基于异构的网络安全策略自适应发布

唐成华 余顺争
(中山大学电子与通信工程系 广州 510275)

摘 要 随着网络实体数量的急剧增长,网络安全策略的请求、更新和执行等操作对策略的发布提出了更高的要求。为了解决网络安全策略发布效率的问题,引入发布影响因子和安全域等概念,提出了基于异构的网络安全策略自适应发布数学模型和结构模型,着重分析了面向属性状态和操作的异构策略表示和生成方法,给出了安全策略快速搜索算法、策略比较及异构策略生成算法、基于安全域的地址分配及数据转发算法。与传统的全策略发布模型相比,大大提高了策略处理效率,并占用较少的网络信道资源。

关键词 安全策略,自适应发布,异构,安全域
中图法分类号 TP393.08 **文献标识码** A

Self-distribution of Network Security Policy Based on Structure-dissimilarity

TANG Cheng-hua YU Shun-zheng
(Department of Electrical and Communication Engineering, Sun Yat-Sen University, Guangzhou 510275, China)

Abstract The operations of security policy's request, update, and execute have put forward higher requirements of the policy distribution. For purpose of resolving the distribution efficiency of the network security policy, the security policy self- distribution mathematic model and structural model were proposed based on structure-dissimilarity, which introduced the concepts of distribution factor, security domain, etc. Expression and making ways of the structure-dissimilarity policy faced on attribute characters and operation were analyzed emphatically. The security policy searching algorithm, comparing algorithm, structure-dissimilarity policy building algorithm, address assigning and data transmitting algorithm based on security domain were presented. Compared with the classical entire distribution model, the proposed methods are superior to enhance the system security policy disposal efficiency, and occupy the lesser resources of network channel.

Keywords Security policy, Self-distribution, Structure-dissimilarity, Security domain

使用高层次的策略规则来调整和控制低层次的系统行为之后,网络安全策略管理技术使得系统安全管理具有高度的可扩展性和灵活性。一方面,如何降低网络冗余数据,充分利用带宽资源及提高网络上有效数据的效率一直是数据传输领域的研究热点,主要集中在多播或广播技术在分布式交互仿真、网上视频转播、股票信息发布等实时分发管理中的研究与应用,通常以减少接收冗余数据的可能性和网络中的数据流量为目标,一般采用基于网格[1]、区域匹配[2]等数据过滤的机制。另一方面,在网络安全策略的描述和制定上做了很多工作,IETF 提出的策略框架定义语言(PFDL)、贝尔实验室开发的策略描述语言(PDL)、英国伦敦皇家学院的 Ponder 语言、OASIS 的 XACML 以及信任策略语言(TPL)、授权规范语言(ASL)、LaSCO 策略图、RSL99、Rei 策略等。所有这些工作都是关于策略的基础性研究,在安全策略管理领域针对策略的优化和发布在理论和应用中还未出现有效的解决方法,而随着网络安全越来越受到重视,以及网络中实体数量的急剧增长,实体间信息交互也更频繁,原有的全策略分发部署方式[3]使得策略决策点(PDP)的负荷严重,并占用大量的与策略执行点(PEP)通信的资源,因此,实现网络传输中的策略数据过滤,降低策略冗余信息,减少策略寻址时间,提高策略的发布效率已是安全策略管理系统的迫切要求。

本文从发布影响因子出发,提出基于异构的安全策略自适应发布数学模型和结构模型,建立异构策略表示方法,给出策略快速搜索算法、异构策略生成算法、策略发布地址分配及数据转发算法,并对其性能进行分析和验证。

1 安全策略自适应发布模型

1.1 数学模型

安全策略自适应发布数学模型为$\langle P,\Delta\sigma,Q,M\rangle$元组:

P 为系统中基于安全域的全部安全策略集合[4],能够根据路径 path 得到指定安全域中关于信息主体或信息客体的策略 $P(path)$,也能根据策略请求命令 τ 得到策略 $P(\tau)$,其

到稿日期:2008-10-28 返修日期:2009-01-22 本文受国家高技术研究发展计划(863)项目(2007AA01Z449),国家自然科学基金-广东联合基金重点项目(U0735002),中国博士后科学基金项目(20070420793)资助。
唐成华(1974—),男,博士后,主要研究方向为网络信息安全等,E-mail:tchbox@163.com;余顺争(1958—),男,教授,博士生导师,主要研究方向为网络信息安全等。

中，$P(path)$和$P(\tau)$均为有序集。

$\Delta\sigma$为策略发布影响因子集合，$\Delta\sigma=\{\sigma i \mid i\in N\}$，对策略的发布内容产生影响，对应着命令$\tau$的策略相关影响因子为$\Delta\sigma(P(\tau))$，$\Delta\sigma(P(\tau))$的取值是动态变化的，决定于策略状态、请求方式、系统操作等。

