

基于 Linux 的传输协议转换模块的设计和实现

杨晓晨,董平,张宏科

(北京交通大学 电子信息工程学院,北京 100044)

摘要:随着互联网的不断发展,涌现出一些具有优异特性的协议栈,例如流控制传输协议、数据报拥塞控制协议。现有的应用程序并不能兼容这些新的协议栈,因此难以享受到新协议带来的好处。文中提出一种设计,在不修改应用程序的情况下使其基于新的协议栈进行通信,从而使现有应用能够方便地享受到新协议的特性,并为新旧网络协议栈的过渡带来便利。采用 SCTP 协议作为蓝本,设计并实现了一种 TCP 到 SCTP 的转换模块,使现有基于 TCP 的应用程序能够使用 SCTP 进行通信,最后采用 WEB 浏览器这一应用程序对协议转换模块的功能进行验证,实验结果证明了协议转换模块的可行性。

关键词:网络协议栈;流控制传输协议;传输控制协议;协议转换;动态加载

中图分类号:TP393.03

文献标识码:A

文章编号:1673-629X(2011)11-0113-04

Design and Implementation of Transport Protocol Conversion Module Based on Linux

YANG Xiao-chen, DONG Ping, ZHANG Hong-ke

(School of Electronic and Information Engineering, Beijing Jiaotong University, Beijing 100044, China)

Abstract: With the continuous development of the internet, it is a fact that there are some new internet protocol stack with excellent features, such as stream control transmission protocol, datagram congestion control protocol. Legacy applications are not compatible with the new protocol stack, making it difficult to enjoy the benefits of the new advantages. It presents a design, which without modifying the application itself, makes legacy applications communicate based on the new protocol, and easily enjoys the characteristics of the new features. This will bring the convenience of the transition of old and new network protocol stack. Take SCTP as the blueprint, design and implement a conversion module of TCP to SCTP, making the legacy applications to call SCTP instead of TCP. In the end use WEB browser to test the functions of the protocol conversion module. The results prove the feasibility of protocol conversion module.

Key words: internet protocol stack; SCTP; TCP; protocol conversion; dynamic loading

0 引言

随着互联网业务的多样化发展,作为传输协议的 TCP/UDP^[1]暴露出自己各方面的缺点,例如头端阻塞、SYN(一种 DOS 攻击)攻击等^[2],针对这些弊端,新的传输协议 SCTP^[3](Stream Control Transmission Protocol)、DCCP^[4](Datagram Congestion Control Protocol)能够很好地克服这些缺点,并且具有更好地性能和特性,例如多流^[5](Multi-streaming)和多宿主^[6](Multi-homing),经过改进的 SCTP 协议还能支持多路径并行传

输^[7](CMT),从而带来更高的网络吞吐量^[8]。尽管新的传输协议具有这些优秀的特性,人们还是很难享受到新协议带来的好处,因为现有的互联网各种各样的应用程序都是基于 TCP/UDP 运作的,必须通过修改应用程序源码的方式才能支持新的协议。修改所有的应用程序使其支持新的协议族和传输协议,是一项非常繁琐的任务,同时很多应用是以二进制发布的,并未给出源码,对这些应用进行修改代码并重新编译是不现实的。

基于上述情况,文中提出一种设计,在不修改应用程序代码的情况下,使应用程序支持新的协议族、新的传输协议,从而给推广新的更为优秀的传输协议带来便利。

1 方案比较与分析

分析 Linux 的系统架构,如图 1 所示。

Linux 中分为用户(或应用程序)空间和内核空间

收稿日期:2011-04-22;修回日期:2011-07-26

基金项目:国家自然科学基金重点项目(60833002);北京市自然科学基金重点(4091003)

作者简介:杨晓晨(1987-),男,硕士研究生,研究方向为下一代互联网;张宏科,博士,教授,博士生导师,主要研究方向有基于 IPv6 的路由协议、网络安全、组播理论与技术、新一代移动互联网络路由协议理论与技术等。

