Real time Protocols

Michael Kropfberger, 9555885
mailto:michael.kropfberger@gmx.net

January 29, 2001

Seminar of Mathematics and Theoretical Computer Science
Prof. Dr. Patrick Horster
(620.710 WS 00/01)

Contents

1 Introduction

Today an increasing number of people using the internet, demands multi-media services, like live-reports, video on demand, but also multidirectional communication like video-telephony, Voice over IP (VoIP) and audio/video/whiteboard conferences with multiple participants.
Standard protocols like HTTP or FTP are not prepared for real-time. They lack quality of service guarantees and good streaming capabilities.
Fortunately, the Internet Engineering Taskforce generated well defined standards for diverse protocols achieving all these wanted features.
This text will describe not only the Real-Time Protocol (RTP) and its feedback mechanism, the Real-Time Control Protocol (RTCP), but will also present the RTP profiles for audio and video conferences and a secure RTP profile using packet encryption.
This secure RTP profile will be explained in further detail, containing the used encryption algorithm and message authentication codes. To understand the secure RTP profile, we also look closer into the RTP format and header specifications and header compression.
After that, there is a short introduction into the Resource reSerVation Protocol (RSVP), which allows quality of service over IP, and finally a high-level protocol, the Real-Time Streaming Protocol, which is needed to control multi-media connections based on RSVP, sent via RTP according to the defined RTP profile.

2 Real-Time Protocol (RTP)

The Real-Time Protocol is defined in RFC 1889 [1]. It provides end-to-end delivery services for all kinds of data with real-time characteristics. RTP itself does not define any behavior and handling of data types per se. So there exist special profiles how to treat different data types. For audio and video conferences, there is already a predefined profile (see section 4).
To make RTP easy to include in actual networks, it might be based on different underlying standard protocol layers, like ST-II, UDP/IP, IPX or ATM AAL5. The widest spread protocol is UDP, a packet-oriented protocol based on IPv4, which offers checksum services but doesn't guarantee, that packets arrive in order or arrive at all. In a multimedia environment, dropped packets are more favorable than resent packets, like TCP/IP would. Chances for a resent packet to arrive in time are small, and to play eg. some video or audio frames $f_t \ldots f_{t+n}$ when the user-shown stream already is at frame f_{t+n+m} makes no sense at all. Resending packets would also increase the needed network bandwidth and would favor network congestion.
IP, and hereby UDP, also offers the interesting feature of multicasting, which means having multiple clients receiving one and the same packet sent by a server. This incredibly decreases network load and also guarantees smaller server-side

connection maintaining costs (eg. a server just streams his live stream, but he has not to care about who is listening).
RTP itself also doesn't guarantee delivery or prevent out-of-order delivery, but includes packet sequence numbers and other useful information for real-time multimedia data to reorder incoming packets.
To monitor and adjust quality of service of RTP streams, there is a real-time control protocol (RTCP)[1], which carries information packets about lost RTP packets, jitter, and so on.

2.1 Mixers and Translators

Mixers are positioned somewhere on the route between the server and the clients, and change the incoming data in some means. So it might be useful, to change high-quality video to a lower resolution, and stream this low-quality video to a client connected via a low bandwidth network. All other clients connected via high bandwidth networks receive the high-quality video. Since every distinct RTP stream has a distinct RTP source identifier (SSRC), every re-mixed stream gets a new identifier, but keeps track of the originating sources as a "contributing source" (CSRC).
Translators forward RTP packets with their source identifiers intact, since they don't change the payload itself. A translator may be used on firewalls which don't allow multicast streams, so the mulicast is translated into unicast and directly sent to the clients behind the firewall (enabling all possibilities of authentication and other means of access control). Another typical usage for translators would be as a gateway between eg. UDP/IP and ATM AAL5.

2.2 Security

RTP does not provide any security features, but based on the underlying protocol or the application, eg. packet encryption might be implemented. How the client and server generates and exchanges keys is not in the scope of RTP and also has to be implemented somewhere else.
You find more about a RTP Profile for secure RTP in section 5.

2.3 RTP Packet Format

Figure 1 shows the RTP header on top of UDP/IP. Only the most important fields of the RTP header itself will be discussed here, for more detailed information please refer to RFC 1889[1].

2.3.1 Payload Type (7bits)

The *payload type* helps the application understanding and interpreting the RTP data right. There are some predefined payload types depending on the used

