7.1-SRS4.0 ICE交互分析

零声学院：音视频高级课程：https://ke.qq.com/course/468797?tuin=137bb271

这一节开始WebRTC协议的详细解析。

WebRTC涉及的协议非常多，我们要将文档和源码结合去分析协议的作用和实现。

涉及的文档：

- **Session Traversal Utilities for NAT (STUN)** https://datatracker.ietf.org/doc/html/rfc5389
- **Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols**
  https://datatracker.ietf.org/doc/html/rfc5245

- **微软文档** https://docs.microsoft.com/en-us/openspecs/office_protocols/ms-ice2/0879ffca-3284-4a59-b685-6bcc782eb38a

1 ICE相关

- ICE交互的目的
- 和sdp的关联
- stun request binding和response binding的作用

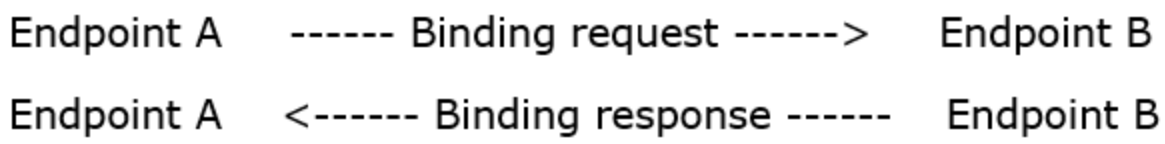
2 ICE状态

1. **Waiting**：还未开始连通性检查，从checklist中选择合适优先级的pair进行检查
2. **In-Progress**：连通性检查已经开始，但还未结束
3. **Succeeded**：该pair 连通性检查已经完成并且成功
4. **Failed**：失败
5. **Frozen**：连通性检查还未开始

3 SFU的ICE主要是发送连通性检查请求

客户端每隔2~3秒向服务器发送一次binding request，检测连通性。

ICE 使用STUN binding request/response，包含Fingerprint检验校验机制。



- 如果A收到B的response，则代表连通性检查成功，否则需要进行重传直到超时。
- 在建立连接时，如果没有响应，则会以RTO时间进行重传，每次翻倍，直到最大重传次数。
  - STUN请求 采用STUN short-term credential方式认证，
  - STUN USERNAME属性 ”RemoteUsername：localUsername”
  - 两端在SDP协商时交换ice-pwd和ice-ufrag，以得对端用户名和密码。
  - STUN 检查请求中需要检查地址的对称性，请求的源地址是响应的目的地址，请求的目的地址是响应的源地址，否则都设置状态为 Failed。

所有的ICE实现都要求与STUN(RFC5389)兼容，并且废弃Classic STUN(RFC3489)。ICE的完整实现既生成checks(作为STUN client)，也接收checks(作为STUN server)，而lite实现则只负责接收checks。这里只介绍完整实现情况下的检查过程.

1. 为中继候选地址生成许可(Permissions).

2. 从本地候选往远端候选发送Binding Request.

在Binding请求中通常需要包含一些特殊的属性,以在ICE进行连接性检查的时候提供必要信息.

- PRIORITY 和 USE-CANDIDATE
  - 终端必须在其request中包含PRIORITY属性,指明其优先级,优先级由公式计算而得. 如果有需要也可以给出特别指定的候选(即USE-CANDIDATE属性).
- ICE-CONTROLLED和ICE-CONTROLLING
  - 在每次会话中，每个终端都有一个身份。有两种身份：即受控方(controlled role)和主控方(controlling role)。主控方负责选择最终用来通讯的候选地址对，受控方被告知哪个候选地址对用来进行哪次媒体流传输, 并且不生成更新过的offer来提示此次告知。发起ICE处理进程(即生成offer)的一方必须是主控方，而另一方则是受控方。 如果终端是受控方，那么在request中就必须加上ICE-CONTROLLED属性。同样，如果终端是主控方，就需要ICE-CONTROLLING属性.
- 生成Credential
  - 作为连接性检查的Binding Request必须使用STUN的短期身份验证.验证的用户名被格式化为一系列username段的联结,包含了发送请求的所有对等端的用户名，以冒号隔开；密码就是对等端的密码.

3. 处理Response.

当收到Binding Response时,终端会将其与Binding Request相联系,通常通过事务ID.随后将会将此事务ID与 候选地址对进行绑定.

