I/O复用代理在网络隔离系统中的应用研究*)

武海燕　谭成翔　汪海航
(同济大学计算机系　上海 201804)

摘　要　应用代理为网络隔离系统中的其他业务提供了运行平台。首先通过比较传统代理技术,提出了适用于网络隔离环境的代理模型,然后分析比较了I/O复用技术,提出了在网络隔离系统中使用epoll实现应用代理的方法,并阐述了详细的实现过程。最后对系统性能进行了研究,提出了应用数据的缓存方法和基于TTL的数据重发机制。

关键词　I/O复用,应用代理,网络隔离,数据缓存

Application Research on I/O Multiplexing Proxy of Network Isolation System

WU Hai-yan　TAN Cheng-xiang　WANG Hai-hang
(Computer Department, Tongji University, Shanghai 201804, China)

Abstract　Application proxy has provided a running platform for other applications in the network isolation system. Firstly, it proposed a suitable proxy model for the network isolation system through comparing traditional proxy solutions. Secondly, it analyzed I/O multiplexing in practical use and proposed a method to realize application proxy in the network isolation system based on the epoll technology. Then, it described the realization process in detail. Finally, it researched the system performance, and proposed data caching method and re-sending mechanism based on the TTL of IP protocol.

Keywords　I/O multiplexing, Application proxy, Network isolation, Data caching

1　引言

网络隔离是网络安全防护的一种新思路,隔离系统部署于可信网(内网)与非可信网(外网)之间,将内外网划分为不同安全等级的信任域,形成了纵深的防御体系,达到了既隔离又交换的目的。通常网络隔离设备在物理上由三个处理单元构成:内网单元、专用交换单元和外网单元。

网络隔离系统的数据交换是以信息摆渡的方式实现的,且只有被系统明确要求传输的信息才可以通过,其信息流一般是通用应用服务[1,2]。用户应用数据的交换是建立在代理之上的。代理的传统实现方法有多种,但是在网络隔离系统中,其实现所采用的技术与方法与传统代理有很大区别。

本文通过分析比较传统代理模型,结合网络隔离的应用环境,提出了在单个进程中使用I/O复用技术实现网络隔离系统中应用代理的方法,然后给出了详细的设计流程和主要功能的实现,最后对系统性能进行了分析研究,提出了应用数据的缓存方法和基于IP协议中TTL设计思想的数据重发机制。

2　相关技术概述

2.1　代理技术

依据代理在TCP/IP参考模型中不同的工作位置,可以把传统代理技术划为分以下几类:①应用层代理;②传输层代理;③SOCKS代理[3]。

其中传输层代理工作在TCP/IP参考模型的传输层,其主要思想是分别代理传输层协议(如TCP,UDP等),对于应用层业务交换则是通过有机地组合传输层各协议的代理而完成的,因此传输层代理的思想相对来说更适合于网络隔离系统。

但是,传统的代理服务程序通常运行在一台服务器上,客户端与服务器的连接可由进程或线程的ID标识,这样可以很好地保持客户端与服务器的会话,也可以很方便地处理各类运行错误。但是网络隔离系统中代理的客户端和服务器端运行于两个不同的物理单元上,代理模型与传统代理大不相同,不能简单把传统代理的实现方法直接应用于网络隔离系统[4]。因此,详细研究网络隔离系统中的代理模型及其需要解决的关键技术是系统实现的首要步骤。

2.2　I/O复用技术

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[5]。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。

epoll是在Linux2.5.44内核中引入的,2.6内核对其做了进一步改进。epoll实现I/O复用虽然比前两者复杂,但是它解决了前两种方法所共有的性能瓶颈,主要表现在以下几个方面:

