16} addresses to conferences, hierarchical division except on a very coarse scale is not feasible.

To a limited extent, multicast address allocation can be compared to the well-known channel multiple access problem. The multicast address space plays the role of the common channel, with each address representing a time slot.

All the following schemes require cooperation from all potential users of the address space. There is no protection against an ignorant or malicious user joining a multicast group.

15.1 Channel Sensing

In this approach, the initiator randomly selects a multicast address from a given range, joins the multicast group with that address and listens whether some other host is already transmitting on that address. This approach does not require a separate address allocation protocol or an address server, but it is probably infeasible for a number of reasons. First, a user process can only bind to a single port at one time, making 'channel sensing' difficult. Secondly, unlike listening to a typical broadcast channel, the act of joining the multicast group can be quite expensive both for the listening host and the network. Consider what would happen if a host attached through a low-bandwidth connection joins a multicast group carrying video traffic, say.

Channel sensing may also fail if two sections of the network that were separated at the time of address allocation rejoin later. Changes in time-to-live values can make multicast groups 'visible' to hosts that previously were outside their scope.

15.2 Global Reservation Channel with Scoping

Each range of multicast addresses has an associated well-known multicast address and port where all initiators (and possibly users) advertise the use of multicast addresses. An initiator first picks a multicast address at random, avoiding those already known to be in use. Some mechanism for collision resolution has to be provided in the unlikely event that two initiators simultaneously choose the same address. Also, since address advertisement will have to be sent at fairly long intervals to keep traffic down, an application wanting to start a conference, for example, has to wait for an extended period of time unless it continuously monitors the allocation multicast group.

To limit traffic, it may seem advisable to only have the initiator multicast the address usage advertisement. This, however, means that there needs to be a mechanism for another site to take over advertising the group if the initiator leaves, but the multicast group continues to exist. Time-to-live restrictions pose another problem. If only a single source advertises the group, the advertisement may not reach all those sites that could be reached by the multicast transmissions themselves.

The possibility of collisions can be reduced by address reuse with scoping, discussed further below, and by adding port numbers and other identifiers as further discriminators. The latter approach appears to defeat the purpose of using multicast to avoid transmitting information to hosts that have no interest in receiving it. Routers can only filter based on group membership, not ports or

other higher-layer demultiplexing identifiers. Thus, even though two conferences with the same multicast address and different ports, say, could coexist at the application layer, this would force hosts and networks that are interested in only one of the conferences to deal with the combined traffic of the two conferences.

15.3 Local Reservation Channel

Instead of sharing a global namespace for each application, this scheme divides the multicast address space hierarchically, allowing an initiator within a given network to choose from a smaller set of multicast addresses, but independent of the application. As with many allocation problems, we can devise both server-based and fully distributed versions.

15.3.1 Hierarchical Allocation with Servers

By some external means, address servers, distributed throughout the network, are provided with non-overlapping regions of the multicast address space. An initiator asks its favorite address server for an address when needed. When it no longer needs the address, it returns it to the server. To prevent addresses from disappearing when the requestor crashes and looses its memory about allocated addresses, requests should have an associated time-out period. This would also (to some extent) cover the case that the initiator leaves the conference, without the conference itself disbanding. To decrease the chances that an initiator cannot be provided with an address, either the local server could 'borrow' an address from another server or could point the initiator to another server, somewhat akin to the methods used by the Domain Name Service (DNS). Provisions have to be made for servers that crash and may loose knowledge about the status of its block of addresses, in particular their expiration times. The impact of such failures could be mitigated by limiting the maximum expiration time to a few hours. Also, the server could try to request status by multicast from its clients.

15.3.2 Distributed Hierarchical Allocation

Instead of a server, each network is allocated a set of multicast addresses. Within the current IP address space, both class A, B and C networks would get roughly 120 addresses, taking into account those that have been permanently assigned. Contention for addresses works like the global reservation channel discussed earlier, but the reservation group is strictly limited to the local network. (Since the address ranges are disjoint, address information that inadvertently leaks outside the network, is harmless.)

This method avoids the use of servers and the attendant failure modes, but introduces other problems. The division of the address space leads to a barely adequate supply of addresses (although larger address formats will probably make that less of an issue in the future). As for any distributed algorithm, splitting of networks into temporarily unconnected parts can easily destroy the unique-

ness of addresses. Handling initiators that leave on-going conferences is probably the most difficult issue.

