文章编号:1001 - 9081(2009)05 - 1244 - 04

Linux 下的高流量数据包监听技术

马 博[1,2], 袁 丁[1]
(1. 四川师范大学 计算机科学学院, 成都 610061; 2. 天融信网络安全技术有限公司 成都研发部, 成都 610063)
(mabo1215@ hotmail. com)

摘 要:研究了 Linux 操作系统中使用底层抓包函数库 Libpcap 处理高量数据包监听的原理,利用网卡设备在网络的旁路处进行数据捕捉后预处理,利用 NAPI 技术实现设备半轮询机制以加快数据在缓冲区的处理速度,最后利用排队论原理计算最优带宽值并设置相关参数以达到最佳处理效率。实验表明,该方法不仅提高数据包的捕捉率,并且在系统资源占用率等多项指标中都有显著改善。

关键词:网络监听;Libpcap;TCP 协议;多线程;半轮询;New API
中图分类号: TP393.03 **文献标志码**:A

Monitor technique with high flow of data packets in Linux

MA Bo[1,2], YUAN Ding[1]
(1. *Computer Department, Sichuan Normal University, Chengdu Sichuan* 610061, *China*;
2. *Research and Development Department, Chengdu TOPSEC Network Security Technology Company Limited, Chengdu Sichuan* 610063 *China*)

Abstract: The principle of packet monitor to handle high volume packets using the underlying library Libpcap capture in Linux operation system was studied. Network Interface Card (NIC) was used to capture data in bypass monitor to carry out pre-processing. Semi-polling with New API (NAPI) was also used to speed up the processing of packets in input buffer. Finally the queuing theory was used to ensure the optimal bandwidth value and relevant parameters were set to achieve the best efficiency. Experimental results demonstrate that the scheme not only increases the rate of packet capture, but also improves the occupancy rate of system resources in many figures significantly.

Key words: network monitoring; Libpcap; TCP protocol; multi-thread, semi-polling; New API (NAPI)

0 引言

网络运行的稳定性和网络攻击的过滤一直是困扰着网络发展的难题。网络监听技术可以使网络使用者和管理者很好地获知网络的状况信息,并且做出相应处理。随着网络数据传输量的增大,互联网研究界认识到对于数据包监听在网络结构中的底层进行数据处理,可以提高数据包的捕捉率[1]。监听数据流在传输中必须根据其数据类型进行封装重组,而传统的数据包捕获和重组机制比如 Raw Socket 和 Win32 平台下捕捉效率较高的网络驱动程序接口规范(Network Device Interface Specification, NDIS)下的数据包过滤技术等[2],都要进行频繁的数据拷贝中断、页面切换,所以造成处理效率相对过低,很多时间和资源花费在处理过程中[3-4]。NDIS 数据包过滤技术对于数据包的处理速率较高,但是在高流量数据包的网络下进行旁路监听,其系统开销较高。旁路监听高量数据包下,由于受 CPU 和内存还有本身的处理机制限制,处理效率下降较大。而本文在网卡(Network Interface Card, NIC)处理层面上,利用 NAPI 设置半轮询机制(Semi-polling)加快数据在缓冲区的处理过程,计算最优带宽值并设置相关参数以达到最佳处理效率。由于对内存拷贝处理的改进,使得旁路监听高量数据包的效率极大提高。

1 核心技术介绍

1.1 以太网网络接口设备的工作模式

在基于网卡的捕捉中通常有两种工作模式,即一般模式和混杂模式。一般模式下,在接收到数据帧时,网卡仅接收目的地址与本地网络接口地址相同的数据帧和广播帧,而丢弃其他的数据帧;在混杂模式下,网卡会接收所有出现在网络接口上的数据帧,并移交给上层协议进一步处理。

以太网上的数据帧监听主要涉及 TCP/ IP 的 IP、ARP 等几个协议的分析[5]。在 Linux 下监听网络,首先设置参数使其工作于混杂模式下便于监听网络上的所有数据帧,然后选择用 Linux 中相关的函数库来截取数据帧[6]。

1.2 捕捉数据包平台比较

传统的数据包捕捉平台主要是基于网路中单节点的操作系统。最早的是 Raw Socket 原始套接字,但是其缺点是只能捕捉 IP 包,对于 ARP 包则处理无效[7]。而 BSD Packet Filter (BPF)机制相对来说也是基于 FreeBSD 下使用比较多的捕捉平台[8],它对于数据包的处理过程也相对简单易行,但是对于高流量数据包的处理,由于依赖于 UNIX 内部的一些机制,比如频繁的系统调用,所以数据处理效率也相对下降[9]。

