网络中主机和服务的状态检测*)

李 琦[1,2] 蒙 杨[1] 卿斯汉[1,3]

(中国科学院信息安全技术工程研究中心 北京100080)[1] (中国科学院研究生院 北京100039)[2]

(中国科学院软件研究所 北京100080)[3]

摘 要 本文提出一种检测网络中计算机和服务的状态的全新方法。该实现在状态检测方面提供了很大的灵活性和通用性,特别对提高服务检测的效率十分有意义。

关键词 对象,插件程序,状态检测

Stateful Inspection of Computer and Service in Network

LI Qi[1,2] MENG Yang[1] QING Si-Han[1,3]

(Engineering Research Center for Information Security Technology, The Chinese Academy of Sciences, Beijing 100080)[1]

(Graduate School, The Chinese Academy of Sciences, Beijing 100039)[2]

(Institute of Software, The Chinese Academy of Sciences, Beijing 100080)[3]

Abstract This paper proposes an new approach to inspect the state of computer and service in network. The implementation provide a flexible and versatile way, and it is especially significant for the efficiency of inspecting state of service.

Keywords Object, Plugin, Inspection of state

1 引言

随着网络技术的飞速发展,局域网中的设备变得越来越复杂,势必引起相应网络中服务的多样性。在这样的网络环境下,监视网络中的对象的运行状态就显得特别的重要。同时,在国内外许多网络管理[1]工具开发商相继在自己的产品里提供了管理功能,但大多都是基于很多相关的产品,具有很大的限制性,不具有很灵活的通用性。本文引用了插件式的管理方式,并且引入了面向对象的概念,对物理网络中的主机以及网络中提供的服务以对象的方式进行管理。本文第2节介绍传统的利用 SNMP 协议基于主机的状态检测;第3节介绍基于对象的检测的理论依据及管理模型;第4节介绍基于对象的状态检测的模型的实现;第5节简述一下测试和实现的性能;最后是总结并对下一步工作进行展望。

2 利用 SNMP 协议基于主机的状态检测

在网络运行的主机和服务检测功能中,IETF(Internet 任务工程组)提出的 SNMP(Simple Network Management Protocol 简单网络管理协议)[2~4]应用最为广泛,已经成为事实上的工业标准。在 SNMP 协议的实现基于主机的检测中,管理端(manager)和客户端(agent)的通讯是一个统一的方式。在管理端的 SNMP 实体通过发送一个协议数据包单元(PDU)(GetRequest、GetNextRequest 或者 SetRequest)[]进行初始请求。GetRequest 和 GetNextRequest 的请求数通常用来从 Agent 获得主机或者服务的管理信息,而 SetRequest 一般用来设定和改变管理信息。在收到一个 PDUs 之后,客户端通过查询管理信息库(MIB Management Information Base)中的主机和服务的状态信息,返回一个回应的 PDU(图1)。这个 PDU 封装了请求的信息或者先前请求的失败信息。

另外 SNMP 协议实现中还有一种基于客户端发起的主动发送 PDU 的情形。当客户端检测到一个异常的事件,就会发送一个 TrapPDU(图2)。

基于 SNMP 协议的检测,很多厂商的设备和设备提供的服务都支持这个国际标准。但 SNMP 只定义了管理信息如何交换,IETF 并没有定义如何检测一个服务的运行状态,需要各个厂商提供各自的扩展的MIB,如果没有厂商提供的

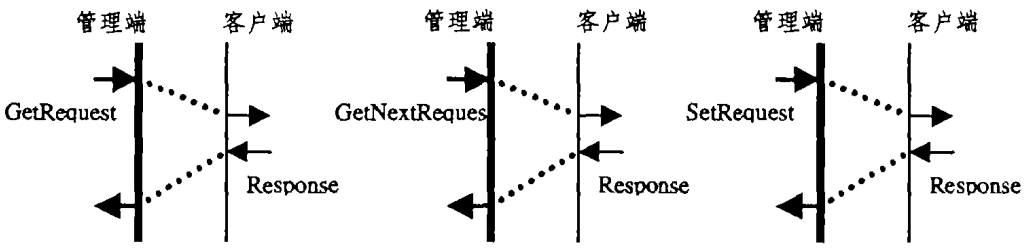

图1 管理端的主动查询模式

*)基金资助:国防科技预研基金资助项目。李 琦 硕士,主要研究领域为网络安全技术;蒙 杨 博士,助理研究员,主要研究领域为网络安全技术;卿斯汉 研究员,博士生导师,主要研究方向为信息安全理论与技术。