15.4 Restricting Scope by Limiting Time-to-Live

Regardless of the address allocation method, it may be desirable to distinguish multicast addresses with different reach. A local address would be given out with the restriction of a maximum time-to-live value and could thus be reused at a network sufficiently removed, akin to the combination of cell reuse and power limitation in cellular telephony. Given that many conferences will be local or regional (e.g., broadcasting classes to nearby campuses of the same university or a regional group of universities, or an electronic town meeting), this should allow significant reuse of addresses. Reuse of addresses requires careful engineering of thresholds and would probably only be useful for very small time-to-live values that restrict reach to a single local area network. Using time-to-live fields to restrict scope rather than just prevent looping introduces difficult-to-diagnose failure modes into multicast sessions. In particular, reachability is no longer transitive, as B may have A and C in its scope, but A and B may be outside each other's scope (or A may be in the scope of B, but not vice versa, due to asymmetric routes, etc.). This problem is aggravated by the fact that routers (for obvious reasons) are not supposed to return ICMP time exceeded messages, so that the sender can only guess why multicast packets do not reach certain receivers.

16 Security Considerations

Security issues are discussed in Section 11.

Acknowledgments

This draft is based on discussion within the AVT working group chaired by Stephen Casner. Eve Schooler and Stephen Casner provided valuable comments.

This work was supported in part by the Office of Naval Research under contract N00014-90-J-1293, the Defense Advanced Research Projects Agency under contract NAG2-578 and a National Science Foundation equipment grant, CERDCR 8500332.

A Timestamp conversion

It is easy to show that with appropriate rounding, the conversion from sample clock to RTP timestamp and back to sample clock is exact. Let x be any sample clock value, an integer. Let c

be the conversion factor between the sample clock and the RTP timestamp, for example, 8.192 for an 8 kHz sample clock. Sender and receiver round to the nearest integer, i.e., add 0.5 and truncate. The received integer sample value y can be expressed as

$$y = \left\lfloor \frac{\lfloor cx + 0.5 \rfloor}{c} + 0.5 \right\rfloor ,$$

where $\lfloor\rfloor$ expresses truncation (floor operation). We can replace the inner truncation operation and write

$$\left\lfloor \frac{cx + \Delta}{c} + 0.5 \right\rfloor ,$$

with Δ computed that the equality holds, with $|\Delta| < 1/2$. (To convince yourself that this holds, write cx as the sum of an integer and fractional part and consider the two cases of the fractional part being less than or greater than one half.) Now,

$$\begin{aligned} y &= \lfloor x + \Delta/c + 0.5 \rfloor \\ &= x + \lfloor \Delta/c + 0.5 \rfloor \\ &= x \end{aligned}$$

The second step follows since x is an integer; the third from the fact that $|\Delta/c| < 0.5$, so that the bracketed term is strictly greater than zero and strictly less than one. This relationship holds for all $c > 1$, i.e., as long as the sampling frequency is less than the RTP clock frequency.

The derivation assumes that floating-point computations are exact. As long as c can be represented exactly as a binary floating point number with less than the precision of the mantissa minus 32 bits, e.g., 21 bits for double-precision floating point, the computation is indeed precise since no information is lost during the computation. (The computation could be carried out in a suitably scaled integer format.)

The conversion works without off-by-one error if the receiver can reconstruct the full sender timestamp and if c is not perilously close to one (see the example below).

The generation and processing of fixed-frequency timestamps follows these steps:

1. At the source, upon initialization and, if desired, periodically between talkspurts thereafter, set the sample counter corresponding to the current sample to be the number of sampling intervals between the beginning of the NTP epoch and the instant that sample was taken (as accurately as that can be determined), where the beginning of the NTP epoch is midnight GMT January 1, 1900. Note that this sample counter will likely exceed 32 bits, so you may want to keep it in a double-precision floating point variable (this is also convenient for the scaling step). This calculation does not have to be done for every talkspurt, just often enough to keep the error due to drift in the sample oscillator within acceptable limits. Also note that this requires the sender to know what time it is.

2. Between the periodic calculations of (1), just increment the sample counter by the number of samples in each block read from the input device.

3. The sender calculates the fixed-frequency timestamp for transmission by scaling by the floating-point ratio of the fixed frequency to the sampling rate, adding 0.5 for rounding and extracting the lower 32 integer bits. The `fmod` function with a modulus of 2**32 can be used for the last step.

4. The receiver extends the 32-bit fixed-frequency timestamp with the high 16 bits of the NTP timestamp as shown in the function `clock_extend` in the program below. The result is a 64-bit integer or a double-precision floating point number. Note that this requires the receiver to know what time it is, but only within +/- half the wrap-around time of the fixed-frequency timestamp.

5. To reconstruct the original sample counter, the receiver scales by the ratio of the sampling rate to the fixed frequency and rounds by adding 0.5. A 32-bit sample counter can be extracted from this result if desired. The sample counter will properly span the whole 32-bit range, without gaps, and be equal to the sender's sample count in all but pathological cases.

