Assignment 3: udp2sip

udp2sip

- Receives UDP packets from a specified transport address (command line)
- Works for unicast and multicast addresses (IPv4, optionally IPv6)
- Virtually “any number” of addresses (typically 1 or 2)
- Option: Short and long form for dumping data packets to stdout or file
  - Short output: include SSRC, RTP payload type, timestamp, seq-no, and M bit
- Terminating the program with Ctrl-C (SIGINT) will cause it to dump a summary of the packets received so far.
  - Count missing packets from gaps in RTP sequence number space
- Accept RTSP-based control connection and forward the data packets to a media client
- Accept a SIP-based TCP control connection and forward the data packets to a SIP soft client (or SIP phone)
- Support most trivial SIP interactions for a single run
  - Incoming: INVITE, ACK, BYE (possibly OPTIONS)
  - Generate: 200 OK or appropriate error code
  - Negotiate codecs and transport address using SDP in a single offer/answer exchange
- Terminate when the dialog terminates
- Do NOT REGISTER; just accept SIP calls to your IP address for any user

`udp2rtsp -a <addr-spec-audio> -v <addr-spec-video> -i <if-addr> -s -l <dumplen> -f <output-file> -r <addr-spec> -S <sip-addr>`

-a, -v: transport address to receive data on; uses the following format

<IPv4 address>/rtp-port[-rtcp-port][/pt]

/port[-rtcp-port][/pt]

<IPv6 address>/port[-rtcp-port][/pt]

<hostname>/port[rtcp-port][/pt]

May each be specified only once.

-i: address of the local interface to use for listening to multicast packets

-s: packet reports in short form: one line per packet:

reception timestamp (μs!), sender, receiver address, packet size

If "-s" is not specified, the long form is implied. In this case, the above line is followed by a hexdump of (parts of) each packet received:

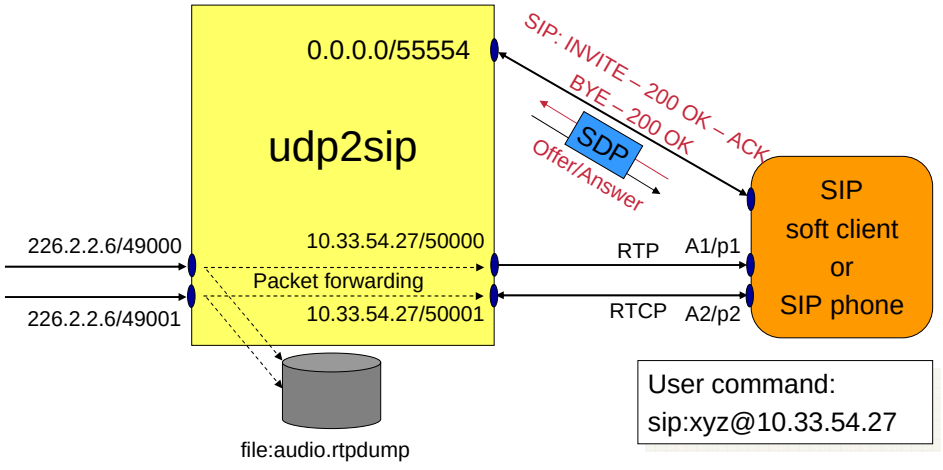
000000 xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx .....

-l: Number of bytes to include in the hexdump

-r, -S: transport address to accept RTSP or SIP connections, respectively

-f: name of the output file to dump to ("-" -> stdout); if not given, be silent

udp2sip -a 226.2.2.6/49000-49001/0 -i 10.33.54.27 -s -f audio.rtpdump -S /55554



Media Stream Control

- SIP is used to
  - Invoke and terminate the media replay
  - Negotiate media stream parameters and addresses (carry offer/answer)
- SDP is used to describe the media stream
  - Carried in RTSP 200 OK message in response to DESCRIBE

Sample Offer from SIP UA

```
v=0\r\n
s=-\r\n
o=jo 1234.. 1234.. IN IP4 <A1>\r\n
t=0 0\r\n
c=IN IP4 <A1>\r\n
m=audio <p1> RTP/AVP 0 8 3 4 8 18\r\n
```

Sample Answer from udp2sip

```
v=0\r\n
s=-\r\n
o=jo 2345.. 2345.. IN IP4 10.33.54.27\r\n
t=0 0\r\n
c=IN IP4 10.33.54.27\r\n
m=audio 50000 RTP/AVP 0\r\n
```


What you need to do…

- Add SIP-specific handling to your process
  - Re-use the message parsing where possible
  - Just do the SIP processing elsewhere (minimal state machine)
  - Add small SDP parser
  - Add offer/answer logic
  - Feed the negotiation result into your packet forwarding engine
  - Simplifying assumptions allowed for the state machine: there will be only one call
- SIP handling
  - 100 Trying, 200 OK, 4xx responses as appropriate
  - Remember Via: header processing, branch parameter, and To: tag
  - Minimal number of headers (as required by RFC 3261) should do
  - Beyond that: ignore what you don't understand (client must live with it)
- Again: some experimentation may be required to get it right
  - Need to use a SIP client that speaks TCP (some don't)

Approach: Similar to RTSP

- Add an empty container function for SIP handling
- Make your message parser usable for SIP
- Write SIP code separately
  - Test this stand-alone
  - To get the SIP interaction right you don't need media streams
- Finally integrate and test
- Test media streams:
  will try provide copyright-free streams or tools to create them