Internet Engineering Task Force
INTERNET-DRAFT

Audio-Video Transport Working Group
J. Rosenberg
Lucent Technologies, Bell Laboratories
H. Schulzrinne
Columbia University

November 26, 1996
Expires: May 26, 1997

Issues and Options for an Aggregation Service within RTP

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 and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

To learn the current status of any Internet-Draft, please check the "1id-abstracts.txt" listing contained in the Internet-Drafts Shadow Directories on ftp.is.co.za (Africa), nic.nordu.net (Europe), munnari.oz.au (Pacific Rim), ds.internic.net (US East Coast), or ftp.isi.edu (US West Coast).

Distribution of this document is unlimited.

Abstract

This memorandum discusses the issues and options involved in the design of a new transport protocol for multiplexed voice within a single packet. The intended application is the interconnection of devices which provide "trunking" or long distance telephone service over the Internet. Such devices have many voice connections simultaneously between them. Multiplexing them into the same connection improves on the efficiency, enables the use of low bitrate voice codecs, and improves scalability. Options and issues concerning timestamping, payload type identification, length indication, and channel identification are discussed. Several possible header formats are identified, and their efficiencies are compared.

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 to the author(s).

1. Introduction

With the tremendous changes in the telecommunications industry, and the recent growth of the Internet, there is a new opportunity for offering long distance telephony over the Internet. Such a service can be offered by allowing users to dial a local access number, connecting them to a device called an Internet Telephony Gateway (ITG). This device prompts the user for a destination telephone number, and then routes the call over the Internet to a similar device at the local exchange of the destination. There, the call is completed when the destination ITG dials the end user. The scenario is depicted in Figure 1.

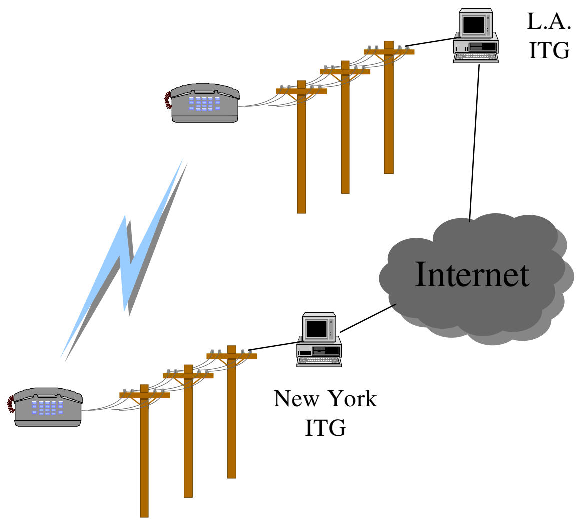


Figure 1: Internet Telephony Gateway

In this application, the Internet is used only for the long distance portion of the telephone call. Access to the service is still via the traditional POTS. Current implementations of this service are using H.323 to set up and tear down a new connection each time a user establishes or terminates a call. However, H.323 is the wrong protocol for many reasons. First, it is far too complex, providing for capabilities and features which cannot be used because both endpoints are analog telephones. Secondly, a significant increase in efficiency (in excess of 30%), can be readily achieved if all of the voice calls between two ITG are multiplexed into a single packet, instead of using a separate connection (and thus separate packets) for each. Such multiplexing reduces overhead by increasing the effective payload *without* a corresponding penalty in packetization delay. In fact, as more users are multiplexed, the payload from a particular user can be reduced in size, or the bitrate reduced, without an efficiency penalty. Furthermore, multiplexing improves scalability. As the number of users increases, the number of packets which arrive at the destination does not increase. This means that computations which are per-packet (such as RTCP statistics collecting, jitter accumulation, header processing, etc.) do not increase. The end result is that multiplexing can simultaneously improve efficiency, reduce delay, and improve scalability. There are some minor side benefits in addition to these major three. For example, in the aggregated scenario, when a particular user enters a silence period, and stops sending data, the flow of packets will not stop unless all of the other users are already in silence (generally, an unlikely event). This means that packets continually arrive, and that delay estimates obtained from those packets can be continuously generated. Algorithms for dynamically adapting the playout buffer at the receiver are based on these delay estimates [1], and can now be reworked to utilize the continuous stream of delays, as opposed to relying on the delays received during talkspurts only. The result is likely to be an improvement in both end to end delay and loss performance.

In order to perform such multiplexing, a new Internet protocol is required. This protocol must provide for the transport of multiple real time streams within a single IP packet. Since the intended application is real-time, the requirements for timing recovery, sequencing, and payload identification are nearly identical to normal single user voice. Since RTP was designed to meet these requirements [2], it makes sense to build this new multiplexing protocol on top of RTP. In fact, RTP allows for different profiles to be defined for a particular application. The goal of this document is to define a variety of options for that new profile, and to compare them.

It is important to note that this application is similar in its requirements to [3], which seeks to multiplex multiple encodings for a particular user into the same IP packet.

2. Terminology

User: One of the individuals who has data within the IP packet.
Connection: The point to point RTP session between two ITG's.
Channel: A "virtual connection" which is established by allowing a user to send data within a packet. There are many channels per connection - this represents the multiplexing.
Channel Identifier: A number which identifies a channel.
Block: The section of the payload of a packet which contains data for a particular user.

3. Requirements:

The transport protocol must provide, at a minimum, the following functionality:

1. Delineation. Data from different users must be clearly delineated.
2. Identification. The channel to which the data belongs must be identified.
3. Variable lengths: The protocol should support variable length blocks from a particular user. This allows for variable rate codecs.
4. Low overhead: Since the protocol is designed for low rate voice, it should have low overhead. This issue is extremely important. New coders are emerging which can support near toll quality at 8 kbps, and acceptable quality at rates even as low as 4 kbps. It is desirable to support such codecs, as they can reduce the cost of providing an ITG service. Furthermore, advances in coding technology indicate that it is desirable to send very low bitrate information (1 kbps or less) during silence periods, so that background noise can be reproduced well (as opposed to sending nothing). Support of such rates requires a protocol with low overhead.
5. Marker: A general purpose marker bit should be available for all users within the connection.
6. Payload Identification. The codec in use for each user should be indicated somehow. It is a requirement to allow for the coding type to change during the lifetime of a channel.

4. Issues

The following section identifies a number of issues which have an impact on the design of the protocol. It also identifies a variety of options for providing the specific services of the protocol.

4.1 How to bind telephone numbers to channel identifiers

There are four options for this problem. First, the telephone number can be included in the per-user header. Second, the telephone number can be signaled reliably by a companion TCP connection before data begins. Thirdly, the phone number can be sent periodically in RTCP in a soft-state fashion. Fourthly, the information can be sent periodically over a reliable TCP based control channel. The first approach avoids any synchronization problems, but has high overhead. The second approach is a more traditional approach, but relies on hard state at the destination ITG. The third approach allows for a refresh of state, but causes longer setup delays in the face of packet loss. The fourth approach guarantees reliable delivery of signaling information, but also generates refreshes to allow for recovery from end-system failures.