美国洛仑兹伯克利国家实验室(Lawrence Berkeley National Laboratory)所编写的专用于数据包截获功能的 API 函数库"Libpcap"的设计目标,就是要使得数据包监听程序在不同的操作系统平台上可以更加容易地进行移植[10]。本文在 Linux 系统下利用 Libpcap 函数库来实现网络数据监听。使用该库一方面是因为 Libpcap 是一个与系统无关的网络数据捕获函数库,为不同的平台提供了一致的编程接口,具有良好的移植性;另一方面是因为 Libpcap 的捕包流程和内核 TCP/IP 协议栈的报文接收过程相对于传统内核而言,它绕过

收稿日期:2008 - 12 - 04;**修回日期**:2009 - 02 - 13。 **基金项目**:四川省计算机软件重点实验室重点项目。
作者简介:马博(1984 -),男,宁夏银川人,硕士研究生,主要研究方向:网络安全、网络通信; 袁丁(1967 -),男,四川成都人,教授,博士,主要研究方向:信息安全、网络安全。

了 TCP 层(UDP 层)和 IP 层的处理过程,直接将数据包从数据链路层拷贝到应用程序缓冲区中,比传统的 Raw socket 方式处理效率要高得多[11],并且避免了 SOCK_ PACKET 只在基于 Linux 的操作系统中有效定义的问题[12-13]。

传统的处理有很多时间都消耗在系统调用、中断、PCI 总线带宽以及内存拷贝上。而其中内存拷贝是在 Libpcap 中占用处理时间最长的过程。通常一个数据包被用户空间应用程序处理需要两次内存拷贝,一次是从网卡缓冲区到系统内核缓冲区,另一次是从内核缓冲区到用户。应用程序每次对 1 024 B内存的拷贝时间大约为 1 μs[14]。

文献[15]关于 Linux 报文捕获研究中提到,Libpcap 直接从链路层捕获的处理方式能够减少数据包在接收过程中所消耗的 CPU 时间,从表 1 中可以看出节省了大约 10% 的处理时间。但是在 Libpcap 捕包过程中,频繁的系统调用、数据拷贝和内核中断处理仍然是系统主要的性能瓶颈。

表 1　收包过程中 TCP/IP 协议栈各个部分的时间代价

协议栈	处理时间/μs	协议栈处理时间占总处理时间的百分率/%
Speedo_rx→Netif_rx	4.1	11.2
net_rx_action	4.0	10.9
ip_rcv	1.7	4.6
Tcp_recvmsg	3.8	10.4
Inet_recvmsg	20.9	57.1
System call	2.1	5.8

1.3　半轮询机制对于现有监听的优化

在 standard Libpcap 的捕捉处理中,要打开块设备等,逐层拷贝,并且每次都产生系统调用来实现数据拷贝,在对换中采用软中断机制。此种措施对于低量的数据包处理效率比较高,但是对于高量数据包,由于频繁的换入换出,使得处理时间大大增加。而全轮询机制方法由于设置了缓冲队列,在高数据量时一次处理整队列的数据,大大减少了频繁的中断调用,但是在网络流量不稳定的情况下会经常造成空等待从而降低了处理速率,因此并不是普遍适用的处理方法。所以在本文中引入半轮询机制优化措施,降低网络层传输的内存拷贝次数及避免频繁的系统调用,从而减小漏包率,提高检测速度[16]。

半轮询方式是一种中断和轮询方式的集成,如图 1。在高负载时使用轮询,在轻负载时使用中断驱动,很好地解决了低负载时轮询方式响应时间和处理器资源消耗的问题以及高负载时中断方式的接收活锁问题。半轮询的优势如下:1)限制了中断的频率,可以看作是一种自适应的中断批处理方式;2)减少了接收活锁的发生可能;3)数据和指令的本地局部性上也得到强化。