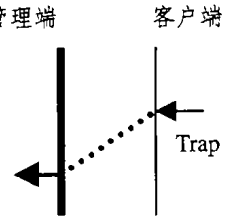

图2 客户端的主动报告模式

MIB，那么相应的服务就不能进行检测；另外这种检测模型需要在每个客户端都要运行一个 Agent 程序来查询 MIB、发送 Response PDU 以及发送 Trap PDU，这样增加了检测的复杂性，并且如果 Agent 实现 MIB 库的不全面，极有可能会在主机上留下安全隐患[6~8]；最后 SNMP 协议只实现了发送 Trap 消息并没有定义如何处理异常，需要管理员指定处理功能。

另外还有一个 CMIP 协议也定义了进行网络管理的信息协议和服务，可参考文[5]。

3 检测的理论依据及管理模型

3.1 检测的理论模型

在这里我们用到了流量的估计发送速率和真实的数据包接受率来评测对象的性能。这个算法我们用到了流量方程(Throughput equation)。引用了文[9]的方程如下：

定理1

$$X=\frac{S}{R*\sqrt{2bp/3}+3*t_{RTO}*P*(1+32p^2)*\sqrt{3bp/8}}$$

上式中参数的含义分别为：

X：计算所得数据包发送速率；S：发送端发送出的数据包大小；R：数据包的往返时间(RRT)；p：由接受端测出的数据包丢失率，介于0和1之间；t：超时定时器(timeout)；b：收到确认包的数量。

超时重传时间 t 在不同的协议栈的实现版本取值是有差异的，在这里我们取值为4R；当前有些版本的协议是采用了延时确认的，即一个包确认2个或者2个以上数据包，但大部分没有使用延时确认，故此处将 b 取为1，这也是比较合理的。

定义1(对象的状态集) 每一种服务和状态为以下四种状态：STATE_OK、STATE_WARNING、STATE_CRITICAL、STATE_UNKNOWN。

在这里我们对所有量化的数据分成以上四组进行性能评测。

另外由于在我们的实现模型下，主机和服务的检测(以下都简称为对象)的性能数据都不能准确地获得，但这些是接近与线性函数变化，因此我们利用多层学习算法进行了性能数据的准确度估算。过程如下：

(1)从测试样本中取出一样例，把它作为开始测试的初始值；

(2)由定理1分别计算各层节点的输出；

(3)计算实际输出和期望输出的误差；

(4)从输出反向计算到第一层，根据一定原则向减小误差方向调整网络的各个连接权值；

(5)对样本集中的每一个样例重复以上步骤，直到对整个误差达到<10%为止。

定义2 所有的对象的非正常的状态(Non-OK)分别处于两种状态：

1)软状态(Soft state)：当对象在$0<t<t_1$的时间处于非正常状态，则我们称该对象处于软状态；

2)硬状态(Hard state)：当对象在$t_1<t<t_2$的时间内处于非正常状态，则我们称该对象处于硬状态。

这里 t_1、t_2分别指经过估计和测试取得一个时间点，而我们主要指处于这种状态的对象在一段时间内将以频率 f_1进行检测，一旦 t_1时候后，软状态将迁移到硬状态，这样该对象将以 f_2的频率监测。

