SNMP++开发包协议实现分析

丁 明[1,2],胡 术[1,2]
(1.四川大学 计算机学院,四川 成都 610064;
2.四川大学 视觉合成图形图像技术国防重点学科实验室,四川 成都 610064)

摘 要:随着互联网的快速发展,网络规模和数量将空前增大,网络的监控和管理将是一个挑战。简单网络管理协议是目前网络中应用最广泛的管理协议,SNMP++是HP公司开发的一套用于SNMP编程的C++开源库,提供了简单易用的接口,被广泛使用于网络管理的开发。在详细了解SNMP协议、编码规则和报文格式的基础上,详细分析SNMP++中报文收发处理过程。在报文传输过程中使用I/O复用技术和超时重传机制,提高了效率和可靠性。通过对开发包的分析,可以更好地利用SNMP++类库进行跨平台的移植和进一步的开发,满足新的网络管理需求。

关键词:简单网络管理协议;网络管理;SNMP++类库;I/O复用
中图分类号:TP31 **文献标识码**:A **文章编号**:1673-629X(2012)04-0001-04

Analysis and Implementation of SNMP++ Development Kit

DING Ming[1,2],HU Shu[1,2]
(1. Department of Computer,Sichuan University,Chengdu 610064,China;
2. National Key Laboratory of Fundamental Science on Synthetic Vision,Sichuan University,
Chengdu 610064,China)

Abstract:With the development of network,the scale of network will increase more quickly,network monitoring and management will be a challenge. SNMP (Simple Network Management Protocol) is the most widely used network management protocol,SNMP++ is a set of C++ class libraries supplied by HP company,provide an easy-to-use interface into SNMP,and widely used in the development of network management. Based on the research of SNMP,encoding rules and the types of SNMP packet,analyze the process of the transmission of the message based on the SNMP++. Using the I/O multiplexing and timeout retransmission mechanism,it greatly improves the efficiency and reliability of the transmission. Through the analysis of the development kit,it will better for use SNMP++ class library for transplantation,further development to meet the new network management requirements.

Key words:simple network management protocol;network management;SNMP++ class library;I/O multiplexing

0 引 言

随着网络技术的快速发展,网络的规模迅速增长及设备越来越多样化,“更可管理性”是下一代互联网的一个重要特征,因此网络管理和监控的需求将不断增加。

SNMP作为一种网络管理协议,能够监控和管理不同网络设备,同时具有简单性和可移植性,因此将成为未来网络管理必选的解决方案[1,2]。文中详细分析该协议的底层实现过程,从而为网络管理架构和部署奠定基础。

收稿日期:2011-08-23;**修回日期**:2011-11-26
基金项目:国家自然科学基金项目(60832011)
作者简介:丁 明(1987-),男,湖北随州人,硕士研究生,研究方向为网络和中间件;胡 术,博士,副教授,研究方向为计算机网络、操作系统及中间件。

1 SNMP简介

1.1 SNMP基本架构

SNMP是目前TCP/IP网络中应用最广泛的网络管理协议,采用UDP协议进行数据传输。在管理站和代理之间传输数据不需要建立端到端的连接。代理端在161端口侦听来自管理站的set/get命令,并给予回应;管理站在162端口侦听代理传来的“通知(inform)”和“陷阱(trap)”信息,见图1。由于UDP是不可靠的,为保证可靠性,在SNMP报文传输时采用了超

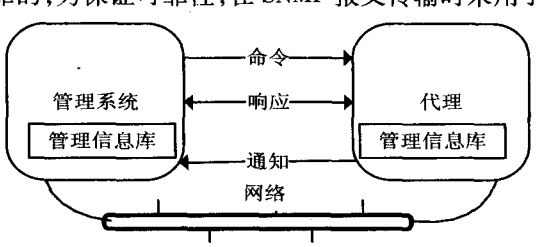

图1 管理站和代理的通信

时重传机制[3]。

1.2 SNMP++简介

SNMP++是最著名SNMP管理站端开发工具之一，其最初由HP公司开发，版本2.8a以后由www.agentpp.com网站提供后续研发和技术支持，目前的版本SNMP++3.2提供对SNMPv3的支持[4,5]。SNMP++是为网络管理应用程序开发者提供的具有SNMP服务的一套C++类的集合。它具有基于面向对象模型建立的SNMP应用程序接口。对于使用TCP/IP协议的Internet来说，SNMP++封装了底层的Socket操作，提供了简单API供开发者使用[6,7]。