A small program that checks the conversion and demonstrates the steps is shown in the listing below. One pathological example where the conversion is off by one sample is $c = 1.00000000046566129$, sample 3221225954.

```
/*
* Timestamp conversion:
* sample clock -> NTP 32/16 -> NTP 16/16 (integer, RTP timestamp)
*  -> NTP 32/16 -> samples (32 bits)
*/
#include <math.h>
#include <limits.h>
#include <sys/types.h>

typedef double CLOCK_t;
#define MAX_32bit 4294967296.
#define MAX_31bit 2147483648

/*
* Return least-significant 32 bits of clock.
*/
u_long clock_32(t)
CLOCK_t t;
{
  return (unsigned long)fmod(t, MAX_32bit);
} /* clock_32 */
```

```
/*
* Extend 32-bit timestamp to CLOCK_t within same clock reference.
*/
CLOCK_t clock_extend(ts, now)
u_long ts;    /* in: timestamp, low-order 32 bits */
CLOCK_t now;  /* in: current local time (same frequency as ts) */
{
  u_long high, low;   /* high and low order bits of 48-bit clock */

  low  = clock_32(now);
  high = now / MAX_32bit;

  if (low > ts) {
    if (low - ts > MAX_31bit) high++;
  }
  else {
    if (ts - low > MAX_31bit) high--;
  }
  return high * MAX_32bit + ts;
} /* clock_extend */


main(int argc, char *argv[])
{
  double c = atof(argv[1]);
  CLOCK_t tx_ntp32_16, tx_ntp16_16;
  CLOCK_t rx_ntp32_16, rx_ntp16_16;
  u_long tx_sample, rx_sample;
  unsigned long rtp;

  printf("c=%g\n", c);
  for (tx_sample = UINT_MAX*0.7499; tx_sample < UINT_MAX; tx_sample++) {
    /* sample clock -> NTP 32/16 */
    tx_ntp32_16 = tx_sample * c;

    /* -> NTP 16/16 */
    tx_ntp16_16 = fmod(tx_ntp32_16, MAX_32bit);

    /* RTP timestamp */
    rtp = tx_ntp16_16 + 0.5;

    /* RTP -> NTP 32/16; with possible clock offset */
    rx_ntp32_16 = clock_extend(rtp, tx_ntp32_16 + 1e7);
```

```
    if (fabs(tx_ntp32_16 - rx_ntp32_16) > 0.5) {
      printf("32.16 tx %15.2f rx %15.2f\n", tx_ntp32_16, rx_ntp32_16);
    }

    /* NTP 32/16 -> sample */
    rx_sample = floor(rx_ntp32_16 / c + 0.5);
    if (tx_sample != rx_sample) {
      printf("tx_sample %12u rx_sample %12u\n", tx_sample, rx_sample);
    }
    if (tx_sample % 100000 == 0) printf("%u\n", tx_sample);
  }
}
```

B Glossary

The glossary below briefly defines the acronyms used within the text. Further definitions can be found in RFC 1392, "Internet User's Glossary". Some of the general Internet definitions below are copied from that glossary. The quoted passages followed by a reference of the form "(G.701)" are drawn from the CCITT Blue Book, Fascicle I.3, Definitions. The glossary of the document "Recommended Practices for Enhancing Digital Audio Compatibility in Multimedia Systems", published by the Interactive Multimedia Association was used for some terms marked with [IMA]. The section on MPEG is based on text written by Mark Adler (Caltech).

4:1:1 Refers to degree of subsampling of the two chrominance signals with respect to the luminance signal. Here, each color difference component has one quarter the resolution of the luminance component.

4:2:2 Refers to degree of subsampling of the two chrominance signals with respect to the luminance signal. Here, each color difference component has half the resolution of the luminance component.

16/16 timestamp: a 32-bit integer timestamp consisting of a 16-bit field containing the number of seconds followed by a 16-bit field containing the binary fraction of a second. This timestamp can measure about 18.2 hours with a resolution of approximately 15 microseconds.

n/m **timestamp:** a $n+m$ bit timestamp consisting of an n-bit second count and an m-bit fraction.

ADPCM: Adaptive differential pulse code modulation. Rather than transmitting → PCM samples directly, the difference between the estimate of the next sample and the actual sample is transmitted. This difference is usually small and can thus be encoded in fewer bits than the sample itself. The → CCITT recommendations G.721, G.723, G.726 and G.727 describe ADPCM encodings. "A form of differential pulse code modulation that uses adaptive quantizing. The predictor may be either fixed (time invariant) or variable. When the predictor

is adaptive, the adaptation of its coefficients is made from the quantized difference signal." (G.701)

adaptive quantizing: "Quantizing in which some parameters are made variable according to the short term statistical characteristics of the quantized signal." (G.701)

A-law: a type of audio →companding popular in Europe.