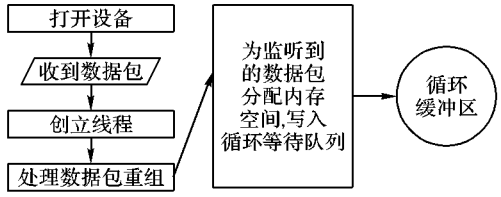

图 1　利用 semi-polling(半轮询)机制处理数据包

1.4　利用排队论计算带宽参数

在高量数据环境中,网络类型复杂多变,为了估算合理的带宽阈值,对于半轮询机制的选择,可以用博弈论中的排队论计算新的数据流量值以达到半轮询中两种机制的最佳切换[17-18]。设共有 N 种应用类型,不同应用的平均传输速率(每秒传输的数据包个数)为 $\lambda_i(i=1,2,\cdots,N)$,平均数据包长度(单位为 Byte,1 Byte=8 bit)为 $l_i(i=1,2,\cdots,N)$,现有的带宽利用率为 $\rho_i(i=1,2,\cdots,N)$,则数据流阈值(单位为 bps)为:

$$B=\sum_{i=1}^{N}\frac{\lambda_i}{\rho_i}l_i\times 8 \tag{1}$$

2　Linux 下的高量网络监听实现过程

2.1　Linux 下数据监听框架及数据包处理过程

如图 2 所述,首先利用 Libpcap 打开网卡等捕捉设备,当为数据分配块处理时,在内核模块中设置缓冲队列,然后计算数据包捕获频率和数据流阈值,通过比较来选择相应的处理方式;而计算捕获的参数通过 SK-BUFF 中的换入率得到,如果换入频繁超过了网络测值,就调用轮询方式,在 2.6 内核下调用 NAPI 开启缓冲队列并且设置轮询,然后从循环缓冲队列中取出 TCP 报文数据部分重组,并且输出相关信息。如果用户退出,则执行撤销缓冲、撤销进程等操作。

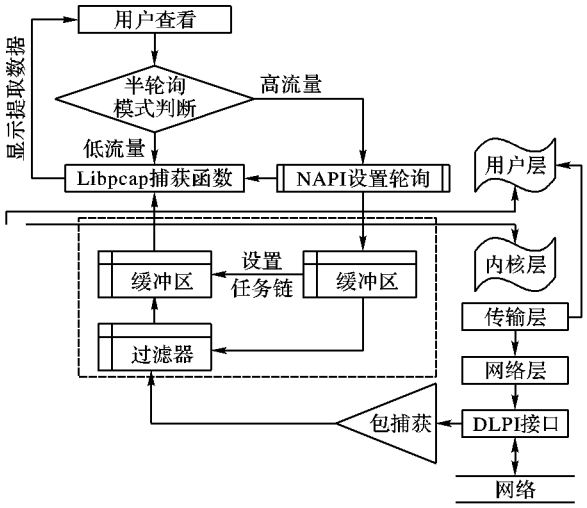

图 2　监听处理结构

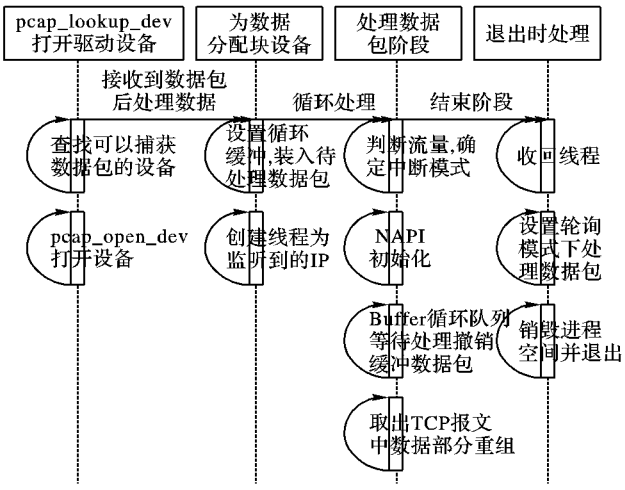

图 3　数据包处理过程

2.2　Linux 下数据监听处理步骤

如图 3 描述相关处理过程图所述,大概步骤如下。

1)打开、读取设备,设置过滤器:pcap_open_live( ), pcap_read( ),pcap_setfilter( )。

2)选择网络接口并且设置网络检测模块,包括初始化检测网络用到的 pcap_lookupdev( ),pcaplookupnet( )等相关函数。其过程为:首先打开套接字(socket),然后调用一系列的 ioctl 来获取套接字状态,以达到检测 TCP/IP 第一层网络设备

的目的。

3)设置轮询模式判断部分：捕获到数据后计算当前数据流量。利用式(1)中排队论理论计算出现有数据流值是否大于数据流阈值B。如果现有值大于数据流阈值B，则启用轮询模式处理，否则使用默认软中断处理，这样在一定程度上可以达到处理的最优化。