2 SNMP消息处理

SNMP消息收发是将编码过后的SNMP报文发送出去并得到应答报文。在SNMP++中提供有两种工作方式：一种是同步收发机制，另一种是异步收发机制。所谓同步收发机制是指发送消息后阻塞等待应答报文的到达，在得到应答或超时之前程序不能做任何事情。所谓异步收发则是在报文发送动作执行后函数立即返回，程序可以继续运行，在应答到达时再通知程序进行处理[8]。

SNMP在消息接收的过程中使用的是UDP协议，使用的端口是161和162。Inform消息和Trap消息使用的是162。SNMP++在处理消息的过程中，设计了一个事件处理链表EventList，处理所有的事件。在事件链表中主要加入两种消息处理队列，一种称为指令消息队列(SNMPMessageQueue)，主要用来处理指令消息的发送及超时重传和消息的接收；一种称为通知消息队列(NotifyMessageQueue)，主要用来处理通知消息的接收和回复。

2.1 消息发送

在SNMP++实现中，消息处理前进行事件链表的初始化和套接口的初始化。每一个PDU(Protocol Data Unit，协议数据单元)都被赋予一个请求ID，使得同一代理的每一个请求能够被唯一识别。请求ID(request_id)使SNMP应用程序能够将发送的请求与应答联系起来，代理发送RequestPDU，经过处理后返回的ResponsePDU中的request_id字段与RequestPDU一致。request_id获取方法是初始化时随机生成一个current_rid，调用函数MyMakeReqId生成request_id。处理过程是：current_rid++，此时遍历snmpEventList，检测current_rid是否被使用。如果没有，就把current_rid赋值给request_id，否则继续增加1。如果current_rid递增超过PDU_MAX_RID，则定时等待一段时间，然后设置current_rid= PDU_MIN_RID。其初始化代码如下：

current_rid = rand()%(PDU_MAX_RID-PDU_MIN_RID-1))+PDU_MIN_RID

其中PDU_MAX_RID = 32767；PDU_MIN_RID = 1000。

2.1.1 发送和接收函数

SNMP++中请求和接收报文主要由下面几个函数完成，其中

1. 发送消息由send_snmp_reques发送：其实现通过调用函数sendto将消息发送出去。

int send_snmp_request(SnmpSocket sock, unsigned char * send_buf, size_t send_len, Address & address)

2. 接收消息由receive_snmp_response(用于接收get、set的回应)、receive_snmp_notification(用于接收inform、trap信息)接收：通过调用函数recvfrom来实现

int receive_snmp_notification(SnmpSocket sock, Snmp &snmp_session, Pdu &pdu, SnmpTarget * * target)

值得注意的是，在receive_snmp_notification中需要保存通信目标的代理(target)信息。该信息用于在接收到通知消息后，进行回调处理时使用。

3. 取消消息发送和接收：int cancel(const unsigned long request_id)在请求完成之前取消相应的请求，即从指令消息队列snmpEventList中将消息删除。函数notify_unregister()，将通知消息队列notifyEventList中消息事件删除。

2.1.2 trap消息发送

Trap消息是代理方检测到一个错误或者故障事件自动向管理站发送非请求消息。调用send_snmp_request函数将消息发送出去即可。对于trap消息的发送，判断sendto函数返回值send_result(在没有错误发生情况下，表示已发送的字节数)，如果返回值小于0，表示失败。