CCIR: Comite Consultativ International de Radio. This organization is part of the United Nations International Telecommunications Union (ITU) and is responsible for making technical recommendations about radio, television and frequency assignments. The CCIR has recently changed its name to ITU-TR; we maintain the more familiar name. →CCITT

CCIR-601: The CCIR-601 digital television standard is the base for all the subsampled interchange formats such as SIF, CIF, QCIF, etc. For NTSC (PAL/SECAM), it is 720 (720) pixels by 243 (288) lines by 60 (50) fields per second, where the fields are interlaced when displayed. The chrominance channels horizontally subsampled by a factor of two, yielding 360 (360) pixels by 243 (288) lines by 60 (50) fields a second.

CCITT: Comite Consultatif International de Telegraphique et Telephonique (CCITT). This organization is part of the United Nations International Telecommunications Union (ITU) and is responsible for making technical recommendations about telephone and data communications systems. X.25 is an example of a CCITT recommendation. Every four years CCITT holds plenary sessions where they adopt new recommendations. Recommendations are known by the color of the cover of the book they are contained in. (The 1988 edition is known as the Blue Book.) The CCITT has recently changed its name to ITU-TS; we maintain the familiar name. →CCIR

CELP: code-excited linear prediction; audio encoding method for low-bit rate codecs; →LPC.

CD: compact disc.

chrominance: color information in a video image. For →H.261, color is encoded as two color differences: CR ("red") and CB ("blue"). →luminance

CIF: common interchange format; interchange format for video images with 288 lines with 352 pixels per line of luminance and 144 lines with 176 pixel per line of chrominance information. →QCIF, SCIF

CLNP: ISO connectionless network-layer protocol (ISO 8473), similar in functionality to →IP.

codec: short for coder/decoder; device or software that → encodes and decodes audio or video information.

companding: contraction of compressing and expanding; reducing the dynamic range of audio or video by a non-linear transformation of the sample values. The best known methods for audio are mu-law, used in North America, and A-law, used in Europe and Asia. →G.711 For a given number of bits, companded data uses a greater number of binary codes to represent small signal levels than linear data, resulting in a greater dynamic range at the expense of a poorer signal-to-nose ratio. [29]

DAT: digital audio tape.

decimation: reduction of sample rate by removal of samples [IMA].

delay jitter: Delay jitter is the variation in end-to-end network delay, caused principally by varying media access delays, e.g., in an Ethernet, and queueing delays. Delay jitter needs to be compensated by adding a variable delay (refered to as → playout delay) at the receiver.

DVI: (trademark) digital video interactive. Audio/video compression technology developed by Intel's DVI group. [IMA]

dynamic range: a ratio of the largest encodable audio signal to the smallest encodable signal, expressed in decibels. For linear audio data types, the dynamic range is approximately six times the number of bits, measured in dB.

encoding: transformation of the media content for transmission, usually to save bandwidth, but also to decrease the effect of transmission errors. Well-known encodings are G.711 (mu-law PCM), and ADPCM for audio, JPEG and MPEG for video. → encryption

encryption: transformation of the media content to ensure that only the intended recipients can make use of the information. → encoding

end system: host where conference participants are located. RTP packets received by an end system are played out, but not forwarded to other hosts (in a manner visible to RTP).

FIR: finite (duration) impulse response. A signal processing filter that does not use any feedback components [IMA].

frame: unit of information. Commonly used for video to refer to a single picture. For audio, it refers to a data that forms a encoding unit. For example, an LPC frame consists of the coefficients necessary to generate a specific number of audio samples.

frequency response: a system's ability to encode the spectral content of audio data. The sample rate has to be at least twice as large as the maximum possible signal frequency.

G.711: → CCITT recommendation for → PCM audio encoding at 64 kb/s using mu-law or A-law companding.

G.721: → CCITT recommendation for 32 kbit/s adaptive differential pulse code modulation (→ ADPCM, PCM).

G.722: → CCITT recommendation for audio coding at 64 kbit/s; the audio bandwidth is 7 kHz instead of 3.5 kHz for G.711, G.721, G.723 and G.728.

G.723: → CCITT recommendation for extensions of Recommendation G.721 adapted to 24 and 40 kbit/s for digital circuit multiplication equipment.

G.728: → CCITT recommendation for voice coding using code-excited linear prediction (CELP) at 16 kbit/s.

G.764: → CCITT recommendation for packet voice; specifies both → HDLC-like data link and network layer. In the draft stage, this standard was referred to as G.PVNP. The standard is primarily geared towards digital circuit multiplication equipment used by telephone companies to carry more voice calls on transoceanic links.