2.3 程序构架和处理过程描述

项目以DataListener函数作为整个程序框架的主体，调用Sniffer模块设置Libpcap初始化并且调用DataPacket模块控制监听的状况和执行相应的任务，然后把相关信息反馈给用户。其中DataPacket模块主要负责数据流量判断并且初始化NAPI设置轮询方式。描述伪码如下：

```
Void DataListener
{ 初始化 Sniffer, DataPacket 模块过滤函数和捕捉设备
  char * filter_app = fexp;                    // 设置过滤表达式
  调用 Parseopts( ) 函数获取命令参数
  利用 Libpcap 函数库中打开网卡设备并且进行监听
  用 pcap_datalink( handle) 设置句柄并获得网络类型;
  if ( 当前数据流量 < 数据流阈值)
    默认软中断处理;
  else
    利用定时器处理半轮询设备:
  初始化计时器 init_timer( &polling_timer) ;
  Semi-polling_timer. data = ( unsigned long) something;
  Semi-polling_timer. function = polling_handler;
  添加半轮询计时器
  void semi-polling_handler( unsigned long data)
  { add_timer( &semi-polling_timer) ; }
  设置半轮询句柄，数据放入循环队列等待处理
  调用 PacketBuffer( ) 处理数据包缓冲区:
  maint_thread_run( )
  调用线程互斥机制处理 pthread_mutex_lock( &pb_mutex) ==0
  调用 getnlp( ) 获链接头文件的指针，其中 n. p 指向网络头部;
  把数据包推入栈 pb -> pushPacket( n) ;
  Void listener_exit
  { pthread_mutex_unlock( &quitflag_mutex) ;
    pthread_cancel( maint_thread_tid) ;
    pcap_close( handle) ;
    Destroy_stack( )
    退出程序;
  }
}
```

上述部分描述了大概的处理过程。通过处理，管理者获得数据包信息后对其进一步的处理就可以提取出一些重要的监听信息或者对于相应数据端连接进行阻止或者规则匹配，比如载入相应过滤规则可以防止网络攻击等功能。

而具体实验关键代码如下：

```
if ( app -> promisc)
  handle = pcap_open_live( iface, SNAPLEN, 1, POL_TO_MS,
  errbuf) ;
else
  handle = pcap_open_live( iface, SNAPLEN, 0, POL_TO_MS,
  errbuf) ;
                                  //打开 NIC 设备获取捕捉数据
if ( ! handle)
  throw PcapError( "pcap_open_live", errbuf) ;
dlt = pcap_datalink( handle) ;
…
//过滤器初始化
struct bpf_program filter;                     //sniffer 过滤器
char * filter_app = fexp;                      //过滤器表达式
bpf_u_int32 mask;                  //sniffing device 的子网掩码
bpf_u_int32 net;                       //sniffing device 的 IP
…
//计算现有缓冲区最佳阈值
avg_lth = getlenth( &pcap_lenth) ;          //获取数据包平均长度
avg_speed = getspeed( &sniffer_speed) ;     //获取当前网络速率
avg_ utilize = avg_speed/max_speed;
while( pcap_flag)
{ B = avg_speed * avg_length/avg_ utilize + B; }
return B * =8;
…                          //getnlp 指针连接头，n. p 指向网络层
struct nlp * n = getnlp( packet, dlt, header) ;

if( ! checknlp( n) )
{ if ( n -> p ! = NULL)
    free( n -> p) ;
  free( n) ;
  assert( pthread_mutex_unlock( &pb_mutex) ==0) ;
  return;
}
```

上面是对于过滤器初始化、计算最佳阈值等几个处理过程关键技术的部分代码描述。

3 实验及结论

3.1 实验准备

为了验证技术对于数据包监听的改进，设置设施如下：Host(CPU：DualCore 1.8 GHz Athlon，NIC：3Com 3c59x Ethernet card，内存512 MB DDR)五台，100 Mb Ethernet ROUTER(Cisco 1760-V)五台。把主路由器A设置为在网络上的关键节点，IP分配为192.168.1.101。对监听服务器设置其IP为192.168.1.103。然后由网内几台测试主机发送由P2P等多媒体流形成的少量数据包和由Spirent公司的smartbit 2000形成的高量混合型数据包，其中包大小都在64 B。数据包捕捉对比平台使用现有处理效率相对较高的Win32 + NDIS和standard Libpcap + Linux 2.4内核，还有本文阐述改进方案Libpcap + Linux2.6 + Semi-polling平台，相应的网络拓扑和用户界面如图4、5所示。

