

RTCP (Realtime Transport Control Protocol)

한화비전 VEDA 1기 B반

김태원

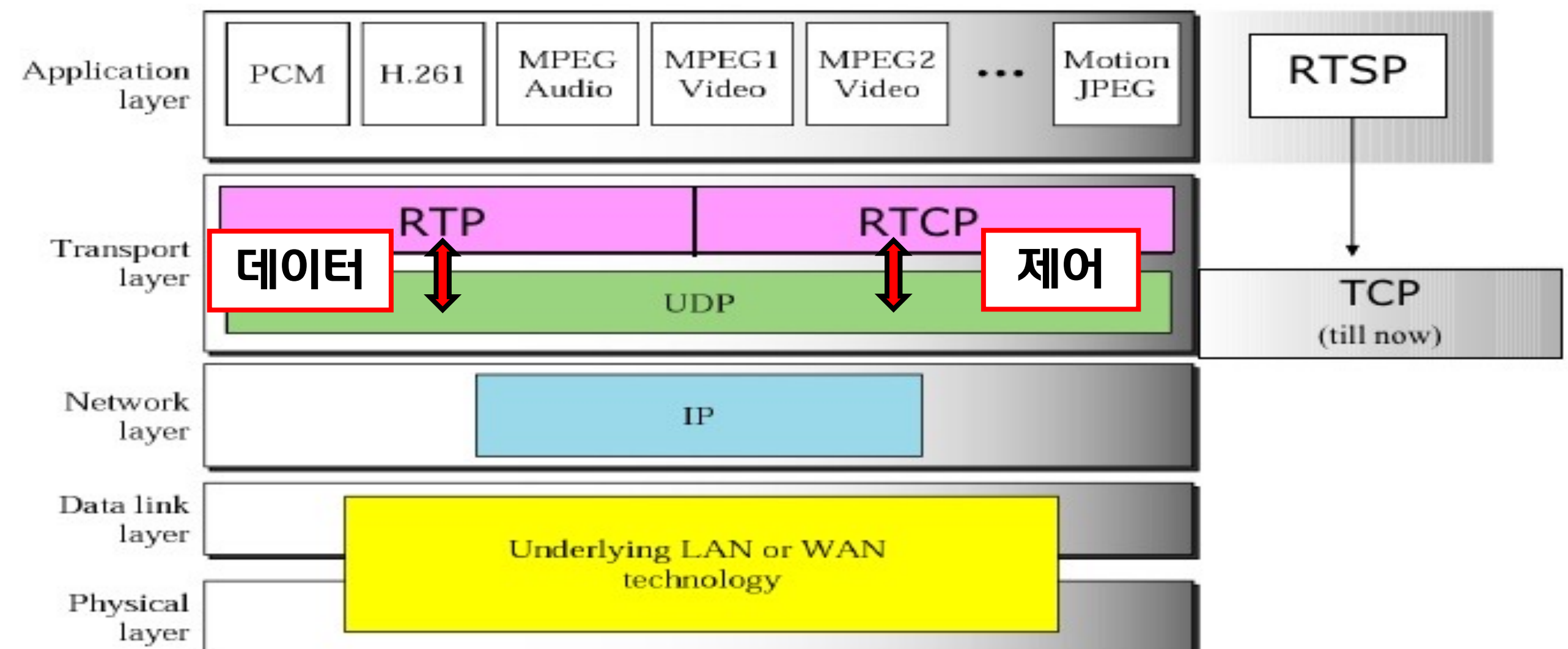
RTP 와 RTCP

RTP (Realtime Transport Protocol)

- 실시간 비디오, 오디오 및 데이터를 전송하는 데 사용하는 프로토콜 (짝수 포트 번호)

RTCP (Realtime Transport Control Protocol)

- RTP 세션의 네트워크 품질 제어를 위한 별도의 제어용 프로토콜 (RTP + 1 포트 번호)



Q. 왜 RTP, RTCP는 주로 UDP를 사용하고
RTSP는 TCP를 사용할까?

- Live Streaming 에서 많이 사용
- RTSP는 RTP의 세션 관리용 프로토콜 → 세션 관리를 위해 패킷 보장 (TCP)

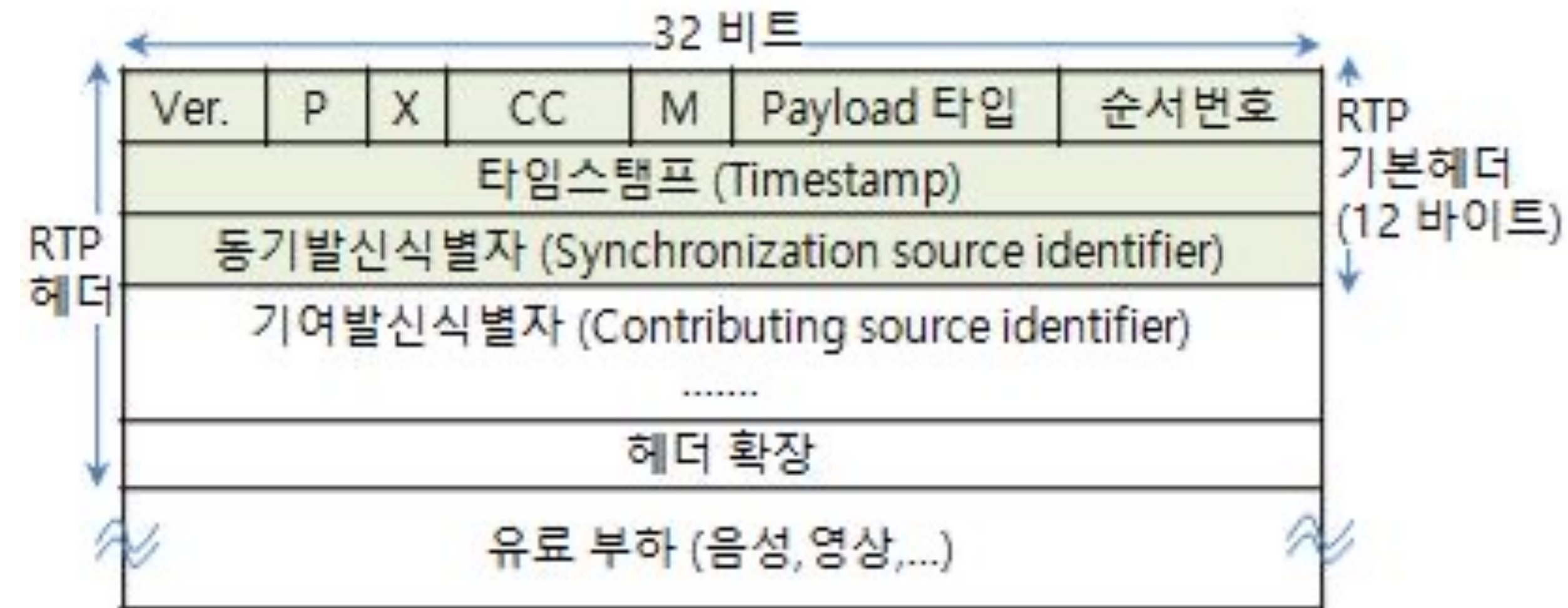
Q. RTP / RTCP는 왜 패킷이 보장되는 TCP를 놔두고 UDP를 사용하는가?