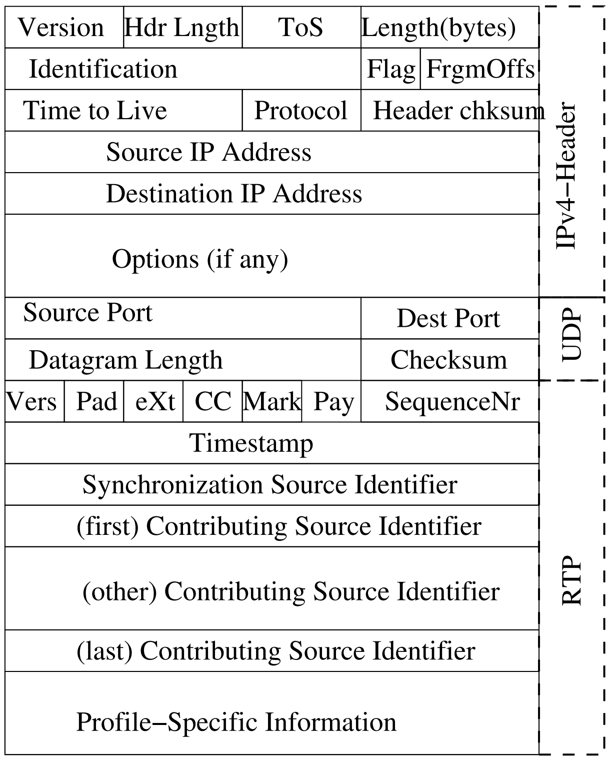

Figure 1: RTP Header on top of UDP/IP

RTP profile, eg. for video and audio conferences [5], there are *payload types* for different audio (eg. GSM, MPEG-1 layer 3) and video encodings (eg. JPEG, MPEG-1, MPEG-2, H.261).

2.3.2 Sequence Number (16 bits)

The *sequence number* is incremented for each RTP packet sent, and may be used by the receiver to detect packet loss and to restore packet sequence. The initial value should be a random number, to to make known-plaintext attacks on encrypted packets more difficult. Even if the server itself doesn't provide packet encryption, the *sequence number* should be randomly chosen, because there might be translators on a packets' way which provide secure tunneling.

2.3.3 Timestamp (32 bits)

The *timestamp* has to be generated by a monotonically and linearly clock. To allow sufficient jitter calculations and synchronization accuracy, the clock frequency should be more fine-grained than eg. a given video frame rate or audio sample rate. For security reasons, the *timestamp* should also be initiated with a random number, so the system clock is only one factor of many and offers a basic clue for calculations. Over multiple RTP packets, the *timestamp* might also not be monotonic, even if the *sequence number* is: eg. MPEG encoded video streams store frames in a different order than the have to be displayed later on.

2.3.4 Synchronization Source Identifier (SSRC) (32 bit)

This is a randomly chosen ID for a RTP stream. The *SSRC* has to be unique so a client, mixer or translator can distinguish between different RTP streams. Even that the chance for multiple *SSRCs* is low, RTP implementations must be prepared to detect and resolve collisions.

2.3.5 Contributing Source Identifiers (CSRC) (32 bit)

As discussed before, mixers include all originating sources as "contributing" ones for the newly generated RTP stream. the *CSRC Count (CC)* field (4 bits) holds the number of stored *CSRCs* and is in the range from 0 to 15. If there are more than 15 sources, all > 15 are lost.

2.4 Header Compression

In the minimal case a IP/UDP/RTP header block has a size of 40 bytes. Taking an 20 ms packetization interval, we have 16 kbit/s only for headers. This is definitely too much overhead for small serial links, so RFC 2508 [3] defines a way for IP/UDP/RTP header compression from 40 bytes down to 2 bytes (or 4 bytes with checksums). The compression protocol works for a static serial line (a modem connection) between a dial-in user and his ISP.
Compression is done by knowing the fact that only fifty percent of all data fields change from one to the other packet, so it is redundant to resend these fields. On the other hand, some values may change, but in a predictable (constant) manner, so the second-order difference is also zero.
So after transferring the full header and the first-oder difference once, it is enough to keep track of a connection with an 8 bit connection ID (*CID*) and when a compressed packet with this *CID* arrives and indicates no second-order differences, the receiver can reconstruct the original header with no loss of information.
If there was a second-order difference, the state change is sent in a little bigger packet. On major changes (eg. new *CIDs* arrive) more packets have to be sent to return to full compression mode.

3 Real-Time Control Protocol (RTCP)

The RTP control protocol (RTCP) is based on the periodic transmission of control packets to all participants in the session, using the same distribution mechanism as the data packets. Data and control packets have to be two different streams, so when we are using UDP/IP, we use two consecutive ports (n for data, $n+1$ for control, where n has to be an even number, so $n+1$ forcably will be an odd number). This also allows monitoring stations, only to receive RTCP packets and gather statistics.
RTCP performs four functions:

- The main feature is to send information about the quality of the describing stream. This feedback can be immediately used for adaptive encodings, to decrease or increase the quality of video or audio or to switch to a different encoding mechanism.

- RTCP also has clear-text information packets, where the most important is the CNAME, which is the canonical name for a RTP stream. So if the SSRC changes because of collision, all participants keep track of the stream by its CNAME.

- Each participant sends his control packets to all the others. This is needed to calculate the rate at which the RTCP packets are sent, so that there aren't too many RTCP packets in comparison to RTP packets on the net.

- Functions 1-3 should be used in all environments, but there is a further possible usage. RTCP could be used to keep track of all joined participants in a loosely controlled session, where virtually everyone may connect and disconnect without membership control or parameter negotiation.

3.1 RTCP Packet Format

3.1.1 Sender and Receiver Reports

The only difference between the RTCP sender report (figure 2) and the RTCP receiver report (figure 3), besides the packet type code, is 20-byte sender information section with relating timestamps and a packet and byte-counter. The *NTP Timestamp* stores the globally defined time when the report was sent. The *RTP Timestamp* corresponds to the timestamp format used in the RTP packets. This is important for synchronizing different RTP streams like audio and video. Then a various number of report blocks (according to the *report count* field) are attached. They store the *SSRC*, some statistics about *packet loss* and *jitter*, and finally the time of the last sent RTCP packet related to this *SSRC* and the time since that last sent RTCP packet. This is very useful for the receiver of this packet to calculate the roundtrip time.