The most reasonable approach seems to be the second - the use of TCP (or any other reliable protocol) for sending signaling information. This approach guarantees that the critical information is received correctly, and in a timely manner. It avoids bandwidth inefficient refresh as well.

4.2 Payload type identification

There are a number of ways to identify the coding of the payload. The first is through static types, identified by bits in the header (like RTP is now). The second approach dynamically adjusts the coding type based on external messages which bind a coding type to a channel identifier. Such external messages can be either UDP or TCP based. A related issue is synchronization of these changes. Either the timestamps or sequence numbers can be used. One approach to performing the synchronization is as follows: The source sends a message reliably to the receiver, indicating that it will change codings at timestamp N, where N is some future timestamp (or SN). The N should be chosen far enough into the future to guarantee that the receiver will get the TCP message before time N. The farther away N is, the more robust the system becomes, but the source also loses its ability to adapt quickly. There are also several options for simple in-band signaling methods which can assist in error recovery. This is based on the assumption that it is better for the receiver to know that the encoding has changed (even though it doesn't know to what), than to know nothing. This avoids playing garbage out. A one or two bit "coding sequence number" can be used in the header. Such a number starts at zero. At the timestamp where the encoding changes, the SN increments, and stays incremented until the next change. In this fashion, we are guaranteed that the source will never play out data using the wrong coding type. Probably just one or two bits of this SN is necessary.

Yet another approach to changing payload types is via "pseudo-dynamic" payloads. Before transmission of data commences, a reliable exchange occurs which downloads a table of possible encodings of the payload type, based on the capabilities of the source. The table then remains active for the lifetime of the connection. This technique can reduce the number of bits required for the payload type, since a particular gateway is likely to support just a few codecs. However, it is still a hard state approach, but it would only fail in the face of end system failure, not network failure.

Our conclusion is that it is desirable to have the PTI field in the payload. This makes it possible to do more robust rate control, which becomes a significant issue when multiple connections are multiplexed together (and therefore the aggregate bitrate increases). It also makes sense to signal a table of encodings for the payload type at the beginning of the connection. Any particular pair of ITG will generally only support a few codecs. Therefore, dynamically setting the codings of the PTI bit makes a more compact representation possible without restricting the set of codecs which may be used.

4.3 Timestamps

Timing is a very complex issue for the multiplexing protocol. The first question related to it is whether the protocol will support mixing of media derived from separate clocks (i.e., voice and video). Although doing this seems attractive, it is complex and in opposition to the philosophy under which RTP was developed. RTP explicitly states that separate media should be placed in separate RTP streams. This allows for different QoS to be requested for each media, and for clocks to be defined based on the media type. Furthermore, this profile is geared towards the aggregation of voice traffic generated from the POTS across the Internet. As a result, the only source of data is from a single, 125us clock.

The next basic question is whether timestamps are needed "globally", i.e., just one per packet independent of the number of users, or "locally", whereby each user within a packet needs their own timestamp. A separate question is the representation of these timestamps in an efficient manner. When considering these questions, the criteria to keep in mind are:

1. Can silence periods be recovered correctly
2. Can resynchronization occur in the face of packet loss
3. What is the impact on playout buffering and jitter computation

The answer to this question depends on the desired capabilities of the protocol. In the most general case, it is possible to have different frame sizes for each user (for example, 20ms, 10ms, and 15ms) within the same packet. These frames can be arbitrarily aligned in time with respect to each other (i.e., the 20ms frame starts 5.3 ms after the beginning of another user's 10 ms frame). The user can send packets off at any point, containing data from those users whose frames have been generated before the packet departure time. A somewhat more restrictive capability is to allow for different frame sizes and time alignments, but to require that any packet contains all the same frame sizes, all aligned in time. The most restrictive case is to require separate RTP sessions for users with different frame sizes. This requires a channel to be torn down and re-setup when it changes codec. The desire to perform flow control on a channel-by-channel basis makes this approach unacceptable, and it is not considered further.

4.3.1 General Case

First consider the general case. Packets can contain frames from some or all of the users, and those frames are not the same length nor time aligned in any way. An example of such a scenario is depicted in Figure 2. In the figure, there are three sources, and the ti correspond to the times of packet emissions. When packets are lost, the variability in the amount and time alignment of data in each packet makes it impossible to reconstruct how much time had elapsed based solely on sequence numbers (such reconstruction IS possible in the single user case). Furthermore, the amount of time elapsed can easily vary from user to user, and therefore local timestamps are needed.

The general case introduces further complications which have to do with jitter and delay computation. Such computations are needed for RTCP reporting and possibly for the estimation of network delays, used in dynamic playout buffers. In the single user case, the jitter is computed between each packet as:

$$D(i,j) = (Rj - Ri) - (Sj - Si)$$

Where the Ri correspond to the reception times at the receiver measured in RTP time, and the Si are the RTP timestamps in the data packets. The delay is computed as the difference between the arrival time at the receiver and generation time, as indicated by the RTP timestamp.

In the multiple user case, these definitions no longer make sense, as there is no single RTP timestamp any longer. Each arriving packet will have a single arriving time (Ri), but multiple sending times (Si,j) for each block j in the i^{th} packet. There are a number of alternatives for delay and jitter computation in this case: compute such information for all users, compute such information for a single user, or generate a single delay and jitter estimate, but have it be based on information from all users. There are pros and cons to each approach.

First of all, it is possible for different blocks to experience different delays (and jitters) even though they are within the same packet. This is because the general scenario allows for significant variability, whereby blocks may either vary in size from packet to packet and within a packet, or not be transmitted immediately after their completion (the latter happens to source B in Figure 2). Thus, it is arguable they it may be desirable to perform adaptive playout buffering separately for each user, which would require the storage and computation of delays for each user.

The second alternative is to compute the delays for a single user, and use that information to size all of the other playout buffers. This may be sub-optimal in terms of delay and loss, depending on what fraction of the total delay and jitter are introduced by the packetization itself. There is a second disadvantage to this approach, however. When that particular user enters a silence period, delay and jitter information is no longer being received, and so estimates of network delay stop adapting. This implies that delay estimates will be old for certain periods of time. An alternative is to change the user from which delay and jitter estimates are being collected.

The third alternative is to compute delay estimates based on some measure derived from all of the users. There are several reasonable approaches. For example, the delay estimate can be computed as:

$$Delay = \max\{j, Ri - Si,j\}$$

which would yield a conservative estimate of the delay for some users. This approach requires storage of only a single set of delay information, although computation still grows with the number of users in a packet.