- Live Streaming는 실시간 전송
 - 라이브는 시간이 매우 중요, 시간 지연 X
- ➔ 패킷은 손실하더라도 지연은 용납 못해! UDP 사용
(간혹 RTP over TCP도 사용함)

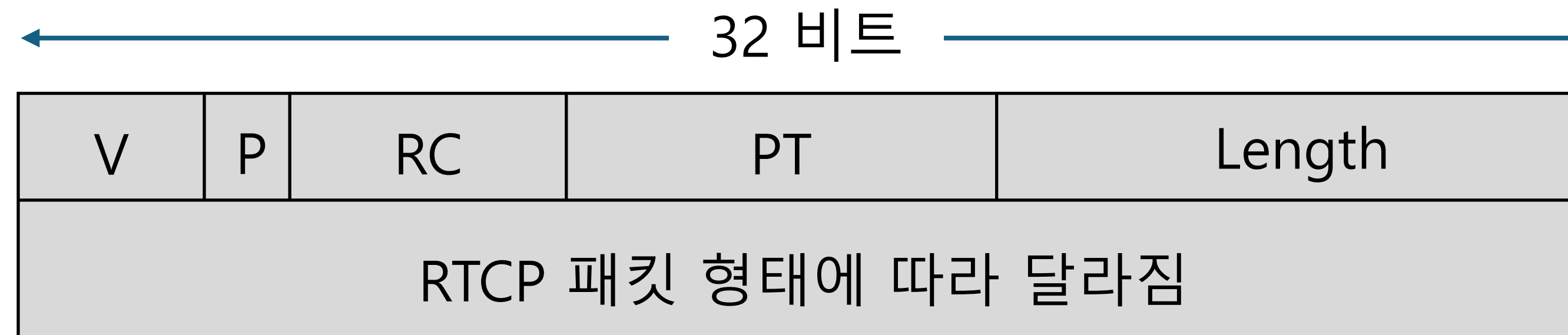
결론 : Live 관점에서는 RTP와 RTCP는
목적에 맞게 주로 UDP를 사용

RTP 와 RTCP 헤더 구조 비교

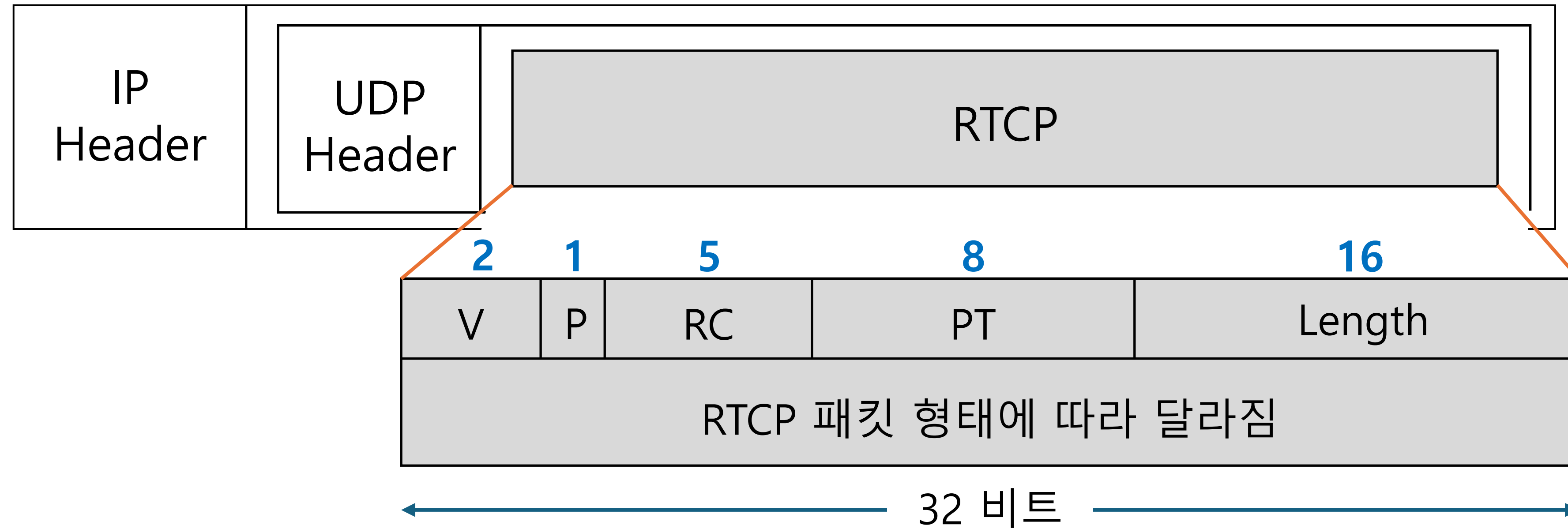
RTP



RTCP



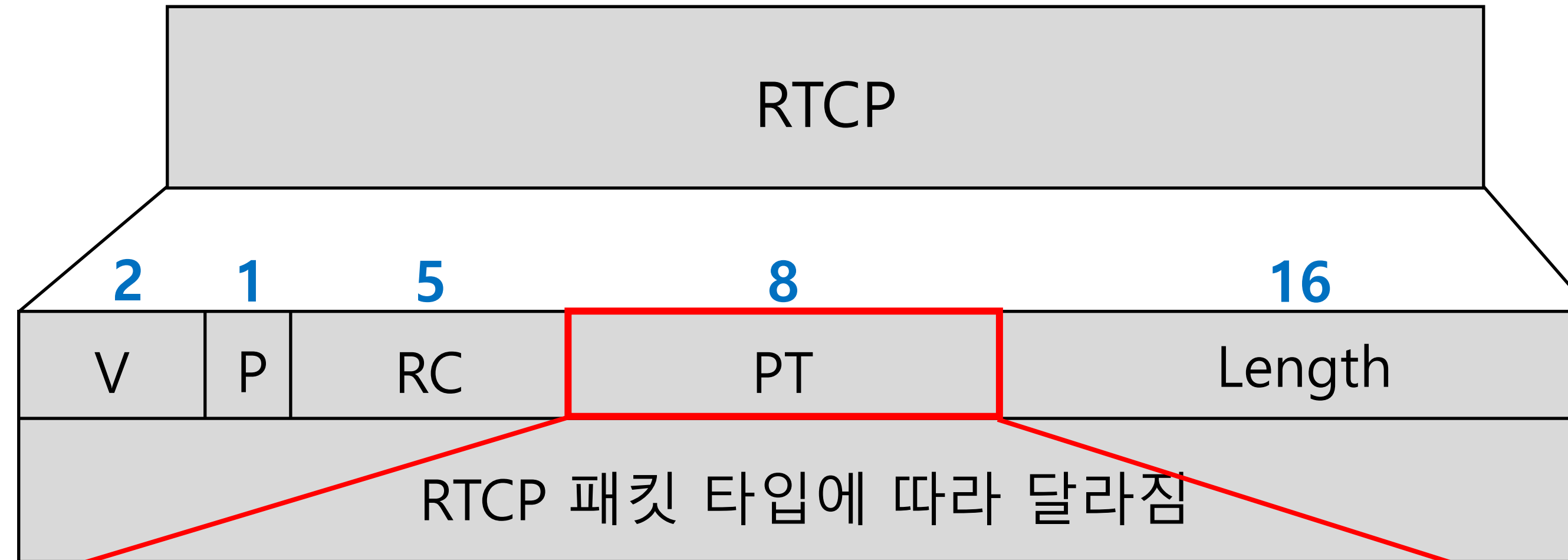
RTCP 패킷 구조



- **V(Version):** RTP의 버전(일반적으로 2로 설정)
- **P(Padding):** 패킷의 끝에 패딩이 있는지 여부를 표시, P=1이면 패딩 존재.
- **RC(Reception Report Count):** 패킷에 포함된 수신 보고서 블록 수
- **PT(Packet Type):** RTCP 패킷 타입
- **Length:** 패킷의 길이

