Boost 实现简单的 udp 代理服务器

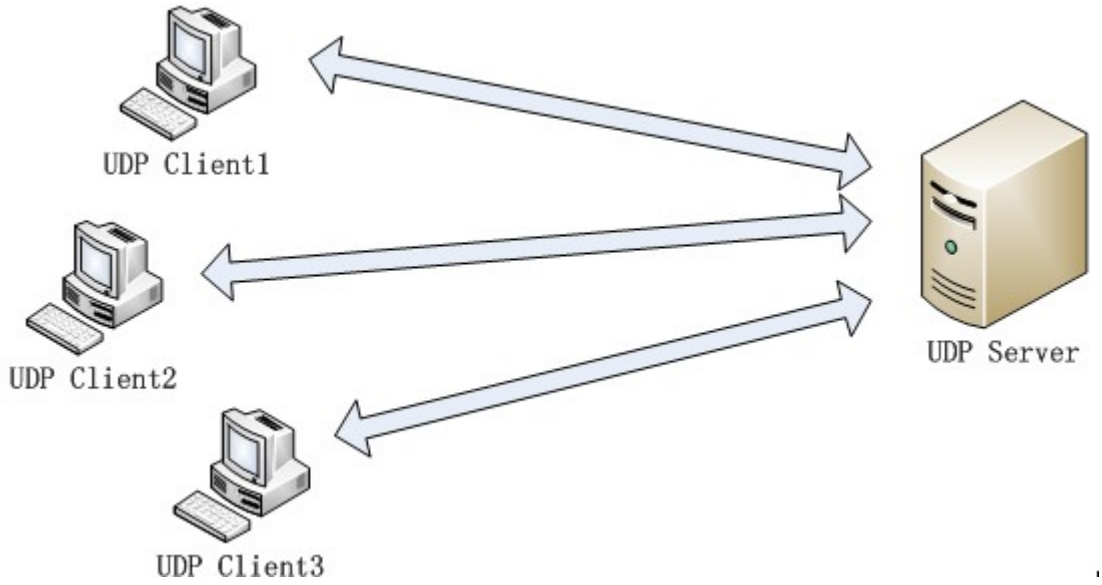
这段时间在开发代理服务器，这方面的资料了解的比较多，这里我总结下 udp 代理服务的实现，也方便我以后查阅。

首先看下基于 Boost 的 TCP 代理服务器：http://tcpproxy.codeplex.com/

我的实现也是参考这个 tcpProxy 的，这个就不多说了，网站上有，大家自己看了。

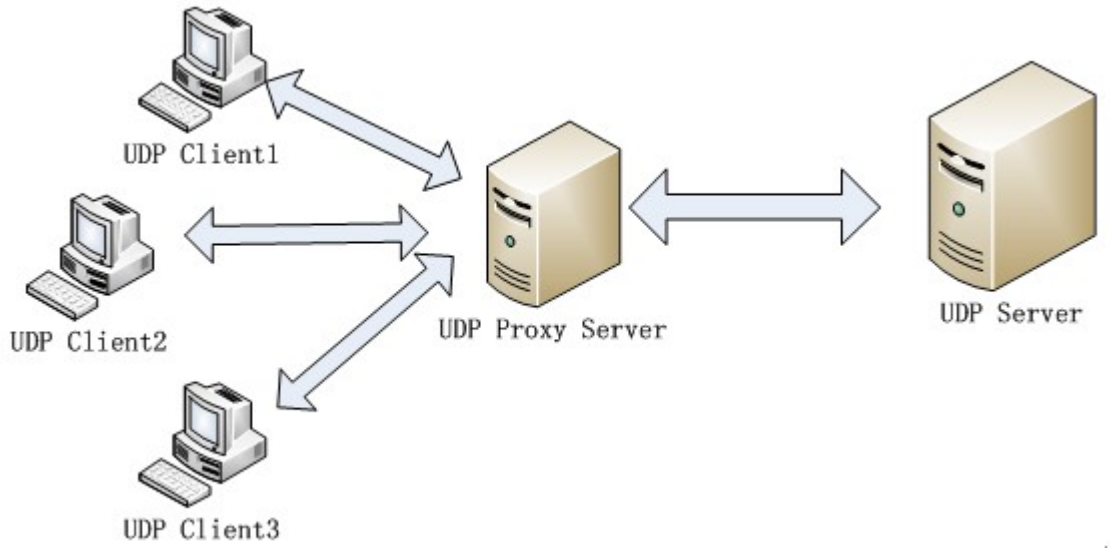
一、通信模型

1、非代理情况下的通信模型



这是典型的 C-S 通信模型，客户端和服务器直接交互。

2、代理情况下的通信模型



这种情况下，服务器和客户端不是直接交互，而是通过代理服务器进行的，代理服务器负责把客户端发来的请求转发给服务器，并把服务的回应返回给客户端。

二、UDP 服务器和客户端 demo

上面分析了通信模型，这里给出一个 echo 的服务器和客户端代码以供下文使用。

1、服务端 demo

这里有一个 python 实现的 echo 服务器，代码如下：

```
#! /usr/bin/python
# a simple udp server
import socket, traceback

s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
s.bind(("192.168.1.100",12345))

while True:
    try:
        message, address = s.recvfrom(1024)
        print "Got data from", address
        print message
        s.sendto(message, address)
    except (KeyboardInterrupt, SystemExit):
        raise
    except:
        traceback.print_exc()
```