(1)epoll能同时处理大量文件描述符,而select只能处理FD_SETSIZE个(Linux中为1024)。

(2)epoll对I/O事件的轮询频率比前两者高。

(3)epoll在每个文件上设置了回调函数,仅当事件发生时回调函数才被调用,因此在处理大量I/O时,系统性能下降比前两者小得多[6]。

因此,epoll比select和poll更适用于网络服务产品,本文的I/O复用将采用epoll来实现。

*)国家信息安全专项基金资助(编号:国家发改委批文[2005]1879号)。**武海燕**　博士生,主要研究计算机网络、信息安全;**谭成翔**　研究员,博士生导师,主要研究计算机网络、信息安全;**汪海航**　教授,博士生导师,主要研究网络安全、电子商务。

3 网络隔离系统中的代理模型

3.1 体系结构

由上所述，网络隔离系统由三个物理单元组成，代理功能在网络隔离系统中的分布如图1所示。

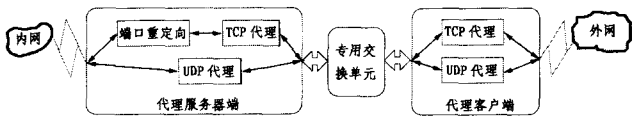

图1 网络隔离系统中的代理模型

3.2 模型描述

在图1中，代理模块分布于内、外两个物理单元上，每个单元又包含TCP代理和UDP代理两个子系统，图中主要模块功能如下所述：

(1)端口重定向。应用层的各类服务通常是与端口绑定的，因此要完成对应用服务的代理，首先要把访问这些端口的数据重新定向到一个统一的端口，然后代理的服务器端程序就可以在此端口启动监听服务。在Linux中，可由iptables中的表nat来实现此项功能，配置规则如下：

iptables-t nat-A PREROUTING-i eth0-p tcp-j REDIRECT-to-ports PORT

PORT即代理服务器端程序要监听的端口。

(2)TCP代理。用于处理TCP协议提供的应用数据，分布于内外两个单元上。TCP是面向连接的可靠的数据传输协议，TCP代理除了完成数据转发外，更重要的是要保持实际的客户端与服务器的连接，即二者的会话。

(3)UDP代理。UDP是一种无连接的传输层协议，因而对UDP的代理只需采用存储转发方式即可，其实现原理与nat类似，本文不做重点讨论。

(4)专用交换单元。主要负责数据的存储转发，不关心协议类型及数据的具体格式等。

4 I/O复用代理的实现

4.1 主要数据结构

为便于下面描述，首先介绍一下系统中用到的主要数据结构。

(1)数据交换格式

代理服务器端与代理客户端数据交换的格式统一定义为

```
typedef static struct
{
  int packettype; //包类型
  int totallen; //总长度
  int clifd; //客户端FD
  int srvfd; //服务器端FD
  struct sockaddr_in cliaddr; //客户端地址
  struct sockaddr_in srvaddr; //服务器地址
  char data[DATALEN]; //应用数据
}PACKET;
```

(2)数据包类型

数据交换格式的第一项为数据包类型，取值如下：

```
enum
{
  CL_NEWCONN, //新连接
  CL_DATA,   //客户端数据
  CL_RDCLOSE,   //客户端读关闭
  CL_WRCLOSE,   //客户端写关闭
  SV_DATA,   //服务器数据
  SV_RDCLOSE, //服务器读关闭
  SV_WRCLOSE //服务器写关闭
};
```

(3)缓存结构

对应用数据的缓存及对缓存的控制定义为

```
typedef struct cache
{
  bool SVRDClose; //服务器读关闭
  bool SVWRClose; //服务器写关闭
  bool CLRDClose; //客户端读关闭
  bool CLWRClose; //客户端写关闭
  int ttl;  //生存时间
  int DataLen; //缓存数据大小
  list<char *> Data; //缓存数据内容
} CACHE;
```