G.821: → CCITT recommendation for the error performance of an international digital connection forming part of an integrated services digital network.

G.822: → CCITT recommendation for the controlled →slip rate objective on an international digital connection.

G.PVNP: designation of CCITT recommendation → G.764 while in draft status.

GOB: (H.261) groups of blocks; a →CIF picture is divided into 12 GOBs, a QCIF into 3 GOBs. A GOB is composed of 3 macro blocks (→MB) and contains luminance and chrominance information for 8448 pixels.

GSM: Group Speciale Mobile. In general, designation for European mobile telephony standard. In particular, often used to denote the audio coding used. Formally known as the European GSM 06.10 provisional standard for full-rate speech transcoding, prI-ETS 300 036. It uses RPE/LTP (residual pulse excitation/long term prediction) at 13 kb/s using frames of 160 samples covering 20 ms.

H.261: → CCITT recommendation for the compression of motion video at rates of $P \times 64$ kb/s (where $p = 1 \ldots 30$. Originally intended for narrowband →ISDN.

hangover: [30] Audio data transmitted after the silence detector indicates that no audio data is present. Hangover ensures that the ends of words, important for comprehension, are transmitted even though they are often of low energy.

HDLC: high-level data link control; standard data link layer protocol (closely related to LAPD and SDLC).

IMA: Interactive Multimedia Assocation; trade association located in Annapolis, MD.

ICMP: Internet Control Message Protocol; ICMP is an extension to the Internet Protocol. It allows for the generation of error messages, test packets and informational messages related to → IP.

in-band: signaling information is carried together (in the same channel or packet) with the actual data. → out-of-band.

interpolation: increase in sample rate by introduction of processed samples.

IP: internet protocol; the Internet Protocol, defined in RFC 791, is the network layer for the TCP/IP Protocol Suite. It is a connectionless, best-effort packet switching protocol [31].

IP address: four-byte binary host interface identifier used by →IP for addressing. An IP address consists of a network portion and a host portion. RTP treats IP addresses as globally unique, opaque identifiers.

IPv4: current version (4) of → IP.

ISDN: integrated services digital network; refers to an end-to-end circuit switched digital network intended to replace the current telephone network. ISDN offers circuit-switched bandwidth in multiples of 64 kb/s (B or bearer channel), plus a 16 kb/s packet-switched data (D) channel.

ISO: International Standards Organization. A voluntary, nontreaty organization founded in 1946. Its members are the national standardards organizations of the 89 member countries, including ANSI for the U.S. (Tanenbaum)

ISO 10646: →ISO standard for the encoding of characters from all languages into a single 32-bit code space (Universal Character Set). For transmission and storage, a one-to-five octet code (UTF) has been defined which is upwardly compatible with US-ASCII.

JPEG: ISO/CCITT joint photographic experts group. Designation of a variable-rate compression algorithm using discrete cosine transforms for still-frame color images.

jitter: → delay jitter.

linear encoding: a mapping from signal values to binary codes where each binary level represents the same signal increment →companding.

loosely controlled conference: Participants can join and leave the conference without connection establishment or notifying a conference moderator. The identity of conference participants may or may not be known to other participants. See also: tightly controlled conference.

low-pass filter: a signal processing function that removes spectral content above a cutoff frequency. [IMA]

LPC: linear predictive coder. Audio encoding method that models speech as a parameters of a linear filter; used for very low bit rate codecs.

luminance: brightness information in a video image. For black-and-white (grayscale) images, only luminance information is required. →chrominance

MB: (H.261) macroblock, consisting of six blocks, four eight-by-eight luminance blocks and two chrominance blocks.

MPEG: ISO/CCITT motion picture experts group JTC1/SC29/WG11. Designates a variable-rate compression algorithm for full motion video at low bit rates; uses both intraframe and interframe coding. It defines a bit stream for compressed video and audio optimized to fit into a bandwidth (data rate) of 1.5 Mbits/s. This rate is special because it is the data rate of (uncompressed) audio CD's and DAT's. The draft is in three parts, video, audio, and systems, where the last part gives the integration of the audio and video streams with the proper timestamping to allow synchronization of the two. MPEG phase II is to define a bitstream for video and audio coded at around 3 to 10 Mbits/s.

MPEG compresses YUV SIF images. Motion is predicted from frame to frame, while DCTs of the difference signal with quantization make use of spatial redundancy. DCTs are performed on 8 by 8 blocks, the motion prediction on 16 by 16 blocks of the luminance signal. Quantization changes for every 16 by 16 macroblock.