Q为最终的发布策略，对应于系统中生成发布策略的运算规则系列，对于命令τ的发布策略表述形式为$Q(P(\tau),\Delta\sigma(P(\tau)))$，它以静态的$P(\tau)$和动态的$\Delta\sigma(P(\tau))$为输入，其中$\Delta\sigma(P(\tau))$直接影响着发布策略的产生流程，并按照一定的算法过滤并构造出策略发布内容为输出。

M为策略的分配方案，描述了策略发布的数据源与目的地的映射关系，表述形式为$M(Q,\bigcup_{i=1}^{n}di)$，其中di为Q的应用目的地，$\bigcup_{i=1}^{n}di$为Q的应用目的地集合。策略发布到PEP之前，首先为该映射关系分配策略应用对象寻址。

1.2 安全策略发布影响因子

安全策略发布影响因子对安全策略发布结果具有决定性的影响，主要有3种发布影响因子：策略状态因子σ_1、策略请求方式因子σ_2和策略内容因子σ_3，如表1所列。

表1 策略发布影响因子

影响因子	取值范围	策略描述
σ_1	−1	锁定态，不可被发布(0策略发布)
	0	静止态，可被发布
σ_2	0	非强制请求
	1	强制请求(全策略发布)
σ_3	0	内容无修改(0策略发布)
	(0,1)	内容有修改，σ_3值越大，修改程度越大(异构策略发布)

定义1 设所请求的策略为$p=P(\tau)$，其中$P(\tau)$是根据请求命令τ从策略集中获取的原策略，经过策略发布因子影响，按一定的规则运算生成的策略为$p^*=f(p,\sigma)$，其中$\sigma=g(\Delta\sigma(p))=g(\sigma_1,\sigma_2,\cdots,\sigma_n)$，$n\in N$，$\sigma$是策略$p$的发布影响因子，$\sigma_1,\sigma_2,\cdots,\sigma_n$的取值范围为$(-1\cup[0,1])$，并依次对策略的生成产生影响，且符合性质：

$$\sigma=g(\sigma_1,\sigma_2,\cdots,\sigma_n)=\begin{cases}0, & \exists\sigma_i=-1 \text{ 或 } \forall\sigma_i=0\\ 1, & \exists\sigma_i=1\\ (0,1), & \exists\sigma_i\in(0,1)\end{cases},i\in n$$

定理1 如果策略p的发布影响因子$\sigma=g(\sigma_1,\sigma_2,\sigma_3)$，则满足：

1) 当$\sigma=0$时，$Q=p^*=0$；

2) 当$\sigma=1$时，$Q=p^*=p$；

3) 当$\sigma\in(0,1)$时，$Q=p^*\neq p$，且当$\sigma\to1$时，有$p^*\to p$，$i=1,2,3$。

证明：当$\sigma=0$时，如果策略被锁定，此时所请求的策略不可被发布，因此$p^*=0$，$Q=0$，即进行0策略发布；如果$\forall\sigma_i=0$，即策略可以被非强制全策略发布，但策略内容与上次发布的同一策略(设为p')内容完全相同，$Q=p^*=0$，执行0策略发布。

当$\sigma=1$时，策略是在非锁定状态下，但被强制请求全策略发布，发布的策略是从策略集中获取的策略p，即$Q=p^*=p$。

当$\exists\sigma\in(0,1)$时，策略可以被非强制全策略发布，但策略内容有变化，即$Q=p^*\neq p$，σ值越大，表示策略内容与策略p'内容相异程度越高，当$\sigma\to1$时，Q内容与策略p'内容完全不同，因此，发布的内容实际上是从策略库中取出的完全策略，即$p^*\to p$。

定义2 异构策略是根据策略请求命令从策略库中获取的策略与已发布的对应策略按照内容的不同而建立的关于内容变化的描述。

定义3 当$Q=p^*=0$时，称为0策略发布，策略发布内容为零状态标识；

当$Q=p^*=p$时，称为全策略发布，策略发布内容是根据策略请求命令从策略库中获取的完全策略；

当$Q=p^*\neq p$时，称为异构策略发布，策略发布内容为异构策略。

1.3 结构模型

根据定义1、2、3和定理1，建立安全策略自适应发布结构模型，如图1所示。

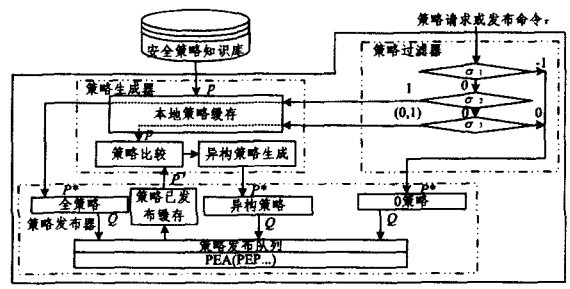

图1 安全策略自适应发布模型

该模型主要由3个模块组成：策略过滤器、策略生成器和策略发布器。模型接收来自PEP的策略请求的传入参数，首先由策略过滤器根据定理1进行快速策略过滤，将通过的参数传递给策略生成器，建立最优的策略发布内容，最终的策略发布内容可能是0策略、全策略或异构策略，流程如下：