定义3 O_i：节点 i 的输出；net_j：节点 j 的出入；w_{ij}：从节点 i 到 j 的连接权值；y_i、y'_i：分别为输出层上节点 k 的期望输出和期望输出，则对节点 j 有(节点 j 位于节点 i 的下一层，且它们相连)：

$$net_j=\Sigma_i w_{ij}o_i \quad O_j=f(net_j)$$

由此可得：

$$e=\frac{1}{2}\sum_k (y'_k-y_k)^2 \quad w_{jk}(t+1)=w_{jk}(t)+\Delta w_{jk}$$

所以输出的 Δw_{jk}为：

$$\Delta w_{jk}=\eta(y'_k-y_k)f'(net_k)O_j$$

实例 图3是量化以后的性能数据图，而图4是使用权值量化以后的迁移数据图，由该图计算出数据变化21%该状态为正常。

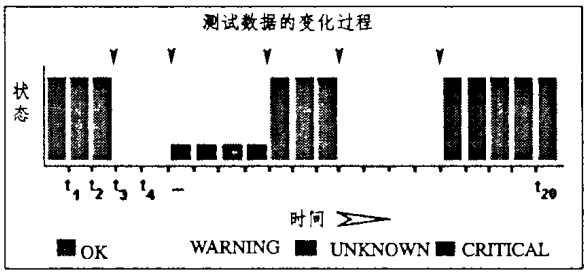

图3 对象的性能变化图

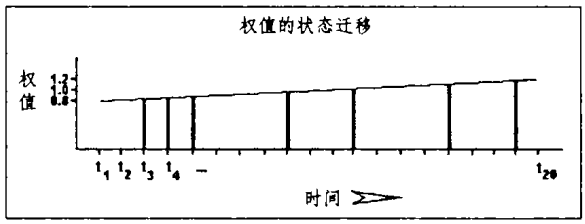

图4 权值运算后的性能变化图

3.2 对象检测的工作过程

我们检测实现的模型是基于插件的方式进行的，其基本原理是基于文[9,10]的想法。把主机以及主机上的服务都定义成一个需要单独检测的对象(如图5)；针对每个对象的检测我们需要一个特定的插件程序(Plugin)，之所以我们称为插件程序，是因为我们实现的后台调度程序没有包括任何内在的主机和服务的检测机制，相反这个程序只是调度外部的插件程序进行检测。

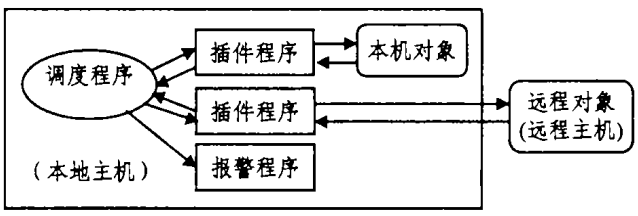

图5 对象状态检测的数据通讯模型

在这里我们实现对象检测的过程：

1) 根据后台执行的调度程序的事件队列，调用相应的插件程序检测对应的对象(本机或者远程的)；

2）插件程序执行检测功能，把检测完的状态返回给调度程序；

3）调度程序收到的消息，如果收到的是正常的消息，则做简单的状态记录，然后继续执行1；

4）否则做进一步处理，如果重复调度的次数大于21次，则执行5；否则把这个检测事件重新放入事件队列，进行优先的调度，然后进行继续检测，执行3；

5）调度程序调度事件异常处理程序或者报警程序。

4 基于对象的状态检测模型的实现

4.1 主要数据结构

4.1.1 调度队列的内部结构的定义

```
typedef struct sched_info_struct{
    int total_services;
        /* 当前检测的服务总数 */
    int total_hosts;
        /* 当前检测的主机总数 */
    unsigned long check_interval_total;
        /* 检测的时间间隔 */
    double average_check_interval;
        /* 平均的检测的时间间隔 */
    double average_inter_check_delay;
        /* 平均的检测的延迟 */
    double inter_check_delay;
        /* 检测的延迟 */
    ……
    }sched_info;
```

4.1.2 消息队列的内部结构的定义

```
typedef struct service_message_struct{
    char host_name;
     /* 主机名 */
    int return_code;
     /* 插件程序返回的结构：0、1、2、3 */
    int exited_ok;
     /* 插件程序返回正确？ */
    time_t check_time;
     /* 该对象开始检测的时间 */
    time_t finish_time;
     /* 该对象结束检测的时间 */
  ……
    }service_message;
```

