TCP NAT 穿透开发库 Libt2u 使用说明

更新日期：2014-08-21 作者：Yexr

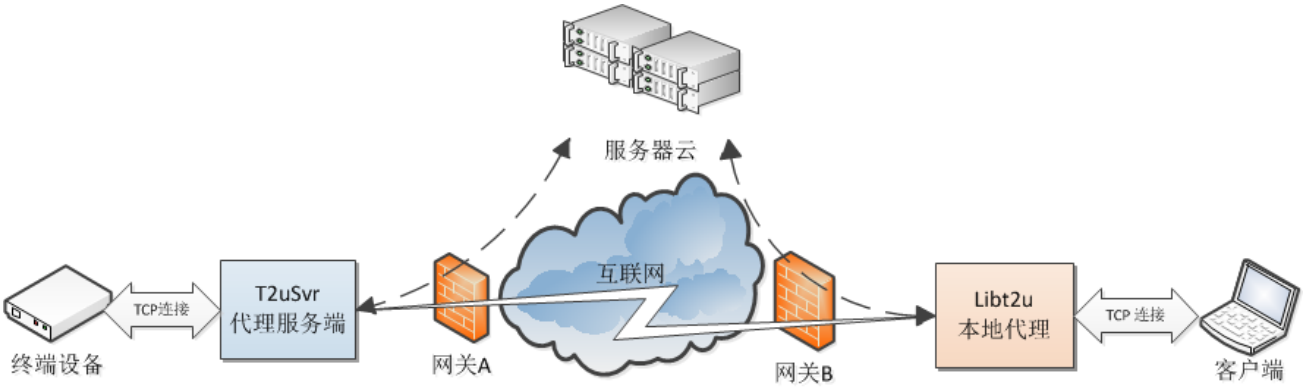
一. 概述

Libt2u 是一个实现 **TCP 穿越网关**并与远端设备建立 P2P 连接的开发库。它可以在**不修改原有通信协议**的基础上，帮助网络终端厂家**实现客户端与设备的 P2P 直连**，节省开发周期，提升用户体验。

与传统的动态域名加路由器端口映射方式相比，具有不可比拟的优越性。为用户省去了申请动态域名，路由器上做端口映射等繁琐的操作，让设备达到即插即用的效果。

二. 系统组成

整个系统由服务器云、设备端程序和客户端 SDK 组成，如图：



1) 服务器云
由一台或多台服务器组成服务器云接受设备端和客户端的注册，协助客户端找到设备并建立 P2P 连接。

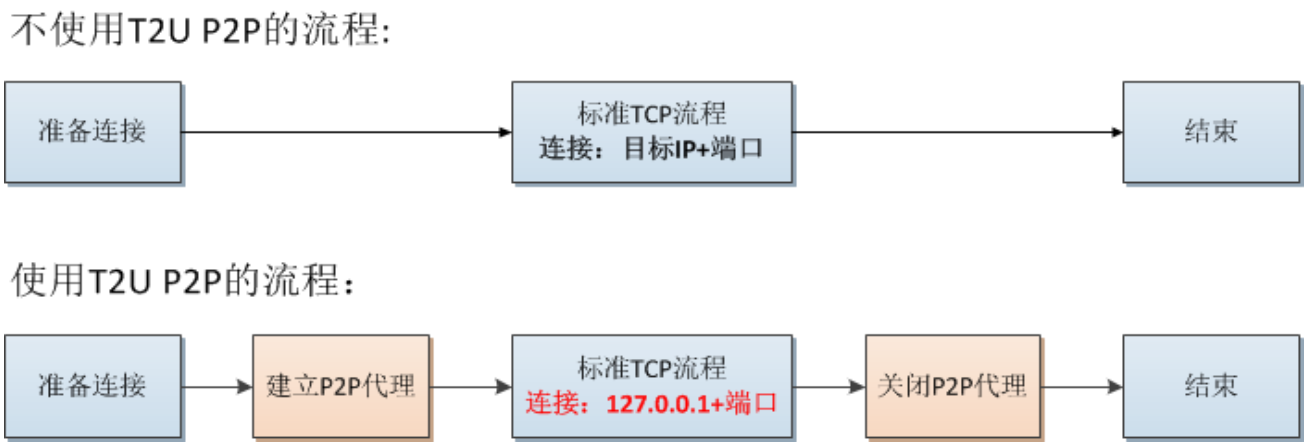
2) 设备端
T2uSvr 设备端程序是一个根据各种嵌入式硬件环境编译成的可执行文件，它其实是一个代理服务端。厂家在**设备端不需要做额外的开发**，只需在固件中加入 T2uSvr 并在开机时自动启动进程。