2.1.3 消息处理引擎

在处理所有请求消息和异步请求消息时定义了一个消息引擎函数来处理。处理消息主要有get、get_next get_bulk、set、inform、report。函数定义如下：

int snmp_engine(Pdu &pdu, long int non_reps, long int max_reps, const SnmpTarget & target, const snmp_callback cb, const void * cb)

调用过程：pdu.set_type(type)；设置PDU操作的类型，然后调用snmp_engine。

return snmp_engine(pdu, non_reps, max_reps, target, callback, callback_data)；参数根据请求消息类型进行设置，后面两个参数是回调函数及其参数。

snmp_engine函数发送处理过程如下：

(1) 对参数的预处理，主要检查参数的合法性以及参数的初始设置。

(2) 如果 action 是 get_bulk,

```
if ( pdu_action == sNMP_PDU_GETBULK) {
        pdu. set_error_status((int) non_reps);
        pdu. set_error_index((int) max_reps);
        }
```

同时生成请求报文的 ID, req_id = MyMakeReqId()。

(3) 对 SNMP 报文进行编码。

(4) 将消息事件加入消息队列中 eventListHolder->snmpEventList()->AddEntry()。

(5) 发送报文消息。如果发送失败,则从队列中删除,除了 response 和 report 消息。

(6) 发送成功。处理消息队列,接收回复及超时重传。如果是 report、response 和异步请求,直接返回 SNMP_CLASS_SUCCESS。

2.2 消息的接收和重传

2.2.1 指令事件处理

事件处理模型见图 2,指令消息在队列中处理流程如下:

1. 通过一个 while 循环来处理事件链表,循环中调用 SNMPProcessEvents 函数。循环结束条件是当前处理的消息不在队列(因为超过超时重传次数而删除)或者消息 received 状态是已接收。

2. SNMPProcessEvents 函数处理过程是首先获取(GetNextTimeout)队列中下一个最先超时事件的时间(sendtime 事件注册时间+设置的超时时间)。然后通过 select 或 poll 函数来设置 timeout。

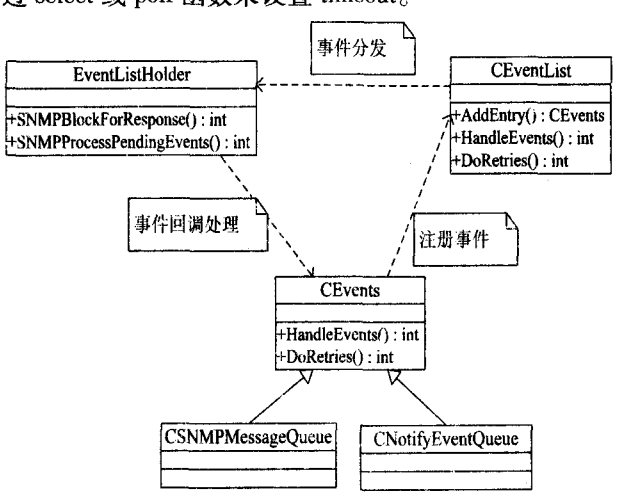

图 2　事件处理模型

3. 处理队列中事件调用函数。

SNMPProcessPendingEvents。该函数通过循环调用 select 或者 poll 函数,判断是否有 fd 处于读写状态,如果有,就处理事件 HandleEvents。如果 nfound<=0(超时或者出错),退出循环。最后调用函数 DoRetries,处理超时和重传。部分代码:

```
nfound = select ( maxfds, &readfds, &writefds,
&exceptfds, &fd_timeout);
if (nfound > 0)
status = m_eventList. HandleEvents( maxfds, readfds, writefds, exceptfds);
```

4. 如果请求事件接收到数据,从队列中获取接收到的消息。msg = m_snmpMessageQueue->GetEntry(req_id);如果 msg 为空,返回状态为超时。否则调用函数 getPdu,获取报文 PDU,然后将消息事件从队列中删除。