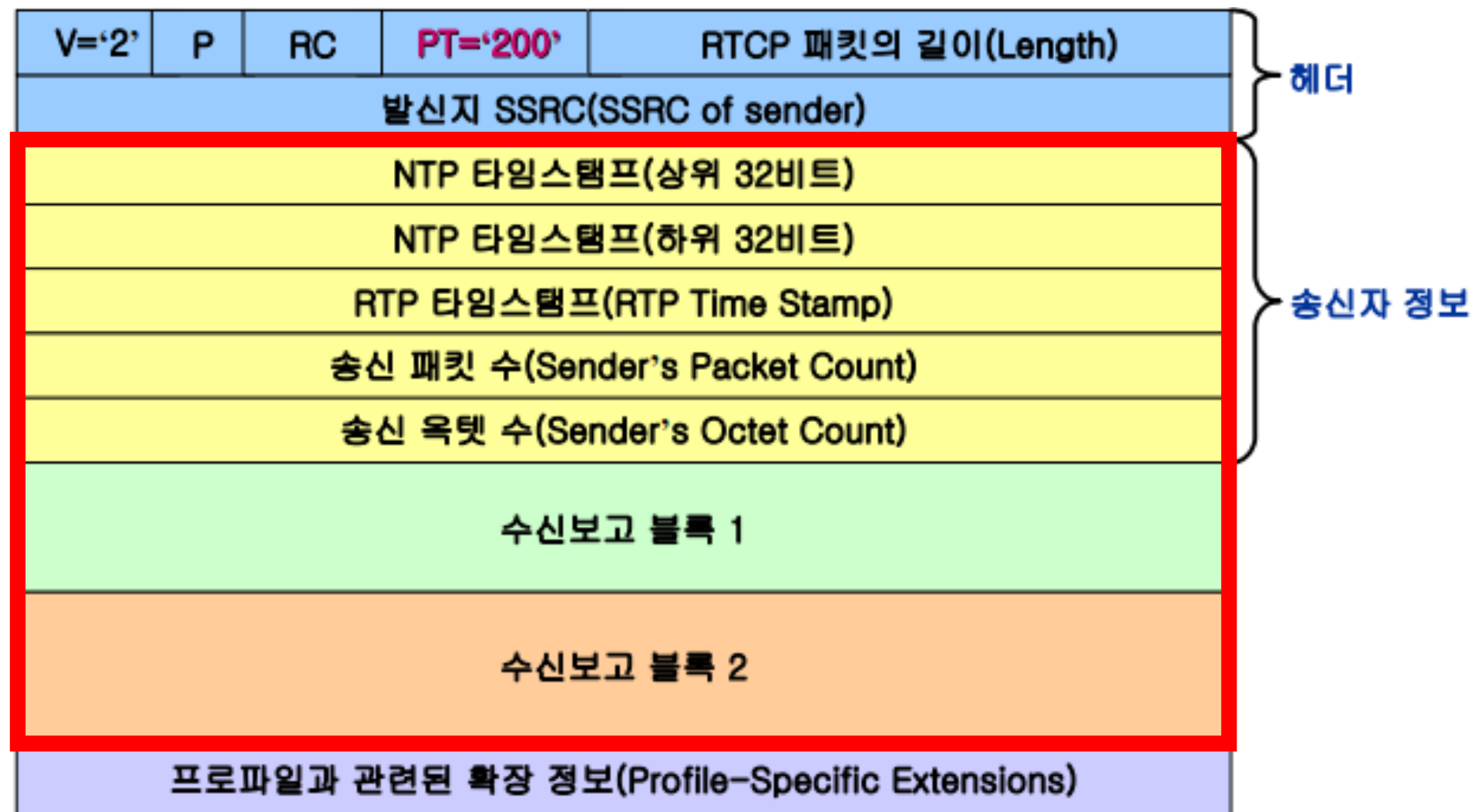
RTCP 패킷 타입(메세지) 종류

PT = 200 : SR
PT = 201 : RR
PT = 202 : SDES
PT = 203 : BYE
PT = 204 : APP

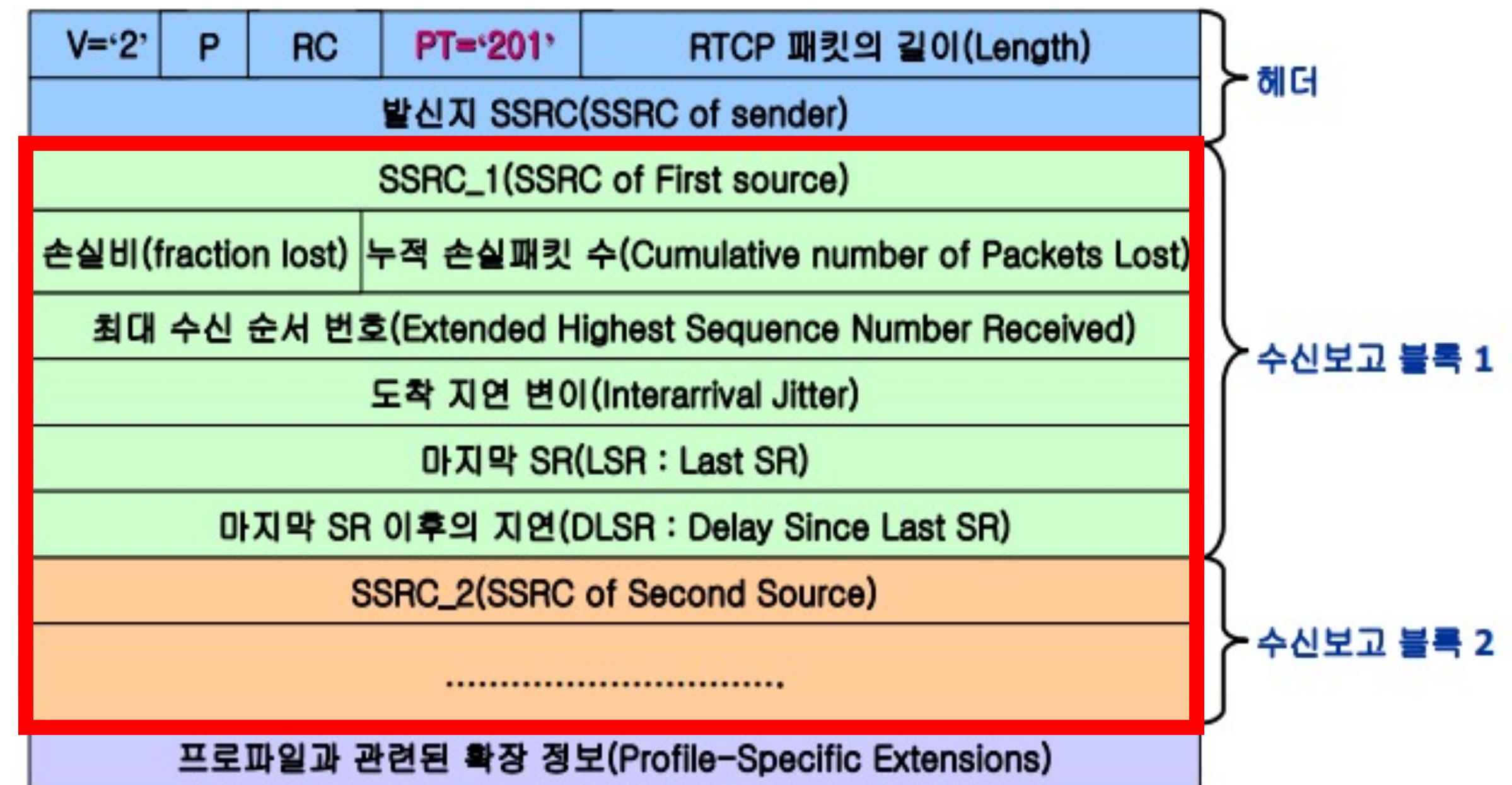


- **SR (Sender Report):** 송신자가 보내는 품질정보 통계 데이터(ex. 송신 패킷 수 등)
- **RR (Receiver Report):** 수신자가 보내는 품질정보 통계 데이터(ex. 패킷 손실률 등)
- **SDES (Source Description):** 소스 정보(사용자 이름, 이메일 등) 제공.
- **BYE:** 세션에서의 종료를 알림
- **APP:** 특정 어플리케이션에 맞게 사용자 맞춤형 데이터를 전송

패킷 타입(PT)에 따라 형식이 다르다?

