基于下一代网络的 SIP 信令服务器设计

杨 川, 唐余亮
(厦门大学信息科学与技术学院通信工程系, 福建 厦门 361005)

摘要: SIP(Session initiation protocol)服务器是下一代网络(NGN)中的关键设备, 其开发所采用的 SIP 协议栈将直接影响其设备性能. 本文从 SIP 协议的基本原理出发, 介绍了一种由 C++ 编写的、稳定性强的开源 SIP 协议栈——ReSIProcate 的架构及其使用方法, 论述了如何使用 Microsoft Visual Studio .NET 2003 开发平台, 利用 ReSIProcate 协议栈实现 SIP 服务器系统的程序设计, 并对该系统信令模块中的注册服务器和代理服务器功能的实现方法进行了重点分析. 实验证明, 该设计系统稳定性好, 扩展性强, 符合 RFC3261 规范, 性能指标均达到设计要求, 实际系统运行良好.

关键词: 下一代网络; SIP; ReSIProcate; 注册服务器; 代理服务器
中图分类号: TP 393 文献标识码: A 文章编号: 0438-0479(2008)01-0045-05

随着全球电信运营市场的逐步开放, 用户已不再满足于现有网络的业务单一, 要求电信运营商能够更快地提供新业务, 以更低的成本提供多样化的语音、数据、视频融合业务. 在这种情况下, 就要打破现有各有线和无线网络专网组网技术的限制, 真正实现多网融合, 从而使下一代网络(NGN)应运而生. NGN 采用开放的网络构架体系, 实现了业务与呼叫控制的分离以及呼叫与承载的分离, 使得业务真正独立于网络, 从而能快速生成各种业务, 支持各种运营模式和商业模式, 有效地降低了网络建设投资和运维成本. 近年来, 它已成为全球通信产业发展的焦点.

目前, ETSI 和 ITU-T 等国家组织都在大力研究基于 SIP(Session initiation protocol)会话的网络融合方案, 以实现现有网络向 NGN 的演进和发展. 通过 SIP 协议 NGN 可同时支持固定和移动的多种接入方式, 实现端到端的语音、视频、数据、状态呈现、即时通信、游戏等业务. ReSIProcate 作为开源的 SIP 协议栈, 以其特有的稳定性, 在商业化的 SIP 终端和网络服务器开发上, 具有广泛的应用前景.

1 SIP 协议概述

SIP 协议是一种客户服务器协议, 用于创建、修改和结束与一个或多个参与者的会话, 具有简单、可扩展等优点, 在 RFC 3261[1] 中定义了两个要素: SIP 用户代理(User agent, UA)和 SIP 网络服务器, 包括代理服务器(Proxy server)、注册服务器(Registrar server)、重定向服务器(Redirect server)、位置服务器(Location server), 如图 1 所示. SIP 消息主要分为两类: 从客户端到服务器端的 6 种请求消息(Request)和从服务器端到客户端的 6 种响应(Response), 详情分别请见表 1、2 所示. 所有的消息都是由起始行(Start-line)、消息头(Message-header)和消息体(Message-body)三部分构成.

表 1 SIP 请求消息
Tab. 1 Request message of SIP

请求消息	消息说明
REGISTER	用于登记用户信息
INVITE	邀请用户加入会话
ACK	对 INVITE 做最后的确认
CANCEL	取消尚未完成的请求
BYE	结束一个已经存在的会话
OPTION	查询服务器功能