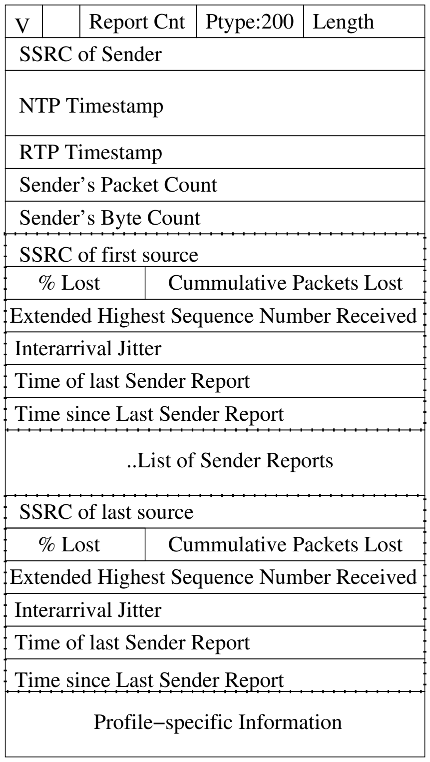

Figure 2: RTCP Sender Report (Packet Type = 200)

3.1.2 SDES: Source Description RTCP Packets

Since every RTP stream has some cleartext properties like its canonical name, RTCP offers a way to distribute those to all participants. Those properties are

- CNAME: the canonical identifier for a RTP stream (eg. jdoe@home:mystream)
- NAME: username, who set up this RTP stream
- EMAIL: electronic mail address of the above user
- PHONE: phone number in the international format (leading + and country code)
- LOC: geographic user location
- TOOL: application name and version
- NOTE: notice or status field about the source

V		R Cnt	Ptype:201	Length
SSRC of Sender				
SSRC of first source				
% Lost		Cummulative Packets Lost		
Extended Highest Sequence Number Received				
Interarrival Jitter				
Time of last Sender Report				
Time since Last Sender Report				
..List of Sender Reports				
SSRC of last source				
% Lost		Cummulative Packets Lost		
Extended Highest Sequence Number Received				
Interarrival Jitter				
Time of last Sender Report				
Time since Last Sender Report				
Profile−specific Information				

Figure 3: RTCP Receiver Report (Packet Type = 201)

- PRIV: private application-specific extensions
- END: indicates the last *SDES item* for this *SSRC/CSRC*.

To keep network load low, there should be an algorithm for sending different SDES packets: eg. always send *CNAME*, but send *EMAIL* every fifth packet, don't send all the others.

3.1.3 BYE: Goodbye RTCP Packets

There is a special BYE packet with the *packet type* set to 203. It might also include a text like "camera malfunction". Mixers should forward those BYE packets unchanged and if necessary, send their own BYE packets for their generated *SSRCs*.

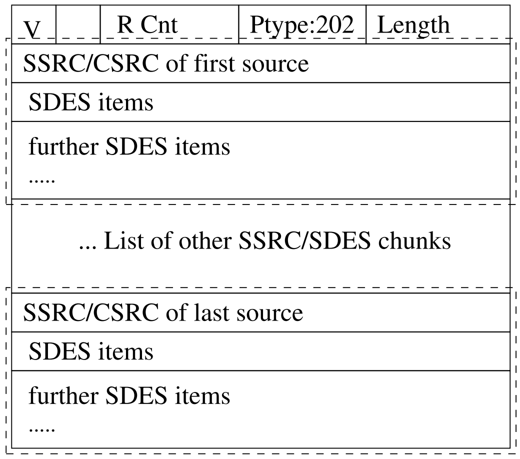

Figure 4: SDES: Source Description RTCP Packet(Packet Type = 202)

3.1.4 APP: Application-defined RTCP Packets

Finally, there is an APP packet, which is open for experimental use and for application-specific functions. It uses the *packet type* 204.

3.2 Security

As for RTP, there is no direct support for authentication, integrity, and confidentiality. All these features have to be supported by the underlying layer.
For simple packet encryption with a stream cypher, there has to be 32-bit random number prepended to all regular RTCP packets. This avoids vulnerability to known plaintext attacks.
It would also make sense to only encrypt eg. RTP packets and SDES packets, but not sender and receiver reports, so other monitoring stations are able to generate statistics.
The RTP internet draft[2], which is the successor of the RFC for RTP [1], recommends the usage of the DES (data encryption standard) in cypher block chaining mode (CBC) as described in section 1.1 of RFC 1423 [4].
The initialization vector is zero because there is already the random prefix for RTCP or the randomly initialized RTP *timestamp* and *sequence number*. If 56 bit keys are not secure enough (and it shouldn't be these days), it is easy to use triple-DES instead.
The key management infrastructure has to be provided by other protocols in the application. As pointed out before, all encryption may be realized in tunneling the data between two RTP translators.
These recommendations are all outdated by a new RTP Profile for secure RTP ,

which exactly defines the encryption in great detail. We will discuss this profile in section 5.