3) 客户端
Libt2u 是客户端 SDK，为客户端与设备端实现 TCP P2P 连接建立通信端口。无需修改客户端与设备的通信协议，只需让客户端连接 Libt2u 建立的端口，即可达到连接设备的目的。

三. 技术原理

T2uSvr 和 Libt2u 作为 2 个代理，分别运行在设备端和客户端，两个代理之间通过 P2P NAT 技术建立连接，依靠自主研发的 UDP 可靠传输和拥塞控制算法进行数据通信，再加上端口转发技术，实现了**将远端设备的任意端口映射到客户机本地**。于是在不知道设备 IP 地址，也没有在路由器上做端口映射的情况下，通过 TCP 连接本地端口达到连接设备的目的。

客户端使用 P2P 和不使用 P2P 在流程上的区别：



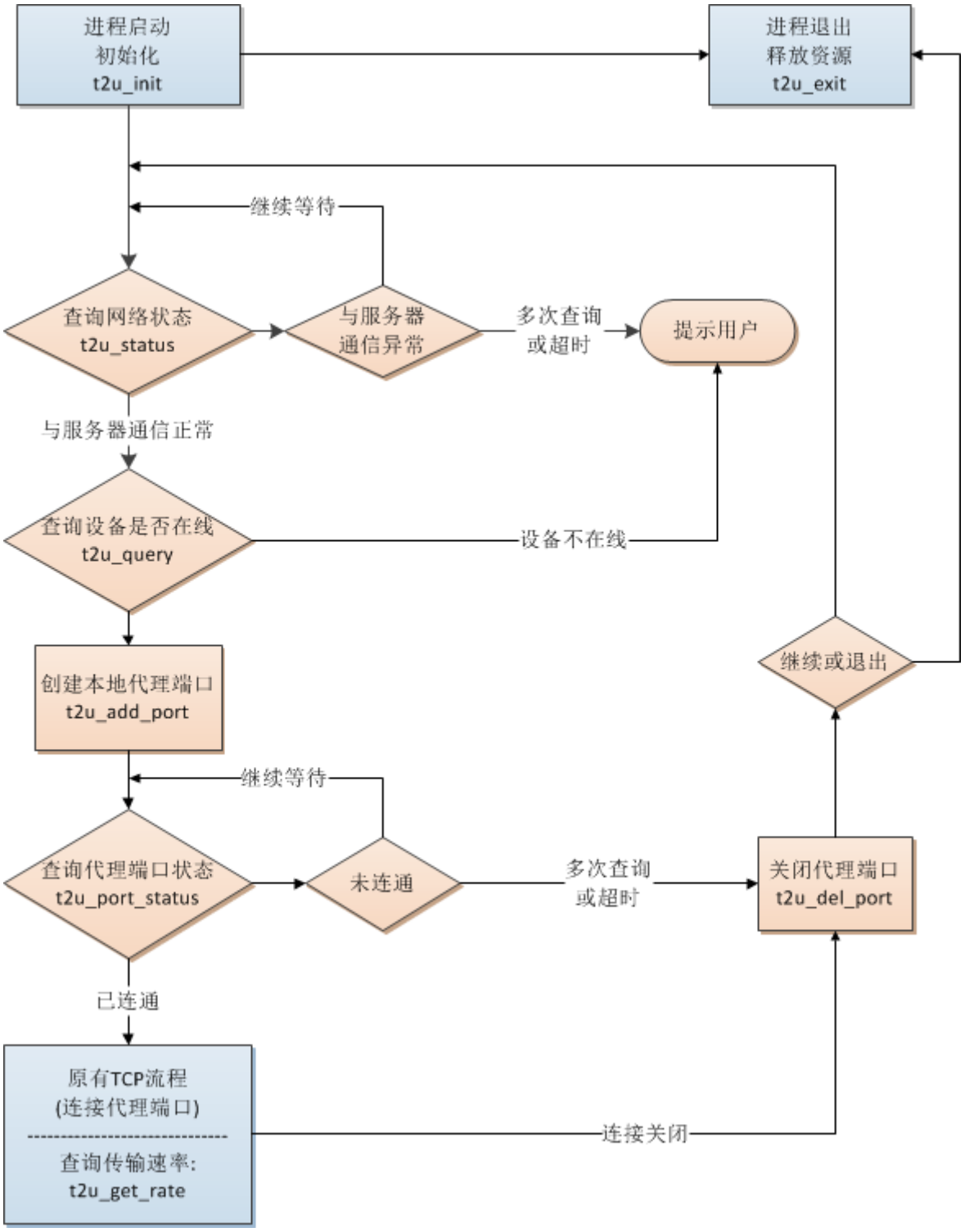
四． 客户端开发流程

1. 进程启动时调用 t2u_init 初始化，之后每次连接前都不需要再调用初始化接口

2. 连接过程调用步骤：
   1) 先调用 t2u_status ，查询客户端与服务器之间是否已经正常连接，如果尚未连接服务器，继续等待，或者提示用户
   2) 查询设备是否在线，调用 t2u_query ，如果不在线，提示用户
   3) 创建 P2P 代理端口，t2u_add_port ，将设备的服务端口映射到客户端本地
   4) 查询端口状态，t2u_port_status (由于创建端口后，建立 P2P 连接需要几秒钟的时间，等待查询到连接已建立)
   5) 按照原协议，TCP 连接本地端口，开始播放视频或其他操作 (连接期间可以调用 t2u_get_rate 查询传输速率)
   6) 连接关闭后，t2u_del_port 删除本地代理端口