两部分。用户空间是用户应用程序执行的地方。用户空间之下是内核空间, Linux 内核正是位于这里。

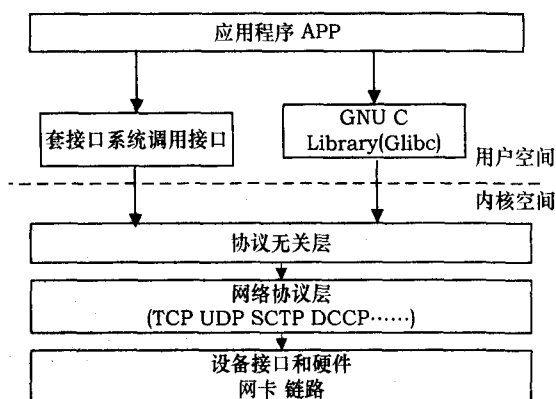


图 1 Linux 系统架构

GNU C Library (Glibc) 提供了应用程序连接内核的系统调用^[9]接口, 还提供了在用户空间应用程序和内核之间进行转换的机制。应用程序通过系统调用接口进入内核, 它实现了一些基本的功能, 例如 read 和 write。系统调用接口之下是内核代码, 可以更精确地定义为独立于体系结构的内核代码。

对于用户空间的应用程序, 一种是经过了 Glibc 进入内核, 另一种是直接通过系统调用再进入内核。现有的网络应用的程序也是这样, 这些应用程序通过调用 Glibc 库里的函数, 进入到内核空间的网络协议栈, 然后根据相应的网络协议栈进行数据的分组转发。

针对 Linux 的这种架构, 有两种方案完成协议转换这一功能。

现有方案: 修改 Linux 内核的网络协议栈中的部分代码, 应用程序进入内核时, 根据需求, 智能地选择各种协议栈进行数据的处理和转发。内核空间的修改可以参考 Ryan W. Bickhart 等人所完成的 TCP-to-SCTP Translation Shim Layer^[10]的工作, 该设计在内核中建立隐藏的 SCTP 套接口与 TCP 相对应, 通过各种状态判断等选择合适的传输协议进行数据的分组转发。

文中的方案: 在用户空间通过动态加载库的手段, 绕过系统的 Glibc, 调用修改过的库函数直接进入内核中新的网络协议栈。

在用户空间或是内核空间做修改, 这两者各有自己的优缺点。在内核做修改, 通用性比较好, 但是内核修改的地方比较多并且复杂, 同时对于普通用户使用也比较繁琐, 必须先编译内核, 才能使用协议转换的功能, 这对新协议栈的推广并无好处。

在用户空间使用动态加载的方法, 用户只需要运行一个脚本控制程序的启动就能完成这一协议栈转换的过程, 这种方法实行起来十分的方便、简单。文中的设计选择第二种方案。

Linux 中, 利用 LD_PRELOAD 这个环境变量, 可以影响程序的运行时的链接 (Runtime linker), 它允许在程序运行前定义优先加载的动态链接库。这个功能主要就是用来有选择性地载入不同动态链接库中的相同函数。通过这个环境变量, 可以在主程序和其动态链接库的中间加载别的动态链接库, 甚至覆盖正常的函数库功能来使用自己的或是更好的函数 (无需别人的源码)。

文中正是基于这样的思路设计出传输协议转换模块, 如图 2 所示。

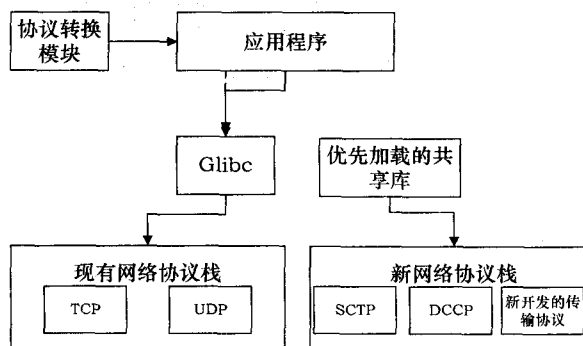


图 2 协议转换模块示意图

现有的应用程序使用 Glibc 中的 socket 函数进入到内核空间, 调用 TCP/UDP 各种处理函数, 进入原网络协议栈。加上协议转换模块以后, 应用程序绕过 Glibc, 进入修改过的动态库, 使用的也是修改过的 socket 函数, 直接进入新的网络协议栈处理数据包。