4 RTP Profile for Audio and Video Conferences (RTP/AVP)

Since RTP itself doesn't know what data it is transporting, there exist well defined profiles for all different kinds of multimedia data. RFC 1890 [5] describes the profile for audio and video conferences. It supports all different audio and video formats and how they have to be stored in the RTP payload data area.
RFC 1890 defines a default 20 ms packetization rate (which might go up to 200 ms to reflect high-compression formats' natural frame size). The sampling frequency should be chosen out of 8000, 11025, 16000, 22050, 24000, 32000, 44100, 48000.
Multi-channel sample-based or frame-based encodings should order their channels from left-to-right, like (left channel 1. sample) (right channel 1. sample) (left channel 2. sample) (right channel 2. sample).
Only some audio and video encodings are shown here, for a complete list refer to RFC 1890 [5].

4.1 Some Audio Encodings

RFC 1890 exactly describes the packet formats of all audio encodings bit by bit.

4.1.1 G.722, G.723, G.726, G.728, G.729

According to ITU-T Recommendations, those encoding standards define audio encoding with rates between 6.4 kbit/s up to 64 kbit/s. Those encodings are used for videophone terminal applications up to internet telephony in high quality.

4.1.2 GSM, GSM-HR, GSM-EFR

The European GSM standard for full-rate speech at 13 kbit/s (33 bytes for 160 samples), half-rate speech (14 bytes) and extended full-rate speech (31 bytes).

4.1.3 MPA

MPA defines the payload for MPEG-1 and MPEG-2 audio in the differently complex layers I, II, and III. The exact payload format for MPEG-1/MPEG-2 audio and video can be found in RFC 2250 [7].

4.1.4 RED

The redundant audio payload format "RED" (defined in RFC 2198 [8]) holds not only compressed audio data for the current interval, but also a highly compressed version of the last interval. This allows to reconstruct lost packets in better quality than silence substitution or amplitude/frequency interpolation.

4.2 Some Video Encodings

The RTP *timestamp* frequency is defined as 90,000 Hz for all the following video encodings. This suffices enough for jitter estimation and calculations with RTCP *timestamps*.

4.2.1 Motion JPEG

MJPEG defines a way of storing consecutive JPEG-encoded still-images. Some initialization tables are only sent once and then each raw image data is sent with only a minimum of necessary header information. Find out more about RTP/MJPEG in RFC 2435 [9].

4.2.2 H.261, H.263

According to ITU-T Recommendations, those encoding standards define video encoding with low bitrates up to 64 kbit/s. These encodings are used for videophone terminal applications and video telephony.

4.2.3 MPV

MPV defines the payload for MPEG-1 and MPEG-2 video ad described in RFC 2250 [7].

5 Secure Real-Time Protocol (SRTP)

The Secure Real-Time Protocol (SRTP) [10], a profile for RTP, provides privacy, message authentication, and replay protection.
SRTP achieves high throughput and low packet expansion by using an additive stream cypher for encryption, a universal hashing based function for message authentication, and an "implicit" index for sequencing based on the RTP sequence number. The encrypted payload contains data defined by another profile like RTP/AVP.
Figure 5 shows the SRTP packet format. It is identical to the RTP format, except the 4-byte authentication tag at the very end. The encrypted block starts after the 12th byte, so the main header with the *sequence number*, *timestamp* and the *SSRC* is still visible. This is needed for parsing and putting the arrived SRTP packet into the right cryptographic context.

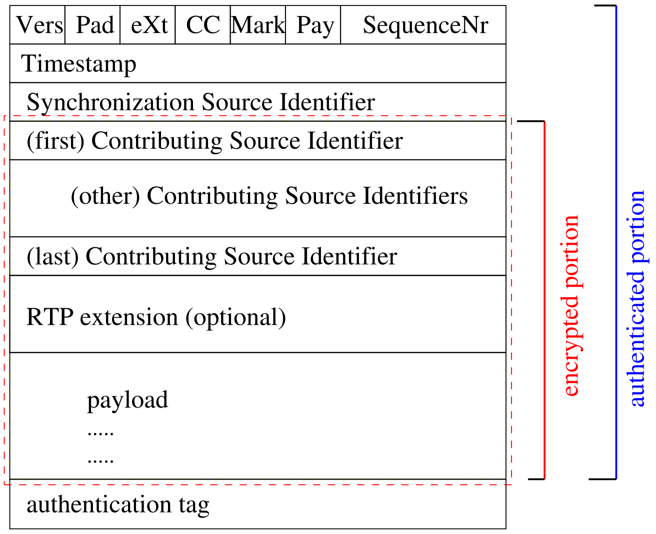

Figure 5: SRTP packet format

The full packet starting from byte 0 up to (including) the payload is cryptographically hashed with a MAC, so it is impossible to change the cleartext header data or concatenate different headers and payloads on a packets way to the receiver.

5.1 Cryptographic context

The sender and receiver has to maintain some cryptographic state information. It contains

- an encryption key k_e
- a message authentication key k_m
- a 32-bit rollover counter r (which counts how many times the 16-bit *RTP sequence number* wrapped around 0xFFFF)
- the last authenticated sequence number s_l
- a replay list L (only receiver side), which keeps track of already processed packets (or better, their *sequence numbers*).

The *SSRC*, the encryption key and the authentication key have to remain fixed for the whole session, and SRTP doesn't provide any means of key management and distribution.