There are three types of coded frames. Intra ("I") frames are coded without motion prediction, Predicted ("P") frames are difference frames to the last P or I frame. Each macroblock in a P frame can either come with a vector and difference DCT coefficients for a close match in the last I or P frame, or it can just be intra coded (like in the I frames) if there was no good match. Lastly, there are "B" or bidirectional frames. They are predicted from the closest two I or P frames, one in the past and one in the future. These are searched for matching blocks in those frames, and three different things tried to see which works best: the forward vector, the backward vector, and the average of the two blocks from the future and past frames, and subtracting that from the block being coded. If none of those work well, the block is intra-coded.

There are 12 frames from I to I, based on random access requirements.

MPEG-1: Informal name of proposed →MPEG (ISO standard DIS 1172).

media source: entity (user and host) that produced the media content. It is the entity that is shown as the active participant by the application.

MTU: maximum transmission unit; the largest frame length which may be sent on a physical medium.

Nevot: network voice terminal; application written by the author.

network source: entity denoted by address and port number from which the → end system receives the RTP packet and to which the end system send any RTP packets for that conference in return.

NTP timestamp: "NTP timestamps are represented as a 64-bit unsigned fixed-point number, in seconds relative to 0 hours on 1 January 1900. The integer part is in the first 32 bits and the fraction part in the last 32 bits." [32] NTP timestamps do not include leap seconds, i.e., each and every day contains exactly 86,400 NTP seconds.

NVP: network voice protocol; original packet format used in early packet voice experiments; defined in [2].

octet: An octet is an 8-bit datum, which may contain values 0 through 255 decimal. Commonly used in ISO and CCITT documents, also known as a byte.

OSI: Open System Interconnection; a suite of protocols, designed by ISO committees, to be the international standard computer network architecture.

out of band: signaling and control information is carried in a separate channel or separate packets from the actual data. For example, ICMP carries control information out-of-band, that is, as separate packets, for IP, but both ICMP and IP usually use the same communication channel (in band).

parametric coder: coder that encodes parameters of a model representing the input signal. For example, LPC models a voice source as segments of voice and unvoiced speech, represented by a set of

parametric coder: coder that encodes parameters of a model representing the input signal. For example, LPC models a voice source as segments of voice and unvoiced speech, represented by filter parameters. Examples include LPC, CELP and GSM. →waveform coder.

PCM: pulse-code modulation; speech coding where speech is represented by a given number of fixed-width samples per second. Often used for the coding employed in the telephone network: 64,000 eight-bit samples per second.

pel, pixel: picture element. "Smallest graphic element that can be independently addressed within a picture; (an alternative term for raster graphics element)." (T.411)

playout: Delivery of the medium content to the final consumer within the receiving host. For audio, this implies digital-to-analog conversion, for video display on a screen.

playout unit: A playout unit is a group of packets sharing a common timestamp. (Naturally, packets whose timestamps are identical due to timestamp wrap-around are not considered part of the same playout unit.) For voice, the playout unit would typically be a single voice segment, while for video a video frame could be broken down into subframes, each consisting of packets sharing the same timestamp and ordered by some form of sequence number. →synchronization unit

plesiochronous: "The essential characteristic of time-scales or signals such that their corresponding significant instants occur at nominally the same rate, any variation in rate being constrained within specified limits. Two signals having the same nominal digit rate, but not stemming from the same clock or homochronous clocks, are usually plesiochronous. There is no limit to the time relationship between corresponding significant instants." (G.701, Q.9) In other words, plesiochronous clocks have (almost) the same rate, but possibly different phase.

pulse code modulation (PCM): "A process in which a signal is sampled, and each sample is quantized independently of other samples and converted by encoding to a digital signal." (G.701)

PVP: packet video protocol; extension of → NVP to video data [33]

QCIF: quarter common interchange format; format for exchanging video images with half as many lines and half as many pixels per line as CIF, i.e., luminance information is coded at 144 lines and 176 pixels per line. →CIF, SIF

RTCP: real-time control protocol; adjunct to → RTP.

RTP: real-time transport protocol; discussed in this memorandum.

sampling rate: "The number of samples taken of a signal per unit time." (G.701)

SB: subband; as in subband codec. Audio or video encoding that splits the frequency content of a signal into several bands and encodes each band separately, with the encoding fidelity matched to human perception for that particular frequency band.

SCIF: standard video interchange format; consists of four →CIF images arranged in a square. →CIF, QCIF

SIF: standard interchange format; format for exchanging video images of 240 lines with 352 pixels each for NTSC, and 288 lines by 352 pixels for PAL and SECAM. At the nominal field rates of 60 and 50 fields/s, the two formats have the same data rate. →CIF, QCIF

slip: In digital communications, slip refers to bit errors caused by the different clock rates of nominally synchronous sender and receiver. If the sender clock is faster than the receiver clock, occasionally a bit will have to be dropped. Conversely, a faster receiver will need to insert extra bits. The problem also occurs if the clock rates of encoder and decoder are not matched precisely. Information loss can be avoided if the duration of pauses (silence periods between talkspurts or the inter-frame duration) can be adjusted by the receiver. "The repetition or deletion of a block of bits in a synchronous or plesiochronous bit stream due to a discrepancy in the read and write rates at a buffer." (G.810) →G.821, G.822

ST-II: stream protocol; connection-oriented unreliable, non-sequenced packet-oriented network and transport protocol with process demultiplexing and provisions for establishing flow parameters for resource control; defined in RFC 1190 [34,35].

