一、exosip实现获取底层UDP IP和端口数据的接口：

1.Exoisp.h末尾

加入：

/// add by HUIHUI 2018-06-01 16:17

/\*

\*get the last udp message remote ip addr

\*/

int eXosip\_event\_getUdpSocket(struct eXosip\_t \*excontext, char \* remoteIpAddr, int \*remotePort);

extl\_udp.c

2. 95行修改：

struct eXtludp {

int udp\_socket;

struct sockaddr\_storage ai\_addr;

char \*buf;

void \*QoSHandle;

unsigned long QoSFlowID;

int udp\_socket\_oc;

struct sockaddr\_storage ai\_addr\_oc;

struct \_udp\_stream socket\_tab[EXOSIP\_MAX\_SOCKETS];

struct sockaddr\_storage lastMessageRemote\_addr; //last message remote addr info added by HUIHUI 2018-06-01 16:12

};

3.添加函数：

int eXosip\_event\_getUdpSocket(struct eXosip\_t \*excontext, char \* remoteIpAddr, int \*remotePort)

{

struct eXtludp \*reserved = (struct eXtludp \*) excontext->eXtludp\_reserved;

struct sockaddr\_storage sa = reserved->lastMessageRemote\_addr;

socklen\_t slen;

if (excontext->eXtl\_transport.proto\_family == AF\_INET)

slen = sizeof (struct sockaddr\_in);

else

slen = sizeof (struct sockaddr\_in6);

char src6host[NI\_MAXHOST] = {0};

int recvport = 0;

memset(src6host, 0, NI\_MAXHOST);

recvport = \_eXosip\_getport((struct sockaddr \*) &sa, slen);

\_eXosip\_getnameinfo((struct sockaddr \*) &sa, slen, src6host, NI\_MAXHOST, NULL, 0, NI\_NUMERICHOST);

strcpy(remoteIpAddr, src6host);

\*remotePort = recvport;

return reserved->udp\_socket;

}

4.udp\_tl\_read\_message函数修改：

每个recvfrom 后面加一句：

reserved->lastMessageRemote\_addr = sa;

static int

udp\_tl\_read\_message (struct eXosip\_t \*excontext, fd\_set \* osip\_fdset, fd\_set \* osip\_wrset)

{

//printf("%s ========\n",\_\_FUNCTION\_\_);

struct eXtludp \*reserved = (struct eXtludp \*) excontext->eXtludp\_reserved;

socklen\_t slen;

int i;

if (reserved == NULL) {

OSIP\_TRACE (osip\_trace (\_\_FILE\_\_, \_\_LINE\_\_, OSIP\_ERROR, NULL, "wrong state: create transport layer first\n"));

return OSIP\_WRONG\_STATE;

}

if (reserved->udp\_socket <= 0)

return -1;

if (excontext->eXtl\_transport.proto\_family == AF\_INET)

slen = sizeof (struct sockaddr\_in);

else

slen = sizeof (struct sockaddr\_in6);

if (FD\_ISSET (reserved->udp\_socket, osip\_fdset)) {

struct sockaddr\_storage sa;

if (reserved->buf == NULL)

reserved->buf = (char \*) osip\_malloc (udp\_message\_max\_length \* sizeof (char) + 1);

if (reserved->buf == NULL)

return OSIP\_NOMEM;

#ifdef TSC\_SUPPORT

if (excontext->tunnel\_handle) {

i = tsc\_recvfrom (reserved->udp\_socket, reserved->buf, udp\_message\_max\_length, 0, (struct sockaddr \*) &sa, &slen);

}

else {

i = recvfrom (reserved->udp\_socket, reserved->buf, udp\_message\_max\_length, 0, (struct sockaddr \*) &sa, &slen);

}

#else

i = (int) recvfrom (reserved->udp\_socket, reserved->buf, udp\_message\_max\_length, 0, (struct sockaddr \*) &sa, &slen);

#endif

reserved->lastMessageRemote\_addr = sa;

//printf("%s ===1111111=== len=%d buf=%s\n", \_\_FUNCTION\_\_, i, reserved->buf);

if (i > 32) {

char src6host[NI\_MAXHOST];

int recvport = 0;

reserved->buf[i] = '\0';

memset (src6host, 0, NI\_MAXHOST);

recvport = \_eXosip\_getport((struct sockaddr \*) &sa, slen);

\_eXosip\_getnameinfo((struct sockaddr \*) &sa, slen, src6host, NI\_MAXHOST, NULL, 0, NI\_NUMERICHOST);

OSIP\_TRACE (osip\_trace (\_\_FILE\_\_, \_\_LINE\_\_, OSIP\_INFO1, NULL, "Message received from: %s:%i\n", src6host, recvport));

\_eXosip\_handle\_incoming\_message (excontext, reserved->buf, i, reserved->udp\_socket, src6host, recvport, NULL, NULL);

/\* if we have a second socket for outbound connection, save information about inbound traffic initiated by receiving data on udp\_socket \*/

if (reserved->udp\_socket\_oc > 0)

{

int pos;

for (pos = 0; pos < EXOSIP\_MAX\_SOCKETS; pos++) {

/\* does the entry already exist? \*/

if (reserved->socket\_tab[pos].remote\_port == recvport && osip\_strcasecmp(reserved->socket\_tab[pos].remote\_ip, src6host)==0) {