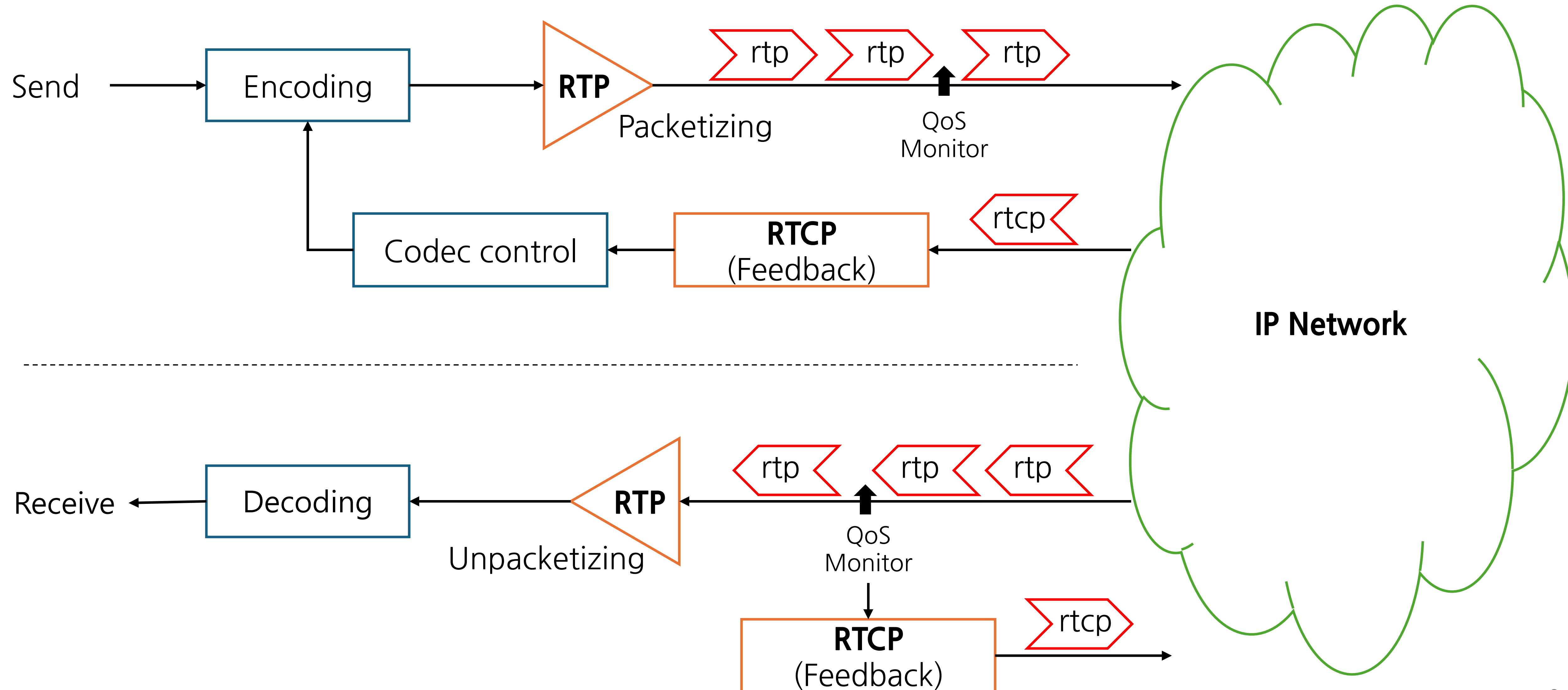


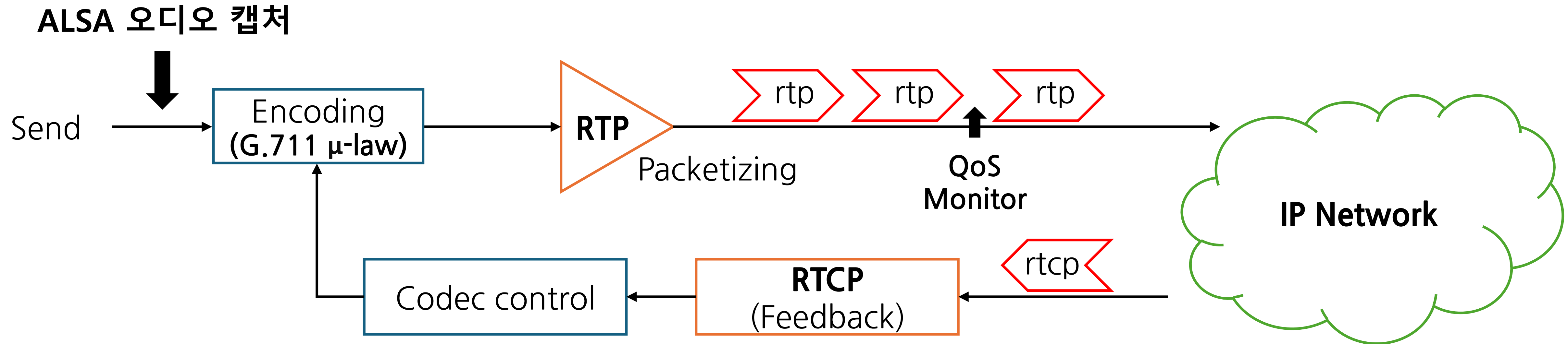
Sender Report(SR)



Receive Report(RR)

RTP and RTCP Operation Process





구성 요소

1. **ALSA(Advanced Linux Sound Architecture)**: 리눅스에서 오디오 캡처를 담당하는 라이브러리
2. **G.711 μ-law 인코딩**: 오디오 데이터를 압축하는 코덱
3. **RTP/RTCP**: RTP를 통해 오디오 데이터를 전송하고 RTCP로 SR(Sender Report) 전송

RTP 헤더 (단일 바이트 vs 구조체 비트 필드)

```
// RTP 헤더 생성 함수
void create_rtp_header(unsigned char *rtp_header, unsigned short seq_num, unsigned int timestamp, unsigned int ssrc) {
    rtp_header[0] = 0x80; // Version 2, no padding, no extension, no CSRC

    rtp_header[1] = 0x00; // Payload type 0 (G.711=PCMU), Marker bit 0

    rtp_header[2] = seq_num >> 8; // Sequence number (16 bits)
    rtp_header[3] = seq_num & 0xFF;

    rtp_header[4] = (timestamp >> 24) & 0xFF; // Timestamp (32 bits)
    rtp_header[5] = (timestamp >> 16) & 0xFF;
    rtp_header[6] = (timestamp >> 8) & 0xFF;
    rtp_header[7] = (timestamp >> 0) & 0xFF;

    rtp_header[8] = (ssrc >> 24) & 0xFF; // SSRC (32 bits)
    rtp_header[9] = (ssrc >> 16) & 0xFF;
    rtp_header[10] = (ssrc >> 8) & 0xFF;
    rtp_header[11] = (ssrc >> 0) & 0xFF;
}
```

// RTP 헤더 구조체 (비트 필드 사용)

```
struct rtp_header {
    uint16_t cc:4; // CSRC count (4 bits)
    uint16_t x:1; // Extension (1 bit)
    uint16_t p:1; // Padding (1 bit)
    uint16_t version:2; // Version (2 bits)
    uint16_t pt:7; // Payload type (7 bits)
    uint16_t m:1; // Marker (1 bit)
    uint16_t seq_num; // Sequence number (16 bits)
    uint32_t timestamp; // Timestamp (32 bits)
    uint32_t ssrc; // SSRC (32 bits)
};
```

```
// RTP 헤더 생성 함수
void create_rtp_header(struct rtp_header *header, unsigned short seq_num, unsigned int timestamp, unsigned int ssrc) {
    header->version = 2;
    header->p = 0;
    header->x = 0;
    header->cc = 0;
    header->m = 0;
    header->pt = 0; // 0 for PCMU (G.711 μ-law)
    header->seq_num = htons(seq_num);
    header->timestamp = htonl(timestamp);
    header->ssrc = htonl(ssrc);
}
```