3. 进程退出时调用 t2u_exit 释放资源

流程图：


进程启动
初始化
t2u_init
进程退出
释放资源
t2u_exit
继续等待
查询网络状态
t2u_status
与服务器
通信异常
多次查询
或超时
提示用户
与服务器通信正常
查询设备是否在线
t2u_query
设备不在线
创建本地代理端口
t2u_add_port
继续或退出
继续等待
查询代理端口状态
t2u_port_status
未连通
多次查询
或超时
关闭代理端口
t2u_del_port
已连通
原有TCP流程
(连接代理端口)
查询传输速率:
t2u_get_rate
连接关闭

五. 接口说明

1) t2u_init 初始化

SDK 初始化并指定服务器地址端口。

注：调用一次即可，不能重复调用。

```
void    t2u_init(
                const char*         svraddr,
                unsigned short      svrport,
                const char*         svrkey
);
```

参数说明：

Svraddr:

[in] 服务器地址，可以是 IP 或域名

Svrport:

[in] 服务器端口

Svrkey:

[in] 服务器密钥（只有使用正确的密钥才能注册到服务器，防止非法用户使用）
如果服务器没有密钥，使用 NULL

返回值：无

示例：

```
t2u_init("122.225.102.4",8000,NULL);
```

2) t2u_set_callback 设置回调函数

设置 P2P 连接成功或断开的回调接口

```
void        t2u_set_callback(
                PT2uCallBack callback
);
```

参数说明：

callback:

[in] T2uCallBack 结构体定义如下：

```
typedef struct _T2U_CALLBACK
{
    void (*OnConnect)(              //连接成功
                unsigned short local_port,      //本地代理端口号
                int usetime                     //建立 P2P 连接所用时间（毫秒）
    );
    void (*OnDisconnect)(           //连接断开
                unsigned short local_port,      //本地代理端口号
                int ret                         /* 断开原因
                                                   -2:    P2P 连接失败
                                                   -3:    P2P 连接中断
                                                   -4:    对方离线
                                                   -5:    设备有密码，密码认证失败
                                                */
    );
}T2uCallBack,*PT2uCallBack;
```

3) **t2u_set_port_range 设置 P2P 端口范围**

设置创建 P2P 通道时客户端使用的端口范围

```
void      t2u_set_port_range(
                    unsigned short min_port,
                    unsigned short max_port
);
```

参数说明：

Min_port:

[in]　　最小端口值

Max_port:

[in]　　最大端口值

4) **t2u_add_port 添加映射端口**

将远端设备的指定端口映射到本地

```
int       t2u_add_port(
              const char*          uuid,
              unsigned short       remote_port,
              unsigned short       local_port
);
```

参数说明：

Uuid:

[in]　　设备 UUID（如:设备序列号）

Remote_port:

[in]　　对方的服务端口

Local_port:

[in]　　映射到本地的端口，0 表示使用随机端口

返回值：

>0:　　映射到本地的端口

-1:　　创建端口失败，本地端口被占用

示例：

```
int port1,port2;

//将序列号为 12814201441 设备的 23 端口映射到本地  23 端口
port1 = t2u_add_port("12814201441",23,23);

//将设备的 8000 端口映射到本地随机端口
port2 = t2u_add_port("12814201441",8000,0);
```

5) **t2u_add_port_ex 添加映射端口**

把远端设备当作代理，通过它把其他网络设备的端口映射到客户端本地

```
int       t2u_add_port_ex(
              const char*          uuid,
              const char*          remote_ip,
              unsigned short       remote_port,
              unsigned short       local_port
);
```

参数说明：

Uuid:

[in] 设备 UUID（如:设备序列号）

Remote_ip:

[in] 跟设备在同一局域网的其他设备的 IP

Remote_port:

[in] 对方的服务端口

Local_port:

[in] 映射到本地的端口，0 表示使用随机端口

返回值：

>0: 映射到本地的端口

-1: 创建端口失败，本地端口被占用

示例：

```
int port;

//通过序列号为 12814201441 的设备，把对方局域网内的另一台设备映射到本地
port = t2u_add_port_ex("12814201441","192.168.1.22",22,22);
```