4.2 代理服务器端的实现

4.2.1 处理流程

代理服务器端TCP代理流程如图2所示。

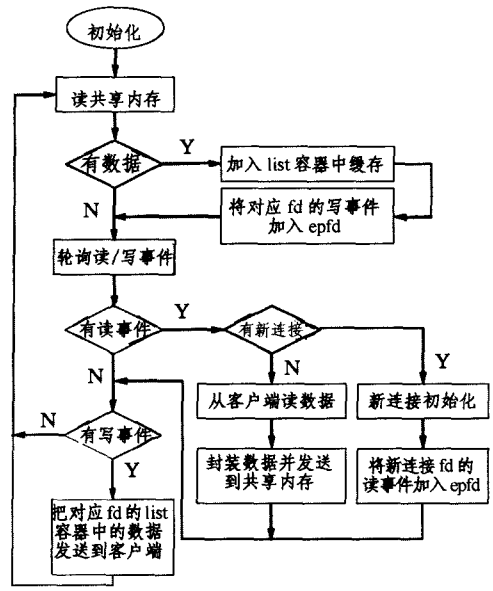

图2 服务器端TCP代理工作流程

4.2.2 流程描述及其实现

图2中，fd为socket文件描述符，epfd为epoll文件描述符。各步骤的详细功能如下所述：

(1)初始化。主要包括：①将共享内存挂接到该进程；②建立服务器监听socket并绑定到本地地址与重定向后的端口；③epfd的创建；④设置该socket为非阻塞工作模式并开始监听。

(2)读共享内存。应用数据由代理进程的一端接收，经过安全性验证后，再由另一端发送，其间将有多个进程读写应用数据，各进程间对应用数据的共享采用共享内存实现，对共享数据的互斥操作采用信号量实现。

(3)如果共享内存有数据，表示外网有数据返回，并且完成了安全性验证，即将要发送到内网，此时要把该数据块加入list容器中缓存，设置数据块所属文件描述符为可写。

(4)轮询读写事件。采用epoll_wait调用检查是否有读写事件准备好，代码为

epollnum=epoll_wait(epfd,evs,MAXCONN,0))

返回值epollnum为准备好读或写的文件描述符的个数，evs为epoll_event类型的数组，元素个数为MAXCONN，属于值-结果(value-result)型的参数，epoll_wait调用后通过访问evs可获得准备好的文件描述符的信息。

(5)有读事件。读事件包括两类：有新连接和socket有数据可读如果有新连接，则将新连接的描述符的读事件加入epoll文件描述符，采用epoll_ctl系统调用实现。代码为

ev.events = EPOLLIN;

ev.data.fd = fd;

epoll_ctl(epfd,EPOLL_CTL_ADD,fd,&ev);

如果没有新连接,表示有 socket 接收缓冲区有数据,此时即可读该块数据,然后填入共享内存,备其他进程处理。

(6)新连接初始化。主要包括:数据交换格式 PACKET 的设置、清空对应 list 容器、设置 socket 为非阻塞等。

(7)有写事件。表示客户端 socket 的发送缓冲区有剩余空间,调用 send 即可发送应用数据。与读事件不同,写事件要同时满足两个条件:有应用数据可写和 socket 的发送缓冲区有空间接收。因此,在图 2 中,当有新连接时只设置相应文件描述符的读事件。

在代理服务器端和代理客户端都要判断 socket 文件描述符是否可读或可写,在第(4)步中的 evs 结构中有一个 events 变量,如果要判断是否可读,则使用 events 与 EPOLLIN 进行位"与"运算;如果返回 EPOLLIN,则 events 至少可读。可写的判断方法同上。