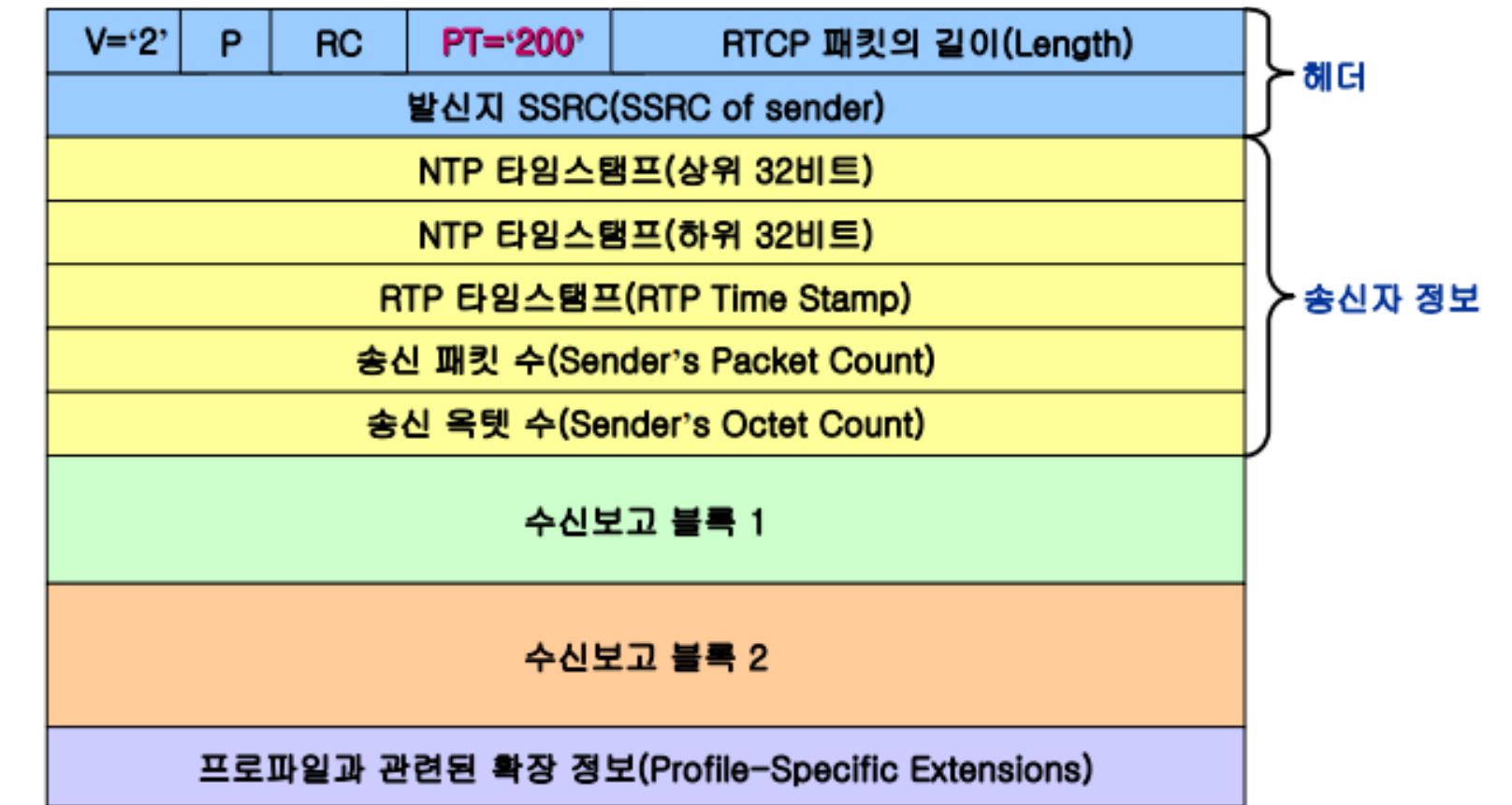

RTCP SR 패킷 구조 코드

```
#define DEST_IP "127.0.0.1" // 라즈베리파이의 IP 주소
#define RTP_PORT 5004 //
#define RTCP_PORT 5005
#define PAYLOAD_SIZE 160 // 20ms of G.711 audio at 8kHz
```

// RTCP SR (Sender Report) 구조

```
struct rtcp_sr {
#if __BYTE_ORDER == __LITTLE_ENDIAN
    uint8_t rc:5;
    uint8_t p:1;
    uint8_t version:2;
#elif __BYTE_ORDER == __BIG_ENDIAN
    uint8_t version:2;
    uint8_t p:1;
    uint8_t rc:5;
#else
#error "Please fix <bits/endian.h>"
#endif

    uint8_t pt;
    uint16_t length;
    uint32_t ssrc;
    uint32_t ntp_timestamp_msw;
    uint32_t ntp_timestamp_lsw;
    uint32_t rtp_timestamp;
    uint32_t sender_packet_count;
    uint32_t sender_octet_count;
} __attribute__((packed));
```



```
// RTCP SR(Sender Report) 생성 함수
void create_rtcp_sr(struct rtcp_sr *sr, uint32_t ssrc, uint32_t rtp_timestamp, uint32_t packet_count, uint32_t octet_count) {
    memset(sr, 0, sizeof(*sr));
    sr->version = 2;
    sr->p = 0;
    sr->rc = 0;
    sr->pt = 200; // 200 for Sender Report
    sr->length = htons(6); // 6 32-bit words
    sr->ssrc = htonl(ssrc);

    struct timespec now;
    clock_gettime(CLOCK_REALTIME, &now);
    uint64_t ntp_time = ((uint64_t)(now.tv_sec + 2208988800ULL) << 32) |
        ((uint64_t)now.tv_nsec * 0x100000000ULL / 1000000000ULL);

    sr->ntp_timestamp_msw = htonl((uint32_t)(ntp_time >> 32));
    sr->ntp_timestamp_lsw = htonl((uint32_t)(ntp_time & 0xFFFFFFFF));
    sr->rtp_timestamp = htonl(rtp_timestamp);
    sr->sender_packet_count = htonl(packet_count);
    sr->sender_octet_count = htonl(octet_count);
}
```

ALSA audio capture

ALSA 오디오 캡처



```

// ALSA 초기화
snd_pcm_t *pcm_handle;
snd_pcm_hw_params_t *params;
int dir;
unsigned int sample_rate = 8000; // G.711의 샘플링 레이트
int rc; // ALSA 오디오 파라미터 적용

// ALSA PCM 디바이스 열기
rc = snd_pcm_open(&pcm_handle, "default", SND_PCM_STREAM_CAPTURE, 0);
if (rc < 0) {
    fprintf(stderr, "Unable to open PCM device: %s\n", snd_strerror(rc));
    exit(1);
}

// ALSA PCM 하드웨어 파라미터 설정
snd_pcm_hw_params_alloca(&params); // 오디오 디바이스의 매개변수 설정을 위한 공간 확보
snd_pcm_hw_params_any(pcm_handle, params); // 기본값으로 초기화
snd_pcm_hw_params_set_access(pcm_handle, params, SND_PCM_ACCESS_RW_INTERLEAVED); // 인터리브드 모드로 설정
snd_pcm_hw_params_set_format(pcm_handle, params, SND_PCM_FORMAT_S16_LE); // 오디오 포맷 16비트로 설정
snd_pcm_hw_params_set_channels(pcm_handle, params, 1); // 모노로 설정
snd_pcm_hw_params_set_rate_near(pcm_handle, params, &sample_rate, &dir); // 샘플링 레이트 설정
rc = snd_pcm_hw_params(pcm_handle, params);
if (rc < 0) {
    fprintf(stderr, "Unable to set HW parameters: %s\n", snd_strerror(rc));
    exit(1);
}
  