6) t2u_add_port_v3 添加映射端口

将远端设备的指定端口映射到本地

```
int     t2u_add_port_v3(
            const char*         uuid,
            const char*         passwd,
            const char*         remote_ip,
            unsigned short      remote_port,
            unsigned short      local_port
);
```

参数说明：

Uuid:

[in] 设备 UUID（如:设备序列号）

Passwd:

[in] 设备密码

Remote_ip:

[in] 连设备本机用 "127.0.0.1" 或 跟设备在同一局域网的其他设备的 IP

Remote_port:

[in] 对方的服务端口

Local_port:

[in] 映射到本地的端口，0 表示使用随机端口

返回值：

>0: 映射到本地的端口

-1: 创建端口失败，本地端口被占用

7) t2u_del_port 删除映射端口

关闭并删除已经映射的端口

```
void      t2u_del_port(
          unsigned short          port
);
```

参数说明：

port:

[in]　　映射在本地的端口号

8) t2u_port_status 查询映射端口状态

查询映射端口是否已经与远端设备连通

```
int       t2u_port_status(
               unsigned short       port,
               PT2uNetStat          pStat
);
```

参数说明：

port:

[in]　　映射在本地的端口号

pStat:

[out]　　T2uNetStat 连接状态，如果不需要可以输入 NULL

```
typedef struct _T2U_NET_STAT
{
    char            ip[20];         //对方 IP
    int             port;           //对方端口
    int             proxy;          //是否通过代理
    float           lost_rate;      //丢包率
    int             bandwidth;      //估算最大上行带宽 KB/s
}T2uNetStat,*PT2uNetStat;
```

返回值：

1:　　已连通

0:　　未连通

-1:　　失败，端口不存在

-2:　　P2P 连接失败，等待 30 秒后自动重连

-3:　　P2P 连接中断，等待 30 秒后自动重连

-4:　　对方离线，等待 30 秒后自动重连

-5:　　设备有密码，密码认证失败

9) t2u_get_rate 查询速率统计

查询某个端口或所有端口的速率统计值

```
int       t2u_get_rate(
               unsigned short          port,
               PT2uNetRate             pRate
);
```

参数说明：

port:

[in] 要查询的端口号，0 表示统计全局速率

pRate:

[out] T2uNetRate 速率统计

```
typedef   struct _T2U_NET_RATE
{
    int64_t          total_recv;        //总接收字节
    int64_t          total_send;        //总发送字节
    float            cur_recv_rate;     //当前接收速率 KB/s
    float            cur_send_rate;     //当前发送速率 KB/s
    float            cur_rate;          //当前总速率  KB/s
    float            avg_recv_rate;     //平均接收速率 KB/s
    float            avg_send_rate;     //平均发送速率 KB/s
    float            avg_rate;          //平均总速率 KB/s
    int              time_sec;          //统计时长,秒
}T2uNetRate,*PT2uNetRate;
```

返回值：

0: 成功

-1: 失败，端口不存在

10) t2u_status 查询与服务器连接状态

```
int      t2u_status();
```

返回值：

1: 与服务器连接，状态正常

0: 未连接服务器

-1: SDK 未初始化

-2: 服务器密钥无效

11) t2u_query 查询设备状态

查询设备当前是否在线

```
int      t2u_query(const char* uuid);
```

参数说明：

Uuid:

[in] 设备 UUID（如:设备序列号）

返回值：

1: 设备在线

0: 设备不在线

-1: 查询失败

12) t2u_query_ex 查询设备状态和附加参数

查询设备当前是否在线，以及获取设备上报的附加参数和设备当前的动态 IP

```
int     t2u_query_ex(
                        const char* uuid,
                        char* buff,
                        int buffsize,
                        char* ipaddr,
                        int ipsize
);
```

参数说明：

uuid:

[in] 设备 UUID（如:设备序列号）

buff:

[out] 保存设备附加参数的缓存地址

buffsize:

[in] 缓存大小

ipaddr:

[out] 保存设备当前动态 IP 的缓存地址

ipsize:

[in] 缓存大小

返回值：

1: 设备在线

0: 设备不在线

-1: 查询失败

13) t2u_search 搜索发现本地设备

发送组播消息，搜索发现本地具有代理服务的设备

搜索结果以文本字符串方式输出到指定缓存地址

每行一条记录，格式为: uuid=xxxx,ip=x.x.x.x

```
int     t2u_search(
            char* buff,
            int buffsize
);
```

参数说明：

buff:

[out] 输出结果的缓存地址

buffsize:

[in] 缓存大小

返回值：

>=0 发现的设备数

-1: 查询失败

14) t2u_exit 退出并释放资源

```
void    t2u_exit();
```