1) 策略过滤器接收策略请求的传入参数，根据该策略发布影响因子决定处理流程。如果$\sigma_1=-1$，则当前策略被锁定，禁止向PEP发布该策略，执行0策略发布，此时$Q=p^*=0$，进入策略发布队列。σ_1的值由策略管理中心进行设置，并通过策略的状态标志来判定。如果$\sigma_1=0$，则当前策略处于静止态，可以被发布，进入σ_2分析。

2) 如果$\sigma_2=0$，策略被非强制请求，处于正常策略请求方式，进入σ_3分析。如果$\sigma_2=1$，策略被强制请求。一般地，当PEP的策略被损坏，或者强制命令某策略完全发布时，$\sigma_2=1$，此时转入策略生成器，直接从本地策略缓存或策略知识库中获取全策略p，交给策略发布器，进入发布队列，$Q=p^*=p$，全策略发布成功后，更新已发布缓存。

3) 如果$\sigma_3=0$，策略内容无修改，在非强制策略请求的情况下，不必发布该策略，执行0策略发布，$Q=p^*=0$。σ_3的状态可通过策略的两个时间戳(最近一次修改时间*policyUpdataTime*和最近一次发布时间*publishTime*)来判定，如果$policyUpdataTime\leqslant publishTime$，则$\sigma_3=0$，否则$\sigma_3=(0,1)$，转入策略生成器，从本地策略缓存或策略知识库中获取策略p，并搜索已发布缓存中对应的策略p'，按照一定的算法生成异构策略p^*，$Q=p^*$，由策略发布器通过PEA(策略执行代理)发送给PEP，同时更新已发布缓存。

2 异构策略分析

2.1 异构策略表示

为了减少网络数据传输量，异构策略的表示应该精炼，并便于生成和在 PEP 端被解码。异构策略的建立，借鉴于关联规则的更新思想。作为数据挖掘中的一个重要内容，关联规则的增量式提取或更新，已有许多研究成果[5]，如频繁模式树法[6]、动态序列合并法[7]、支持度函数的改造 FP 树算法[8]，这些建树和规则更新的算法实现都很复杂，表达不够直观。本节提出一种面向策略属性及状态位的异构策略表示方法，信息量少，生成快捷，并便于 PEP 端接收、解码和更新策略。

根据安全策略的定义及描述规范[4]，基本策略包含一个策略规则，每个策略规则由 *infoSubject*，*infoObject*，*condition*，*constraint*，*action* 和 *exception* 6 种属性组成，而复合策略由多个策略规则组成。以复合策略为发布对象来研究异构策略的表示。异构策略的内容，决定于策略 p 与 p' 的内容变化，需要从策略规则的各属性及属性值的变化情况着手。

1）策略规则属性状态控制(CSC)

策略规则的属性状态表明，相对于 p' 而言，p 中对应的策略规则的属性是否有改变。给定一个 6 位二进制数，其 D0—D5 位分别对应策略规则的 6 种属性，当 $D=0$ 时，表示对应的策略规则属性值没有改变，此时异构策略中对该属性值不作描述；当 $D=1$ 时，表示对应的策略规则属性值有变化，异构策略需要对该属性值进行描述。

2）策略规则属性操作控制(COC)

策略规则属性值的变化，体现在对属性值的操作行为上。策略规则属性操作，是异构策略用来表示 PEP 接收到发布策略后，对其属性值进行何种操作，从而保持策略的更新。策略规则属性操作控制由一个二位二进制数表示，如果 D1＝1，则表示对属性进行修改，否则，当 D0＝0 时，表示删除属性值，当 D0＝1 时，为添加属性值。