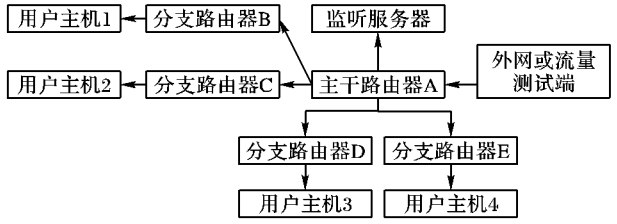

图4 监听架设网络拓扑

3.2 实验数据表及其分析

实验中监听捕捉都设置为混杂模式，表2~3是最后得出的测试数据。

图6中的Input Rate是输入数据包，可以看出在小数据率的情况下，Linux2.6.x + Libpcap + semi-polling和NDIS两种情况处理相当。当传输数据量增大后，传统的Libpcap和NDIS明显下降，而轮询机制下降不明显，数据增大到73 kpps时候，NDIS模式下CPU开销太大，资源耗尽不能处理更多捕捉数据，从而进入阻塞状态，而半轮询技术呈线性递减速率，而在124 kpps中是传统的Libpcap捕捉的2倍多，在千兆网卡和吉比特网络环境下预测该方案处理极限大概为325 kpps。

60.191.220.155:8208	192.168.1.101:4181	ESTABLISHED	1s	40 B/s
192.168.1.101:4725	64.233.189.100:80	CLOSING	1s	0 B/s
192.168.1.101:4717	64.233.189.147:80	CLOSING	1s	0 B/s
192.168.1.101:4718	66.249.89.127:80	CLOSING	1s	0 B/s
192.168.1.101:4565	218.249.43.237:80	CLOSING	1s	0 B/s
192.168.1.101:4711	64.233.189.147:80	CLOSING	1s	0 B/s
192.168.1.101:4715	64.233.189.147:80	CLOSING	1s	0 B/s
192.168.1.101:4713	64.233.189.147:80	CLOSING	1s	0 B/s
192.168.1.101:4732	60.191.220.155:3333	ESTABLISHED	5s	0 B/s
122.227.23.108:9999	192.168.1.101:1133	ESTABLISHED	6s	0 B/s
192.168.1.101:4591	66.249.89.127:80	CLOSING	9s	0 B/s
192.168.1.101:4553	218.249.43.237:80	CLOSING	9s	0 B/s
192.168.1.101:4557	218.249.43.237:80	CLOSING	10s	0 B/s
192.168.1.101:4646	210.51.48.118:80	CLOSING	12s	0 B/s
192.168.1.101:4734	222.73.37.73:9903	ESTABLISHED	14s	0 B/s
192.168.1.101:4654	220.181.37.200:80	CLOSING	15s	0 B/s
192.168.1.101:4547	218.249.43.237:80	CLOSING	17s	0 B/s
192.168.1.101:4637	64.233.189.165:80	CLOSING	17s	0 B/s
192.168.1.101:4575	211.103.181.148:80	CLOSING	17s	0 B/s
192.168.1.101:4579	211.103.181.148:80	CLOSING	19s	0 B/s
192.168.1.101:4625	64.233.189.165:80	CLOSING	21s	0 B/s
192.168.1.101:4574	211.103.181.148:80	CLOSING	24s	0 B/s
192.168.1.101:4573	220.181.37.200:80	CLOSING	24s	0 B/s
192.168.1.101:4578	211.103.181.148:80	CLOSING	25s	0 B/s
192.168.1.101:4585	218.249.43.237:80	CLOSING	25s	0 B/s
192.168.1.101:4726	60.191.220.155:3333	CLOSING	26s	0 B/s
192.168.1.101:4576	211.103.181.148:80	CLOSING	27s	0 B/s
192.168.1.101:4561	218.249.43.237:80	CLOSING	29s	0 B/s