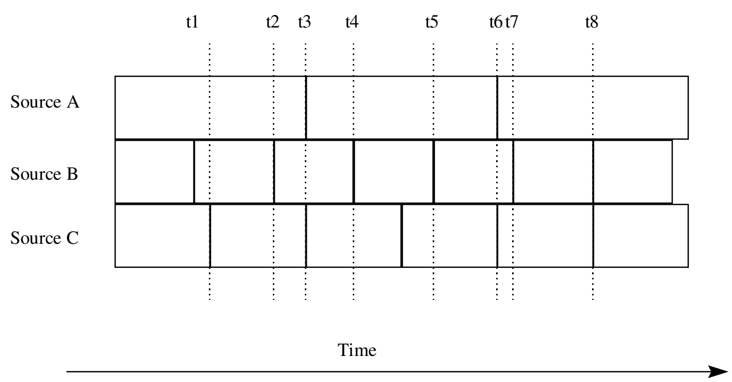


Figure 2: Global Timestamp Problem

Sending local timestamps also requires extra bits in the block headers. It is possible, however, to use *offsets* for the local timestamps. A global timestamp can be used in the RTP header (the field already exists), and each user has a modifier to indicate position in time relative to that timestamp.

A related question is how big to make the offset field. This offset is bounded by the difference in time between the earliest and latest samples within a packet. Clearly, this itself is bounded by the packetization delay at the source. For this application, if we assume a 125us sample clock, and bound packetization delays to 100ms, the offset field is bounded by 800 ticks, requiring 10 bits.

4.3.2 More Restrictive Case

As a more restrictive case, we allow blocks to be present in a packet if their frame sizes are identical and aligned in time. Note that this does not imply identical codecs or identical block sizes in terms of bytes; many voice codecs operate with a 20ms or 50ms frame size. This case would allow all frame sizes of the same size and time alignment, independent of the codec, into a packet.

This simplifies the timing issue tremendously. Now, the scenario is much more like the single user application. The sequence numbers and the frame size completely determine the timing when at least one user is active. But, when all users enter silence, a global timestamp is needed to indicate the duration of the silence period. The global timestamp is sufficient to reconstruct the timing in the face of losses. Therefore, in this case, only a global timestamp is required.

It is desirable to support a variety of different frame sizes within such an aggregated connection, however. The way to do this in this case is to simply mandate that different packets can contain different frame sizes; the only restriction is within a packet. This is not as simple as it may seem at first. Once this is done, the relationship between sequence numbers and timing is lost. Consider an example. There are two frame sizes, 10ms and 30ms. Packet N contains 10ms frames, as does packet N+1 and N+2, however, N+3 contains 30ms frames. Thus, although the difference in sequence number between the first and fourth is three, the relative timing is not 10ms*3 or 30ms*3. Due to this fact, the measurement of jitter is complicated (for the same reasons described in Section 4.3.1), as it should not be done between two packets with different frame sizes. It also makes recovery techniques based on sequence number more complex. To resolve this problem, we use a natural concept in RTP, which is the synchronization source (SSRC). The approach is to have a separate SSRC for each frame size in use. Then, sequence numbers are interpreted for each SSRC separately. This resolves the problem with the relationship between timing and sequence numbering. It also makes jitter and delay computations simpler - they are now done for each SSRC separately. Furthermore, multiple jitter (and delay, loss, etc.) values are reported to the source, one for each frame size. This is also desirable, since the different frame sizes will cause different packetization delays and packet sizes, which may cause those packets to see different delays and losses in the network than other packets.

This case has both advantages and drawbacks when compared to the general case. As an advantage, timing is greatly simplified, and the approach falls much in line with the original intentions of RTP. However, it causes losses in efficiency for systems with a variety of different frame sizes in operation simultaneously. Such a situation arises naturally when flow control is applied to each source individually, as opposed to altering the rate and codec type for all of the active sources.

4.4 Channel ID

The question of channel identification may seem at first trivial - simply use a 32 bit number, much like the SSRC, and be done with it. However, 32 bits adds significant overhead. Reduction of the number of bits for the channel ID becomes a complex issue. Unlike the single user case, the connection may remain active for long periods of time (days or months). The result is that channel ID’s will need to be reused during the lifetime of the connection. It is critical to ensure that data from different channels is not confused because of this. Large channel ID spacing helps to resolve this issue (although it can not eliminate it), so an added side effect of reducing the number of channel ID’s possible is an increase in the likelihood of such confusion.

The first question to be addressed is how many simultaneous users can one expect to find in a single packet.

4.4.1 Number of Users

There are several ways to come up with some minimums and maximums.

Delay-bound

Clearly, as we add more users, the store and forward delays increase since the packet size gets larger. Therefore, if we bound the per-hop delay, and provide a lower bound on the codec bitrate and packetization delay, an upper bound on the number of users can be obtained. Consider a 2.4 kbps codec, with a 20ms frame size. This is a reasonable minimum combination. Next, consider 50ms store and forward delays. For a T1, this limits the number of users within a packet to 965. For a T3, it is 30 times this, or nearly 29,000. If silence suppression is used, the number of users within a packet is roughly half the number of active users (on average), thus requiring twice as many channel identifiers (1930 and 58,000). This bound doesn't seem to tight. Intuitively, even 965 seems too large.

Efficiency bound

The entire purpose of multiplexing is to improve upon efficiency. Therefore, we should be able to support at least as many users as is necessary to get good efficiency. Consider the typical case, a 16 kbps codec, with a 20ms packetization delay. This results in 320 bits of data per user. If we assume IP/UDP/RTP (20+8+12=40 bytes = 320 bits), plus an additional word (32 bits) of overhead per user, the efficiency vs. N becomes:

$$E = (320N / ((320 + 32)N + 320))$$

This reaches an asymptote of 90%. It is desirable to be within a few percent of this, say 88%. Solving for N, this requires 7 users in a packet, so that we must support at least 14 active channels (again, due to stat mux). The lower bound, therefore, on the number of users is around 14.

MTU Bound

In many cases, there is a maximum packet size. This is usually around 1500 bytes. If we consider a very low bitrate codec, the minimum block size from any particular user is 32 bits (otherwise, overheads become very large, and we lose word alignment, so 32 bits is a good minimum). Dividing 1500 bytes by 4 bytes, we obtain a maximum of 375 users. Multiplying by two, the number of active channels needed is around 750.

Based on these bounds, we need to simultaneously support at least 10 users, and at most 750. This would imply that at least 8 to 10 bits of channel ID are required.