4.3 代理客户端的实现

4.3.1 处理流程

代理客户端 TCP 代理流程如图 3 所示。

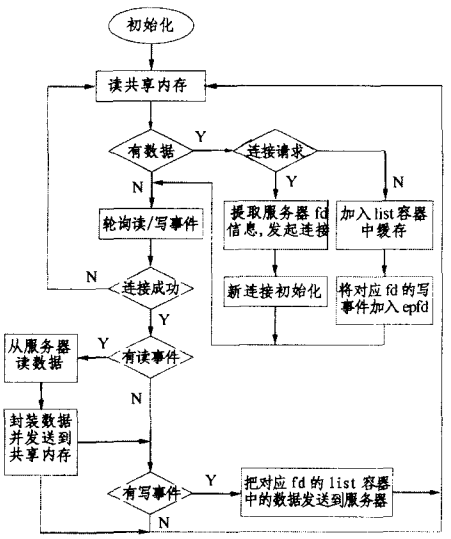

图 3 客户端 TCP 代理工作流程

4.3.2 流程描述及其实现

代理服务器端连接的发起是被动的,而代理客户端连接的发起是主动的,因此二者在处理流程和实现细节上都存在差别,下面只叙述二者的不同之处:

(1)初始化。主要包括:①将共享内存挂接到该进程;②epfd 的创建。

(2)连接请求。在代理服务器端接收到的第一个应用层数据包是用来发起新连接的,在本系统中该类型的包被标示为 CL_NEWCONN。代理客户端通过检测包类型来确定是否是新连接。如果是新连接,则提取服务器信息,调用 connect()连接服务器。因为 socket 为非阻塞工作方式,所以 connect()会立即返回。

(3)连接成功与否的判断。在非阻塞式工作方式下,connect 会立即返回 EINPROGRESS 错误,TCP 三路握手在内核中进行。对连接是否成功的判断可采用另一个 epoll 实现,判断规则为:①当连接成功时描述字变得可写;②当连接建立遇到错误时,描述字变得既可读又可写。

4.4 会话的保持

TCP 协议是面向连接的,除了客户端与代理服务器端、代理客户端与服务器保持连接外,代理服务器端与代理客户端也要保持连接,这样才能保证客户端与服务器会话顺利地进行。在此,会话的保持包括连接的建立、数据的传输、半连接的正常与异常关闭(shutdown)以及全连接的正常与异常关闭(close)四个阶段。

4.1 节中的数据结构 PACKET 可用来控制整个 TCP 的会话过程。包类型分配如下:

(1)用作数据交换的数据包类型有:①新连接 CL_NEWCONN;②客户端数据 CL_DATA;③服务器数据 SV_DATA;

(2)其余标志用作控制信息,应用数据长度为 0,包长度为 48 字节。

TCP 为全双工通信,因此对于连接的关闭也就有全连接与半连接之分,主要遵循以下 4 条规则:

(1)如果客户端写关闭,则关闭代理服务器端的连接,并通知代理客户端关闭对应于服务器的连接;

(2)如果客户端读关闭,则关闭代理服务器端的读连接,并通知代理客户端关闭对应于服务器的写连接;

(3)如果服务器读关闭,则关闭代理客户端的连接,并通知代理服务器端,代理服务器端检查对应文件描述符的缓存是否为空,空则关闭连接,非空则标识为最后一次发送,发送完毕关闭连接。

(4)如果服务器写关闭,则代理客户端关闭对应的写连接,并通知代理服务器端,关闭对应的读连接。

5 性能研究

5.1 数据缓存方法

网络隔离系统处理的数据来自于 TCP/IP 应用层,这些数据已经没有了 TCP/IP 包的格式,每次接收的最大值是通过用户程序设定的。因此,对于基于 TCP 的业务来说,接收到应用数据就表示 TCP 确认过程已经完成。此时如果应用数据发生错误或丢失,不论是客户端还是服务器都无法请求重发,只能重新建立连接,发送所有数据。

对于一个非阻塞的 TCP 套接口而言,如果其发送缓冲区中没有剩余空间时,输出函数调用将返回一个 EWOULDBLOCK 错误。此时要求把应用数据缓存,保证应用数据不被人为地丢失,直到 epoll 检测到缓冲区可写时,再进行发送。本系统中的代理数据的缓存采用 list 容器实现。list 容器是一个单向链表,可以完成 FIFO 功能,每当接收到应用数据时就将该数据块加入 list 容器末尾。实现代码为

```
char * p=new char[packetlen];
memset(p,0,packetlen);
memcpy(p,phead->data,packetlen);
cache[phead->clifd].Data.push_back(p);
```