```

```

// RTP 패킷 전송 루프
while (1) {
    // ALSA라이브러리를 통해 오디오 데이터 캡처
    rc = snd_pcm_readi(pcm_handle, buffer, frames);
    if (rc == -EPIPE) { // 오버런일 경우 처리
        fprintf(stderr, "Overrun occurred\n");
        snd_pcm_prepare(pcm_handle);
        continue;
    } else if (rc < 0) { // 에러가 발생했을 때의 처리
        fprintf(stderr, "Error from read: %s\n", snd_strerror(rc));
        continue;
    } else if (rc != frames) { // 요청한 것보다 적은 프레임을 읽었을 때
        fprintf(stderr, "Short read, read %d frames\n", rc);
    }
}
  
```

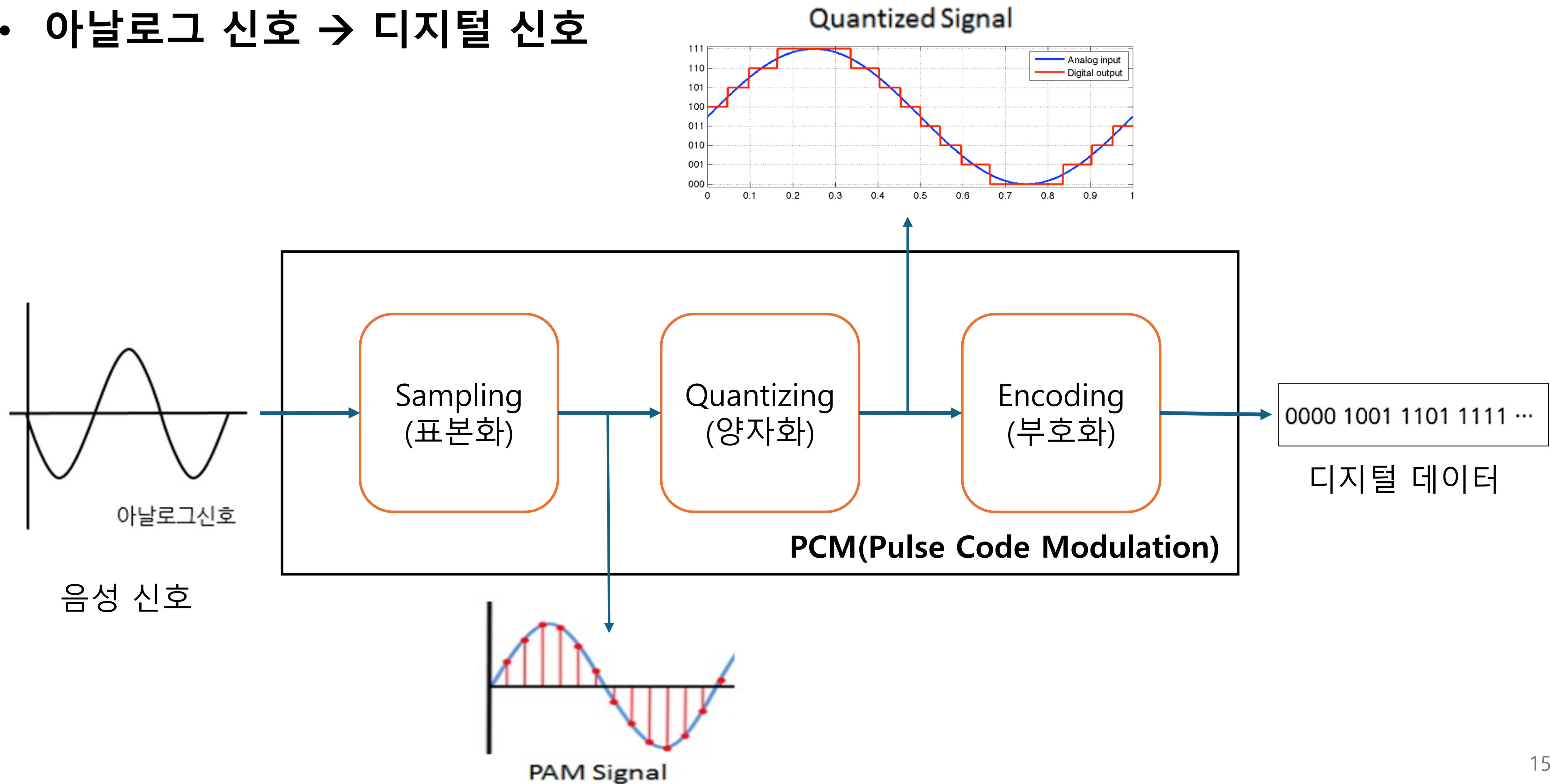

- **G.711은, PSTN 망(전화망)에 적용되는 가장 기초적인 방식**
 - PCM (Pulse Code Modulation, 펄스 부호 변조) 방식이라고도 함

주요 특징

- **대역폭** : 300Hz ~ 3400Hz 대역의 음성 대역 신호
- **샘플링 주파수** : 8kHz
 - 이유 : 주로 **전화 통화**에서 사용 (사람 목소리는 **300~3400Hz**에서 재현하기에 충분)
- **양자화 비트 수** : 각 표본을 8bit로 부호화
- **양자화 방식** : 비선형 양자화 → μ (mu)-law, A-law
- **전송 비트율** : $8000(\text{sample/s}) \times 8(\text{bit}) = 64 \text{ kbps}$

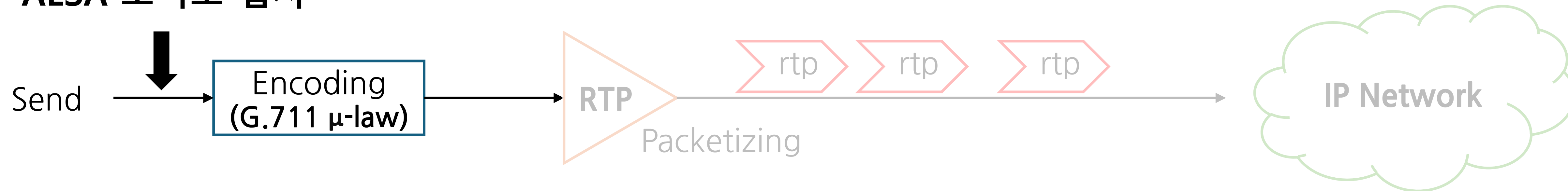
PCM(Pulse Code Modulation)