5.2 Cryptographic Background

The default encryption cypher is the AES block cypher in counter mode. Advanced Encryption Standard (AES) [11] is not a particular encryption algorithm, but defines guidelines for an upcoming AES-compliant block cypher. This algorithm will be used by the US Government and hopefully by a wide-spread community world-wide. AES specifies the need for a symmetric key block-cypher with a 128-bit block size and three key sizes of 128, 192 and 256 bits. An AES report [12] describes all possible candidates in detail. Presented on October 2000, the winner is "Rijndael" [13], and will be proposed as the new Advanced Encryption Standard (AES).

5.2.1 AES winner: Rijndael

A data block to be processed using Rijndael [13] is partitioned into an array of bytes, and each of the cipher operations is byte-oriented.
Rijndael is a substitution-linear transformation network with 10, 12 or 14 rounds, depending on the key size.
Rijndaels round function consists of four layers. In the first layer, an 8x8 S-box is applied to each byte. The second and third layers are linear mixing layers, in which the rows of the array are shifted, and the columns are mixed. In the fourth layer, subkey bytes are XORed into each byte of the array. In the last round, the column mixing is omitted.
Rijndael was submitted by Joan Daemen (Proton World International) and Vincent Rijmen (Katholieke Universiteit Leuven).

5.2.2 Block cyphers in Segmented Integer Counter Mode (SIC)

To increase resistancy to redundant plaintext attacks, the used block-cypher is used in an additive manner.
The well-known approach is the cypher block chaining mode (CBC), where first the plaintext block B_i is XORed with the previous block B_{i-1} and then encrypted.
This approach is not feasible for multi-media data, since we can't (and don't want to) guarantee, that no packets are dropped. So without receiving all blocks $B_j (0 <= j < i)$, we wouldn't be able to restore the block B_i.
To overcome this obstacle, we use a unique counter, encrypt this and then XOR ($\oplus$) the result with our actual plaintext block (see figure 6).
One might say that successive ctr and $ctr + 1$ have a small Hamming difference, and might facilitate differential cryptoanalysis. However, this concern is only valid if the underlying cipher is differentially weak, so it's not the fault of the Counter Mode approach, but has to be faced by eg. AES criterias for the upcoming standard.

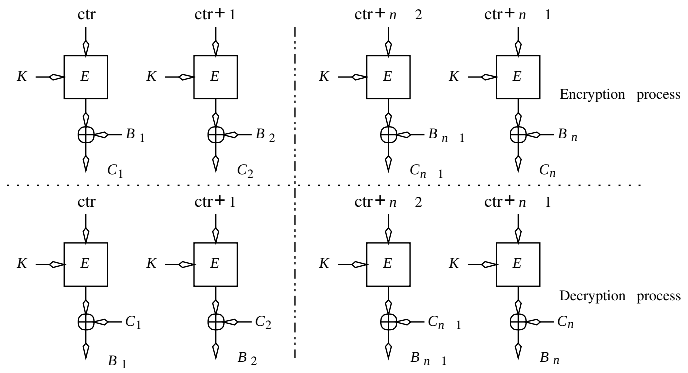

Figure 6: Encryption and Decryption Process in Counter Mode

In the SRTP case, the counter is set to

$$ctr = [(r * 65536) + seq] * 4096 + i$$

where r is the rollover counter, seq is the *RTP sequence number* and i is a counter for each 128-bit block. Since the maximum size of an IP packet is 64KB, this splits up to a maximum of 4096 blocks. IP Jumbo frames would require different values, but are not likely to be used for RTP-based multimedia traffic.
Since the counter has to be unique in its whole life time, and our rollover counter r is 32 bits, we have a maximum of $2^{48} = 281,474,976,710,656$ SRTP packets before the *SSRC* and the k_e and k_m keys have to be changed. This means to completely start a new RTP streaming session, but taken a 20 ms packetization time (eg. audio profile), this would be every 178,510 years. Non the less, if this happens, the actual session has to be renewed.

5.2.3 Message Authentication

The default Message Authentication Code (MAC) is UMAC [14]. It is very fast (eg. one clock cycle for one byte) and also offers extra security not only by a cryptographic key, but also a "nonce", in our case the already used counter *ctr*. UMAC uses the following function:

$$UMAC = E_{AES}(k_m, ctr) \oplus UHASH(k_m, B_i)$$

Where first our AES-compliant algorithm E_{AES} is used on counter *ctr* with the authentication key k_m, and the result is XORed with the 4 byte keyed hash value of our block B_i, where the hashing is obviously the most time consuming part onf this operation.

UHASH is a keyed hash function, which takes a string of arbitrary length and returns a fixed-length string of UMAC-OUTPUT-LEN bytes (for us, 4 bytes).
UHASH does its work in three layers. First, a hash function called *NH* is used to compress input messages into strings which are typically many times smaller than the input message. *NH* is optimized for speed, since it has to deal with the biggest junk of bytes.
Second, the compressed message is hashed with an optimized "polynomial hash function" into a fixed-length 16-byte string. Finally, the 16-byte string is hashed using an "inner-product hash" into a string of length 2 bytes. These three layers are repeated (with a slightly modified subkey from our master key k_m) until the outputs total UMAC-OUTPUT-LEN bytes. In our case, we need 2 rounds for our 4 byte hash value.
Note: Because the repetitions of the three-layer scheme are independent (aside from sharing some internal key), it follows that each "word" of the final output can be computed independently. Hence, to compute a prefix of a UMAC tag, one can simply repeat the three-layer scheme fewer times. Thus, computing a prefix of the tag can be done significantly faster than computing the whole tag. This allows the receiver to trade guaranteed authenticity with speed, because he might only calculate 2 out of 4 bytes UHASH value (by just doing one round of the three-layer algorithm).
Note: even if we only calculate a prefix hash value, we can XOR it with the same length prefix of $E_{AES}(k_m, ctr)$!
To do so makes sense to avoid denial of service attacks, because packets can be processed more quickly until replay is detected.

