遗传算法在蜜网动态负载均衡中的应用

曾蛟龙，胡荣贵，谷　裕，许成喜
（电子工程学院 网络系，合肥 230037）

摘　要：针对蜜网动态负载均衡过程中产生的额外通信开销问题，首先分析了蜜网动态负载均衡的特点，建立了基于最小通信开销的动态负载均衡数学模型；然后设计和实现了一种利用遗传算法解决该问题的新方法。实验测试表明，与贪心算法相比，遗传算法可获得更小通信开销的负载分配方案，能进一步减少蜜网动态负载均衡中负载迁移次数，降低额外通信开销。

关键词：蜜网；遗传算法；贪心算法；动态负载均衡；Honeyd
中图分类号：TP309　　**文献标志码**：A　　**文章编号**：1001-3695(2012)06-2253-05
doi:10.3969/j.issn.1001-3695.2012.06.067

Application of genetic algorithm in honeynet dynamic load balancing

ZENG Jiao-long, HU Rong-gui, GU Yu, XU Cheng-xi
(*Dept. of Network, Eectronic Engineering Institute, Hefei* 230037, *China*)

Abstract: For the problem of additional communication overhead in dynamic load balancing process in honeynet, first, this paper analyzed the traits of honeynet and built the mathematical model of the problem based on minimum communication overhea. Then, it designed and implemented a new method used genetic algorithm to solve the problem. At last, experimental tests show that the genetic algorithm can obtain better allocation scheme which has smaller communication overhead than greedy algorithm, it can reduce the number of load migration further and additional communication overhead in dynamic load balancing process in honeynet.

Key words: honeynet; genetic algorithm; greedy algorithm; dynamic load balancing; Honeyd

蜜网是网络安全领域中一种基于蜜罐技术的主动防御资源，是由多个蜜罐构成的网络体系架构[1]。在本文中，蜜网是指包含有多个低交互蜜罐（如 Honeyd[2]）的诱骗网络[3~5]，其中各个低交互蜜罐之间通过网络互连，每个低交互蜜罐可同时模拟大量虚拟主机。蜜网动态负载均衡是指在蜜网运行过程中，根据低交互蜜罐所在物理主机的状态，适时地在各个低交互蜜罐之间迁移虚拟主机，实现负载均衡，提高蜜网的可交互性和保真度[6]。

目前，国内外针对蜜网动态负载均衡问题的研究还较少，在文献[6]阐述的各种蜜网中，系统通过重定向技术将部分服务和流量转移至系统其他节点，实现了一种较为简单的负载均衡。这种均衡仅是一种静态的负载均衡策略，不能满足蜜网在运行中的负载均衡需要。本文主要解决蜜网的动态负载均衡问题。

动态负载均衡（dynamic load balancing，DLB）是一个经典的组合优化难题，是 NP 完全问题[7]。它主要应用于分布式并行计算领域[8~10]，其主要目标是如何在多台计算机之间更合理地分配负载，避免系统出现某些计算节点过载、某些节点轻载的现象，从而提升系统整体性能。在 DLB 过程中产生的额外通信开销将降低系统动态负载均衡性能，并且在蜜网中，随着各节点间网络延时的增大，额外通信开销制约蜜网 DLB 性能的影响也将进一步增大。因此，如何最大限度地降低 DLB 过程中各节点之间的通信开销成为影响蜜网 DLB 性能的一个重要问题。目前针对降低 DLB 过程中额外通信开销问题的解决思路主要是采用贪心算法进行处理，如文献[11]提出的动态负载均衡算法，采用重负载优先分配策略，有效地提升了数据并行程序的执行效率；文献[12]提出的 LRS 算法，采用轻负载优先接收的分配原则，有效地解决了过载节点数远大于轻载节点数时的负载分配问题；文献[13]则综合重负载优先和轻负载优先的两种策略，提出了一种自适应的负载分配算法，有效地降低了负载均衡中的通信开销。虽然利用贪心算法能解决负载分配问题，但因为上述几种算法无法同时满足贪心选择性质和最优子结构性质，所以得到的负载分配方案往往是局部最优的，而且对解决某些特殊情况下的负载分配问题的效果不理想。

1　蜜网动态负载均衡

蜜网动态负载均衡主要由低交互蜜罐、代理和负载均衡中心三个部分实现，如图 1 所示。该图只是侧重显示了蜜网中与动态负载均衡相关的部分。

低交互蜜罐可以同时模拟上千个虚拟主机，通过配置，每一个虚拟主机可模拟任意服务、操作系统。常见的低交互蜜罐有 Honeyd、LaBrea[6] 等。

代理负责向负载均衡中心反馈低交互蜜罐主机状态；接

收稿日期：2011-12-04；**修回日期**：2012-01-12

作者简介：曾蛟龙（1984-），男，湖南邵东人，硕士研究生，主要研究方向为信息安全（315712486@qq.com）；胡荣贵（1966-），男，安徽合肥人，教授，硕导，博士，主要研究方向为信息安全；谷裕（1985-），男，安徽合肥人，讲师，硕士，主要研究方向为信息安全；许成喜（1987-），男，安徽合肥人，硕士研究生，主要研究方向为信息安全.