2.2.2 通知事件的处理

通知事件队列主要处理两种事件:一种是监听 trap 事件即接收 trap 消息,一种是接收 inform 消息并返回确认。其处理过程是首先注册监听事件,通过函数 notify_register 来实现,然后将监听事件通过函数 AddEntry 加入到 NotifyMessageQueue。然后调用线程处理函数 bool start_poll_thread(const int select_timeout)处理事件。

1. int notify_register(const OidCollection &trapids, const TargetCollection &targets, const snmp_callback callback, const void * callback_data)

注册之前,需要删除以前的会话 session 建立的过滤,notifyEventList()->DeleteEntry(this);通知消息加入队列时,需创建监听套接字 m_notify_fd,同时绑定 162 端口。

在注册的时候,需要定义回调函数来处理接收到的数据。在回调处理前 notify_filter 函数来实现过滤。过滤通过后,调用回调函数处理数据。如果是 trap 消息,就直接获取接收消息;如果是 inform 消息,获取接收消息的同时调用 snmp->response(pdu, target);返回一个确认消息。

2. 通过调用函数 CreateThread/ pthread_create 创建线程,根据平台而定。在线程函数中调用 SNMPProcessEvents 函数完成事件的监听和数据的收发。

```
while (pSnmp->is_running())
pSnmp -> eventListHolder -> SNMPProcessEvents (pSnmp->m_iPollTimeOut);
```

3. 通知事件列表处理。利用 HandleEvent 处理事件,如果有数据可读,利用函数 receive_snmp_notification 接收数据。然后调用回调函数 callback 处理接收到的数据。如果接收失败,回调处理的原因 reason = SNMP_CLASS_TL_FAILED;接收成功 reason = SNMP_CLASS_NOTIFICATION。

2.2.3 消息的重传处理

SnmpTarget 类是一个抽象类,为 CTarget 类提供接口。在 CTarget 类中主要设置目标主机的地址,设置读

写公共体,设置超时时间和重传的次数,其中公共体community是一种安全机制[9,10]。

消息引擎的输入参数SnmpTarget &target,用来设置超时信息。超时计算m_sendTime.refresh(),m_sendTime += (m_target->get_timeout() * 10)。在事件队列处理完成后,即没有读写事件发生时,首先遍历消息队列,获取超时时间与当前时间进行比较。如果超时,判断该消息m_received状态,若为接收状态,表示收到回复,此时更新超时时间。如果没有接收到回复,进行重传处理,如果重传的次数为0,表示不再重传,Callback(SNMP_CLASS_TIMEOUT),进行超时回调处理;如果重传次数不为0,将设置的retries减1,更新超时时间SetSendTime(),重新发送请求。最后将超时的消息从队列中删除。

2.2.4 I/O复用

Unix/Linux系统中的I/O复用是一种基本的I/O模型,I/O复用的实现是基于非阻塞I/O的。目前在Linux下实现I/O复用主要有三种方式:

(1)基于select调用;

(2)基于poll调用;

(3)基于epoll调用。

这里主要分析的是select函数,该函数允许进程指示内核等待多个事件的任何一个发生,并仅在有一个或多个事件经历一段指定时间后才唤醒它[11,12]。函数原型如下:

```
int select(int maxfd, fd_set * readfds, fd_set * writefds, fd_set * exceptfds, const struct timeval * timeout);
```

其中参数maxfd至少比事件队列中最大的文件描述符大1,参数timeout有三种情况:

(1)timeout设置的时间为0,检查描述字后立即返回;

(2)timeout为空指针NULL,永远等待,直到至少有一个文件描述符就绪;

(3)timeout为正整数,等待固定时间,因此可以实现一个定时器。

图3是一个事件处理的流程。

2.2.5 异步事件处理

在处理异步事件时,其实在接收到数据后,异步事件通过回调callback函数处理数据,然后从事件队列中删除。如果是异步的inform事件,则调用函数reponse,发送一个确认回复信息。通过异步函数进行操作过程如下:定义一个callback函数,处理接收的数据;调用相应异步函数;调用事件处理函数SNMPProcessPendingEvents处理异步事件。异步回调函数类型定义:

```
typedef void (* snmp_callback)(int reason, Snmp * session, Pdu &pdu, SnmpTarget &target, void * data);
```