5.3 Replay Detection

A packet is "replayed", when someone in between the network path keeps a passing by packet and re-injects it later on.
The replay list L is a simple bit field over the last SRTP_WINDOW_SIZE *sequence numbers SRTPseq* (*SRTPseq* is defined as $r * 65,536 + seq$).
This is implemented as a sliding window approach, always covering the window between $SRTPseq - SRTP\_WINDOW\_SIZE$ and $SRTPseq$. Every incoming packet with a *RTPseq* smaller than $SRTPseq - SRTP\_WINDOW\_SIZE$ should also be treated as replayed.
Every detected replay should be logged. Still, our protocol will then paranoically overreact on erroneously cloned packets on some misconfigured routers, so there should be a "knob" for tuning this at the application.
The SRTP draft suggests, that the SRTP_WINDOW_SIZE must be at least 64, but might be higher.

So how big should SRTP_WINDOW_SIZE really be? This depends mainly on the receiver buffer size and hereby the requirement for "real" real-

time. The bigger the buffers, the more time it takes to fill them up for the first time, so this increases the communication lag.
For unidirectional communication (eg. radio), a bigger lag (like 3 seconds) and hereby a higher buffer size is allowed and tolerated, but video conferences can't tolerate more than, let's say, 500 ms lagging.
The recommended 20ms packetization time leads to 50 packets per second. So for video conferences, receiving data which is out of the 500 ms buffer, has to be discarded anyway. This would lead to throw away every packet older than $s_l - 25$, so SRTP_WINDOW_SIZE should be 32 (for efficiencys sake).
So for the above example, why should we detect and log replay for packets we would discard anyway? So maybe the "MUST be at least 64" statement is a little strong.

6 Secure Real-Time Control Protocol (SRTP)

The Secure Real-Time Control Protocol (SRTCP) [10] is described in the same profile as SRTP, and provides the same features, namely privacy, message authentication, and replay protection.
It is based on the same ideas like SRTP, so there is the normal RTCP header, with an attached authentication tag and an SRTCP index (figure 7). The additional index was not necessary in the SRTP packets, because they generated their index with the *RTP sequence number* and the rollover counter).

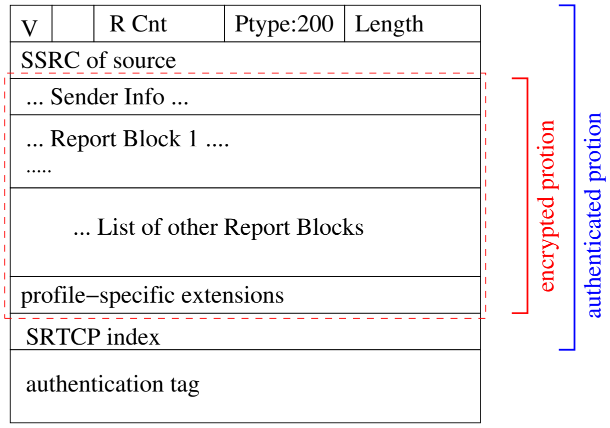

Figure 7: SRTCP Packet Format

SRTCP also uses an AES compliant additive block cypher in counter mode (SIC), and the UMAC message authentication code.
Since RTCP packets are not sent that frequently as RTP packets are, we rely on the 32-bit *SRTCP index*, which gives us $2^{32} = 4,294,967,296$ SRTP packets

before the index would be re-used for encryption. If an SRTCP stream reaches this maximum, the last packet has to be a RTCP BYE packet and the key pairs and connection status has to be reinitialized.

7 Real-Time Resource ReSerVation Protocol (RSVP)

As mentioned earlier, the Real-Time Protocol (RTP) runs on different lower-level transport layers like UDP/IP and ATM AAL5. Thus, RTP has no means to guarantee any quality of service. ATM supports different QoS types but UDP/IP, based on best-effort internet technology, doesn't. Even if IP runs over ATM, all standard "features" of IP are translated to ATM, so is the best-effort packet-forwarding property.
To overcome this obstacle, RFC 2205 [15] defines a Resource Reservation Protocol (RSVP) based on IP. To adopt this approach to IP-over-ATM, RFC 2382 [16] maps this RSVP to ATM.
RSVP supports three traffic types: best effort (as known from standard IP, but with reserved paths), rate-sensitive and delay-sensitive.

7.1 rate-sensitive (guaranteed bit rate) Reservations

An application might request 100 kbps of bandwidth, and even if it then sends 200 kbps for a certain time, only 100 kbps will go through and the rest is delayed. So we trade timeliness for a guaranteed bit rate.

7.2 delay-sensitive Reservations

For example, the MPEG-2 codec sends frames with a certain fixed frame rate, but there are different sizes in the frames. Very simplified, there are full frames (eg. 7 Mbps) and then some following delta-frames (3 Mbps), because those frames only store the difference to the preceeding frame. But still, it is important that all sent frames arrive in time. There exists a RSVP controlled-delay service (non-real time) and a predictive service (real time).