表 2 SIP 响应状态码
Tab. 2 Response code of SIP

响应状态码	状态码说明
1XX	暂时响应
2XX	成功响应
3XX	重定向响应
4XX	客户端出错
5XX	服务器端出错
6XX	全局故障

SIP 与分组交换网上的其他会话控制协议(如 H.323)相比, 最大的优点是具有良好的可扩展性, 用户可

收稿日期: 2007-06-25
Email: yangcd1983@tom.com

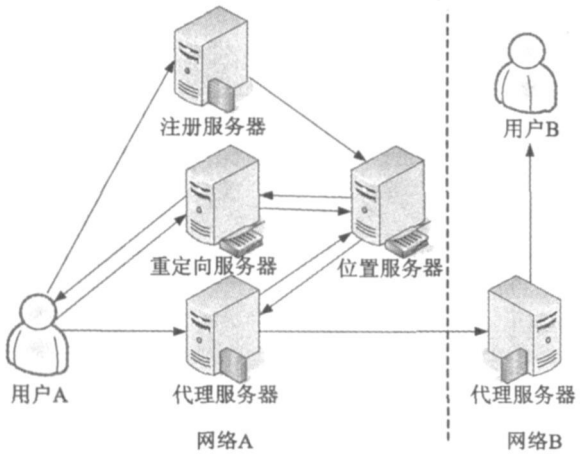

图 1 SIP 协议的工作流程

Fig. 1 Working flow of SIP

根据不同的需要自行扩展 SIP 消息方法, 以达到所要实现的目的. 但其扩展必须以 RFC3261 为基础, 不得违背 RFC3261 的基本精神. 随着 SIP 协议应用环境的不断扩大, 互联网工程任务组(Internet engineering task force, IETF) 相继提出了如下 5 种扩展消息: INFO, 用于传递会话中产生的与会话相关的控制信息; MESSAGE, 通过在其请求消息体中承载即时消息内容实现即时消息; REFER, 指示接受方通过使用在请求中提供的联系地址联系第三方; SUBSCRIBE, 用来向远端端点预定其变化的通知; NOTIFY, 发送消息以通知预定者它所预定的状态的变化.

SIP 网路服务器是由注册、代理、重定向及位置服务器这些逻辑实体构成, 在网络中, 这些实体各施其职, 最终目的是要实现用户代理(UA) 之间会话链路的建立、修改、结束等控制. 图 1 是 SIP 协议的工作流程, 很直观地表现出这 4 个逻辑实体的功能.

UA 在发起呼叫前, 首先要向所在域的注册服务器进行注册, 告知其当前具体的 IP 地址, 使 IP 地址与该 UA 的统一资源标识符(URI) 建立对应关系. 对于呼叫, SIP 协议支持两种运作模式: (1) SIP Proxy 模式, 主叫用户代理通过代理服务器将其呼叫请求消息转发至被叫用户代理; (2) SIP Redirect 模式, 主叫用户代理在重定向服务器的辅助下得到被叫用户地址, 直接将其呼叫请求发送到被叫用户代理[2].

从图 1 可以看出 SIP 服务器在整个网络中的作用是实现注册、代理转发呼叫的功能. 下文将针对注册服务器和代理服务器两个逻辑实体的程序设计进行论述.

2 ReSIProcate 协议栈

ReSIProcate 是按照 RFC 3261(SIP) 和 RFC 2327 (SDP)[3] 标准, 使用标准的 C++ 语言编写的协议栈, 其源代码公开, 性能十分稳定. 该协议栈提供了各种 SIP 消息的编程接口, 用户根据这些接口来实现应用编程开发. 整个 ReSIProcate 协议栈的核心部分是 DUM(Dialog usage manager) 和 SIP Stack, 如图 2 所示.

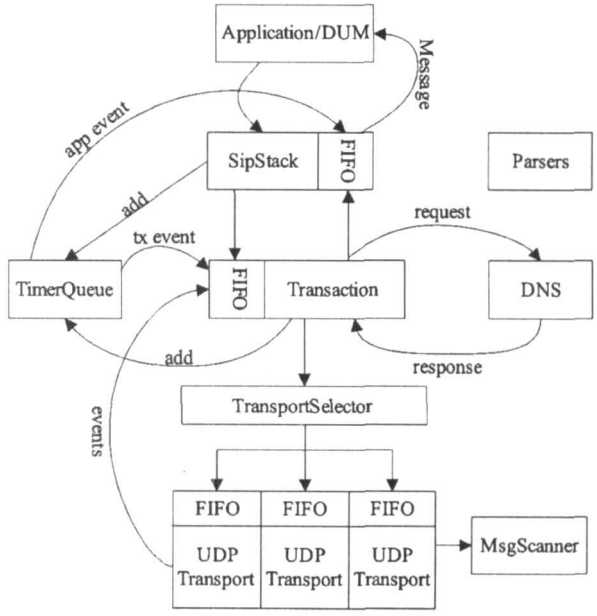

图 2 ReSIProcate 协议栈的架构

Fig. 2 Architecture of ReSIProcate

DUM 是事务层管理类, 是整个协议栈的上层管理者, 实现 SIP 的用户注册、会话邀请、会话结束及其应答等一系列动作. 其派生关系如下: DUM 产生 Dialog Set, Dialog Set 产生 Dialog, Dialog 产生 Session, Session 又分为 Client Session 和 Server Session. DUM 类还定义了很多句柄管理类, 如 ServerInviteSessionHandle, 通过它我们能得到对象 ServerInviteSession, 从而对其进行操作, 这是一种在事件响应中非常有用的方法[4].