服务端把收到的数据输出并发给客户端。

2、客户端 demo

```
#! /usr/bin/python
# a simple udp client
import socket,time

client = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
client.connect(('192.168.1.100', 12345))

while True:
        client.sendall('3')
        print time.time(),' : send success'
        print time.time()," : ",client.recv(1024)
        time.sleep(3)
#client.close()
```

客户端负责给服务端发数据并接收服务器返回的数据。

三、UDP 代理服务器 demo

这里采用 boost 库实现，用 asio 中的异步机制。

1、成员变量及参数说明

类名称：**m_udpProxyServer**

私有成员：

ip::udp::socket downstream_socket_; //代理服务器和客户端的链接

ip::udp::socket upstream_socket_; //代理服务器和远端服务器（echo server）的链接

string _remotehost; // 远端服务器（echo server）的 ip 地址

int _remoteport; //远端服务器（echo server）的端口

ip::udp::endpoint downstream_remoteUdpEndpoint; //客户端信息

ip::udp::endpoint upstream_remoteUdpEndpoint; //服务端信息

unsigned char downstream_data_[max_data_length]; //上传链路 buffer

unsigned char upstream_data_[max_data_length]; //下载链路 buffer

2、启动本地服务

代理服务器在这个模型中既充当客户端的服务器，又充当远端服务器（echo server）的客户端，所以代理服务器既要有本地监听端口供客户端连接，又要向远端服务器（echo server）发起链接，转发客户端发来的数据。

```
m_udpProxyServer(io_service& io,const string& localhost,
        const int& localport,const string& remotehost,const int& remoteport):
        downstream_socket_(io,ip::udp::endpoint(ip::udp::v4(),localport)), //启动本地服务
        upstream_socket_(io), //初始化连接远端服务器的 socket
        _remotehost(remotehost),
        _remoteport(remoteport),
        upstream_remoteUdpEndpoint(ip::address_v4::from_string(_remotehost),_remoteport)
        {
                start_downstream_receive();
        }
```

3、连接远端服务器

当接收到客户端发来的数据后，触发代理服务器向远端服务器的连接。

```
//接收客户端发来的数据
void start_downstream_receive()
{
        //如果接收到客户端的数据则触发向服务器的链接
        downstream_socket_.async_receive_from(
                boost::asio::buffer(downstream_data_,max_data_length),
                downstream_remoteUdpEndpoint,
                boost::bind(&m_udpProxyServer::upstream_connect,this,
```

```
                boost::asio::placeholders::bytes_transferred,
                boost::asio::placeholders::error));
}
//连接远端服务器
void upstream_connect(const size_t& bytes_transferred,
        const boost::system::error_code& error)
{
        if (!error )
        {
                upstream_socket_.async_connect(
                        upstream_remoteUdpEndpoint,
                        boost::bind(&m_udpProxyServer::handle_upstream_connect,
                        this,bytes_transferred,boost::asio::placeholders::error));
        }
        else
        {
                std::cerr << "Error: " << error.message() << std::endl;
        }
}
```

4、数据转发

将从客户端接收到的数据转发给远端服务器，并接收远端服务器的返回数据，转发给客户端。

```
void handle_upstream_connect(const size_t& bytes_transferred,
        const boost::system::error_code& error)
{
        //将从客户端接收到的数据转发给远端服务器
        upstream_socket_.async_send_to(
                        boost::asio::buffer(downstream_data_,bytes_transferred),
                        upstream_remoteUdpEndpoint,
                        boost::bind(&m_udpProxyServer::handle_upstream_send,
                        this,boost::asio::placeholders::error));
}

void handle_upstream_send(const boost::system::error_code& error)
```

```
{
	if (!error )
	{
		//从服务器接收返回数据
		upstream_socket_.async_receive_from(
			boost::asio::buffer(upstream_data_,max_data_length),
			upstream_remoteUdpEndpoint,
			boost::bind(&m_udpProxyServer::handle_upstream_receive,
			this,
			boost::asio::placeholders::bytes_transferred,
			boost::asio::placeholders::error));

	}
	else
	{
		std::cerr << "Error: " << error.message() << std::endl;
	}
}

void handle_upstream_receive(const size_t& bytes_transferred,
	const boost::system::error_code& error)
{
	if (!error )
	{
		//把从服务器接收到的返回数据转发给客户端
		downstream_socket_.async_send_to(
			boost::asio::buffer(upstream_data_,bytes_transferred),
			downstream_remoteUdpEndpoint,
			boost::bind(&m_udpProxyServer::handle_downstream_send,
			this,
			boost::asio::placeholders::error));
	}
	else
	{
		std::cerr << "Error: " << error.message() << std::endl;
```

```
        }
}

void handle_downstream_send(const boost::system::error_code& error)
{
        if (!error )
        {
                //接收客户端发来的数据
                downstream_socket_.async_receive_from(
                        boost::asio::buffer(downstream_data_,max_data_length),
                        downstream_remoteUdpEndpoint,
                        boost::bind(&m_udpProxyServer::handle_downstream_receive,this,
                        boost::asio::placeholders::bytes_transferred,
                        boost::asio::placeholders::error));
        }
        else
        {
                std::cerr << "Error: " << error.message() << std::endl;
        }
}

void handle_downstream_receive(const size_t& bytes_transferred,
        const boost::system::error_code& error)
{
        //将从客户端接收到的数据转发给远端服务器
        upstream_socket_.async_send_to(
                boost::asio::buffer(downstream_data_,bytes_transferred),
                upstream_remoteUdpEndpoint,
                boost::bind(&m_udpProxyServer::handle_upstream_send,
                this,boost::asio::placeholders::error));
}
```

完整代码：https://gist.github.com/3888929