收、执行负载均衡中心发送的指令，并协同其他代理一起完成负载迁移任务；动态控制低交互蜜罐的配置信息，如动态增加或删除某些虚拟主机、修改虚拟主机的服务等，从而实现动态管控低交互蜜罐的目的。在这里，负载迁移是指在低交互蜜罐主机之间迁移虚拟主机。

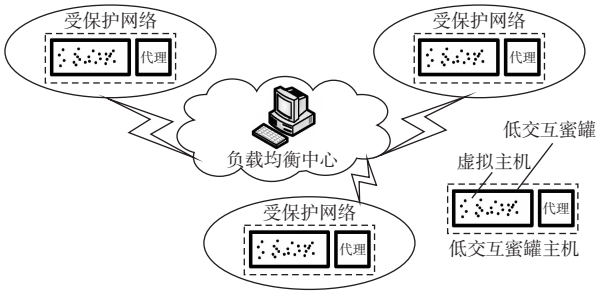

图1 蜜网动态负载均衡体系结构

负载均衡中心是蜜网动态负载均衡的核心，它根据 DLB 策略，适时向各个代理下达负载迁移指令。

由此可见，蜜网动态负载均衡是由负载均衡中心根据各个低交互蜜罐主机状态，适时向各个代理发送负载迁移指令，代理接收指令后，向相应代理进行负载迁移并由低交互蜜罐加载执行，从而完成蜜网的动态负载均衡机制。

2 DLB 数学模型

本文以最大限度降低动态负载均衡产生的额外通信开销为目标，结合蜜网动态负载均衡特点，提出了该问题的数学模型。

定义 1 过载向量。过载向量 $H_n(t)$ 记为 $H_n(t)=(h_1(t),\cdots,h_i(t),\cdots,h_n(t))$，其中 $h_i(t)$ 表示 t 时刻导致第 i 号低交互蜜罐主机过载的负载量，且向量中各元素降序排列。

定义 2 轻载向量。轻载向量 $L_m(t)$ 记为 $L_m(t)=(l_1(t),\cdots,l_i(t),\cdots,l_m(t))$，其中 $l_i(t)$ 表示 t 时刻第 i 号低交互蜜罐主机所能接收的负载量。

$H_n(t)$ 和 $L_m(t)$ 都是经过归一化处理后的结果，该处理过程使得向量中的每一个元素属于(0,100)。

定义 3 负载分配矩阵。负载分配矩阵 $X_{n\times m}(t)$ 表示 t 时刻过载向量 $H_n(t)$ 到轻载向量 $L_m(t)$ 的映射关系，即 t 时刻的一个负载分配方案，可表示为

$$X_{n\times m}(t)=\begin{pmatrix} x_{11} & \cdots & x_{1m} \\ \vdots & & \vdots \\ x_{n1} & \cdots & x_{nm} \end{pmatrix}$$

其中：$x_{ij}\in[0,1]$ $(i\in[1,n],j\in[1,m])$ 表示节点 i 向 j 进行负载迁移的百分比。若 $x_{ij}\neq 0$，意味着过载节点 i 向轻载节点 j 进行负载分配，反之则无负载分配。

当 $\sum_{i=1}^{n}h_i(t)\leqslant\sum_{i=1}^{m}l_i(t)$ 时，该问题表达式如下：

$$\min\sum_{i=1}^{n}\sum_{j=1}^{m}[x_{ij}] \tag{1}$$

s. t.

$$\sum_{j=1}^{m}x_{ij}=1 \quad i\in\{1,2,\cdots,n\} \tag{2}$$

$$\sum_{i=1}^{n}h_i(t)\times x_{ij}\leqslant l_j(t) \quad j\in\{1,2,\cdots,m\} \tag{3}$$

$$x_{ij}\geqslant 0 \tag{4}$$

其中：式(1)为目标函数，式(2)～(4)为约束条件。式(1)表示取迁移次数最少的负载分配方案；式(2)表示过载向量必须分配完毕；式(3)表示分配到某一轻载节点的负载量不能超过其可接收的负载量。

注：当 $\sum_{i=1}^{n}h_i(t)>\sum_{i=1}^{m}l_i(t)$ 时，可优先处理过载向量中的重负载节点，转换成式(1)～(4)能表达的处理情况。

3 遗传算法

针对上述 DLB 数学模型中解空间大及遗传算法的全局搜索能力强的特点，本文采用遗传算法[14]进行求解，并对算法中若干关键问题进行了分析和解决，如染色体编码和可行性验证、适应度函数设计、利用贪心算法生成初始种群以及交叉运算等。模型中的一个分配矩阵对应算法中的一条染色体，记为 $A_{n\times m}(t)$。