7.3 Lifetime of a reservation

After a reservation request, RSVP path messages are sent through the network to set up the reservation in each routing node. RSVP maintains a soft-state, so if the end-system doesn't send packets on a regular basis, a timeout occurs and all routers discard the reservation. If a router breaks down, this is detected by connected routers and the routing path is reassigned. RSVP teardown messages remove the path and reservation state without waiting for the cleanup timeout period.
Of course, this all enforces routers which understand RTSP. If not, RSVP can be tunnelled over a "non-RSVP" network.

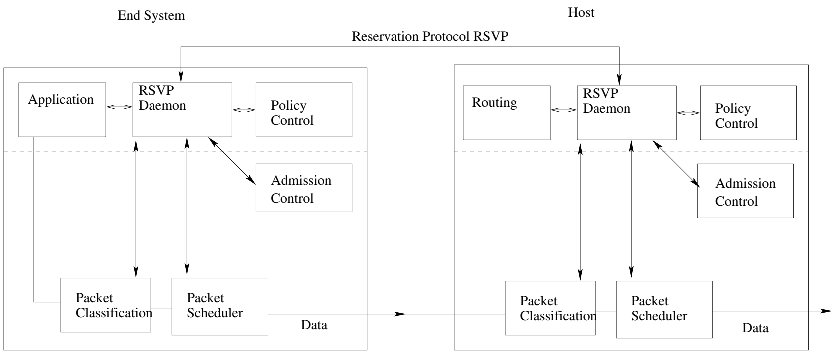

Figure 8: RSVP reservation between the first router and the host

Each node capable of resource reservation has several local procedures for reservation setup and enforcement (see figure 8). *Policy control* determines whether the user has administrative permission to make the reservation. In the future, authentication, access control and accounting for reservation will also be implemented by policy control. *Admission control* keeps track of the system resources and determines whether the node has sufficient resources to supply the requested QoS.
The RSVP daemon checks with both procedures. If either check fails, the RSVP program returns an error notification to the application that originated the request. If both checks succeed, the RSVP daemon sets parameters in the packet classifier and packet scheduler to obtain the requested QoS. The packet classifier determines the QoS class for each packet and the packet scheduler orders packet transmission to achieve the promised QoS for each stream.

8 Real-Time Streaming Protocol (RTSP)

The Real-Time Streaming Protocol (RTSP) [17] establishes and controls either a single or several time-synchronized streams of continuous media such as audio and video. The stream itself is typically not included, but might be also be interleaved into the control stream. The stream might also be sent via RTP, or any other real-time protocol like RealNetworks RDP.
RTSP provides "VCR-style" remote control functionality, like play, pause, fast forward, reverse, and absolute positioning.
The protocol has a similar syntax and operation to HTTP/1.1, and the default port is 554. Each presentation and media stream is identified by an RTSP URL like rtsp://music.bigserver.com/new-smash-hit.ram
For error codes, HTTP error codes like "404 File not found" are re-used, but there are also new ones like "453 Not enough bandwidth".

8.1 Important Methods

In the following, we will list up the most important methods to get information about existing streams, setting up streaming sessions and then play/pause/stop them.

8.1.1 DESCRIBE

The DESCRIBE method retrieves the description of a presentation or media object identified by the request URL from a server.

```
C->S: DESCRIBE rtsp://server.example.com/fizzle/foo RTSP/1.0
      CSeq: 312
      Accept: application/sdp

S->C: RTSP/1.0 200 OK
      CSeq: 312
      Date: 23 Jan 2001 15:35:06 GMT
      Content-Type: application/sdp
      Content-Length: 376

      v=0
      o=mhandley 2890844526 2890842807 IN IP4 126.16.64.4
      s=SDP Seminar
      i=A Seminar on the session description protocol
      u=http://www.cs.ucl.ac.uk/staff/M.Handley/sdp.03.ps
      e=mjh@isi.edu (Mark Handley)
      c=IN IP4 224.2.17.12/127
      t=2873397496 2873404696
      a=recvonly
      m=audio 3456 RTP/AVP 0
      a=control:rtsp://server.example.com/fizzle/foo/audio
      m=video 2232 RTP/AVP 31
      a=control:rtsp://server.example.com/fizzle/foo/video
```

So the client asks the server for a media object *foo* and expects a description in the *application/sdp* Session Description Protocol (SDP) [19]. The server answers with a unique sequence number *CSeq* and a *SDP* block with information about the title, author, and (in this case) multicast IP address, and time length. In this example, we have two streams in the RTP/AVP format, and their exact location.

8.1.2 SETUP

Now the client can set up the two RTP/AVP streams (audio and video) using the SETUP command.

```
C->S: SETUP rtsp://server.example.com/fizzle/foo/audio RTSP/1.0
      CSeq: 302
      Transport: RTP/AVP;unicast;client_port=4588-4589

S->C: RTSP/1.0 200 OK
      CSeq: 302
      Date: 23 Jan 2001 15:35:06 GMT
      Session: 47112344
      Transport: RTP/AVP;unicast;
        client_port=4588-4589;server_port=6256-6257

C->S: SETUP rtsp://server.example.com/fizzle/foo/video RTSP/1.0
      CSeq: 303
      Transport: RTP/AVP;unicast;client_port=4590-4591

S->C: RTSP/1.0 200 OK
      CSeq: 303
      Date: 23 Jan 2001 15:35:10 GMT
      Session: 47112344
      Transport: RTP/AVP;unicast;
        client_port=4590-4591;server_port=6258-6259
```

The client installs two streams for the two objects in the container (audio and video) and the server sets them up, using the same session number.