4.1.3 日程队列的结构定义

```
typedef struct timed_event_struct{
    int event_type;
       /* 区分是服务检测还是主机检测 */
    int recurring;
       /* 设定是否需要立即重新设置检测 */
    void event_data;
       /* 检测的事件类型 */
    time_t run_time;
       /* 运行的时间 */
    struct timed_event_struct *next;
     }timed_event;
```

4.2 调度算法

由于篇幅关系调度算法就不详细描述了，参见图6的算法流程图。

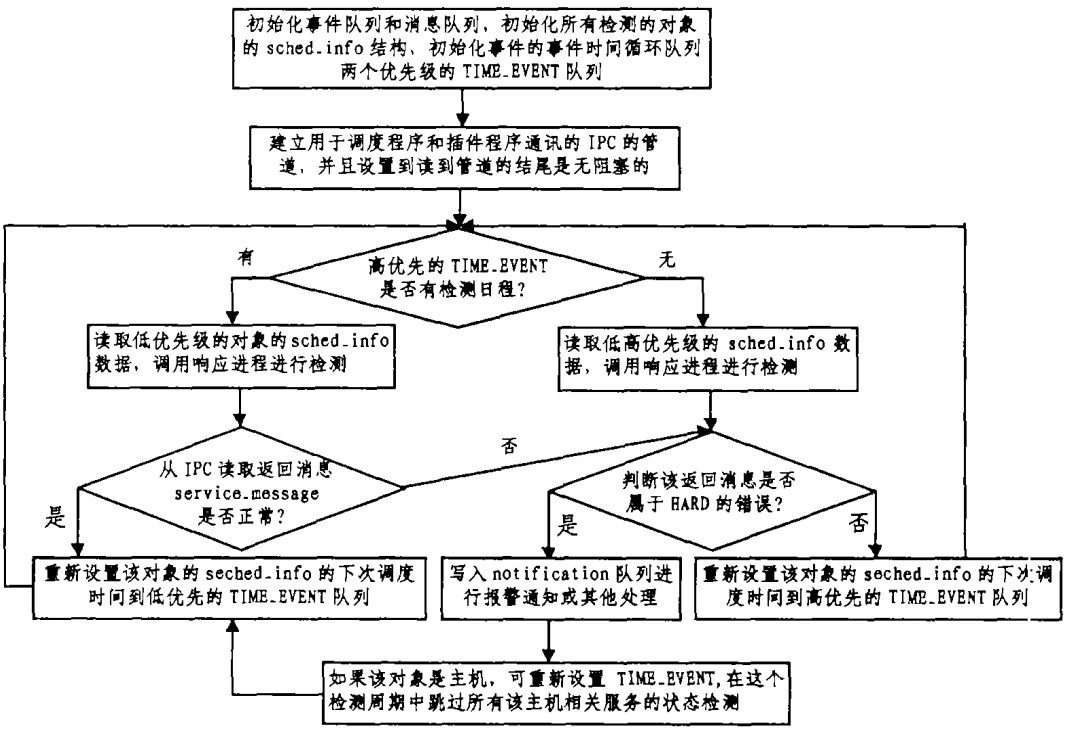

图6 调度程序的运行流程

4.3 插件程序的描述和实现

插件程序实现的出发点在于：对于每个对象能有对应的插件程序来进行检测。调度程序根据返回的状态值以及状态值的变化频度来判定该对象的服务状态。我们实现了一个对象异常的检测和确认的算法，由于篇幅关系，本文不涉及描述。

我们实现了一般常见需要检测的对象的插件程序，根据标准的协议实现可以检测 SMTP、POP3、HTTP、LDAP 等服务的专用插件程序。由于不能实现所有服务的检测，我们实现了三种情况对所有的对象提供了默认的有针对性的状态检测。下面就针对这三种默认的检测情况进行简要的描述。