4.4.2 Channel ID Reuse Problem

It is important to guarantee that data from a particular channel is never routed to a different channel; this would mean that a user may hear pieces of conversations from different users, an error we consider catastrophic. Such misrouting becomes possible when a channel is torn down, and a new channel is set up soon after using the same channel ID. Such a scenario is depicted in Figure 3. Sometime after channel K is torn down, a new channel is set up using the same channel ID, K. If the data packets (dotted lines) are being delayed significantly, blocks from the old channel K may still be present in the data stream after the new channel K is established. These blocks will then be played out to the new user of channel K. Protocol support is needed to guarantee that this can never happen.

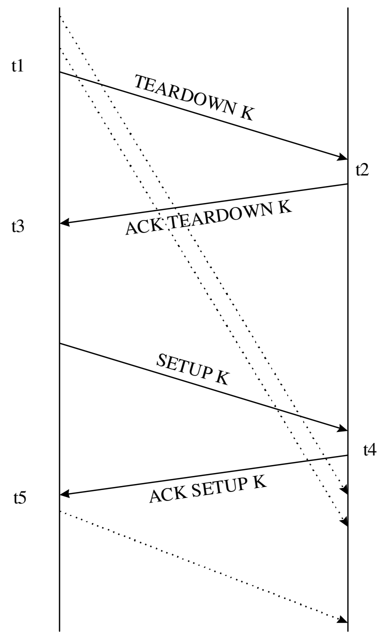


Figure 3: Channel ID Reuse Problem

The solution lies in an intelligent signaling protocol. The protocol must support a two-way handshake for all control messages. In addition, three simple rules must be obeyed at a source when setting up or tearing down connections:

1. When a source sends a teardown message, it stops sending data in the UDP stream for that channel. Furthermore, in the signaling message, it indicates the sequence number of the packet which contained the last block for that channel, call this sequence number K.
2. A source cannot re-use a channel identifier until it has received an acknowledge from the destination that that particular channel was successfully torn down.
3. A source cannot send begin to send data from a particular channel in the UDP stream until it has received an acknowledge from the destination that the setup is complete.

A few simple rules must also be used at the receiver:

1. When a receiver gets a teardown message, it checks the highest SN received so far (call this sequence number M). If M > K, the channel is torn down, and any further blocks containing that channel ID are discarded. If M < K, blocks from that channel are accepted until the received SN exceeds K. Once this happens, the channel is torn down and no further blocks with that channel ID are accepted.
2. When a setup message is received, the destination will begin to accept blocks with the given channel identifier, but only if the sequence numbers of the packets in which they ride is greater than K.

The use of the sequence numbers allows the receiver to separate the old channel K blocks from the new ones. This guarantees that the destination will not misroute packets. An additional benefit is that the end of speech will not be clipped if the last data packets arrive after the teardown is received. This protocol is quite simple to implement, although it requires a table at the receiver of the values of K for each channel ID.

Alternate solutions to this reuse problem exist which can operate when the above restrictions are relaxed. The simplest approach is to have the source keep a linked list of free channel ID's. The list is initialized to contain all channel ID's, in order. When a new channel is required to be established, the channel ID is taken from the top of the list. When a channel is torn down, its ID is placed at the bottom of the list. This makes the time between channel ID reuse as long as possible, and reduces the probability of confusion. With this method, it is no longer necessary to include sequence numbers in the tear down messages. Also, the receiver does not need to maintain a table.

4.4.3 Channel ID Coding

This section discusses some of the options for coding the channel ID field.

4.4.3.1 Fixed Length

The fixed length approach is the most straightforward. A fixed number of bits is assigned to the channel ID. Issues surrounding the number of bits required have been discussed above.

4.4.3.2 Implicit + Present Mask

In reality, the channel ID's are very redundant. Both source and destination know the set of active connections and their channel identifiers from the signalling messages. Therefore, if the blocks are placed in the packet in order of increasing channel ID, very little information actually needs to be sent. In fact, without silence suppression, channel activity and the presence of a block in a packet are likely to be equivalent, in which case NO information actually needs to be sent about channel ID's.

Unfortunately, there are some practical problems with this. First, silence suppression is used. Secondly, even if it weren't, it is possible for the voice codecs at the ITG not to have their framing synchronized (as in the general case above), so that a packet may not contain data from all users. Thirdly, the source and destination do NOT have a consistent view of the state of the system. There is a delay while signaling messages are in transit.

A few simple mechanisms can be used to overcome these complexities. In the header of the packet, a mask is sent. Each bit in the mask indicates whether data from a channel is present in the packet or not. Mapping of channel ID's to bits is done by sorting the channel ID's, and mapping the lowest number to the first bit, next lowest to the second, etc. Therefore, if a channel has no data for that packet, its bit is set to zero. Given that the source and destination agree on how many connections are active at all points in time, the number of bits required is known to both sides.

The next step is to deal with the differences in state. An additional field, called the "state-number", perhaps 5 bits, is sent in the header of the packet. This field starts at zero. Lets say at some point in time, its value is N. The source wishes to tear down a channel. It sends the tear down message to the destination, but continues to send data for that channel (or it may choose to send nothing, but must set the appropriate bit in the mask to zero). When the destination receives the message, it replies with an acknowledge. When the acknowledge is received by the source, the source considers the channel torn down, and no longer sends data for it, nor considers it in computing the mask. In the packet where this happens, the source also increments the state-number field to N+1. The destination knows that the source will do this, and will therefore consider the state changed for all packets whose value of the field is N+1 or greater.

When the next signaling message takes effect, the field is further increased. Even if packets are lost, the value of the state-number field for any correctly received packet completely tells the destination the state of the system as seen in that packet. Furthermore, it is not necessary to wait for a particular setup or teardown to be acknowledged before requesting another setup or teardown.

The number of bits for the state-number field should be set large enough to represent the maximum number of state changes which can have taken effect during a round trip time. As an alternative, an additional exchange can occur. After the destination receives a packet with state number greater than N, it destroys the state related to N, and sends back, reliably, a "free-state N" message, indicating to the destination that state N is now de-allocated, and can be used again. Until such a message is received, the source cannot reuse state N. This is essentially a window based flow control, where the flow is equal to *changes in state*. With this addition, the number of bits for the state number can be safely reduced, and it is guaranteed that the destination will *never* confuse the state, independent of the number of state-number bits used. However, the use of too few state bits can cause call blocking or delay the teardown of inactive channels.

This problem in state difference appears to be similar to the channel ID reuse problem described in Section 4.4.2. However, there is an important difference. In the channel ID reuse problem, if the packet containing the last block of a user arrives before the signaling message tearing down that connection, there is no problem. The destination will generally play out silence until the signaling message is received. Here, however, the destination must know that blocks are no longer present in the data stream independent of when the signaling messages arrive.