3）异构策略的结构表示

复合策略的异构策略结构表示如图 2 所示，首先给出策略的标识号 *policyID*，这种标识号被 PEP 用来确认策略的身份。*ruleNum* 表明该异构策略所包含的规则数量，当 *ruleNum*＝1 时，既表示该异构策略只有一条策略规则，也表明 p 与 p' 之间只有一条规则内容发生变化。*ruleID* 为规则的唯一标识号，与 p 和 p' 中的规则相对应。*ruleString* 是策略规则属性状态控制数，对应策略规则的 6 种属性，如果某一位值为 0，则相应的属性操作控制数及属性值在异构策略中不描述，例如 D3＝0 时，则在异构策略中不描述规则的 *constraint* 属性值，属性操作控制数及属性值按照 *ruleString* 的位的顺序列出。

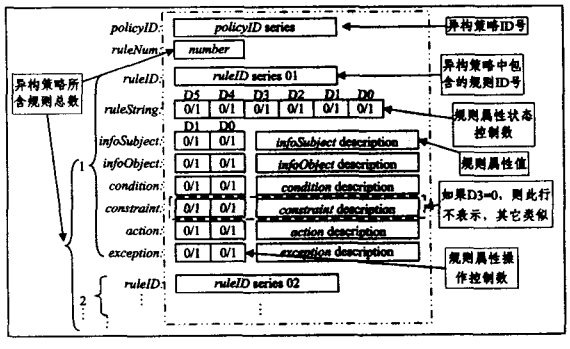

图 2　复合策略的异构策略结构表示

2.2　异构策略生成

异构策略的生成由策略生成器来完成，主要包括策略缓存搜索、策略比较和异构策略生成 3 个步骤。

1）策略缓存搜索

自适应策略发布模型设计了本地策略缓存，避免频繁地从远程策略知识库中读取策略，只有在本地缓存中找不到相应的策略时才执行远程读取，同时还设置了一个已发布策略缓存。虽然策略之间可能具有一定的相关性，但这种相关性是一种功能上的相关性，因此，对于缓存中的策略搜索，可以执行非顺序的搜索算法，而对于同一策略中的不同规则，容易根据它们的 *ruleID* 建立大小顺序，但处于不同策略中的规则的 *ruleID* 可能依次相等，此时如果采用完全顺序搜索法，在相同 ID 的规则中搜索指定的规则必须附加另外的约束条件，因此必须对 *ruleID* 和 *policyID* 一起考虑，鉴于此，设计了基于哈希(Hash)索引的链策略搜索算法。

对普通链表来说，二分法搜索是不适用的，因为不能很快地找到位于链表中间的数据项。因此，为了使用二分法搜索，在每个对象的结点键域(key)的基础上增加一个链域(link)，用来保存排列的信息，而所有的对象组成一个对象数组，即 link[i]指向索引(index)为 i 的对象 key[i]，在初始化时，对链表的键域 key[i]进行插入排序。index 为 0 的对象作为链表访问的入口点，其 key 值保存 key 的数据类型的最大数，记为 MaxNum。

基于哈希索引的链策略搜索算法，依赖于策略和策略规则的存储结构，如图 3 所示，具有相同哈希值的策略规则，按照大小顺序构成一个链表。

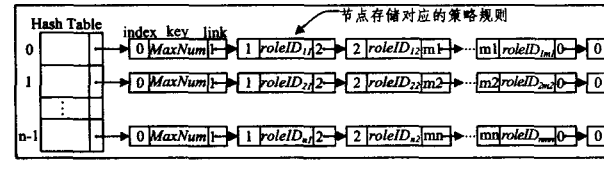

图 3　基于 HASH 索引的链策略存储结构

首先定义哈希函数，取 *policyID* 与 *ruleID* 异或运算，以哈希表长度取模：

```
Integer HashKey(policyID,ruleID){
int index = (policyID XOR ruleID) mod MAX;/*MAX 为哈希表空间，取素数*/
return index;}
```

系统初始化，对本地策略缓存或已发送缓存策略规则集 A 使用哈希函数 HashKey 构造哈希表 H，同时对链表进行插入排序，算法如下：

算法 1　初始化，建立基于 Hash 索引的链策略表

输入：策略规则集 A。

输出：基于 Hash 索引的链策略表 H。

For ($r\in A$) {

1. 获取规则 r 的 *policyID* 及 *ruleID*；

2. idx = HashKey(*policyID*,*ruleID*)；$H[idx] = r$；/*取得哈希值，并插入该规则*/

3. 如果 $H[idx]$ 上已存储 2 个以上规则，LinkInsertionSort($H[idx]$.*roleID*)；

/*按 *roleID* 的大小进行插入式排序*/}

算法 2　按给定的 *ruleID* 搜索出对应的规则 r

输入：*ruleID*、基于 Hash 索引的链策略表 H。

输出：符合 *ruleID* 的规则 r。

1. 获取给定 *ruleID* 的规则对应的 *policyID*；

2. *node* = H[HashKey(*policyID*,*ruleID*)]；/*取得链表中的

第一个节点 * /

3. j = BinarySearch(*node*. key[1], *node*. key[2], …, *node*. key[m], *ruleID*);