其中,参数reason是回调产生的原因,SNMP_CLASS_TIMEOUT请求超时,SNMP_CLASS_ASYNC_RESPONSE接收到了回复信息,SNMP_CLASS_NOTIFICATION接收到了通知(trap/inform),SNMP_CLASS_SESSION_DESTROYED表示SNMP的会话销毁。

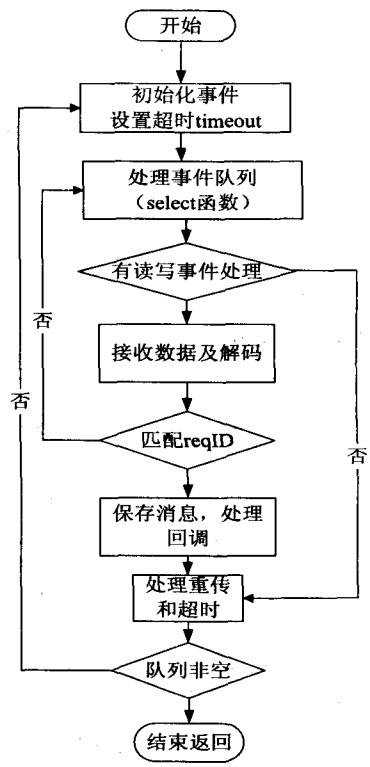

图3 事件处理流程图

3 结束语

随着目前计算机网络规模的扩大,网络管理的地位越来越重要。因此,基于SNMP++开发包,文中详细地分析了SNMP协议的报文收发处理的详细过程,提供了一个清晰的底层实现原理和过程。在此基础上,对报文的收发处理做了相关平台的测试、移植和二次开发,为将来的网络监控开发和管理奠定了基础。

参考文献:

[1] Yang Jiahai, Zhang Hui, Zhang Jinxiang, et al. Towards next generation internet management: CNGI-CERNET2 Experiences[J]. Journal of Computer Science and Technology, 2009,24(3):482-494.

[2] Wu Jianping, Ren Gang, Li Xing. Building a next generation internet with source address validation architecture[J]. Science in China (Series F: Information Science), 2008, 51(11):1681-1691.