Super CIF: video format defined in Annex IV of →H.261 (1992), comprising 704 by 576 pixels.

synchronization unit: A synchronization unit consists of one or more →playout units that, as a group, share a common fixed delay between generation and playout of each part of the group. The delay may change at the beginning of such a synchronization unit. The most common synchronization units are talkspurts for voice and frames for video transmission.

TCP: transmission control protocol; an Internet Standard transport layer protocol defined in RFC 793. It is connection-oriented and stream-oriented, as opposed to UDP [36].

TPDU: transport protocol data unit.

tightly controlled conference: Participants can join the conference only after an invitation from a conference moderator. The identify of all conference participants is known to the moderator. →loosely controlled conference.

transcoder: device or application that translates between several encodings, for example between → LPC and → PCM.

UDP: user datagram protocol; unreliable, non-sequenced connectionless transport protocol defined in RFC 768 [37].

vat: visual audio tool written by Steve McCanne and Van Jacobson, Lawrence Berkeley Laboratory.

vt: voice terminal software written at the Information Sciences Institute.

VMTP: Versatile message transaction protocol; defined in RFC 1045 [38].

waveform coder: a coder that tries to reproduce the waveform after decompression; examples include PCM and ADPCM for audio and video and discrete-cosine-transform based coders for video; →parametric coder.

Y: Common abbreviation for the luminance or luma signal.

YCbCr: YCbCr coding is employed by D-1 component video equipment.

C Address of Author

Henning Schulzrinne
GMD Fokus
Hardenbergplatz 2
10623 Berlin
Germany
telephone: +49 30 25499 219
facsimile: +49 30 25499 202
electronic mail: hgs@fokus.gmd.de

References

[1] H. Schulzrinne and S. Casner, "A transport protocol for real-time applications." Internet draft (work-in-progress) *draft-ietf-avt-rtp-*.txt*, Sept. 1993.

[2] D. Cohen, "A network voice protocol: NVP-II," technical report, University of Southern California/ISI, Marina del Ray, California, Apr. 1981.

[3] N. Borenstein and N. Freed, "MIME (multipurpose internet mail extensions) mechanisms for specifying and describing the format of internet message bodies," Network Working Group Request for Comments RFC 1341, Bellcore, June 1992.

[4] R. Want, A. Hopper, V. Falcao, and J. Gibbons, "The active badge location system," *ACM Transactions on Information Systems*, vol. 10, pp. 91–102, Jan. 1992. also Olivetti Research Limited Technical Report ORL 92-1.

[5] R. Want and A. Hopper, "Active badges and personal interactive computing objects," Technical Report ORL 92-2, Olivetti Research, Cambridge, England, Feb. 1992. also in *IEEE Transactions on Consumer Electronics*, Feb. 1992.

[6] J. G. Gruber and L. Strawczynski, "Subjective effects of variable delay and speech clipping in dynamically managed voice systems," *IEEE Transactions on Communications*, vol. COM-33, pp. 801–808, Aug. 1985.

[7] N. S. Jayant, "Effects of packet losses in waveform coded speech and improvements due to an odd-even sample-interpolation procedure," *IEEE Transactions on Communications*, vol. COM-29, pp. 101–109, Feb. 1981.

[8] D. Minoli, "Optimal packet length for packet voice communication," *IEEE Transactions on Communications*, vol. COM-27, pp. 607–611, Mar. 1979.

[9] V. Jacobson, "Compressing TCP/IP headers for low-speed serial links," Network Working Group Request for Comments RFC 1144, Lawrence Berkeley Laboratory, Feb. 1990.

[10] P. Francis, "A near-term architecture for deploying Pip," *IEEE Network*, vol. 7, pp. 30–37, May 1993.

[11] IMA Digital Audio Focus and Technical Working Groups, "Recommended practices for enhancing digital audio compatibility in multimedia systems," tech. rep., Interactive Multimedia Association, Annapolis, Maryland, Oct. 1992.

[12] C. J. Sreenan, "Synchronization services for continuous media: an experimental evaluation," technical memorandum, AT&T Bell Laboratories, Murray Hill, NJ, Sept. 1993.