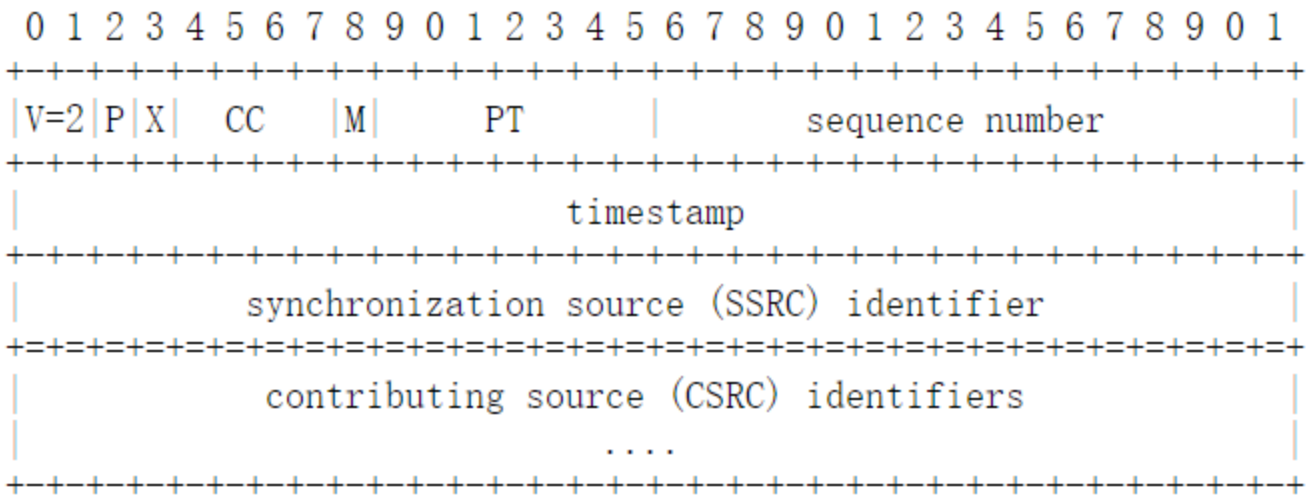
- 失败响应
  - 如果STUN传输返回487(Role Conflict)错误响应,终端首先会检查其是否包含了ICE-CONTROLLED或ICE-CONTROLLING 属性.如果有ICE-CONTROLLED,终端必须切换为controlling role;如果请求包含ICE-CONTROLLING属性, 则必须切换为controlled role。切换好之后,终端必须使产生487错误的候选地址对进入检查队列中, 并将此地址对的状态设置为Waiting.
- 成功响应,一次连接检查在满足下列所有情况时候就被认为成功:
  - STUN传输产生一个Success Response
  - response的源IP和端口等于Binding Request的目的IP和端口
  - response的目的IP和端口等于Binding Request的源IP和端口

终端收到成功响应之后，先检查其mapped address是否与本地记录的地址对有匹配，如果没有则生成一个新的候选地址。 即对等端的反射地址.如果有匹配,则终端会构造一个可用候选地址对(valid pair)。通常很可能地址对不存在于任何检查列表中，检索检查列表中没有被服务器反射的本地地址，这些地址把它们的本地候选转换成服务器反射地址的基地址，并把冗余的地址去除掉。

4 服务器UDP包处理入口

入口SrsRtcServer::on_udp_packet(SrsUdpMuxSocket* skt)

1. 判断是不是is_rtp_or_rtcp，通过(len >= 12 && (data[0] & 0xC0) == 0x80)
   For RTP or RTCP, the V=2 which is in the high 2bits, 0xC0 (1100 0000)



2. 判断是不是is_rtcp ， (len >= 12) && (data[0] & 0x80) && (data[1] >= 192 && data[1] <= 223);
   根据PT值判断是不是rtcp包。

3. 如果不是is_rtp_or_rtcp，判断是不是stun包，srs_is_stun， size > 0 && (data[0] == 0 || data[0] == 1)

For STUN packet, 0x00 is binding request, 0x01 is binding success response.

调用SrsStunPacket::decode(const char* buf, const int nb_buf) 解析解析USERNAME USE-CANDIDATE ICE-CONTROLLED ICE-CONTROLLING

4. 继续解析 stun， return session->on_stun(skt, &ping)

STUN解码SrsStunPacket::decode(const char* buf, const int nb_buf)

```
C++ | 复制代码
```

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|0 0|     STUN Message Type     |         Message Length        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Magic Cookie                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                     Transaction ID (96 bits)                  |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