其中 phead 为 4.1 中的 PACKET 结构体,cache 数组类型为 CACHE 结构体,pachetlen 为接收到的应用数据长度,phead->data 为应用数据内容。

当 epoll 检测到套接口可写时,则循环地将 list 中的数据块逐一发送到套接口发送缓冲区,直至检测到 EWOULDBLOCK 错误,表示缓冲区已满,则退出循环,然后再调用 epoll 不断检查。如果检测到对方套接口出错(如意外关闭)而不能写任何数据时,则清除对应 list 中的全部内容,并释放

(下转第 35 页)

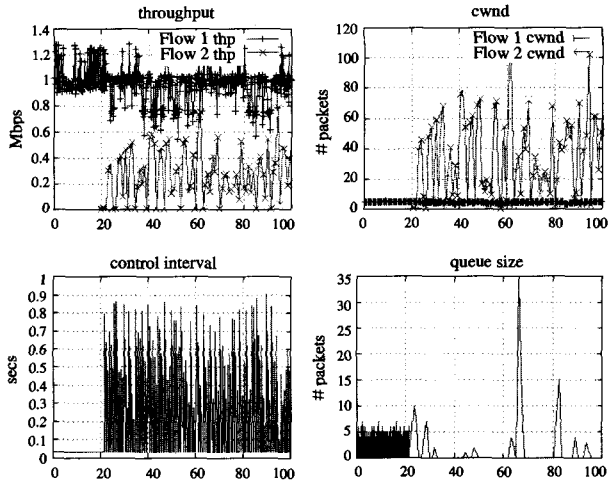

图 5　大 RTT 数据流加入系统的性能

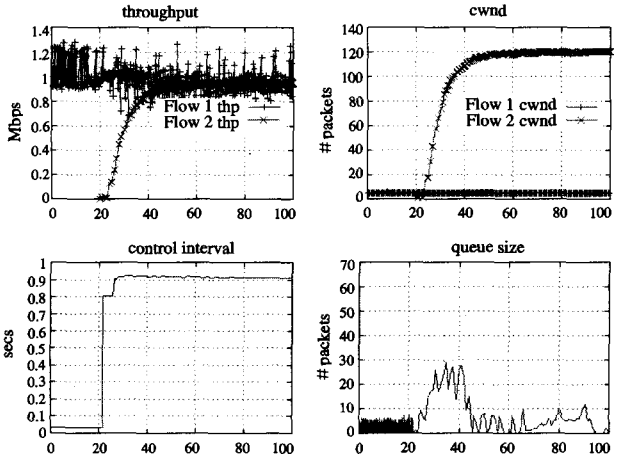

图 6　改进后大 RTT 数据流加入系统的性能

结束语　现有 XCP 协议对数据流的 RTT 差异有一定的鲁棒性，当数据流差异比较大时会导致系统性能恶化。本文通过分析数据 RTT 和控制周期之间的关系，提出了一种基于加权平均的自适应调整控制周期的改进方案。该方法没有给路由系统带来更多的计算负担，有效提高了系统对 RTT 差异的鲁棒性。分析表明，Shuffle Bandwidth 在网络高负载时决定着系统收敛的速度。我们将继续研究通过修改 Shuffle Bandwidth 的计算方法来提高系统在网络高负载时对新加入数据流的反应时间。

参 考 文 献

[1] 张慧翔，戴冠中，等. 具有显示反馈的拥塞控制系统研究进展. 计算机科学(已录用)

[2] Katabi D, Handley M, Rohrs C. Congestion control for high bandwidth-delay product networks. ACM SIGCOMM Computer Communications Review, 32(4): 89-102