[3] 王焕然, 徐明伟. SNMP网络管理综述[J]. 小型微型计算

(下转第8页)

```
if(a >= 5 + b)
{  a = a - 6; }
c = Fun(a,b);
a = b + c;
}
```

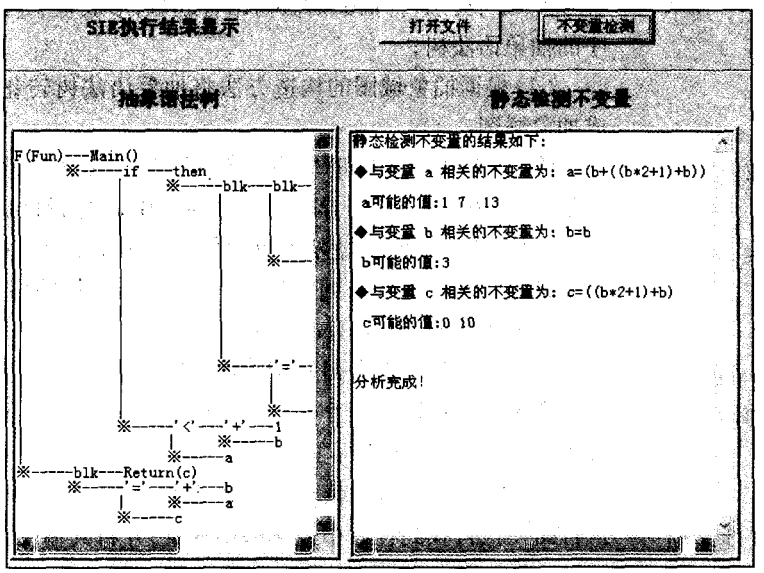

图2　SIE执行结果

4　结束语

通过文中的研究,利用抽象解释能够有效而准确地检测出程序中蕴含的非函数依赖不变量。但该方法同时也存在不足的地方,如该方法目前只针对于非函数依赖的不变量的检测。今后的工作就是针对其它类型的不变量,基于文中的思想,寻找合适的方法对其检测,最终形成一个对所有类型的不变量都适用的检测工具。

参考文献:

[1] 王　伟,刘久富,娄坚波,等. 基于多 Agent 的软件测试系统[J]. 计算机技术与发展,2011,21(4):37-39.

[2] 胡国庆. 动态检测非函数依赖程序不变量[J]. 电脑与信息技术,2008,16(2):47-50.

[3] Hangal S,Lam M S. Tracking down software bugs using automatic detection[C]//Proceedings of the 24th international conference on software engineering. [s. l.]:[s. n.],2002:291-301.

[4] 刘树锟,阳小华. 动态不变量检测工具 Daikon 的分析及应用[J]. 电脑开发与应用,2006,19(1):6-8.

[5] 刘树锟,陈继锋,阳小华. 非函数依赖程序不变量动态检测技术研究[J]. 计算机工程与应用,2008, 44(35):158-162.

[6] 陈　才. 一种区间型程序不变量检测方法[J]. 计算机与现代化,2010(3):184-187.

[7] 常　硕,赵　彬,辛文逵. 抽象解释技术在嵌入式软件测试中的应用[J]. 中国测试技术,2007,33(6):93-95.

[8] Cousot P,Cousot R. Abstract interpretation:a unified lattice model for static analysis of programs by construction or approximation of fixpoints[C]// Proc of the 4th POPL. Los Angeles: ACM Press,1977:238-252.

[9] 赵修伟. 基于抽象解释的实时软件 WCET 研究[D]. 大连:大连理工大学,2009.

[10] Cousot P,Cousot R. Abstract interpretation frameworks[J]. Journal of Logic and Computer,1992,2(4):511-547.

[11] 李梦君,李舟军,陈火旺. 抽象解释理论的程序验证技术[J]. 软件学报,2008,19(1):17-26.

[12] 杨　波,张明义,谢　刚. 抽象解释理论框架及其应用[J]. 计算机工程与应用,2010,46(8):16-20.

(上接第4页)

机系统,2004,25(3):358-366.

[4] 吴　昊,杨　凯,胡　术,等. SNMP 跨平台移植和交换机端口 IP 探测[J]. 计算机技术与发展,2008,18(11):18-21.

[5] 周志成,王　卓,汪秉文,等. 基于 SNMP++类库的简单网络管理平台的实现[J]. 计算机技术与发展,2006,16(3):158-160.

[6] 周家庆,贾　洞. 基于 Linux 的 SNMP 消息收发的实现[J]. 计算机与现代化,2003(6):26-29.

[7] 张　彤,吴世荣. 基于 SNMP 计算机网络流量监控系统研究[J]. 计算机技术与发展,2011,21(1):88-91.

[8] Stevens W R,Bill F,Rudoff A M. UNIX 网络编程第1卷:套接 API[M]. 第3版. 杨继张译. 北京:清华大学出版社,2006.

[9] 李明江. SNMP 简单网络管理协议[M]. 北京:电子工业出版社,2007.

[10] Presuhn R. Version 2 of the protocol operations for the simple network management protocol (SNMP)[S]. RFC3416,2002.

[11] 王　芬,赵梗明. 基于 SNMPv3 网络管理系统的研究和应用[J]. 计算机技术与发展,2006,16(4):199-202.

[12] 武海燕,谭成翔,汪海航. I/O 复用代理在网络隔离系统中的应用研究[J]. 计算机科学,2008,35(7):22-24.