/ * 二分法搜索链表，得到 *ruleID* 在链表中的位置 * /

4. return *node*. key[j]. getRule(); / * 取出并返回存储在链表中的规则 * /

2）策略比较及异构策略生成

策略的比较主要是指策略规则的内容不同的比较，找到不同的策略规则属性值，并按照异构策略的结构建立描述。

算法 3 比较 p 与 p' 的不同，建立它们的异构策略 p^*

输入：本地策略缓存策略 p、已发送策略缓存策略 p'。

输出：异构策略 p^*。

1. 获取策略 p 的 *policyID* 及所包含的规则数量 *ruleNum*，存入 p^*；

2. For ($r \in p$) {

2.1 获取规则 r 的 *ruleID*，并存入 p^*；

2.2 建立规则属性状态控制数 *ruleString*；

2.3 按照算法 2 获取 p' 中 *ruleID* 对应的策略规则 r'；

2.4 for（规则中的所有属性值）{

比较属性值，如果没有变化，*ruleString* 中相应位置为零，返回 2.4。如果属性值有变化，确定属性值从 r' 中的内容变为 r 中的内容应操作的行为（update，delete 或 add）和内容，建立规则属性操作控制数，与操作内容一起存入 p^*；}}

3 策略发布机制

3.1 发布队列

发布队列记录了策略发布器待发布的任务清单，主要包括待发布的策略、对应的发布目的地以及当前的发布状态。发布队列中的策略发布状态，主要受当前网络信道资源（sources）和策略优先级（priority）的限制，其中 priority 决定了待发布策略在队列中所处位置（location）。网络安全管理策略根据其实施的对象或保护的资源的重要程度，被设定成不同的优先级，策略发布器能依据它们的优先级对发布队列进行智能排队，在当前负荷允许的情况下，优先级高的策略先被发布，因此，一个处于发布队列中的策略 Q 被发布的约束条件是：

$\begin{cases} priority(Q)=\mathrm{Max} \\ f(\cdot)>0 \end{cases}$，其中 $f(\cdot)$ 为当前系统的信道资源可用函数。

3.2 地址分配及数据转发

PEA 负责转发安全设备的策略请求，并接收来自策略服务器的策略决策，按照与策略发布模块一致的格式对策略决策进行解码，转化为本地命令在安全设备上执行。策略决策、全策略或异构策略由策略发布器通过 COPS 协议发布到指定安全域中的 PEA 中，再由 PEA 负责解码发送到目标 PEP 上执行，PEA 起着 PEP 地址分配和数据转发的作用。

基于安全域的 PEA 策略发布地址的分配基本原则是：为每个安全域（或 PEA）分配一个固定地址，而对于域内的每个 PEP 结点，按照基于接收方（PEP）的方式分配地址。

设 E_i^l 为安全域 i 内的 PEP 节点 l，安全域 i 内所有 PEP 节点为 $\sum_{l=1}^{m_i} E_i^l$，其中 m_i 是安全域 i 内的 PEP 节点数，安全域 i 内 PEA 进行地址分配及转发策略数据的算法描述如下：

算法 4 地址分配及数据转发

1. 为每个安全域分配地址；

2. $Set_i^l = \phi$； / * 初始化 * /

3. $Set_i^l = \sum_{l=1}^{m_i} E_i^l$； / * 安全域 i 内的 PEP 节点集合 * /

4. 如果 $Set_i^l \neq \phi$，则 PEA 根据 Set_i^l 分配发布地址，建立地址集 $\sum_{l=1}^{m_i} P_i^l$，并指导 E_i^l 与 PEA 建立联系；/ * P_i^l 是节点 E_i^l 的分配地址 * /

5. PEA 接收到发布策略数据，如果 $Set_i^l = \phi$，则返回第 3 步，否则对接收的策略数据进行解码，并转发到对应地址，在接收和转发过程中，PEA 要对策略数据进行出错校验。

对于接收到的策略发布数据，必须进行出错校验，目前 CRC、数据冗余纠错（其中采用前向纠错（FEC）方法较多）等技术在多媒体数据流校验应用中已比较成熟[9]，这两种方法在合适的信噪比条件下能够纠正和检查出绝大多数的误码率。PEA 对接收的策略数据进行校验，如果出现错误，则申请策略服务器重新发布该策略。某一策略从总体上重新发布的次数，称为该策略的重传因子。