SIP Stack 则是协议栈内部的管理类, 是协议栈与用户的通道, 它充分考虑到了要完整实现 RFC 3261 定义的各种 SIP 元素和概念, 以及让这些元素相互关联起来成为一个有机的整体. 从图 2 可以看出, SipStack 这个类从用户层不断获取消息, 并在协议栈内部将这些消息向 DUM 或传输层传送[5].

在利用 ReSIProcate 实现 SIP 服务器时, 关键是实现 Proxy 类和 UI(用户接口) 的交互以及完成事件回调控制. 要处理 SIP 的一系列事件, 还必须继承相关事件句柄接口. 它们都是一些抽象类, 如: 服务端会话的事件接口(ServerInviteSessionHandler)、注册处理事件接口(ServerRegistrationHandler) 等等. 继承这些

接口, 说明服务器将要响应相关事件.

3　SIP 服务器设计

从图 1 所示的 SIP 的工作流程可以看出, 一个 SIP 服务器系统至少需要实现用户注册和呼叫代理功能, 下面我们将重点针对这两部分来论述 SIP 服务器系统的设计.

3.1　总体设计及模块结构

整个 SIP 服务器系统主要由 WEB 管理模块、数据库模块和信令处理模块等 3 个模块组成[6]. 各功能模块的相互关系如图 3 所示.

●WEB 管理模块: 是一个具有人性化、方便化的人机界面, 管理者可通过它远程控制、管理、维护服务器. 该模块采用 ASP. NET 编写.

●数据库模块: 主要负责系统运行参数、WEB 管理者用户及 SIP UA 地址等信息的存储、更新、查询, 它包含了位置服务器这个逻辑实体.

●信令处理模块: 是系统的核心软件模块, 可分为两个子模块: Reg Server 模块和 Proxy Server 模块. 信令处理模块主要负责接收来自网络的 SIP 信令, 再按要求交予 Reg Server 或 Proxy Server 模块进行处理, 将产生出新的 SIP 信令送回网络进行传输.

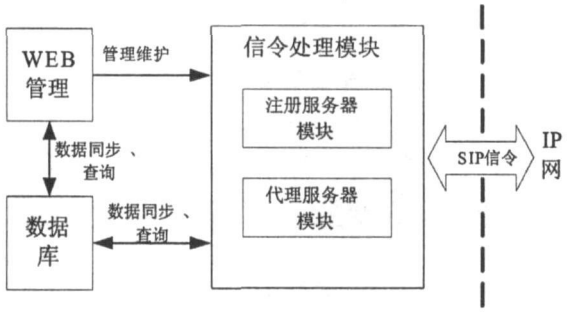

图 3　SIP 服务器系统功能模块

Fig. 3　Function module of SIP server system

在上述 3 个模块中, 信令处理模块是实现 SIP 的核心模块, 下面将通过介绍本模块的实现, 来进一步说明 ReSIProcate 协议栈的使用方法.

3.2　SIP 信令处理模块程序设计

利用 ReSIProcate 协议栈设计 SIP 信令处理模块时, 程序设计的基本流程是: 首先对 ReSIProcate 协议栈进行初始化, 然后, 初始化注册服务器和代理服务器两个功能实体, 最后, 启动相应线程循环等待事件发生.

3.2.1　协议栈的初始化

(1) 在自定义函数 Data addDomains(TransactionUser & tu, Store & store) 中确定服务器的域; (2) 在主函数中通过 SipStack stack( ) 声明一个协议栈实例; (3) 定义传输层协议, 在 IPv4 和 IPv6 之上支持多种传输协议(UDP、TCP、TLS、DTLS), 如通过 stack. addTransport(UDP, 5050, V4, StunEnabled) 函数可定义在 IP v4 上采用 UDP 协议, 信令端口为 5050; (4) 声明整个协议栈运行的线程实例: StackThread stackThread(stack).

3.2.2　注册服务器实例和代理服务器实例

整个协议栈的核心是两个事务用户实例: 注册服务器实例和代理服务器实例, 其初始化是:

(1) 注册服务器初始化(registrar):

```
Registrar registrar; // 创建一个实现注册功能的实例
InMemoryRegistrationDatabase regData;
  // 声明一个注册信息内存数据库
SharedPtr< MasterProfile> profile(new MasterProfile);
  // 定义 SIP 信令结构
```

在这之后, 为了确保注册服务器只能接收到 REGISTER 信令, 需要以下步骤:

```
resip:: MessageFilterRuleList ruleList;
resip:: MessageFilterRule:: MethodList methodList;
methodList. push_back( resip:: REGISTER);
ruleList. push_back( MessageFilterRule( resip:: MessageFilterRule:: SchemeList(), resip:: MessageFilterRule:: Any, methodList));
```