8.1.3 PLAY

After that, both streams are started together and synchronized.

```
C->S: PLAY rtsp://server.example.com/fizzle/foo RTSP/1.0
      CSeq: 835
      Session:47112344
      Range: ntp=0-

S->C: RTSP/1.0 200 OK
      CSeq: 835
      Date: 23 Jan 2001 15:35:16 GMT
```

8.1.4 PAUSE

PAUSE communication looks like PLAY. When a stream is paused, all reservations and already set up paths and streams stay alive but aren't used.

8.1.5 TEARDOWN

TEARDOWN communication looks like PLAY. But here, all resources associated with the session are freed. The session identifier is not valid anymore and to restart a stream, a new SETUP request has to be sent.
How firewalls should handle RTSP, its ports and packets, is described in [20].

9 Conclusion

In this paper we have read about well defined real-time protocols, the Real-Time Protocol (RTP) and its control facilities, the Real-Time Control Protocol (RTCP), a profile for video and audio and an extensive part about the secure RTP profile using an additive block-cypher encryption in counter mode. Tamper proveness and authentication is done by the message authentication code UMAC. SRTP also prevents replays by maintaining replay lists, but doesn't define key management and distribution, which has to be done at application level.
Then we discussed the Resource ReSerVation Protocol (RSVP) for Quality of Service demands over IP and a HTTP-like Real-Time Streaming Protocol (RTSP) providing "VCR-style" remote control functionality.
Still the internet community is not satisfied yet, because it still lacks of proper applications, which really take advantage out of all these standards.
Namely, there is only RealNetworks already using RTSP, but instead of RTP, they favor their own protocol RDP.
RSVP is supported by most of the major router companies (eg. Cisco) but again, it is not really used by a wide community.
Finally, we can only wait for the things to come and we can say that there is a theoretically hard basis for even secure real-time streaming over the internet.

References

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

[2] H. Schulzrinne, S. Casner, R. Frederick,and V. Jacobson, IETF Internet-Draft, draft-ietf-avt-rtp-new-08.ps, "RTP: A Transport Protocol for Real-Time Applications", July 2000

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

[4] D. Balenson, RFC 1423: "Privacy Enhancement for Internet Electronic Mail: Part III: Algorithms, Modes, and Identifiers", February 1993

[5] H. Schulzrinne, S. Casner, RFC 1890: "RTP Profile for Audio and Video Conferences with Minimal Control", January 1996

[6] H. Schulzrinne, S. Casner, IETF Internet-Draft, draft-ietf-avt-profile-new-09.ps, "RTP Profile for Audio and Video Conferences with Minimal Control", July 200

[7] D. Hoffman, G. Fernando,V. Goyal, M. Civanlar, RFC 2250, "RTP Payload Format for MPEG1/MPEG2 Video", January 1998

[8] C. Perkins, I. Kouvelas, O. Hodson, V. Hardman, M. Handley, J.C. Bolot, A. Vega-Garcia, and S. Fosse-Parisis, RFC 2198: "RTP Payload for Redundant Audio Data", IETF, September 1997

[9] L. Berc, W. Fenner, R. Frederick, S. McCanne, P. Stewart, RFC 2435: "RTP Payload Format for JPEG-compressed Video", October 1998

[10] D. McGrew, D. Oran, IETF Internet-Draft, draft-mcgrew-avt-srtp-00.txt, "The Secure Real Time Protocol", November 2000

[11] Advanced Encryption Standard (AES) Development Effort, http://csrc.nist.gov/encryption/aes/

[12] J. Nechvatal, E. Barker, L. Bassham, W. Burr, M. Dworkin, J. Foti, E. Roback, "Report on the Development of the Advanced Encryption Standard (AES)", October, 2000

[13] J. Daemen, V. Rijmen, "AES Proposal: Rijndael, AES algorithm submission", September 1999

[14] T. Krovetz,J. Black, S. Halevi, A. Hevia, H. Krawczyk, P. Rogaway, IETF Internet-Draft, draft-krovetz-umac-01.txt, "UMAC: Message Authentication Code using Universal Hashing", October 2000

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

[16] E. Crawley, L. Berger, S. Berson, F. Baker, M. Borden, J. Krawczyk, RFC 2382: "A Framework for Integrated Services and RSVP over ATM", August 1998

[17] H. Schulzrinne, A. Rao, and R. Lanphier, RFC 2326: "Real Time Streaming Protocol (RTSP)", IETF, April 1998

[18] H. Schulzrinne, A. Rao, and R. Lanphier, IETF Internet-Draft, draft-ietf-mmusic-rtsp-09.ps, "Real Time Streaming Protocol (RTSP)", February 1998

[19] M. Handley, V. Jacobson, RFC 2327: "Session Description Protocol (SDP)", IETF, April 1998

[20] "Using RTSP with Firewalls, Proxies, and Other Intermediary Network Devices", RealNetworks Inc., 1998, Version 2.0/rev.2

[21] Henning Schulzrinne, "Internet Media-on-Demand: The Real-Time Streaming Protocol", Sept. 1997, Dept. of Computer Science, Columbia University