OSIP\_TRACE (osip\_trace (\_\_FILE\_\_, \_\_LINE\_\_, OSIP\_INFO1, NULL, "inbound traffic/connection already in table\n"));

break;

}

}

if (pos == EXOSIP\_MAX\_SOCKETS) {

OSIP\_TRACE (osip\_trace (\_\_FILE\_\_, \_\_LINE\_\_, OSIP\_INFO1, NULL, "inbound traffic/new connection detected (%s:%i\n", src6host, recvport));

for (pos = 0; pos < EXOSIP\_MAX\_SOCKETS; pos++) {

if (reserved->socket\_tab[pos].out\_socket ==0) {

reserved->socket\_tab[pos].out\_socket=reserved->udp\_socket;

snprintf(reserved->socket\_tab[pos].remote\_ip, sizeof(reserved->socket\_tab[pos].remote\_ip), "%s", src6host);

reserved->socket\_tab[pos].remote\_port=recvport;

OSIP\_TRACE (osip\_trace (\_\_FILE\_\_, \_\_LINE\_\_, OSIP\_INFO1, NULL, "inbound traffic/new connection added in table\n"));

break;

}

}

}

}

}

else if (i < 0) {

#ifdef \_WIN32\_WCE

int my\_errno = 0;

#else

int my\_errno = errno;

#endif

OSIP\_TRACE (osip\_trace (\_\_FILE\_\_, \_\_LINE\_\_, OSIP\_ERROR, NULL, "Could not read socket (%i) (%i) (%s)\n", i, my\_errno, strerror (my\_errno)));

if (errno==0 || errno==34) {

udp\_message\_max\_length = udp\_message\_max\_length\*2;

osip\_free(reserved->buf);

reserved->buf = (char \*) osip\_malloc (udp\_message\_max\_length \* sizeof (char) + 1);

}

if (my\_errno == 57) {

\_udp\_tl\_reset (excontext);

}

}

else {

OSIP\_TRACE (osip\_trace (\_\_FILE\_\_, \_\_LINE\_\_, OSIP\_INFO1, NULL, "Dummy SIP message received\n"));

}

}

if (reserved->udp\_socket\_oc > 0 && FD\_ISSET (reserved->udp\_socket\_oc, osip\_fdset)) {

struct sockaddr\_storage sa;

if (reserved->buf == NULL)

reserved->buf = (char \*) osip\_malloc (udp\_message\_max\_length \* sizeof (char) + 1);

if (reserved->buf == NULL)

return OSIP\_NOMEM;

#ifdef TSC\_SUPPORT

if (excontext->tunnel\_handle) {

i = tsc\_recvfrom (reserved->udp\_socket\_oc, reserved->buf, udp\_message\_max\_length, 0, (struct sockaddr \*) &sa, &slen);

}

else {

i = recvfrom (reserved->udp\_socket\_oc, reserved->buf, udp\_message\_max\_length, 0, (struct sockaddr \*) &sa, &slen);

}

#else

i = (int) recvfrom (reserved->udp\_socket\_oc, reserved->buf, udp\_message\_max\_length, 0, (struct sockaddr \*) &sa, &slen);

#endif

//printf("%s ===2222222=== len=%d buf=%s\n", \_\_FUNCTION\_\_, i, reserved->buf);

reserved->lastMessageRemote\_addr = sa;

if (i > 32) {

char src6host[NI\_MAXHOST];

int recvport = 0;

reserved->buf[i] = '\0';

memset (src6host, 0, NI\_MAXHOST);

recvport = \_eXosip\_getport((struct sockaddr \*) &sa, slen);

\_eXosip\_getnameinfo((struct sockaddr \*) &sa, slen, src6host, NI\_MAXHOST, NULL, 0, NI\_NUMERICHOST);

OSIP\_TRACE (osip\_trace (\_\_FILE\_\_, \_\_LINE\_\_, OSIP\_INFO1, NULL, "Message received from: %s:%i\n", src6host, recvport));

\_eXosip\_handle\_incoming\_message (excontext, reserved->buf, i, reserved->udp\_socket\_oc, src6host, recvport, NULL, NULL);

}

else if (i < 0) {

#ifdef \_WIN32\_WCE

int my\_errno = 0;

#else

int my\_errno = errno;

#endif

OSIP\_TRACE (osip\_trace (\_\_FILE\_\_, \_\_LINE\_\_, OSIP\_ERROR, NULL, "Could not read socket (%i) (%i) (%s)\n", i, my\_errno, strerror (my\_errno)));

if (errno==0 || errno==34) {

udp\_message\_max\_length = udp\_message\_max\_length\*2;

osip\_free(reserved->buf);

reserved->buf = (char \*) osip\_malloc (udp\_message\_max\_length \* sizeof (char) + 1);

}

if (my\_errno == 57) {

\_udp\_tl\_reset\_oc (excontext);

}

}

else {

OSIP\_TRACE (osip\_trace (\_\_FILE\_\_, \_\_LINE\_\_, OSIP\_INFO1, NULL, "Dummy SIP message received\n"));

}

}

return OSIP\_SUCCESS;

}