- 아날로그 신호 → 디지털 신호

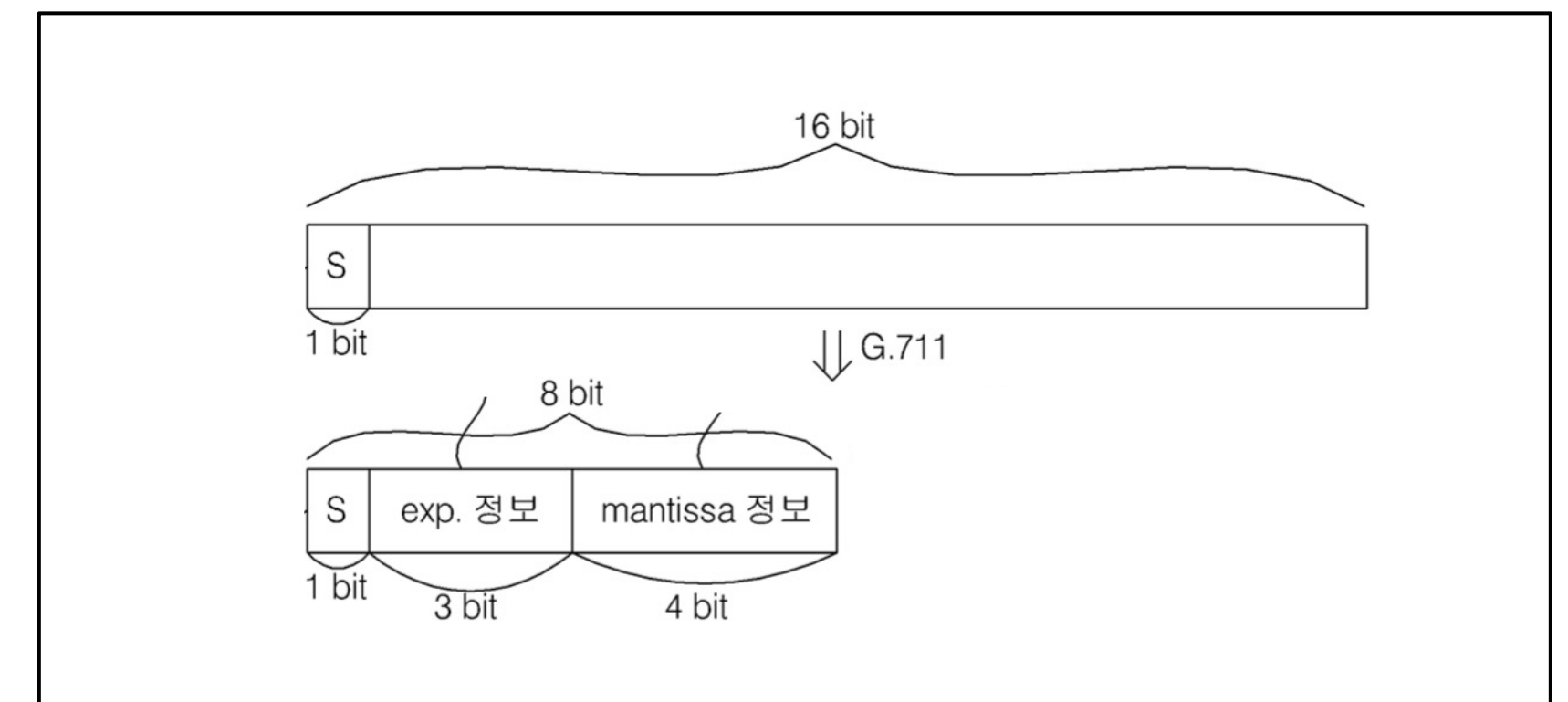


G.711 (μ -law) Encoding

ALSA 오디오 캡처



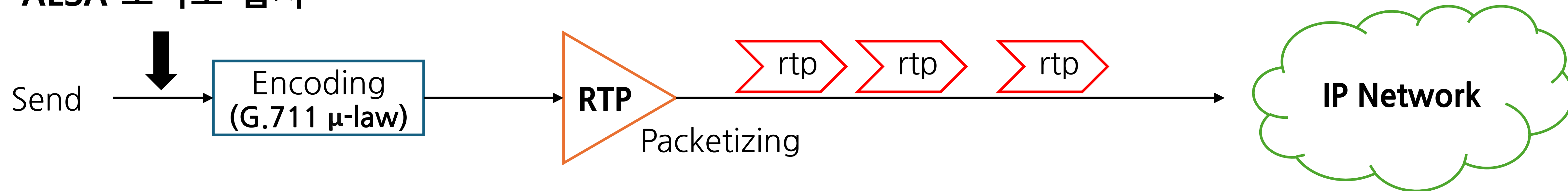
```
// G.711  $\mu$ -law 인코딩 함수
unsigned char g711_ulaw(int sample) {
    const int MAX = 32767; // PCM 오디오 샘플의 최대 값
    const int BIAS = 0x84; // 인코딩에서 사용하는 바이어스 값, 압축시 왜곡을 줄이기 위해 사용
    int sign = (sample >> 8) & 0x80; // 샘플의 상위 8비트 추출 (부호 확인)
    if (sign != 0) sample = -sample; // 음수일 경우 양수로 변환
    if (sample > MAX) sample = MAX; // 최대값 제한
    sample += BIAS; // 바이어스 추가: 작은 신호 왜곡을 줄이기 위함
    int exponent = 7; // 지수(exponent) 계산
    int mask;
    for (; exponent > 0; exponent--) {
        mask = 1 << (exponent + 3);
        if (sample >= mask) break;
    }
    int mantissa = (sample >> (exponent + 3)) & 0x0F; // 가수 계산 (하위 4비트)
    return ~(sign | (exponent << 4) | mantissa); // 부호, 지수, 가수를 결합하여 최종 인코딩 값 생성
}
```



16비트 신호를 → 8비트로 압축

RTP Packetizing

ALSA 오디오 캡처



```
// 캡처한 데이터를 G.711 μ-law으로 인코딩
for (int i = 0; i < PAYLOAD_SIZE; i++) {
    payload[i] = g711_ulaw(buffer[i]);
}

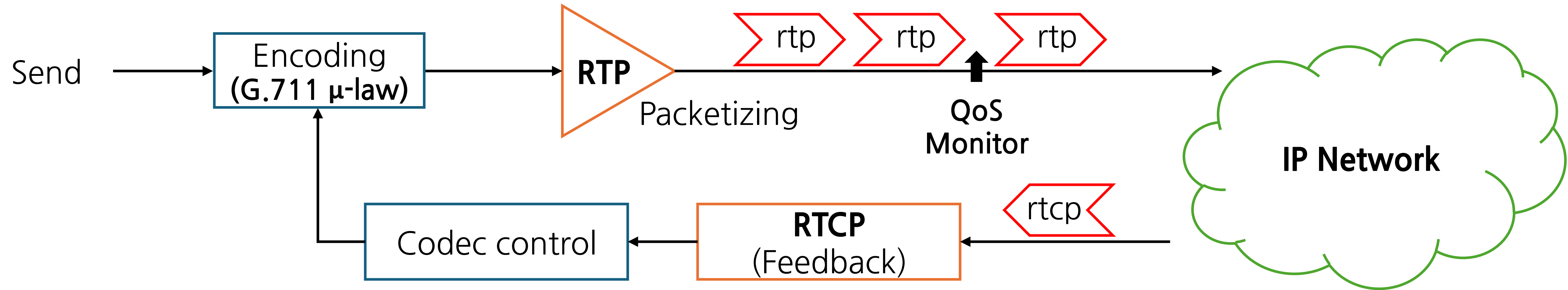
// RTP 헤더 생성
create_rtp_header(&rtp_hdr, seq_num, timestamp, ssrc);

// RTP 패킷화(Packetizing)
memcpy(rtp_packet, &rtp_hdr, sizeof(struct rtp_header));
memcpy(rtp_packet + sizeof(struct rtp_header), payload, PAYLOAD_SIZE);

// RTP 패킷 전송
if (sendto(rtp_sockfd, rtp_packet, sizeof(rtp_packet), 0, (struct sockaddr *)&rtp_dest_addr, sizeof(rtp_dest_addr)) < 0) {
    perror("RTP sendto() error");
    exit(1);
}

printf("RTP packet %d sent\n", ntohs(rtp_hdr.seq_num));
```

RTCP SR(Sender Report) Feedback



```
// 50패킷(1초)마다 RTCP SR 전송
if (packet_count % 50 == 0) {
    create_rtcp_sr(&rtcp_sr, ssrc, timestamp, packet_count, octet_count);
    if (sendto(rtcp_sockfd, &rtcp_sr, sizeof(rtcp_sr), 0, (struct sockaddr *)&rtcp_dest_addr, sizeof(rtcp_dest_addr)) < 0) {
        perror("RTCP SR sendto() error");
        exit(1);
    }
    printf("RTCP SR sent\n");
}

usleep(20000); // 20ms 지연 (50 패킷/초)
}
```


결과 (with Wireshark)

The image displays the Wireshark network traffic analysis interface. The main window shows a list of captured packets on the left, with packet 766 selected. The packet details pane on the right shows the structure of the selected RTCP packet (Sender Report). The packet bytes pane at the bottom shows the raw data of the packet. To the right of the Wireshark window, there are two terminal windows. The top terminal window shows a list of RTP packets being sent. The bottom terminal window shows the command to run FFmpeg to receive RTP packets.