图 5　监听 TCP 数据包建立连接状态界面

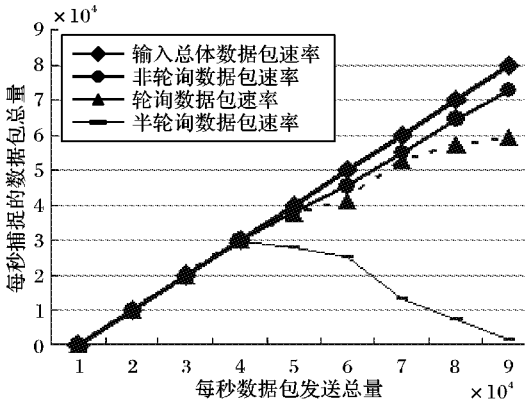

图 6　数据捕捉性能:轮询状态和非轮询状态下对比

表 2　混杂模式下数据包在不同传输率下的捕捉率

数据包量/kpps	Linux2.6.x + Libpcap + semi-polling + NAPI	Windows 2000 + NDIS	传统的 Linux 2.4.x + standard Libpcap
5	99.5%	99.2%	98.6%
26	75.1%	84.7%	61.0%
73	69.2%	58.1%	47.6%
124	42.1%	21.2%	12.4%

表 3　监听程序占用系统表

系统设备	发送包速率/kpps	内存占用率/%	CPU 占用率/%	启动线程数
Linux2.6.x + Libpcap + semi-polling + NAPI	20	72.2	82.0	469
	70	99.1	99.7	1653
Windows 2000 + NDIS	20	78.0	92.0	427
	70	100.0	100.0	1513
传统的 Linux 2.4.x + standard Libpcap	20	84.0	96.0	395
	70	99.6	100.0	1396

3.3　实验分析

上述实验表明在相同情况下,传统的 Linux2.4.x 因为使用非轮询机制,增大了系统中断处理时间,从而使得捕捉包性能较低,特别在处理高位流量数据的情况中,系统内存和 CPU 占用率很高。Windows NDIS 捕捉虽然性能比较高,并且捕捉率达到了 71.2%,但是其系统开销也是非常高的,内存和 CPU 分别达到了 78% 和 92%,而且因为 Windows 系统使用内存开销太大,所以在 20 kpps 以上的数据传输上面拦截性能下降明显,并且容易造成系统崩溃甚至死机等现象。而基于 Linux 2.6.x 内核引用 NAPI 的 semi-polling 机制下,对于 Libpcap 循环捕捉多线程对换处理,在高量数据监听处理上性能有所突破,73 kpps 流量下捕捉率也达到了 79.2%。而且相对现在捕捉性能最好的 Windos 的 NDIS 结构,本文使用的半轮询机制在系统开销方面也相对来说要小,并且数据流量继续增大的情况下也不会很快造成处理的极限从而进入阻塞状态。

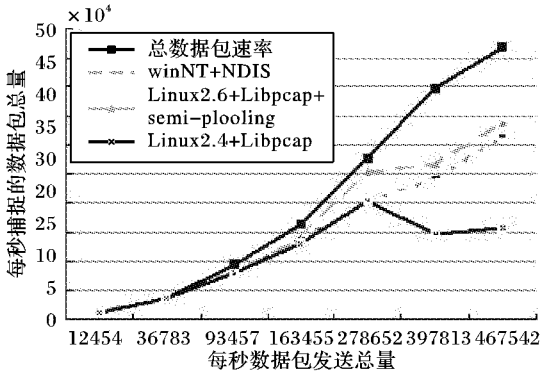

图 7　混杂模式下监听数据包占有量

4　结语

本文着重描述了 Linux 下的高量数据包监听技术及实现,实验证明通过对于部分处理机制的改进后不仅明显提高了对数据监听的处理效率,而且系统资源占用率等多项指标都有明显改善,在网络状况检测和入侵检测系统的改进方面都会带来处理瓶颈的突破。

参考文献:

[1] TSURU M, OIE Y. The characteristics of the Internet measurement technology research and development trends [J]. Journal of Information Processing Society, 2001, 42(2): 192 - 197.

[2] 张健,李焕洲. 网络嗅探原理及其检测和预防[J]. 四川师范大学学报: 自然科学版, 2003, 26(1): 90 - 92.

[3] KAZUYUKI S, HASEGAWA T, MURATA M, *et al*. TCP overlay network to connect the mechanism of division of labor and performance analysis, IEICE Technical Report IN03-198 [R]. IEICE, 2004: 745 - 747.

[4] 魏文清,王长征. Linux 下的 TCP/IP 架构与网络监听技术[J]. 计算机与现代化, 2005(12): 59 - 61.

[5] STEVENS W R. TCP/IP Illustrated[M]. [S.l.]: Addison-Wesley Press, 1998: 142 - 157.