4.3.1 对象为主机的情况 针对对象为主机的插件程序，我们主要应用利用发 PING 数据包来探测主机的丢包的百分比和 RTT(Round Trip Time)的平均值(ms)，另外我们还实现了类似 TRACTROUTE 在 Service_Message 中返回路由路径。

```
/* 实现根据包的接受与预期的差值进行分组的伪代码 */
Check_Pint(int t;char * host;int P)
/* t 可选的设置超时时间，host 指定检测的主机名，host 指定可选的发包的个数 */
{
  int result=STATE_UNKNOWN;
  signal(SIGALRM,timeout_alarm_handler);//设置超时处理;
  while(Soft-State){
          if(pl >wpl)
              result=STATE_WARNING;
```

（下转第139页）

(1)$(x \vee_{h,k} y) \to_{h,k} z \leqslant (x \to_{h,k} z) \wedge_{h,k} (y \to_{h,k} z)$;

(2)$x \to_{h,k} (y \wedge_{h,k} z) \leqslant (x \to_{h,k} y) \wedge_{h,k} (x \to_{h,k} z)$;

(3)$(x \wedge_{h,k} y) \to_{h,k} z \geqslant (x \to_{h,k} z) \vee_{h,k} (y \to_{h,k} z)$;

(4)$x \to_{h,k} (y \vee_{h,k} z) \geqslant (x \to_{h,k} y) \vee_{h,k} (x \to_{h,k} z)$。

证明:(1)因为 $x \to_{h,k} y = \min(1, (\max(0, 1 - x^m + y^m))^{(1/(nm))})$, $x \vee_{h,k} y = 1 - (\max(0, (1-x)^m + (1-y)^m - 1))^{1/(nm)}$, $x \leqslant x \vee_{h,k} y$, $y \leqslant x \vee_{h,k} y$ 和定理4蕴涵的单调性可知,$(x \vee_{h,k} y) \to_{h,k} z \leqslant x \to_{h,k} z$, $(x \vee_{h,k} y) \to_{h,k} z \leqslant y \to_{h,k} z$ 且由于 $x \wedge_{h,k} y \leqslant \min(x, y)$。

所以 $(x \vee_{h,k} y) \to_{h,k} z \leqslant (x \to_{h,k} z) \wedge_{h,k} (y \to_{h,k} z)$

仅当 $h=k=0.5$ 时,不等式中等号成立。

(2)因为 $x \to_{h,k} y = \min(1, (\max(0, 1 - x^m + y^m))^{1/(nm)})$, $x \wedge_{h,k} y = (\max(0, x^m + y^m - 1))^{1/(nm)}$,和定理4蕴涵的单调性可知,$x \to_{h,k} (y \wedge_{h,k} z) \leqslant (x \to_{h,k} y)$, $x \to_{h,k} (y \wedge_{h,k} z) \leqslant (x \to_{h,k} z)$ 且由于 $x \wedge_{h,k} y \leqslant \min(x, y)$。

所以 $x \to_{h,k} (y \wedge_{h,k} z) \leqslant (x \to_{h,k} y) \wedge_{h,k} (x \to_{h,k} z)$

仅当 $h=k=0.5$ 时,不等式中等号成立。

同理可证不等式(3)和(4)。定理7证毕。

结论 蕴涵是研究逻辑学的重点和难点[3~8]。本文对一级泛蕴涵运算进行研究,证明了泛与和泛蕴涵组成伴随对,还证明了泛蕴涵的正则性、单调性和代数性质,对深入研究泛逻辑学的形式系统和代数结构具有重要的意义。

参考文献

1 Zadeh L A. Outline of a New Approach to the Analysis of Systems and Decision Processes. IEEE Trans Systems Man Cybernet, 1973,3:28~44

2 Wang G J. On the Logic Foundation of Fuzzy Reasoning. Information Sciences,1999,117:47~88

3 王国俊. 三I方法与区间值模糊推理. 中国科学(E辑),2000,30(4):331~340

4 王国俊. MV-代数、BL-代数、R_0-代数于多值逻辑. 模糊系统与数学, 2002, 16(2):1~15

5 He Hua-can, Liu Yong-huai, He Da-qing. Generalized Logic in Experience Thinking. Science in China(Series),1996,39(2):225~234