Wireshark Packet List:

No.	Time	Source	Destination	Protocol	Length	Info
745	18.796611601	127.0.0.1	127.0.0.1	RTP	214	PT=ITU-T G.711 PCMU, SSRC=0x3039, Seq=6217, Time=116640
746	18.816748056	127.0.0.1	127.0.0.1	RTP	214	PT=ITU-T G.711 PCMU, SSRC=0x3039, Seq=6218, Time=116800
747	18.836875623	127.0.0.1	127.0.0.1	RTP	214	PT=ITU-T G.711 PCMU, SSRC=0x3039, Seq=6219, Time=116960
748	18.857013171	127.0.0.1	127.0.0.1	RTP	214	PT=ITU-T G.711 PCMU, SSRC=0x3039, Seq=6220, Time=117120
749	18.877130348	127.0.0.1	127.0.0.1	RTP	214	PT=ITU-T G.711 PCMU, SSRC=0x3039, Seq=6221, Time=117280
750	18.897234526	127.0.0.1	127.0.0.1	RTP	214	PT=ITU-T G.711 PCMU, SSRC=0x3039, Seq=6222, Time=117440
751	18.917363870	127.0.0.1	127.0.0.1	RTP	214	PT=ITU-T G.711 PCMU, SSRC=0x3039, Seq=6223, Time=117600
752	18.937772475	127.0.0.1	127.0.0.1	RTP	214	PT=ITU-T G.711 PCMU, SSRC=0x3039, Seq=6224, Time=117760
753	18.958134968	127.0.0.1	127.0.0.1	RTP	214	PT=ITU-T G.711 PCMU, SSRC=0x3039, Seq=6225, Time=117920
754	18.978591276	127.0.0.1	127.0.0.1	RTP	214	PT=ITU-T G.711 PCMU, SSRC=0x3039, Seq=6226, Time=118080
755	18.998757139	127.0.0.1	127.0.0.1	RTP	214	PT=ITU-T G.711 PCMU, SSRC=0x3039, Seq=6227, Time=118240
756	19.019184855	127.0.0.1	127.0.0.1	RTP	214	PT=ITU-T G.711 PCMU, SSRC=0x3039, Seq=6228, Time=118400
757	19.039802127	127.0.0.1	127.0.0.1	RTP	214	PT=ITU-T G.711 PCMU, SSRC=0x3039, Seq=6229, Time=118560
758	19.059964750	127.0.0.1	127.0.0.1	RTP	214	PT=ITU-T G.711 PCMU, SSRC=0x3039, Seq=6230, Time=118720
759	19.080327113	127.0.0.1	127.0.0.1	RTP	214	PT=ITU-T G.711 PCMU, SSRC=0x3039, Seq=6231, Time=118880
760	19.100505310	127.0.0.1	127.0.0.1	RTP	214	PT=ITU-T G.711 PCMU, SSRC=0x3039, Seq=6232, Time=119040
761	19.120996655	127.0.0.1	127.0.0.1	RTP	214	PT=ITU-T G.711 PCMU, SSRC=0x3039, Seq=6233, Time=119200
762	19.141200315	127.0.0.1	127.0.0.1	RTP	214	PT=ITU-T G.711 PCMU, SSRC=0x3039, Seq=6234, Time=119360
763	19.161622290	127.0.0.1	127.0.0.1	RTP	214	PT=ITU-T G.711 PCMU, SSRC=0x3039, Seq=6235, Time=119520
764	19.181752875	127.0.0.1	127.0.0.1	RTP	214	PT=ITU-T G.711 PCMU, SSRC=0x3039, Seq=6236, Time=119680
765	19.202031423	127.0.0.1	127.0.0.1	RTP	214	PT=ITU-T G.711 PCMU, SSRC=0x3039, Seq=6237, Time=119840
766	19.202060905	127.0.0.1	127.0.0.1	RTCP	70	Sender Report
767	19.222277546	127.0.0.1	127.0.0.1	RTP	214	PT=ITU-T G.711 PCMU, SSRC=0x3039, Seq=6238, Time=120000

Wireshark Packet Details (Selected Packet 766):

- Destination Port: 5005
- Length: 36
- Checksum: 0xfe37 [unverified]
- [Checksum Status: Unverified]
- [Stream index: 2]
- [Timestamps]
- UDP payload (28 bytes)
- Real-time Transport Control Protocol (Sender Report)
 - 10.. = Version: RFC 1889 Version (2)
 - ..0. = Padding: False
 - ...0 0000 = Reception report count: 0
 - Packet type: Sender Report (200)
 - Length: 6 (28 bytes)
 - Sender SSRC: 0x00003039 (12345)
 - Timestamp, MSW: 3938034009 (0xeab9a159)
 - Timestamp, LSW: 3078132567 (0xb7789357)
 - [MSW and LSW as NTP timestamp: Oct 16, 2024 02:20:09.716683586 UTC]
 - RTP timestamp: 120000
 - Sender's packet count: 750
 - Sender's octet count: 120000
 - [RTCP frame length check: OK - 28 bytes]

Terminal 1 (Top):

```
RTP packet 6222 sent
RTP packet 6223 sent
RTP packet 6224 sent
RTP packet 6225 sent
RTP packet 6226 sent
RTP packet 6227 sent
RTP packet 6228 sent
RTP packet 6229 sent
RTP packet 6230 sent
RTP packet 6231 sent
RTP packet 6232 sent
RTP packet 6233 sent
RTP packet 6234 sent
RTP packet 6235 sent
RTP packet 6236 sent
RTP packet 6237 sent
RTCP SR sent
RTP packet 6238 sent
RTP packet 6239 sent
RTP packet 6240 sent
RTP packet 6241 sent
RTP packet 6242 sent
RTP packet 6243 sent
```

Terminal 2 (Bottom):

```
taewonkim@raspberrypi:~$ ffmpeg -i rtp://127.0.0.1:5004 -f alsa default
ffmpeg version 5.1.6-0+deb12u1+rpt1 Copyright (c) 2000-2024 the FFmpeg d
evelopers
built with gcc 12 (Debian 12.2.0-14)
configuration: --prefix=/usr --extra-version=0+deb12u1+rpt1 --toolchai
n=hardened --incdir=/usr/include/aarch64-linux-gnu --enable-gpl --disabl
e-stripping --disable-mmal --enable-gnutls --enable-ladspa --enable-liba
om --enable-libass --enable-libbluray --enable-libbs2b --enable-libcaca
--enable-libcdio --enable-libcodec2 --enable-libdav1d --enable-libflite
--enable-libfontconfig --enable-libfreetype --enable-libfribidi --enable
-libglslang --enable-libgme --enable-libgsm --enable-libjack --enable-li
bmp3lame --enable-libmysofa --enable-libopenjpeg --enable-libopenmpt --e
nable-libopus --enable-libpulse --enable-librabbitmq --enable-librist --
enable-librubberband --enable-libshine --enable-libsnappy --enable-libso
xr --enable-lspspeex --enable-lsrt --enable-libssh --enable-lsvtav1
--enable-libtheora --enable-libtwolame --enable-libvidstab --enable-libv
orbis --enable-libvpx --enable-libwebp --enable-libx265 --enable-libxml2
```


노이즈가 많이 끼는 이유?

1. 좋지 않은 마이크 ?

2. 패킷 손실 ?

3. send에 맞는 receive ? Jitter buffer 처리

- 지터(Jitter) 현상: 데이터 패킷이 도착하는 시간이 불규칙해지는 현상

(목소리가 끊기거나 **영상 통화** 중에 화면이 멈춤)

- 해결 방법: 일정 시간동안 도착한 음성 패킷을 "**인코딩 시간 순서대로 정렬**" 하여 버퍼에 저장한 후 재생
- 패킷 손실과 지연 문제를 완화하기 위해 일시적으로 데이터를 저장

1. **RTP** : 실시간 비디오, 오디오 데이터를 전송하는 프로토콜, UDP 사용

2. **RTCP** : RTP의 품질을 제어하며, UDP 사용 (RTP와 한 쌍)

3. **RTCP의 패킷 구조와 패킷 타입**

4. **PCM** : 표본화 → 양자화 → 부호화