There are some drawbacks to this approach. They require the source and destination to maintain state. Any error in processing at either end, or a hardware failure, causes a complete loss of synchronization. This "hard-state" nature of the protocol can be relaxed by having the source send the complete state of the system with each signaling message, along with the "state-number" field for which this state takes effect. This guarantees that even in the event of end-system failure, the system state will be refreshed whenever a new connection is set up or torn down. Furthermore, the state can be sent periodically to improve performance.

4.5 Length Indicators

There are many ways to actually code the length indicators. The first question, however, is the range of lengths which must be coded.

4.5.1 Range of Length Indicators

Here, there is a clear tradeoff between flexibility and efficiency. A larger range can accommodate a variety of different media (such as video) where lengths may be large. However, this comes at the expense of a long length field, which may require another word of header to hold. For voice, one would expect a maximum bitrate to be 64 kbps, and around 50ms packetization delay. This yields exactly 100 words of data. Therefore, an eight bit field is probably sufficient for most voice applications.

4.5.2 PTI Based Lengths

In many applications, the amount of data present depends on the voice codec in use. Frame based coders will generally send a frame at a time. Since the codec type is indicated by the PTI field, it may not always be necessary to send length information at all. Even for non-frame based codecs, such as PCM, default data sizes can be set in the standard (as in RFC 1890 [4]). An extension bit can be used to indicate a non-standard length, so that when set, a length field follows. This allows for efficient coding of the most common cases, but allows for variable lengths with little additional cost.

4.5.3 Variable Length w/ Indicator

In this approach, a variable length header is used. All of the length indicators for all of the blocks are placed together in the beginning of the packet. However, the first four bits of this header field indicate the number of bits used for each length field. What follows are the length fields themselves, each using the number of bits indicated by the first four bits. This approach scales well, using a small overhead when the block lengths are small, and a larger overhead when they are larger. The drawback is a variable length header field, plus additional complexity in the parsing. An example of this technique is depicted in Figure 4. In the first example, the four bit indicator field has a value of three, so that the length fields are all three bits long. The four lengths are then 2,6,3, and 8. In the second example, the 4 bit indicator has a value of two, so that the length fields are all two bits long. The four lengths are thus 3,2,1, and 3.

	4b	3b	3b	3b	3b
Example 1	0011	010	110	011	100

	4b	2b	2b	2b	2b
Example 2	0010	11	10	01	11

Figure 4: Variable Length w/ Indicator

4.5.4 Remaining Packet Length Based Lengths

UDP always informs RTP of how many bytes are in the payload. This itself restricts the possible length of the first block, since its length must be less than the total packet length minus the RTP header. Furthermore, as each block is placed into the packet, the possible set of lengths that it can have shrinks - it must always be less than the remaining length in the packet. This approach, therefore, codes each length field with log2 of the number of bits remaining in the packet. This approach works extremely well when there is a long packet followed by several shorter ones, whereas the previous approach performs poorly in this case. Furthermore, it eliminates the length indicator present in the previous approach. However, it is even more complex than the previous technique. It can result in no savings under some conditions, especially since the header fields must be rounded to 32 bits.

Consider an example. The total size of the packet is 31 words. Inside of it are three blocks, the first whose length is 17, the second 8, and the third, 6. We would code the length field with 5 bits. After this block is read, the remaining amount of data in the packet is 14 words. Therefore, the next length field is coded with 4 bits. After this block, the remaining amount of data in the packet is 6 words, so the final length field is coded with three bits. The total is therefore 5+4+3 = 12 bits. In the previous approach (Section 4.5.3), the entire length field would have required 4 bits for the indicator (whose value would be 5), followed by 3 five bit fields, for a total of 19 bits.

One may question this example since the overhead of the length fields itself is not taken into account when computing the remaining length of the packet. While this can be incorporated, it makes things even more complex, and it is not actually necessary. All that is required is that the length fields are coded with log2(M), where M is any bound on the remaining amount of data which can be deterministically computed from past information. A simple bound is the packet length minus the *data* seen thus far (one can also subtract away any fixed length fields), precisely the metric used in the example above.

4.5.5 Table Based Approach

Realistically, most systems will operate with codecs that generate data in a fixed set of lengths (a frame size, for example). In that case, the set of lengths which can appear in the packet are usually very restricted. To take advantage of this fact, a table can be transmitted to the receiver reliably before transmission commences. This table can indicate the actual length of a block, and its coding. The symbols transmitted in the data packets are then used in this table to look up the actual lengths. This can reduce the length field to 2 or 3 bits. These lengths then all occur next to each other in the header. The technique now relies on state at the receiver, and the parsing process is further complicated by table lookups. In addition, the approach only works if you know the set of lengths before the system begins operation. If you allow the table to be dynamically modified during a session, synchronization problems occur, and the system becomes quite complex.

Further gains can be achieved through the use of Huffman codes instead of fixed length codes This only makes sense when different codecs (and correspondingly different lengths) are used with different frequencies. An example of such a situation is when the codec changes to a higher rate because of music-on-hold; a rare event in general.

4.6 Marker Bit

The marker bit has a general functionality, but is normally used to indicate the beginning of a talkspurt. It seems like a good idea to include this bit for each user.

4.7 Location of Per User Overhead

There will generally be overhead on a per-user basis (information such as channel ID, length, etc.). This information can be located in one of three places. First, it can all reside in front of the block to which it is applicable. Second, it can all be pasted together and reside up front in the header of the packet. The third is a hybrid solution, where some of it resides up front (such as channel ID), and some resides in front of the data. There are various pros and cons to the different approaches. The hybrid approach can be complex, since data is split into multiple places. The case where all the header is up front has a few minor advantages. First, it allows for a complete separation of the data from the header. The implementation is likely to be a little less complex, since extracting blocks does not require actually moving through the payload.

5. Options

5.1 **Option I: Mixer Based**

This option is the most straightforward to implement, but has the most overhead. The basic premise is to reuse the mixer concept introduced in RTP. Each user is considered a contributing source, and the gateway is considered a mixer. However, instead of mixing the media, separate data from each user appear in the payload. The 32 bit CSRC identifies each user, acting as the channel ID. Data from each user is organized into blocks. Each block has its own 32 bit header, which includes the length (12 bits) in units of 32 bit words, Marker bit (1b), TimeStamp Offset (12b), and Payload Type (7b). Furthermore, the payload type and marker bit are stricken from the RTP header (since they only make sense for an individual user), and the CC field expanded to fill the missing bytes. This allows for a 12 bit CC field, or 4096 users in a packet. Thus, the packet would look like:

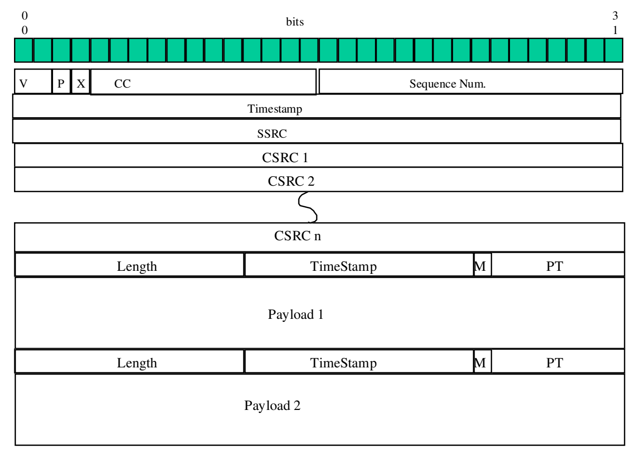


Figure 5: Option I

This approach allows for the most amount of generality in terms of variable length coders and coders with different frame sizes (see Section 4.3.1). The channel ID is longer than necessary, but using the concept of a contributing source for the channel ID necessitates the use of the additional bits. There are several variations on option I, many of which have been mentioned above:

I.A: Put the CSRC with each 32 bit length+M+PT field, instead of all of them being at the beginning. This has some pros and cons. As an interesting artifact of this change, it is no longer necessary to have a CC field. The length passed up by UDP is sufficient to recover the point at where you stop checking for additional blocks from users in the payload. In fact, the length field in the last block is not strictly necessary either.

I.B: Do the opposite of I.A. Put the length+M+PT field up front along with the CSRC fields, with the pattern being CSRC 1, length 1, CSRC 2, length 2, etc. Here again, the CC field is not strictly necessary.

I.C: The CSRC field can be shrunk to 8 bits. This allows for either 4 or two channel ID’s to be coded in the space of one word, whereas only one could in the current size of the field.

I.D: The CSRC field can be shrunk to 16 bits.

5.2 Option II: One word header

This option eliminates the large channel ID field present in the previous option. In the RTP header, the CC bit is set to zero, the marker bit has no meaning, and the payload type is TBD (possible uses include an indication of the number of blocks in the packet). The RTP timestamp corresponds to the generation of the first sample, among all blocks, enclosed in this packet. A one word header precedes each block of data. The number of blocks is known by parsing them until the end of the RTP packet. The one word field has a channel ID (8 bits), length (8 bits), Marker (1 bit), timestamp offset (11 bits), and payload type (4 bits). Channel ID number 255 is reserved, and causes the header to be expanded to allow for greater length, payload type, and possibly channel ID encodings. The specific format for this expanded header is for further study. Given the compacted payload type space, it may be a good idea to allow negotiation of the meaning for the payload type at the beginning of the connection. It may be worthwhile to expand the length field at the expense of the channel ID - this issue is for further study.

The format of the packet is thus:

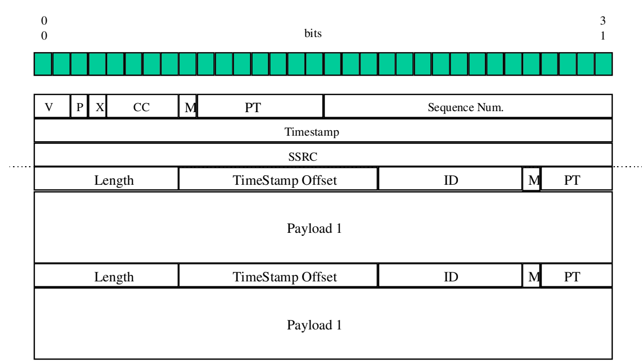


Figure 6: Option II

5.3 Option III - Restricted Case

Option II has the advantage of being able to support multiple frame sizes within a single packet. However, it comes at the expense of a 32 bit header (which can be large for low bitrate codecs), and at a reduced payload type field. This option has a 16 bit header, but does not support different frame sizes within a packet. It therefore falls into the category described in Section 4.3.2. Of the 16 bit header, the first bit is an expand bit (to be described shortly), and the second bit is the marker bit. The following 6 bits indicate payload type, and the remaining 8 are for channel ID. When the expand bit is set, an additional 16 bits are present, which indicate the length of the block. When expand is clear, the length is derived from the payload type. Since there is no timestamp offset, all the blocks in the packet must be time aligned and have the same frame lengths. Different sized frames are supported by using a different SSRC for each frame length (see Section 4.3.2). In the RTP header, the CC field is always zero. The marker bits and payload type are undefined. The timestamp indicates the time of generation of the first sample of each block. SSRC is randomly chosen, but always different for each frame size.

The block headers are all located at the beginning of the packet, and follow each other. If the total length of the fields is not a multiple of 32 bits, it is padded out to 32. The structure of the header is such that fields never break across packet boundaries. An example of such a packet is given in Figure 7. There are 7 blocks in this example. The first two have standard lengths based on the PT field. The next one uses the expansion bit to indicate the length. The fourth uses the PT field, the fifth the expansion bit, and the last two use the PT field. The last 16 bits of the header are padded out.

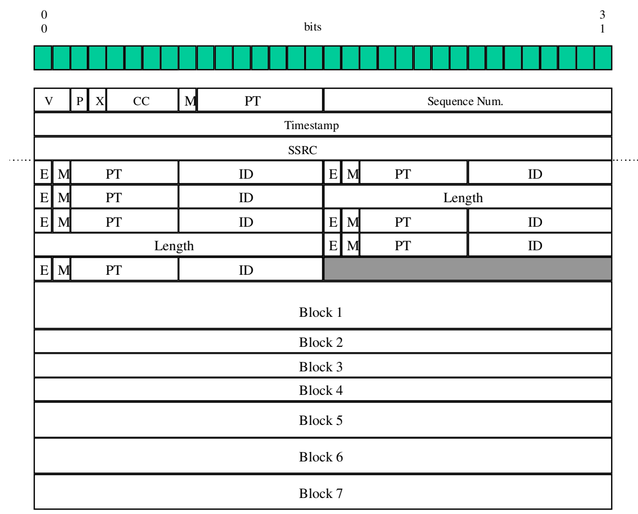


Figure 7: Option III

5.4 Option IV - Stacked RTP

This approach uses a duplicate of the RTP header as the per-block header. It is therefore extremely inefficient (12 bytes per block), but has several advantages: different media types can be mixed, since the timestamps are no longer related, and little processing is required if the sources being combined came from a single user RTP source. It also works well when one of the users is actually a mixer (for example, a conference bridge), since the CSRC can be used. Its main advantage is the reduction in overhead due to the IP and UDP headers. In addition to the standard RTP header, an additional header is required for length indication. This header has a number of 16 bit fields, each of which indicates a length for its corresponding block (including the 12 byte RTP header). The number of such 16 bit lengths fields is known by continuing to look for additional length fields until the total length of the packet passed up from UDP has been accounted for. If an odd number of such length fields is required, then an additional 16 bits of padding is inserted to make the length header a multiple of 32 bits.