[6] FOMENKOV M, KEYS K, MOORE D, *et al*. Longitudinal study of Internet traffic from 1998 - 2001: A view from 20 high performance sites [EB/OL]. [2008 - 09 - 12]. http://www.sfc.wide.ad.jp/~kaizaki/Paper/files/nlanr_overview.pdf

[7] 汪世义,秦品乐. 基于 Linux 的高速网络包捕获技术研究[J]. 微型电脑应用, 2006, 22(3): 51 - 52.

[8] McCANNE S, JACOBSON V. The BSD packet filter: A new architecture for user-level packet capture [C/OL]// Proceedings of the 1993 Winter USENIX Technical Conference. [S.l.]: USENIX, 1993 [2008 - 09 - 02]. http://www.tcpdump.org/papers/bpf-usenix93.pdf.

[9] IANNACCONE G, DIOT C, GRAHAM I, *et al*. Monitoring very high speed links[C]// Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement. New York: ACM Press, 2001: 267 - 271.

(下转第 1250 页)

若令 $\varepsilon(k) = Zw(k)$ 作为故障检测残差,其中,Z 为权矩阵,则按照下述规则可以判断系统是否发生故障。

$$\begin{cases} (\varepsilon(k)) \leqslant \bar{\varepsilon}, & \text{系统正常} \\ (\varepsilon(k)) > \bar{\varepsilon}, & \text{系统故障} \end{cases} \tag{22}$$

其中,$\| \varepsilon(k) \| = \sqrt{\varepsilon(k)^{\mathrm{T}}\varepsilon(k)}$ 是向量的欧氏范数,$\bar{\varepsilon}$ 为选定的故障检测阈值。

4 仿真示例

网络控制系统的被控对象为:

$$\begin{cases} \dot{\boldsymbol{x}}(t) = \begin{bmatrix} 0 & 10 \\ -10 & -20 \end{bmatrix}\boldsymbol{x}(t) + \begin{bmatrix} 0 \\ 10 \end{bmatrix}\boldsymbol{u}(t) + \begin{bmatrix} 1 \\ 10 \end{bmatrix}\boldsymbol{f}(t) \\ \boldsymbol{y}(t) = [1 \quad 0]\boldsymbol{x}(t) \end{cases}$$

设传感器采样周期 $T_s = 0.1$ s,在控制器端设置大容量缓存,令 $N = 10$,则 $T_c = 0.01$ s,用 T_c 离散化后的对象模型为:

$$\begin{cases} \boldsymbol{x}(k+1) = \begin{bmatrix} 0.9953 & 0.0905 \\ -0.0905 & 0.8144 \end{bmatrix}\boldsymbol{x}(k) + \\ \qquad\qquad \begin{bmatrix} 0.0047 \\ 0.0905 \end{bmatrix}\boldsymbol{u}(k) + \begin{bmatrix} 0.0147 \\ 0.0900 \end{bmatrix}\boldsymbol{f}(k) \\ \boldsymbol{y}(k+1) = [1 \quad 0]\,\boldsymbol{x}(k) \end{cases}$$

若原系统的控制律为 $K = [-0.1259 \quad -0.2718]$,事件1发生率 $\xi = 0.1$,则事件2发生率 $1 - \xi = 0.9$。选择满足式(15)的 $\delta_1 = 1.05, \delta_2 = 0.85$,利用 LMI 工具箱的 feasp 求解器计算可得:

$$\boldsymbol{P} = \begin{bmatrix} 81.233 & 0.68638 \\ 0.68638 & 88.124 \end{bmatrix}$$

观测器增益矩阵:

$$\boldsymbol{L} = [0.93945 \quad -0.00732]^{\mathrm{T}}$$

令权矩阵 $\boldsymbol{Z} = \boldsymbol{I}$,假定系统在 $t = 4$ s 时发生阶跃型突变故障,并选择故障检测阈值 $\bar{\varepsilon} = 0.02$。分别采用普通故障观测器和本文的故障观测器,故障检测的仿真结果如图2、3所示。