安全域内 PEP 节点间的发送和接收数据均按基于 PEP 节点的方式进行地址分配，PEA 维护一个域内 PEP 节点作为接收方的地址分配表。

3.3 可行性分析

通过基于安全域的系统结构划分，PEA 策略发布地址分配量会大大减少。设 N 为系统内的 PEP 结点数目，每个安全域内节点数为 n，则每个安全域内 PEP 结点地址最大使用量为 (2^n-1)，为每个安全域自身分配的地址数量共有 N/n，因此总的地址分配量为 $(2^n-1)N/n+N/n=2^nN/n$。表 2 给出了未使用安全域的单一 PEA 地址分配与基于一级安全域的多 PEA 地址分配的需求量比较。

表 2 地址分配量比较

N	单一 PEA	一级安全域	
		n=5	n=10
10	1,023	64	1,024
15	32,767	96	—
20	1,048,575	128	2,048
25	22,554,431	160	—
30	1,073,741,823	192	3,072

使用基于安全域的 PEA 策略发布地址分配及数据转发的方式，有利于策略数据的可靠传送。在功能上可以对 PEA 进行扩展，一方面，PEA 具有一定的数据缓冲能力，由它负责本安全域内的 PEP 的策略重发请求，从而减轻策略服务器的策略重发负担，减少重传因子值，有效降低策略重发带来的延迟，有利于系统整体性能和网络带宽利用率的提高，另一方面，可以对发送的策略请求命令进行加密处理，并能对接收的策略数据进行解密，从而提高系统的安全性。

在使用单一 PEA 的情况下，Law 和 Saxena[10] 指出了能够通过增加策略服务器（PSs）的数量来提高 PSs 与 PEA 之间的平均数据吞吐量（包括策略请求和策略发布数据等），而且在实验中发现，使用多 PSs 的平衡，能明显延长在策略请求过载时系统的崩溃时间，同时，增加 PEA 的数量并不明显影响系统的平均数据吞吐率，因此，可以通过增加 PEA 的数量来提高系统的灵活性和扩展性。

4 实验与结果

4.1 实验方法

为了验证安全策略自适应发布模型的高效性，将全策略

分发部署模式与本文的方法进行比较。其中全策略分发部署模式是对策略请求或策略发布命令，始终采取从策略知识库或本地策略缓存中获取策略并全部下发的方式。实验中对策略服务器发送一定量的策略请求，在线分析两种不同的发布模式，比较策略服务器对策略请求的处理效率，以及策略发布数据对网络信道资源的占用情况。

4.2 实验结果

首先采用全策略发布模式，各防火墙每分钟发送一次策略请求，策略服务器接受请求，找到所请求的全策略并通过PEA转发给防火墙。策略服务器采用基于Hash索引的链策略搜索算法对本地策略缓存进行搜索，找到策略后将其全部下发给PEA。在PEA端，对所接收到的全策略进行确认，用来评价策略服务器对策略请求的处理效率。从第一次发出策略请求后，每隔30秒统计一次策略请求处理量。实验中对下行数据包，即PEA接收的数据包进行测试，用来评价策略发布过程中PEA接收策略数据的流量，并反映对网络信道资源的占用率。

最后采用安全策略自适应发布模式，在其它条件相同的情况下，对策略请求加入策略过滤、异构策略生成、发布等过程。为了模拟策略服务器中策略被修改的情况，采用SQL语句的事务处理过程，由数据库程序调用，定时对策略知识库中防火墙策略规则进行部分或全部字段值修改。

两种模式下PEA收到策略发布的累计确认量如表3所列，表中给出了第一次策略请求后7分钟内的累计确认量。

表3 策略请求的处理量

时间(min)	全策略发布	自适应策略发布
0.5	775	505
1	1,121	733
1.5	1,911	1,537
2	2,186	1,689
2.5	2,946	2,371
3	3,312	2,730
3.5	4,081	3,585
4	4,187	4,021
4.5	4,921	4,891
5	5,094	5,180
5.5	5,770	5,858
6	5,996	6,310
6.5	6,538	6,995
7	6,957	7,540

根据表3的数据可计算出两种模式下策略服务器对策略请求的分时处理效率，如图4所示。在全策略发布模式，策略服务器对所有的请求都作相同的处理，由于并发的策略请求，以及网络流量的影响，使得对请求的处理量始终小于请求量。在自适应发布模式，系统运行初期由于需要初始化较多的支撑环境，因此对策略的请求处理量小于全策略发布模式。但当系统运行环境稳定，所有的防火墙都经过了至少一次请求和确认周期之后，自适应发布模式的策略请求处理效率开始明显高于全策略发布模式，可以看出，这个时间点大约是在第一次请求发生后3分钟左右。实验中发现，随着运行时间的延长，自适应发布模式的策略处理效率的优势越明显，而且，如果降低防火墙规则的修改频率，会显著提高策略请求的处理效率。