(2) 代理服务器初始化(proxy):

代理服务器中存在一系列处理器模式, 在程序中主要有以下 3 种模式: RequestProcessor chain、ResponseProcessor chain、TargetProcessor chain, 分别负责处理请求消息、应答消息和发送请求消息. 对其进行初始化设置:

```
ProcessorChain requestProcessors( Processor:: REQUEST_CHAIN);
ProcessorChain responseProcessors( Processor:: RESPONSE_CHAIN);
ProcessorChain targetProcessors( Processor:: TARGET_CHAIN);
```

完成上述对代理服务器的处理器设置后, 要声明代理服务器的实例:

```
Proxy proxy(stack,
resip:: Uri("sip: example. com"),
requestProcessors,
responseProcessors,
targetProcessors,
store. mUserStore, 180);
stack. registerTransactionUser(proxy);
```

3.2.3　启动相应循环线程

在注册服务器和代理服务器实例分别初始化完成后,将两者启动:

```
DialogUsageManager* dum = new DialogUsageManager(stack);
DumThread* dumThread = new DumThread(* dum);
stackThread.run();
proxy.run();
if (dumThread)
{
dumThread- > run(); //所有的进程都是在 dumThread
                    //里一直循环等待直到事件发生
}
```

3.2.4 注册功能和代理功能的程序实现架构

(1)注册功能

服务器接到终端发送来的注册消息,即将该消息交由注册服务器实例进行处理,注册服务器实例根据该 UA 是否为合法用户,有接受和拒绝两种处理方式:

●void Registrar::onAdd(resip::ServerRegistrationHandle sr, const resip::SipMessage& reg);

●void Registrar::onRemove(resip::ServerRegistrationHandle sr, const resip::SipMessage& reg).

注册服务器除了具有接受、拒绝用户注册之外,还有接受用户重注册(即当用户转移到其它 IP 地址时或是注册有效期即将结束时,可向注册服务器进行重注册以便及时更新信息)和查询用户信息的处理方式:

●void Registrar::onRefresh(resip::ServerRegistrationHandle sr, const resip::SipMessage& reg);

●void Registrar::onQuery(resip::ServerRegistrationHandle sr, const resip::SipMessage& reg).

(2)代理功能

代理服务器的主要实现是通过该类的成员函数 void thread()来完成,该函数通过对 SIP 消息的判断来进行处理:

```
if(sip) //如果是 SIP 消息,将进行以下处理
{
if(sip- > isRequest()) //如果该 SIP 消息是请求消息,
                       //将进行以下处理
{
……//对 SIP 消息的结构和字段的正确与否进行判断,
//若正确将继续进行.处理,否则将删除该 SIP
//消息,并回复错误应答给 UA,让其重发.
……//根据不同的请求消息,代理服务器将进行不同的
    //处理和回复.
}
else if(sip- > isResponse()) //如果该 SIP 消息是应答消息,将进行以下处理.
{
}
……
}
```