Figure 2: Format of STUN Message Header
```

1. 解析消息类型（14bit）： message_type = stream->read_2bytes();
2. 解析消息长度（2字节）： uint16_t message_len = stream->read_2bytes();
3. 解析Magic Cookie （4字节）： string magic_cookie = stream->read_string(4);
4. 解析Transaction ID（12字节）： transcation_id = stream->read_string(12);

STUN Message Header 总共占用20字节，

接下来分析属性

Comprehension-required range (0x0000-0x7FFF):

0x0000: (Reserved)

0x0001: MAPPED-ADDRESS

0x0002: (Reserved; was RESPONSE-ADDRESS)

0x0003: (Reserved; was CHANGE-ADDRESS)

0x0004: (Reserved; was SOURCE-ADDRESS)

0x0005: (Reserved; was CHANGED-ADDRESS)

0x0006: USERNAME （分析local_ufrag和remote_ufrag）

0x0007: (Reserved; was PASSWORD)

0x0008: MESSAGE-INTEGRITY

0x0009: ERROR-CODE

0x000A: UNKNOWN-ATTRIBUTES

0x000B: (Reserved; was REFLECTED-FROM)

0x0014: REALM

0x0015: NONCE

0x0020: XOR-MAPPED-ADDRESS

Comprehension-optional range (0x8000-0xFFFF)

0x8022: SOFTWARE

0x8023: ALTERNATE-SERVER

0x8028: FINGERPRINT

This section registers four new STUN attributes per the procedures in [RFC5389].

0x0024 PRIORITY

0x0025 USE-CANDIDATE

0x8029 ICE-CONTROLLED lite agent MUST take the controlled role

0x802A ICE-CONTROLLING full agent MUST take the controlling

- SrsRtcConnection::on_stun(SrsUdpMuxSocket* skt, SrsStunPacket* r)
- SrsRtcConnection::on_dtls(char* data, int nb_data)
- SrsRtcConnection::on_rtcp(char* data, int nb_data)
- SrsRtcConnection::on_rtp(char* data, int nb_data)
- SrsRtcConnection::on_connection_established()
- SrsRtcConnection::on_rtcp_feedback_twcc(char* data, int nb_data)
- SrsRtcConnection::on_rtcp_feedback_remb(SrsRtcpPsfbCommon *rtcp)

STUN编码SrsStunPacket::encode(const string& pwd, SrsBuffer* stream)

5 SDP的ICE信息

`ice-ufrag、ice-pwd 分别为ICE协商用到的认证信息`

offer

audio和video是一样的，这里只列举audio的

a=ice-ufrag:K6c3

a=ice-pwd:UbMotp8VJxqc37FjOMnQ4Kfa

a=ice-options:trickle 通知对端支持trickle，即sdp里面描述媒体信息和ice候选项的信息可以分开传输

answer

audio和video是一样的，这里只列举audio的

a=ice-lite

a=ice-ufrag:cg7j3f9v

a=ice-pwd:6sb5qm80816910z0284j5i05l4905kpl

两端在SDP协商时交换ice-pwd和ice-ufrag，以得对端用户名和密码。

计算stun包里面的MESSAGE-INTEGRITY时，需要自己本地的ice-pwd去计算HMAC-SHA1，生成对应的属性值串，用来检查消息的完整性，检验被篡改。

6 重点debug

binding request

SrsStunPacket::decode(const char* buf, const int nb_buf)
srs_rtc_stun_stack.cpp:

#0 SrsStunPacket::decode (this=0x7ffff7fdf9a0, buf=0x5555561ca570 "", nb_buf=100) at src/protocol/srs_rtc_stun_stack.cpp:195
#1 0x000055555581b50d in SrsRtcServer::on_udp_packet (this=0x5555560b50d0, skt=0x7ffff7fdfb70)
at src/app/srs_app_rtc_server.cpp:387
#2 0x00005555557b86ac in SrsUdpMuxListener::cycle (this=0x5555560fdec0) at src/app/srs_app_listener.cpp:636
#3 0x000055555571380a in SrsFastCoroutine::cycle (this=0x5555561b7500) at src/app/srs_app_st.cpp:270
#4 0x00005555557138a6 in SrsFastCoroutine::pfn (arg=0x5555561b7500) at src/app/srs_app_st.cpp:285
#5 0x000055555583bc48 in _st_thread_main () at sched.c:363
#6 0x000055555583c4e4 in st_thread_create (start=0x7ffff6e689d8 <__libc_multiple_threads>, arg=0x5b0000006e, joinable=119,
stk_size=124) at sched.c:694

SrsStunPacket::decode (this=0x7ffff7fdf9a0, buf=0x5555561ca570 "", nb_buf=100 字节)

SrsStunPacket::encode_binding_response(const string& pwd, SrsBuffer* stream)
message_type = 1

USERNAME 用户名，用于消息完整性，在webrtc中的规则为“对端的ice-ufrag：自己的ice-ufrag”，其中ice-ufrag已通过提议/应答的SDP信息进行交互
uint16_t type = 6， len=13， val = 6670ji13:/oN4

没有处理
type = 0xc057 len=4 val ="\000\001\000\n"

ICE-CONTROLLING offerer是controlling ,解决 ICE流程中会话双方role冲突的解决方法
type 802A， len 8，val "\212\363\262oY}\245", <incomplete sequence \317>

没有处理 PRIORITY优先级 https://tools.ietf.org/html/rfc5245#section-19
type 0x0024，len 4， val "n\177\036\377"

USE-CANDIDATE 表示使用该通道开始建联DTLS链接
type 0x0025 len0,

没有处理MESSAGE-INTEGRITY，这里实际是客户端的ice-pwd经过HMAC-SHA1，生成对应的属性
type 8， len 20， val "\212\177Dh\016\362\024fn\325b\017\314\017\326\035\320",
<incomplete sequence \366\233>

没有处理FINGERPRINT，这个只是crc32校验后的数据，目的是防篡改
type 0x8028 len 4, val 3\376\023?
到这里就没有数据可以处理了。

binding response

#0 SrsStunPacket::encode_binding_response (this=0x7ffff7fdf220,
pwd="2hz6u30u425a556930n3eyuzb7amt961", stream=0x55555620f720) at
src/protocol/srs_rtc_stun_stack.cpp:283
#1 0x00005555556c2b59 in SrsStunPacket::encode (this=0x7ffff7fdf220,
pwd="2hz6u30u425a556930n3eyuzb7amt961",
stream=0x55555620f720) at src/protocol/srs_rtc_stun_stack.cpp:275
#2 0x00005555557e845a in SrsRtcConnection::on_binding_request (this=0x5555562115e0,
r=0x7ffff7fdf9a0)
at src/app/srs_app_rtc_conn.cpp:2634
#3 0x00005555557e4f8a in SrsRtcConnection::on_stun (this=0x5555562115e0,
skt=0x7ffff7fdfb70, r=0x7ffff7fdf9a0)
at src/app/srs_app_rtc_conn.cpp:1988
#4 0x000055555581b677 in SrsRtcServer::on_udp_packet (this=0x5555560b50d0,
skt=0x7ffff7fdfb70)
at src/app/srs_app_rtc_server.cpp:406
#5 0x00005555557b86ac in SrsUdpMuxListener::cycle (this=0x5555560fdec0) at
src/app/srs_app_listener.cpp:636
#6 0x000055555571380a in SrsFastCoroutine::cycle (this=0x5555561b7500) at
src/app/srs_app_st.cpp:270