这样的设计理念并非只能在 Linux 中实现, 如果 Windows 的内核中也能支持 SCTP 等新协议, 使用一套对应的绕过 Glibc 的 API 函数接口, 同样可以在 Windows 下设计优先加载的共享库, 从而使现有的 Windows 应用程序采用新协议进行通信。

2 方案实现

动态加载允许应用程序在运行过程的任何时候去加载和使用指定的库。使用 dl 系列的一组接口函数^[11], 使得应用程序可以采用动态加载的技术。共享库流程如图 3 所示。

```
函数: void _net_load_libs(void)
{
    if (NULL != lib_handle) return;
    if (! (lib_handle = dlopen("libc.so", RTLD_
        LAZY)))
    {
        if (! (lib_handle = dlopen("libc.so.6", RTLD_
            LAZY))) {
            fprintf(stderr, "error loading libc! \n");
            exit(1);
        }
    }
}
```

```

}

if (! (real_socket = dlsym (lib_handle, "socket")))

fprintf(stderr, "socket() not found in libc! \n");
exit (1);

/* 其他需要修改的网络接口 API, 如 bind、setsockopt、accept 等 */ .....

```

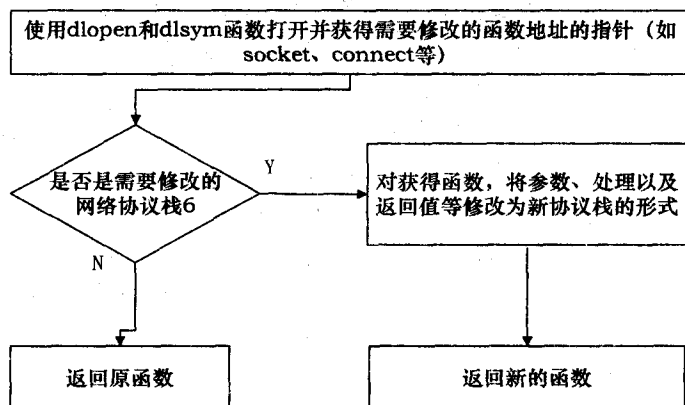


图 3 共享库流程图

功能描述:使用 dlopen 函数将网络接口 API 对应的库加载到内存,再用 dlsym 获得指定函数在内存中的位置(指针)。dlopen 必须在 dlerror, dlsym 和 dlclose 之前调用,表示要将库装载到内存,准备使用。如果要装载的库依赖于其它库,必须首先装载依赖库。如果 dlopen 操作失败,返回 NULL 值;如果库已经被装载过,则 dlopen 会返回同样的句柄。在 dlopen 之后,库被装载到内存。dlsym 可以获得指定函数(symbol)在内存中的位置(指针)。如果找不到指定函数,则 dlsym 会返回 NULL 值。在这里使用 dlsym 获得如 socket、bind、accept 等接口函数。

利用这些指针,对 socket、bind、connect、accept 等一系列与网络协议栈相关的函数原型进行修改,将这些函数中网络协议栈的参数改为新协议栈需要的参数。

例如针对原来的协议族是 v4 改为 v6,需要把所有函数参数中的 AF_INET 改为 AF_INET6,并且要将原来的 sockaddr_in 的地址格式改为相应的 sockaddr_in6 的地址格式。涉及到协议族的转换,修改上要稍许麻烦一些,但是实际上很多应用程序本身能够很好地同时支持 v4 和 v6,所以通常要修改的地方仅仅是传输协议。传输协议的转换不涉及协议族,修改起来更为方便一些。

文中的测试例子是将 v4 和 v6 的 TCP 全部修改为使用 SCTP 传输协议。部分代码如下:

```

int socket(int domain, int type, int protocol)

{
    _net_load_libs(); /* 获的 socket 函数在内存中的位置 */
    if (((PF_INET == domain) || (PF_INET6 == domain)) && (SOCK_STREAM == type)) {
        protocol = IPPROTO_SCTP;
    }

    return (real_socket)(domain, type, protocol);
}