[3] Katabi D. Decoupling Congestion Control and Bandwidth Allocation Policy with Application to High Bandwidth-Delay Product Networks. Ph. D. Dissertation. Massechussetts Institute of Technology, March 2003

[4] Falk A, Katabi D, Pryadkin Y. Specification for the Explicit Control Protocol (XCP). draft-falk-xcp-02. txt (work in progress), November 2006

[5] Chiu, Jain. Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks. Journal of Computer Networks and ISDN, 17(1): 1-14

[6] Balakrishnan H, Dukkipati N, McKeown N, et al. Stability Analysis of Explicit Congestion Control Protocols. Stanford University Department of Aeronautics and Astronautics Report: SUDAAR 776, September 2005

[7] The network simulator ns-2. 30. http://www.isi.edu/nsnam/ns

[8] Allman M, Paxson V, Stevens W. TCP Congestion Control. RFC 2581. April 1999

[9] Allman M, Floyd S, Partridge C. Increasing TCP's Initial Window. RFC 3390, October 2002

[10] Welzl M. Network Congestion Control. John Wiley & Sons Ltd, 2005

(上接第 24 页)

该套接口占用的其它资源。

5.2　数据重发机制

数据重发机制是利用相应的缓存数据来保证应用数据更加准确无误地到达目的地的。send 调用出错时，有两种可能：

一是对等端(IP 地址与端口)已经异常关闭，此时已无法重发，缓存的数据也没有任何用处，此时的主要操作有：

(1)清空文件描述符指示的 list 容器的内容；

(2)发送客户端/服务器已经关闭写的类型包；

(3)从读/写事件中清除该文件描述符；

(4)根据 4.4 判断规则使用 close 或 shutdown 调用关闭 socket 连接。

二是发送缓冲区已满或剩余空间太小不足以容下缓存中数据块，因此要保存未成功发送的数据等待重发。重发的次数由系统全局变量(SENDTIMES)设定，每个连接的 CACHE 初始化时，都要把它的成员变量 ttl 设置为 SENDTIMES。发送每出现一次非 EWOULDBLOCK 错误时，ttl 自减 1。当 ttl 等于 0 时，则认为对等端已无法正确接收任何数据，可以进行连接的关闭和资源回收操作了。

结束语　本文研究了采用 I/O 复用技术来实现网络隔离系统中应用代理的方法。使用基于 I/O 复用技术比采用其他技术有着更明显的优势，主要表现在：首先由文中分析可知，I/O 复用技术更适合于网络隔离系统的代理模型；其次，I/O 复用技术避免了使用多线程(进程)时系统进行线程(进程)切换时带来的时间开销，同时也比多线程(进程)方案容易控制；再次，I/O 复用技术把读写事件，接收连接和发起连接等过程交给内核处理，效率上有着显著的提高；最后，使用 I/O 复用传输多路数据的方案更加容易集成其他安全策略，如内容关键字检索、头域关键字过滤以及其他访问控制策略等。因此，从易操作性、安全性、可控制性以及运行效率等方面分析可知，基于 I/O 复用技术实现的代理更适合于网络隔离系统。

参 考 文 献

[1] GB/T 20279-2006. 信息安全技术网络和终端设备隔离部件安全技术要求

[2] GB/T 20277-2006. 信息安全技术网络和终端设备隔离部件测试评价方法

[3] Fung K P, Chang R K C. A Transport-level Proxy for Secure Multimedia Streams. IEEE Internet Computing, November/December 2000

[4] Lindskog S, Grinnemo K-J, Brunstrom A. Data Protection Based on Physical Separation Concepts and Application Scenarios // Gervasi O, et al., eds. ICCSA 2005, LNCS 3483, 2005: 1331-1340

[5] Stevens W R, Fenner B, Rudoff A M. Unix Network Programming // Third Edition. Volume 1, The Sockets Networking API. Addison-Wesley, 2004

[6] Johnson M K, Troan E W. Linux Application Development. Second Edition. Addison-Wesley, 2005