```
#7  0x00005555557138a6 in SrsFastCoroutine::pfn (arg=0x5555561b7500) at
src/app/srs_app_st.cpp:285
#8  0x000055555583bc48 in _st_thread_main () at sched.c:363
#9  0x000055555583c4e4 in st_thread_create (start=0x7ffff6e689d8
<__libc_multiple_threads>, arg=0x5b0000006e, joinable=119,
    stk_size=124) at sched.c:694
#10 0x00007ffff6e63c40 in ?? () from /lib/x86_64-linux-gnu/libc.so.6
#11 0x0000000000000000 in ?? ()

(gdb) print sendonly_skt->get_peer_ip().c_str()$28 = 0x7ffff7fdf100 "113.246.105.182"
sendonly_skt->get_peer_port()       17477

stun_binding_response.set_message_type(BindingResponse);     回应
stun_binding_response.set_local_ufrag(r->get_remote_ufrag());         客户端的ufrag
stun_binding_response.set_remote_ufrag(r->get_local_ufrag());   服务器的ufrag
stun_binding_response.set_transcation_id(r->get_transcation_id()); // 回应请求的transcation_id
// FIXME: inet_addr is deprecated, IPV6 support client的 ip,   是客户端的ip地址，比如
stun_binding_response.set_mapped_address(be32toh(inet_addr(sendonly_skt-
>get_peer_ip().c_str())));
stun_binding_response.set_mapped_port(sendonly_skt->get_peer_port()); // client的端口
stun_binding_response.encode(get_local_sdp()->get_ice_pwd(), stream)

    uint32_t crc32 = srs_crc32_ieee(stream->data(), stream->pos(), 0) ^ 0x5354554E;
     string fingerprint = encode_fingerprint(crc32);
```

FINGERPRINT：指纹认证，此属性可以出现在所有的 STUN 消息中，该属性用于区分 STUN 数据包与其他协议的包。属性的值为采用 CRC32 方式计算 STUN 消息直到但不包括FINGERPRINT 属性的的结果，并与 32 位的值 0x5354554e 异或。

7 参考

ICE之STUN协议---Binding Request

https://blog.csdn.net/glw0223/article/details/90728328

ICE之STUN协议---Binding Success Response

https://blog.csdn.net/glw0223/article/details/90730814