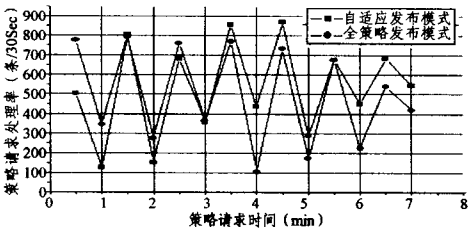

图4 策略请求处理效率

根据抓取的下行数据包，统计数据大小的流量，如图5所示。在系统运行初期，两种模式下行峰值流量均基本保持在30～35Mbps之间，系统稳定之后，全策略发布模式流量始终维持在28Mbps左右，但在3～10分钟内，自适应发布模式流量有一个明显的下降过程，达到15Mbps左右，在实验10分钟之后，未对策略知识库进行修改，继续经过一段时间的观察，下行数据流量逐渐保持在一个较低的水平，信道资源占用率大约只是全策略发布模式下的三分之一。

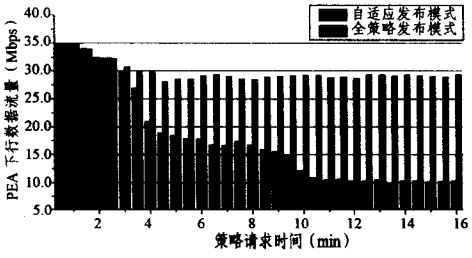

图5 PEA接收数据流量

结束语 针对网络安全策略的发布效率问题，本文考虑其影响因子，提出了自适应安全策略发布数学模型和结构模型，为了精炼策略传输数据，建立了面向属性状态和操作的异构策略表示方法，基于缓存技术的策略快速搜索和异构策略生成算法避免了频繁的搜索过程，基于安全域的地址分配及数据转发算法为高效发布提供了保障，时间复杂度好，具有可行性和可靠性。实际上，该模型也适用于多级分布式安全策略管理系统环境，实验表明，基于异构的安全策略自适应发布模型具有较高的策略处理效率和较低的网络资源占用率。同时发现，策略的请求处理效率受策略请求的上行数据影响，即PSs的处理量与峰值流量有关，如何平衡PEA的策略请求与PSs的策略处理关系，是下一步工作的重点。

参考文献

[1] Boukerche A, Dzermajko C, Kaiyuan L. An enhancement towards dynamic grid-based DDM protocol for distributed simulation using multiple levels of data filtering[J]. Parallel Computing, 2006, 32(11/12): 902-919

[2] Zhang Yachong, Sun Guoji, Zhang Yajun, et al. A new algorithm of data distribution management for distributed interactive simulation[C]//Proc. of the 5th Intelligent Control and Automation. Hangzhou, China, 2004

[3] Ondi A, Menezes R, Ford R. Network Distribution of Security Policies via Ant-like Foraging Behavior[C]//Proc. of the 2th International Conference on Interent and Web Applications and Services. Morne, Mauritius, 2007

[4] Tang Chenghua, Yao Shuping, Cui Zhongjie, et al. A Network Security Policy Model and Its Realization Mechanism[C]// Proc. of Information Security and Cryptology. Beijing, China, LNCS 4318, 2006

（下转第138页）

选择 J-M, M-B 和 NHPP 模型作为模型综合的输入，通过不断的训练，获得预测结果。将 3 种模型得到的预测值和利用 SaMS 方法得到的预测值进行归一化处理，做预测图，得到图 4。

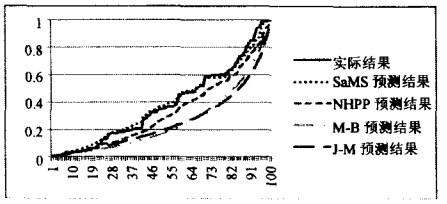

图 4　CSR1 失效数据集预测结果

同样利用 RMSE 方法来比较 3 种单一模型和 SaMS 的预测结果，如表 4 所列。

表 4　各模型的 RMSE 值

模型	J-M	MB	NHPP	SaMS
RMSE	0.0574	0.0586	0.0398	0.00247

根据以上的预测图可以直观地看出，利用本文提出的方法得到的拟合效果是最好的。通过 RMSE 值也说明了本文方法比单一优选模型具有更好的效果。本文利用模型评价准则来评估各种不同模型对失效数据的预测能力，选择优选模型集作为神经网络的输入，而不仅仅依靠模型评价准则选择单一模型得到预测结果；利用优选模型集良好预测能力来训练人工神经网络，克服了较差模型对神经网络训练的干扰，既可以利用神经网络较好的自学习功能，又避免了对优选模型权重的确定。