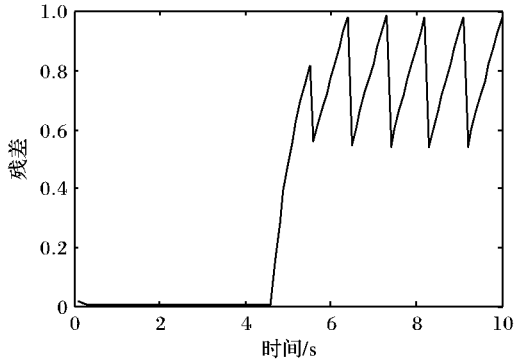

图2 采用普通故障观测器的仿真结果

从图2、3可以看出,采用等分采样故障观测器能更早检测出故障的发生,且残差曲线更为平滑。

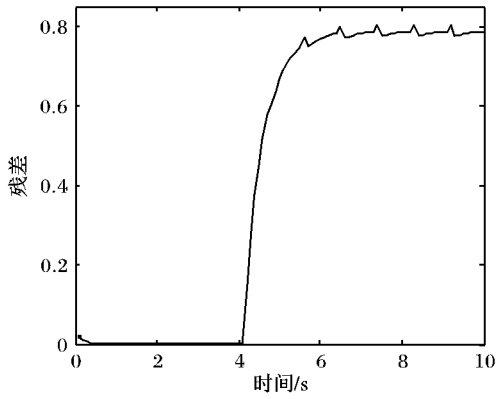

图3 采用等分采样故障观测器的仿真结果

5 结语

本文针对一类时延网络控制系统,假定各节点采用时间驱动方式,在控制器端提高数据读取频率,相当于将原采样周期进行了等分,从而可将输出时延近似看成新采样周期的整数倍。利用这种方法,可以减少采用时间驱动对于时延的放大作用,设计出更为准确可靠的故障观测器。此外,通过将原系统的故障检测等效为数据丢失的故障检测问题,利用异步动态系统理论,对观测器稳定性进行了证明。在本文基础上,可以进一步考虑系统具有干扰的情况,设计基于等分采样周期的鲁棒故障观测器。

参考文献:

[1] YUE DONG, HAN QING-LONG, PENG CHEN. State feedback controller design of networked control systems [J]. IEEE Transactions on Circuits and Systems-II, 2004, 51(11): 640 - 644.

[2] RABELLO A, BHAYA A. Stability of asynchronous dynamical systems with rate constraints and application [J]. IEEE Proceeding on Control Theory Application, 2003, 150(5): 546 - 550.

[3] BRANICKY M S, PHILIPS S M, ZHANG WEI. Scheduling and feedback co-design for networked control systems [C]// Proceedings of the 41st IEEE Conference on Decision and Control. Washington, DC: IEEE Press, 2002, 2: 1211 - 1217.

[4] HAO YE, DING S X. Fault detection of networked control systems with network-induced delay [C]// 8th International Conference on Control, Automation, Robotics and Vision: ICARVC 2004. Washington, DC: IEEE Press, 2004, 1: 294 - 297.

[5] BAO YONG, DAI QIU-QIU, CUI YING-LIU, *et al.* Fault detection based on robust states observer on networked control systems [C]// International Conference on Control and Automation: ICCA 2005. Washington, DC: IEEE Press, 2005, 2: 1237 - 1241.

(上接第1247页)

[10] Libpcap[CP/OL]. [2008 - 09 - 12]. http://sourceforge.net/projects/libpcap/.

[11] CORBET J, RUBINI A. Linux device driver program[M]. [S.l.]: O'Reilly Media, 2003: 65 - 66.

[12] TCPDUMP/LIBPCAP[CP/OL]. [2008 - 09 - 12]. http://www.tcpdump.org/.

[13] 王发琪. 网络监听技术在 Linux 系统下的实现[J]. 科技资讯, 2006(29): 109 - 110.

[14] 杨建华, 谢高岗, 李忠诚. 基于 Linux 内核的流量分析方法[J]. 计算机工程, 2006, 32(8): 67 - 69.

[15] 杨武, 方滨兴, 云晓春, 等. 基于 Linux 系统的报文捕获技术研究[J]. 计算机工程与应用, 2003, 39(26): 28 - 30.

[16] 刘玮, 郭莉. 半轮询方式提高 Linux 以太网桥性能[J]. 计算机应用, 2005, 25(z1): 50 - 51.

[17] ESTAN C, VARGHESE G. New directions in traffic measurement and accounting[J]. ACM SIGCOMM Computer Communication Review, 2002, 32(4): 323 - 336.

[18] 施永益, 黄忠东. 基于排队论和 QoS 的电力系统主干网带宽估算[J]. 电力系统自动化, 2002, 26(18): 50 - 53.