```

相对应的把 accept、bind 等函数也修改为新协议栈的模式,在将这些函数编译成为动态加载的共享库。

最后使用脚本^[12]去控制环境变量 LD_PRELOAD 在函数运行前去优先加载共享库,如图 4 所示。

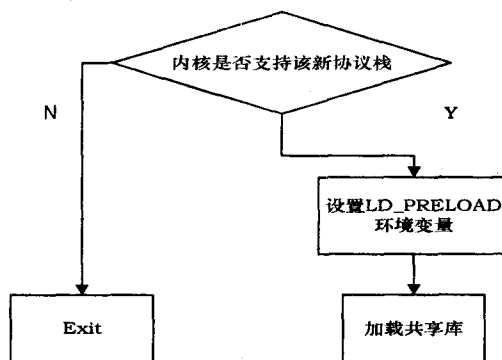


图 4 控制 LD_PRELOAD 加载动态库

控制脚本如下:

```

#!/bin/sh
LIBDIR=/usr/local/lib/
BINDIR=/usr/local/bin
if ! ${BINDIR}/checksctp 2> /dev/null
then
    export LD_PRELOAD=${LIBDIR}/change.so
    ${BINDIR}/checksctp;
else
    exit 1;

```

首先判断内核是否有 SCTP 内核,如果有就设置环境变量使程序优先加载已经修改过的库函数,达到转换传输协议的目的,否则直接运行原来的程序。

3 测试结果

采用两台 Linux 的主机进行测试,一台提供 web 服务,地址为 192.168.0.133,另一台使用协议转换模块启动 firefox 进行 web 测试,地址为 192.168.0.135,验证协议转换模块的功能。

使用转换模块启动浏览器 #change firefox, 在地址栏输入服务器的地址 `http://192.168.0.133`, 显示了正常的 apache 测试页面, 接下去再访问一张较大的图片, 都能正常的显示。下面是使用 wireshark 所抓到的数据包, 如图 5 所示。

1	0.000000	192.168.0.135	192.168.0.133	SCTP	INIT
3	0.077460	192.168.0.135	192.168.0.133	SCTP	COOKIE_ECHO
4	0.079797	192.168.0.133	192.168.0.135	SCTP	COOKIE_ACK
5	0.235600	192.168.0.135	192.168.0.133	SCTP	DATA
6	0.238084	192.168.0.133	192.168.0.135	SCTP	SACK
2	0.357364	192.168.0.133	192.168.0.135	SCTP	INIT_ACK
7	0.466037	192.168.0.133	192.168.0.135	SCTP	DATA
8	0.466463	192.168.0.133	192.168.0.135	SCTP	DATA
9	0.466621	192.168.0.133	192.168.0.135	SCTP	DATA
10	0.544945	192.168.0.135	192.168.0.133	SCTP	SACK
11	0.546328	192.168.0.135	192.168.0.133	SCTP	SACK
12	0.546524	192.168.0.135	192.168.0.133	SCTP	SACK
153	7.646465	192.168.0.135	192.168.0.133	SCTP	SACK
154	7.646477	192.168.0.135	192.168.0.133	SCTP	SACK
155	7.646488	192.168.0.135	192.168.0.133	SCTP	SACK
156	22.279646	192.168.0.133	192.168.0.135	SCTP	SHUTDOWN
157	22.280851	192.168.0.135	192.168.0.133	SCTP	SHUTDOWN_ACK
158	22.281205	192.168.0.133	192.168.0.135	SCTP	SHUTDOWN_COMPLETE

图 5 数据包统计

在上述的数据包中, 可以看到建立连接的四次握手(数据包 1~4), 数据的传输以及最后关闭连接(数据包 156~158)的过程。测试结果说明使用这一协议转换模块, 确实将原本使用 TCP 传输协议的浏览器转换为使用新的传输协议 SCTP, 验证了所设计方案的功能。