4 系统测试及性能分析

通过局域网对该 SIP 服务器进行了协议测试、通信流程测试、性能测试和可靠性测试,各项性能指标均达到或超过同类产品,通信流程及协议实现完成符合

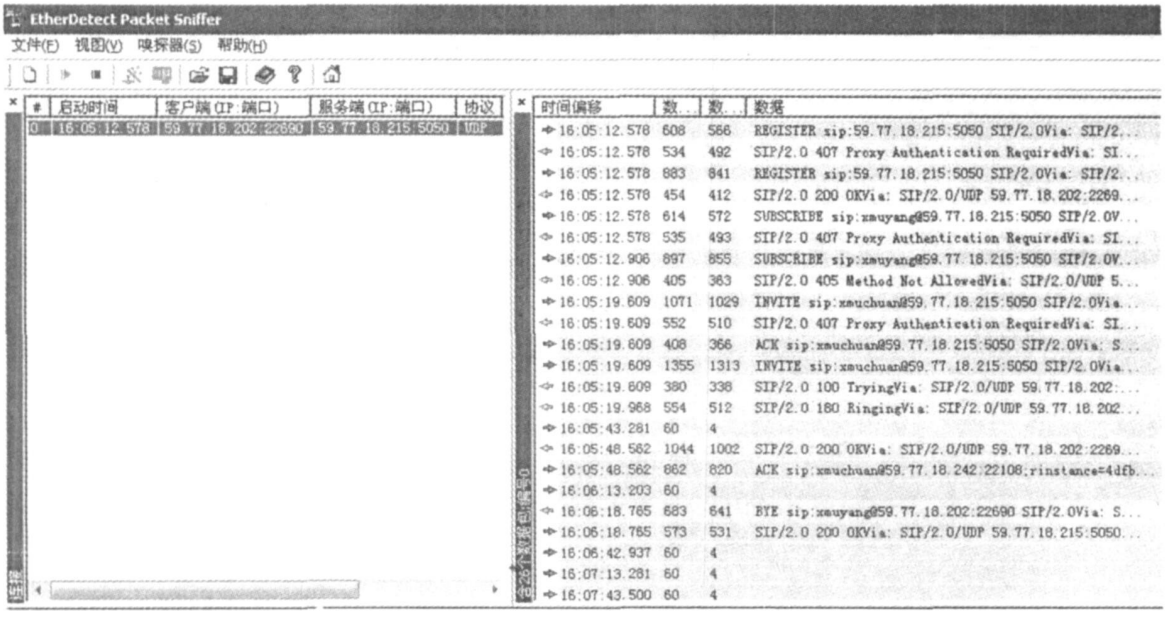

图 4 SIP 报文

Fig. 4 SIP messages

RFC3261 的规定. 图 4 是利用 EtherDetect Packet Sniffer 软件获取的用户注册及呼叫流程的 SIP 报文. 从图可见, 客户端 xmuyang 向注册服务器 59. 77. 18. 215 发送注册请求, 得到注册成功响应; 接着客户端 xmuyang 向另一客户端 xmuchuan 发送会话邀请, 邀请成功; 最后由 xmuchuan 提出结束会话请求.

从图 4 所示的时间数据也可看出, 该 SIP 服务器的处理速度非常快, 经统计得知信令转发的平均时延小于 20 ms. 其它重要的性能数据统计如下: 通过同时并发向服务器发起呼叫, 直至 SIP 服务器无法处理, 测出其呼叫处理能力为 168 次/ s, 在 2 h 的时间内, 每隔 2 s 发起一组 150 路呼叫, 呼叫持续 10 s, 统计其接通率为 99. 96% .

此外, 我们还对所设计的服务器进行了功能测试、常规测试和环境测试, 其结果均达到电信级设备的基本要求.

5　结束语

本文介绍了 SIP 服务器系统的设计思想, 并从服务器的 SIP 信令处理模块的设计出发, 重点论述了利用 ReSIProcate 协议栈实现其中注册服务器和代理服务器功能的程序设计方法. ReSIProcate 作为 SIP 开源协议栈家族中的新成员, 对于国内的 SIP 研究者来说并不十分熟悉, 国内对它的研究文献和相关产品较少. 通过研究, 我们成功地将这种稳定性很好的 SIP 协议栈应用到实际通信系统的开发中, 并取得了良好的设备性能和使用效果.

可以预见, 随着下一代网络的建设和大规模应用的来临, SIP 协议及其应用开发研究必将成为未来网络技术研究的焦点.

参考文献:

[1] Rosenberg J, Schulzrinne H. SIP: Session Initiation Protocol[S]. RFC3261, IETF, 2002.

[2] Gonzalo Canarillo. SIP 揭秘[M]. 北京: 人民邮电出版社, 2003.

[3] Handley M, Jacobson V. SDP: Session Description Protocol[S]. RFC2327, IETF, 1998.

[4] IETF. The reSIProcate SIP stack[EB/OL]. [2007-06-20]. http://www.resiprocate.org/Main_Page.

[5] IETF. The dialog usage manager (DUM) [EB/OL]. [2007-06-20]. http://www.resiprocate.org/Main_Page.

[6] 毕远国. 一种嵌入式 SIP 服务器的设计与实现[D]. 沈阳: 东北大学, 2005.

The Design of SIP Server Based on Next Generation Network

YANG Chuan, TANG Yu-liang

(Department of Communication Engineering, School of Information Science and Technology, Xiamen University, Xiamen 361005, China)

Abstract: SIP(session initiation protocol) server is a pivotal equipment in the next generation network (NGN). SIP protocol stack which is used in the designing of the SIP server can affect the performance of the server. First, the paper presents the principle of SIP, and then introduces the architecture and usage of ReSIProcate which is an open source SIP protocol written by C++ and is very stable. In the end, the paper discusses how to use Microsoft Visual Studio .NET 2003 to design a SIP server system by using ReSIProcate, and analyses the realization of the functions of registrar and proxy. Experiment proving, the design is stability and expansibility and accords with RFC3261 criterion, it meets all the requirements of designing, and it really works all right.

Key words: next generation network; SIP; reSIProcate; registrar; proxy