The format of such a packet is given in Figure 8.

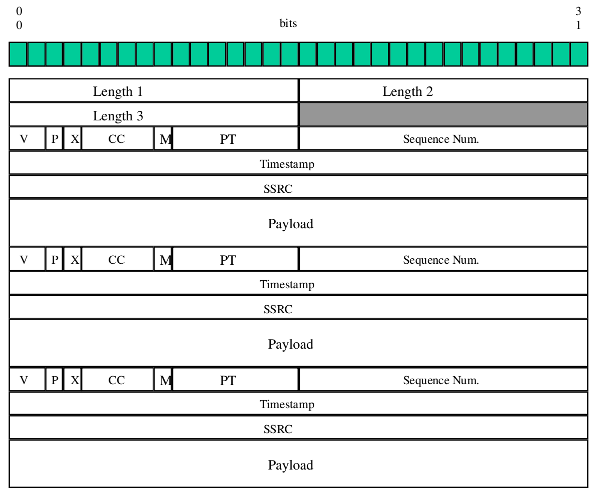


Figure 8: Option IV

5.5 Option V: Compacted

This option uses the Implicit + Mask approach outlined in Section 4.4.3.2 to code the channel ID. In all other respects it is similar to Option III. Now, however, the per-block header can be reduced to one byte: 1 bit of expansion, 1 bit of marker, and 6 bits of payload type. Furthermore, the length field (present when the expansion bit is set) is reduced to 8 bits from 16 in Option III. This reduction saves on space, but it also guarantees that fields remain aligned on byte boundaries. The mask bits are present in the beginning of the packet, and they are preceded by a 8 bit state-number. If the number of active channels is not a multiple of 32, the mask field is padded out to a full word. This approach is extremely efficient, but the channel identification procedure is more complex and requires additional signaling support.

A diagram of a typical packet for this option is given in Figure 9. The marker bits are indicated with lowercase m's. There are four active channels, each of which is present in this packet (all four mask bits would then be 1). The first block has a standard length, but the second has its expansion bit set, so that an 8 bit length field follows. The remaining two blocks have normal 8 bit headers. The last 24 bits of the header are padded to a word boundary.

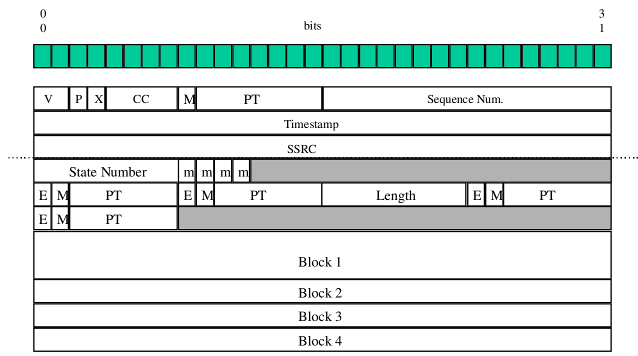


Figure 9: Option V

6. Comparison of Options

In this section, the options are compared in terms of efficiency. Issues relating to complexity, scalability, and generality have already been discussed in previous sections. The analysis here consists of two parts. The first is a table, indicating the efficiency of each option for a variety of speech codecs. Several tables are included for different numbers of users. The second analysis consists of a series of graphs which consider the efficiency vs. bitrate, assuming a fixed frame size and a certain number of users. This analysis helps to indicate the range of codecs which may be reasonably supported with each option.

6.1 Specific Codecs

In both Table 1 and Table 2, the efficiency vs. codec for all three options is tabulated. For G.711, G.726, G.728 and G.722, the frame size listed is a multiple of the actual frame size of the codec, which is too small to be sent one at a time. The efficiency is computed as the number of words of payload such a codec would occupy, times the number of users, divided by the total packet size (i.e., it does not consider inefficiencies due to padding the payload portion). Note that Option V is always superior in efficiency. The efficiencies are generally 1 to 10 percent apart. Table 1 considers the case where there are 10 users, and Table 2 considers the case where there are 24.

Codec	Bitrate (kbps)	FrameSize (ms)	Option I	Option I.C	Option I.D	Option II	Option III	Option IV	Option V
G.711	**64**	**20**	**93.02%**	**94.56%**	**94.12%**	**95.24%**	**96.39%**	**90.50%**	**96.84%**
G.726,	**32**	**20**	**86.96%**	**89.69%**	**88.89%**	**90.91%**	**93.02%**	**82.64%**	**93.88%**
G.728,	**16**	**18.75**	**76.92%**	**81.30%**	**80.00%**	**83.33%**	**86.96%**	**70.42%**	**88.47%**
G.729	**8**	**10**	**50.00%**	**56.60%**	**54.55%**	**60.00%**	**66.67%**	**41.67%**	**69.72%**
G.723	**5.3**	**30**	**62.50%**	**68.49%**	**66.67%**	**71.43%**	**76.92%**	**54.35%**	**79.33%**
G.723	**6.3**	**30**	**66.67%**	**72.29%**	**70.59%**	**75.00%**	**80.00%**	**58.82%**	**82.16%**
ITU 4kbps	**4**	**20**	**50.00%**	**56.60%**	**54.55%**	**60.00%**	**66.67%**	**41.67%**	**69.72%**

G.722	64	15	90.91%	92.88%	92.31%	93.75%	95.24%	87.72%	95.84%
GSM Full Rate	13	20	75.00%	79.65%	78.26%	81.82%	85.71%	68.18%	87.35%
TCH Half Rate	5.6	20	57.14%	63.49%	61.54%	66.67%	72.73%	48.78%	75.43%
IS54	7.95	20	62.50%	68.49%	66.67%	71.43%	76.92%	54.35%	79.33%
IS96	8.5	20	66.67%	72.29%	70.59%	75.00%	80.00%	58.82%	82.16%
EVRC	8.5	20	66.67%	72.29%	70.59%	75.00%	80.00%	58.82%	82.16%
PDC Full Rate	6.7	20	62.50%	68.49%	66.67%	71.43%	76.92%	54.35%	79.33%
PDC Half Rate	3.45	40	62.50%	68.49%	66.67%	71.43%	76.92%	54.35%	79.33%