6 何华灿,刘永怀,何大庆. 经验性思维中的泛逻辑. 中国科学(E辑),1996,26(1):72~78

7 何华灿,王华,等. 泛逻辑学原理. 科学出版社,2001

8 何华灿,刘永怀,魏宝刚,胡麒,王瑛. 泛"蕴含"运算和泛"串行推理"运算研究. 软件学报,1998,9(6):469~473

9 王国俊. 非经典数理逻辑与近似推理. 科学出版社,2000

(上接第81页)

```
    if(pl <upl)   //与期望的差值大于 upl
        result=STATE_CRITICAL
    if((pl <0)&&(wpl>upl)){
        if(!result==STATE_CRITICAL)){
            result=STATE_WARNING;
        }
    }else
 result=STATE_CRITICAL;
  if((pl<wpl)&&(pl>=0))   //如果丢包没超过告警的百分比
 result=STATE_OK;
}
return result;
}
```

4.3.2 *对象为基于 TCP 服务的情况* 以下我们用伪码来描述基于主机状态的检测程序,返回上面提到的四种状态值之一。

针对对象为 TCP 服务的插件程序,根据提供的 TCP 的端口进行连接来获得连接信息来判断服务的状态。以下我们用伪码来描述基于 TCP 服务的状态的检测程序,返回四种状态值之一。

现在很多服务都通过 SSL 通道进行连接,然后所有的数据都通过 SSL 通道进行通讯。

```
/* 检测基于 SSL/TLS 对象的检测伪代码 */
Check_TCP(char *host,int *proto)
{
  初始化 SSL 上下文;
  设置超时处理;
  打开握手协议的套接字;
  进行正常的数据通讯;
  while(1){
     通过接受的数据通过3.1描述的模型来判断服务运行状态;
  }
  返回结果;
}
```

4.3.3 *对象为基于 UDP 服务的情况* 针对对象为 UDP 服务的插件程序,实现和 TCP 服务的插件程序原理差不多,也是根据提供的端口进行连接获得连接信息来判断服务的状态。由于篇幅关系,这里我们就不重复描述了。

5 进一步工作的展望

在具体的实现过程中,虽然对象的检测可以使用默认的检测插件,但针对不同的对象最好用有针对性的插件程序来检测,如何保证插件程序的正确生成是进行对象状态检测的一个重点,而且在这方面我们可以进一步做优化处理。

另外在确定对象检测状态时,我们目前只是利用相对比较简单权值计算方法,在性能和实现上还有改进的余地。另外没有准确的主机和服务的性能数据,以及提高检测的准确性和及时性也是我们下一个阶段要解决的问题。

结论 网络中的主机和服务状态检测在信息技术飞速发展的今天日益重要。本文提出的状态检测方法具有以下优点:1)检测各个主机和服务非常灵活,特别是新增的主机和服务;2)没有对被检测的客户主机上性能有很大的影响,并且不用在客户主机上装任何客户程序;3)最后能对不稳定的主机和服务实现一定程度的预测。

参考文献

1 Pras A. Network Management Architectures: [Ph. D-thesis]. Netherland,1995

2 Schmidt K, Mauro D J K. SchmidtEssential SNMP oreilly, July 2001

3 Reynolds R E. RFC 1156: Tools for Monitoring and Debugging TCP/IP Internets and Interconnected Devices, June 1993

4 McCloghrie K, Rose M, Waldbusser S. RFC 1450: Management Information Base for version 2 of the Simple Network Management Protocol (SNMPv2), April 1993

5 Warrier U, Besaw L, LaBarre L, Handspicker B. RFC 1189: Common Management Information Services and Protocols for the Internet, Oct. 1990

6 http://www.cert.org/advisories/CA-2002-03.html

7 http://www.microsoft.com/technet/security/bulletin/ms02-006.asp

8 http://www.cisco.com/warp/public/707/ios-snmp-community-vulns-pub.shtml

9 Clark D D. The Design Philosophy of the DARPA Internet Protocols. In:the Proc. of ACM SIGCOMM '88, Aug. 1988

10 Paxson V. Bro: A System for Detecting Network Intruders in Real-Time,. Computer Networks (Amsterdam, Netherlands: 1999) 1999,31:23~24

11 Provos N. Honeyd: A Virtual Honeypot Daemon. July 2003