结束语　本文总结了软件可靠性模型目前常有的一些选择和综合方法，提出了一个自适应的可靠性增长模型选择和综合框架 SaMS，利用实际获得的软件失效数据并通过实验结果验证了它的有效性。目前的工作包括对模型选择问题中评价准则的研究，选择增强型的神经网络方法对模型进行更好的综合。

参 考 文 献

[1] ANSI/IEEE. Standard Glossary of Software Engineering Terminology[S]. ANSI/IEEE: STD-729-1991, 1991

[2] Musa J D. Software Reliability Engineering (V2) [M]. Author House, 2004

[3] Jelinski M Z. Software reliability research[C]// Statistical Computer Performance Evaluation. New York: Academic Press, 1972: 465-484

[4] Cai K, Wen C, Zhang M. A critical review on software reliability modeling [J]. Reliability Engineering and System Safety, 1991: 357-371

[5] Lyu M R. Handbook of Software Reliability Engineering [M]. New York: IEEE Computer Society Press, McGraw Hill, 1996

[6] Wood A. Predicting Software Reliability [J]. IEEE Computer, 1996, 29(11): 69-77

[7] Goel A L, Okumoto K. A Time Dependent Error Detection Model for Software Reliability and Other Performance Measures [J]. IEEE Transactions on Reliability, 1979, 28(3): 206-211

[8] Musa J D. A theory of software reliability and its application [J]. IEEE Transactions on Software Engineer, 1975, 1(3): 312-327

[9] Ascher H, Feingold H. Repairable System Reliability: Modeling, Inference, Misconceptions, and their Causes [M]. Marcel Dekker Inc, 1984: 103-111

[10] Yamada S, Ohtera H, Narihisa H. Software Reliability Growth Models with Testing Effort[J]. IEEE Transactions on Reliability, 1986, 35(1): 19-23

[11] Kececioglu D. Reliability Engineering Handbook [M]. NJ: Prentice-Hall, 1991

[12] Yamada S. Software quality reliability measurement and assessment: Software reliability growth models and data analysis [J]. Journal of Information Processing, 1991, 14(3): 254-266

[13] Yamada S, Ohba M, Osaki S. S-shaped reliability growth modeling for software error detection [J]. IEEE Transactions on Reliability, 1983, 32(5): 475-478

[14] Lyu M R, Nikora A. CASRE: A Computer-Aided Software Reliability Estimation Tool [J]. Computer-Aided Software Engineering, 1992: 264-275

[15] Khoshgoftaar T M, Woodcock T G. Software Reliability Model Selection: A Case Study [J]. ISSRE, 1991: 183-191

[16] Stringefellow C, Andrews A. An Empirical Method for Selecting Software Reliability Growth Models [J]. Empirical Software Engineering, 2002, 7(4): 319- 343

[17] Andersson C. A replicated empirical study of a selection method for software reliability growth models [J]. Empirical Software Engineering, 2007, 12(8): 161- 182

[18] Asad C A, Ullah M I. An Approach for software reliability model selection [C]// Proceedings of the 28th Annual International Computer Software and Applications Conference. COMPSAC'04. Sep. 2004

[19] Yin Qian, Li Jiang. A New Cascade Software Reliability Model [C]// Third International Conference on Natural Computation. ICNC'2007. Aug. 2007: 241-245

[20] Su Yu-Shen, Huang Chin-Yu, et al. An Artificial Neural-Network-Based Approach to Software Reliability Assessment [C]// TENCON'2005. Nov. 2005

(上接第 78 页)

[5] Adnan M, Alhajj R, Barker K. Performance analysis of incremental update of association rules mining approaches[M]. Intelligent Engineering Systems. Mediterranean Sea, 2005

[6] 朱玉全，孙志挥，季小俊. 基于频繁模式树的关联规则增量式更新算法[J]. 计算机学报，2003，26(1)：91-96

[7] Lin Ming-Yen, Lee Suh-Yin. Incremental update on sequential patterns in large databases by implicit merging and efficient counting[J]. Information Systems, 2004, 29(5): 385-404

[8] 易彤，徐宝文，吴方君. 一种基于 FP 树的挖掘关联规则的增量更新算法[J]. 计算机学报，2004，27(5)：703-710

[9] Seong - Ryong K, Loguinov D. Modeling Best - Effort and FEC Streaming of Scalable Video in Lossy Network Channels[J]. IEEE/ACM Transactions, 2007, 15(1): 187-200

[10] Law K L E, Saxena A. Scalable Design of a Policy-Based Management System and its Performance[J]. IEEE Communications, 2003, 41(6): 72-79