接下去使用转换模块, 协议栈使用 SCTP-CMT, 在端到端 1.5Mbps 带宽, 35ms 延迟的网络状况下下载不同大小的文件, 与现有应用程序进行比较, 结果如图 6 所示。横坐标显示的是下载文件的大小, 纵坐标为该文件的传输时间。可以看到在不修改应用程序的前提下, 使用转换模块, 应用程序享受到了新协议栈优异的传输性能。

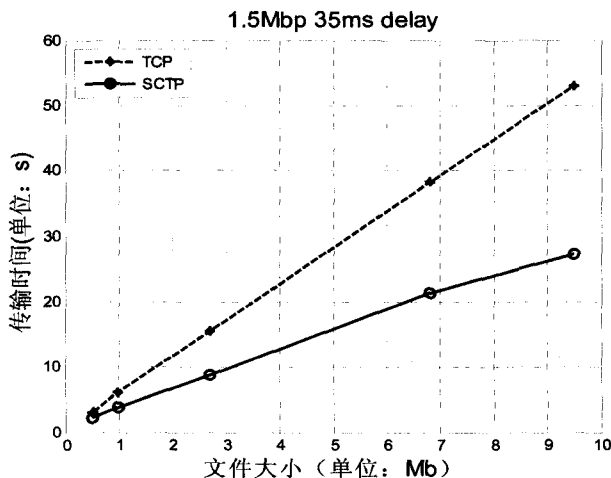


图 6 文件传输时间

4 结束语

文中从 Linux 的系统架构出发, 设计了一种独立于应用程序外的协议转换模块。此设计以绕过 GLIBC 的方式动态加载新的函数库, 从而达到转换传输协议的目的。

使用这一转换模块, 能够简单灵活地使现有的应用程序使用新的传输协议作为数据传输的载体, 这对新的协议栈的推广、应用等都带来了便利, 让更多的应用程序做很少的修改甚至不做修改就能享受新协议所带来的特性。

更进一步来说这样的设计理念并非只能用于修改协议栈, 当对应用程序的某一块功能不满意时, 也可以采用类似的方法将这一部分的功能写成一个库, 然后绕过原来的库函

数调用新的库, 从而获得新的特性。

参考文献:

- [1] 赵 飞, 叶 震. UDP 协议与 TCP 协议的对比分析与可靠性改进[J]. 计算机技术与发展, 2006, 16(9): 219-221.
- [2] 夏 云, 孙力娟, 叶晓国, 等. SCTP 协议分析与仿真研究[J]. 计算机技术与发展, 2009, 19(11): 27-30.
- [3] Stewart R. Stream Control Transmission Protocol (SCTP)[S]. RFC4960, 2007.
- [4] Kohler E, Handley M, Floyd S. Datagram Congestion Control Protocol (DCCP)[S]. RFC4940, 2006.
- [5] 王 璐. 流控制传输协议高级特性介绍[J]. 技术交流, 2008(1): 14-17.
- [6] 朱桂勇, 吴庆波. 基于 SCTP 多宿特点的多路径同时传输研究[J]. 计算机技术与发展, 2007, 17(3): 5-9.
- [7] Iyengar J R, Shah K C, Amer P D, et al. Concurrent Multi-path Transfer Using SCTP Multihoming[C] // SPECTS2004. San Jose, USA: [s. n.], 2004: 265-273.
- [8] 鄢 欢, 高德云, 宋 飞. 基于 SCTP 多路径并行传输的性能评估[J]. 计算机技术与发展, 2010, 20(11): 29-32.
- [9] 张步忠, 金海平. Linux 内核系统调用扩展研究[J]. 计算机技术与发展, 2007, 17(5): 163-165.
- [10] Bickhart R W, Amer P D, Stewart R R. Transparent TCP-to-SCTP Translation Shim Layer[D]. Delaware: University of Delaware, 2005.
- [11] Stevens W R. UNIX 环境高级编程[M]. 尤晋元, 译. 北京: 机械工业出版社, 2002.
- [12] Haviland K, Gray D, Salama B. UNIX 系统编程[M]. 舒明, 译. 北京: 电子工业出版社, 2003.