Table 1: 10 Users

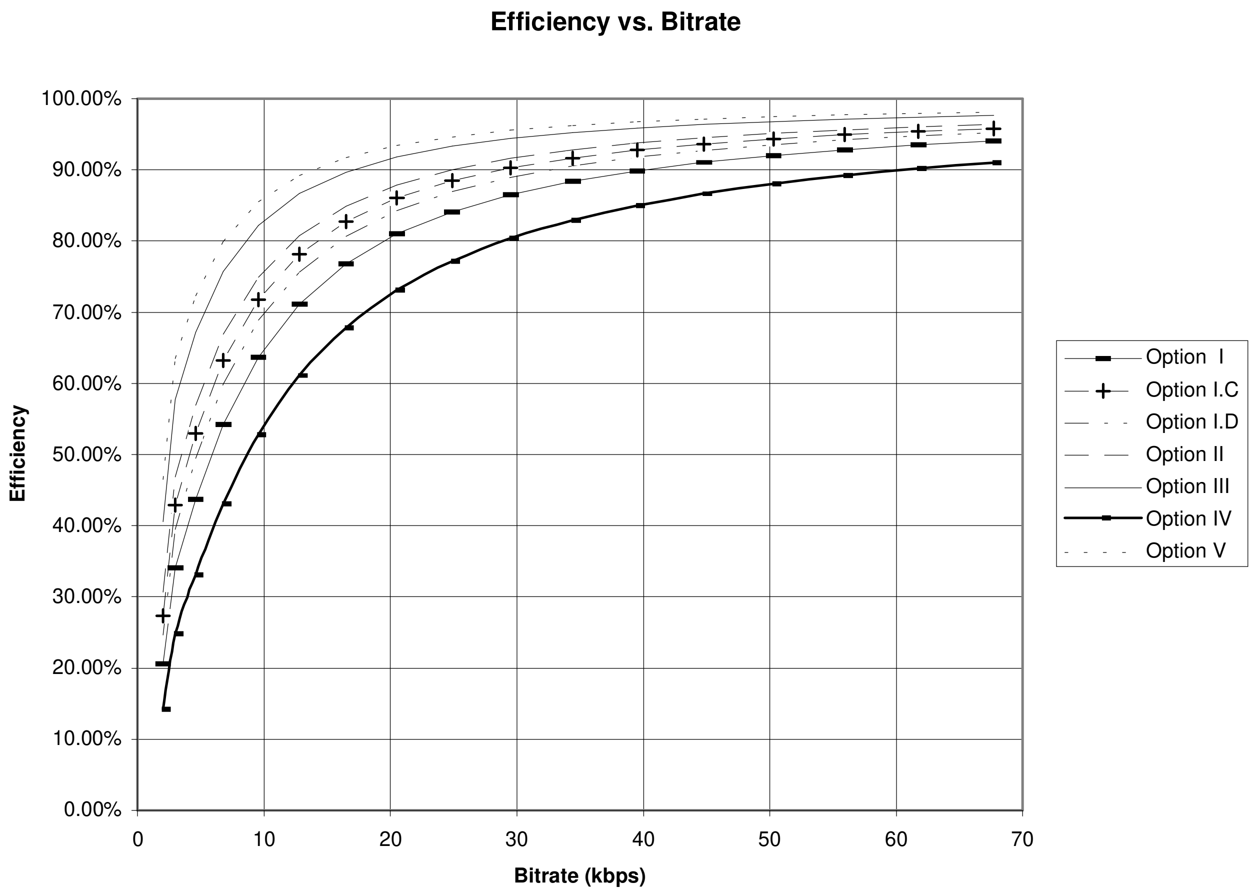
Codec	Bitrate (kbps)	FrameSize (ms)	Option I	Option I.C	Option I.D	Option II	Option III	Option IV	Option V
G.711	64	20	94.30%	96.00%	95.43%	96.58%	97.76%	91.34%	98.26%
G.726	32	20	89.22%	92.31%	91.25%	93.39%	95.62%	84.06%	96.57%
G.728	16	18.75	80.54%	85.71%	83.92%	87.59%	91.60%	72.51%	93.37%
G.729	8	10	55.38%	64.29%	61.02%	67.92%	76.60%	44.17%	80.87%
G.723	5.3	30	67.42%	75.00%	72.29%	77.92%	84.51%	56.87%	87.57%
G.723	6.3	30	71.29%	78.26%	75.79%	80.90%	86.75%	61.28%	89.42%
ITU 4kbps	4	20	55.38%	64.29%	61.02%	67.92%	76.60%	44.17%	80.87%
G.722	64	15	92.54%	94.74%	93.99%	95.49%	97.04%	88.78%	97.69%
GSM Full Rate	13	20	78.83%	84.38%	82.44%	86.40%	90.76%	70.36%	92.69%
TCH Half Rate	5.6	20	62.34%	70.59%	67.61%	73.85%	81.36%	51.34%	84.93%
IS54	7.95	20	67.42%	75.00%	72.29%	77.92%	84.51%	56.87%	87.57%
IS96	8.5	20	71.29%	78.26%	75.79%	80.90%	86.75%	61.28%	89.42%
EVRC	8.5	20	71.29%	78.26%	75.79%	80.90%	86.75%	61.28%	89.42%
PDC Full Rate	6.7	20	67.42%	75.00%	72.29%	77.92%	84.51%	56.87%	87.57%
PDC Half Rate	3.45	40	67.42%	75.00%	72.29%	77.92%	84.51%	56.87%	87.57%

Table 2: 24 Users

6.2 Efficiency vs. Bitrate

The following figure considers the efficiency of the protocol vs. bitrate. For this case, the frame size is fixed at 20ms, and the number of users at 24. As the bitrate varies, the block size varies, and therefore the efficiency does as well. The efficiency here is computed in a slightly different manner than the graph above. Here, the efficiency is the bitrate times the frame size (without padding to 32 bits), divided by the same quantity plus the packet and block overhead. This avoids the otherwise sawtooth behavior of the graph, which makes it very difficult to read.

The graph is very illustrative. The ordering of the efficiencies is no surprise; option V is always superior. However, the difference between the options is interesting. Despite the difference in overhead by a factor of two, Option V and Option III are very close in efficiencies over a wide range of bitrates. This is due to the fact that it requires a lot of users at low bitrates to overcome the IP/UDP/RTP header overhead, and at higher bitrates, the payload sizes are large enough to make the difference in block headers inconsequential.



7. References

[1] R. Ramjee, J. Kurose, D. Towsley, H. Schulzrinne, “Adaptive Playout Mechanisms for Packetized Audio Applications in Wide Area Networks”, Proceedings of IEEE Infocom, 1994
[2] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, “RTP: A Transport Protocol for Real-Time Applications”, Audio Visual Working Group Request for Comments RFC 1889, IETF, January 1996
[3] M. Handley, V. Hardman, I. Kouvelas, C. Perkins, J. Bolot, A. Vega-Garcia, S. Fosse-Parisis, “Payload Format Issues for Redundant Encodings in RTP”, Work In Progress
[4] H. Schulzrinne, “RTP Profile for Audio and Video Conferences with Minimal Control”, Audio Visual Working Group Request for Comments RFC 1890, IETF, January 1996