[13] W. A. Montgomery, "Techniques for packet voice synchronization," *IEEE Journal on Selected Areas in Communications*, vol. SAC-1, pp. 1022–1028, Dec. 1983.

[14] D. Cohen, "A protocol for packet-switching voice communication," *Computer Networks*, vol. 2, pp. 320–331, September/October 1978.

[15] International Electrotechnical Commission, "Iec 461: Time and control code for video tape recorders," 1986.

[16] ISO/IEC JTC 1, *ISO/IEC DIS 11172: Information technology — coding of moving pictures and associated audio for digital storage media up to about 1.5 Mbit/s*. International Organization for Standardization and International Electrotechnical Commission, 1992.

[17] Apple Computer, *Inside Macintosh: QuickTime*. Apple Computer, Cupertino, California, 1.5 ed., Oct. 1992.

[18] J. Escobar, D. Deutsch, and C. Partridge, "Flow synchronization protocol," in *Proceedings of the Conference on Global Communications (GLOBECOM)*, (Orlando, Florida), pp. 1381–1387 (40.04), IEEE, Dec. 1992.

[19] L. Delgrossi, C. Halstrick, R. G. Herrtwich, and H. Stüttgen, "HeiTP: a transport protocol for ST-II," in *Proceedings of the Conference on Global Communications (GLOBECOM)*, (Orlando, Florida), pp. 1369–1373 (40.02), IEEE, Dec. 1992.

[20] G. J. Holzmann, *Design and Validation of Computer Protocols*. Englewood Cliffs, New Jersey: Prentice Hall, 1991.

[21] A. Nakassis, "Fletcher's error detection algorithm: how to implement it efficiently and how to avoid the most common pitfalls," *ACM Computer Communication Review*, vol. 18, pp. 63–88, Oct. 1988.

[22] J. G. Fletcher, "An arithmetic checksum for serial transmission," *IEEE Transactions on Communications*, vol. COM-30, pp. 247–252, Jan. 1982.

[23] J. Linn, "Privacy enhancement for Internet electronic mail: Part III — algorithms, modes and identifiers," Network Working Group Request for Comments RFC 1115, IETF, Aug. 1989.

[24] D. Balenson, "Privacy enhancement for internet electronic mail: Part III: Algorithms, modes, and identifiers," Network Working Group Request for Comments RFC 1423, IETF, Feb. 1993.

[25] S. Kent, "Privacy enhancement for internet electronic mail: Part II: Certificate-based key management," Network Working Group Request for Comments RFC 1422, IETF, Feb. 1993.

[26] J. Linn, "Privacy enhancement for Internet electronic mail: Part I — message encipherment and authentication procedures," Network Working Group Request for Comments RFC 1113, IETF, Aug. 1989.

[27] R. Rivest, "The MD5 message-digest algorithm," Network Working Group Request for Comments RFC 1321, IETF, Apr. 1992.

[28] North American Directory Forum, "A naming scheme for c=US," Network Working Group Request for Comments RFC 1255, North American Directory Forum, Sept. 1991.

[29] N. S. Jayant and P. Noll, *Digital Coding of Waveforms*. Englewood Cliffs, New Jersey: Prentice Hall, 1984.

[30] P. T. Brady, "A model for generating on-off speech patterns in two-way conversation," *Bell System Technical Journal*, vol. 48, pp. 2445–2472, Sept. 1969.

[31] J. Postel, "Internet protocol," Network Working Group Request for Comments RFC 791, Information Sciences Institute, Sept. 1981.

[32] D. L. Mills, "Network time protocol (version 3) – specification, implementation and analysis," Network Working Group Request for Comments RFC 1305, University of Delaware, Mar. 1992.

[33] R. Cole, "PVP - a packet video protocol," W-Note 28, Information Sciences Institute, University of Southern California, Los Angeles, California, Aug. 1981.

[34] C. Topolcic, S. Casner, C. Lynn, Jr., P. Park, and K. Schroder, "Experimental internet stream protocol, version 2 (ST-II)," Network Working Group Request for Comments RFC 1190, BBN Systems and Technologies, Oct. 1990.

[35] C. Topolcic, "ST II," in *First International Workshop on Network and Operating System Support for Digital Audio and Video*, no. TR-90-062 in ICSI Technical Reports, (Berkeley, California), 1990.

[36] J. B. Postel, "DoD standard transmission control protocol," Network Working Group Request for Comments RFC 761, Information Sciences Institute, Jan. 1980.

[37] J. B. Postel, "User datagram protocol," Network Working Group Request for Comments RFC 768, ISI, Aug. 1980.

[38] D. R. Cheriton, "VMTP: Versatile Message Transaction Protocol specification," Request for Comments RFC 1045, SRI International, Menlo Park, California, Feb. 1988.