3.1 染色体编码

为减少遗传算法解空间的大小和增大染色体变异成功的可能性，染色体的每一位基因采用二进制编码，用 a_{ij} $(a_{ij}\in\{0,1\})$ 表示。若 $a_{ij}=0$ 表示节点 i 不向节点 j 进行负载分配；$a_{ij}=1$ 则相反。每一条染色体所含有的基因个数称为染色体的长度，分配矩阵的每一行视为一个基因组。图 2 显示了当 $H_3(t)=(6,4,3)$，$L_3(t)=(7,6,2)$ 时其中三条染色体的编码实例。

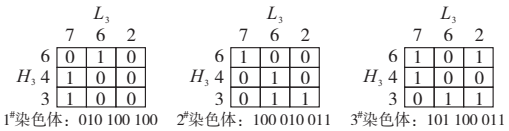

图2 染色体编码实例

图 2 显示，每一条染色体的长度为 9，含有 3 个基因组。1#、2#、3#染色体依次需要进行 3、4、5 次负载迁移操作。

3.2 染色体可行性验证算法

染色体的可行性验证算法的目的是判断染色体是否满足模型中的约束条件。按照计算步骤可划分成初步验证、降维和线性方程组判断三个子算法，其中前两个子算法的时间复杂度远低于最后一个子算法。

初步验证子算法完成对染色体所确定的负载分配方案进行初步验证，主要是检查每一个过载节点是否进行了负载迁移和过载节点迁移的负载量是否超出对应的轻载节点所能承受的总负载量。初步验证通过的染色体可能满足模型中的约束条件，若不通过，则一定不满足约束条件。算法描述如下：

算法 BaseValidate($H_n(t)$,$L_m(t)$,$A_{n\times m}(t)$)
//对染色体 $A_{n\times m}(t)$ 的可行性进行初步验证
//输入：$H_n(t)$,$L_m(t)$,$A_{n\times m}(t)$
//输出：返回 1 表示初步验证通过，-1 不通过
for($i=1;i<=n;i++$){
  if ($\sum_{j=1}^{m}a_{ij}==0$ or ($\sum_{j=1}^{m}a_{ij}\times l_j(t)$) $<h_i(t)$) return -1;
}
return 1;

由图 2 可知，图中所示的 1#、2#、3#染色体均能通过初步验证子算法。初步验证子算法通过的染色体下一步进入降维处理，其主要目的是降低染色体中“1”基因的数目，降低后续处理方法的计算规模。降维子算法描述如下：

算法 ReduceDimension($H_n(t)$,$L_m(t)$,$A_{n\times m}(t)$)
//对染色体 $A_{n\times m}(t)$ 进行降维处理
//输入：$H_n(t)$,$L_m(t)$,$A_{n\times m}(t)$
//输出：降维后的染色体 $A_{n\times m}(t)'$
for($i=1;i<=n;i++$){
  if($\sum_{j=1}^{m}a_{ij}==1$ and $a_{ik}=1$){$a_{ik}=0;l_k(t)'=l_k(t)-h_i(t);$}

}

return $A_{n\times m}(t)$';

图 3 显示了 1#、2#、3#染色体经过降维后的结果。

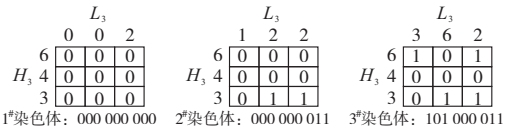

图3　1#、2#、3#染色体降维结果

线性方程组判断子算法是染色体可行性验证算法的最后阶段。该子算法主要是根据 DLB 数学模型的约束条件构造一个线性方程组对染色体进行可行性验证,在该子算法的验证过程中将使用如下推论。

推论　在一个线性方程组所对应的通解中,若非自由未知量的表达式中含有属于(0,1)的常数,则该线性方程组必存在一组未知量都属于(0,1)的解。

该推论的证明只需设通解中所有的自由未知量为一个无限趋于 0 的正数,则非自由未知量的取值必定属于(0,1)。

该子算法的执行步骤如下:

a)若染色体无 1 基因,转步骤 h)。

b)依据 DLB 数学模型构造线性方程组 $A\begin{pmatrix}Z\\Y\end{pmatrix}=b$,令染色体中的 1 基因为基因变量 $Z_i(Z_i\in(0,1))$,若该染色体所对应的分配矩阵中某一列含有 1,则对应地添加一个非基因变量 Y_i $(Y_i>0)$。

c)将增广矩阵$(A|b)$化简成最简行矩阵。

d)若 $R(A)\neq R(A|b)$,转步骤 f)。

e)若最简行矩阵中基因变量所对应的常数属于(0,1),非基因变量所对应的常数为正数,转步骤 h),否则转步骤 g)。

f)返回 -1,算法结束。

g)返回 0,算法结束。

h)返回 1,算法结束。

该子算法返回 -1 表示该染色体不可行,返回 1 表示该染色体可行,返回 0 表示该染色体有可能可行。图 4 显示了线性方程组判断子算法对图 3 中 2#染色体的验证过程。

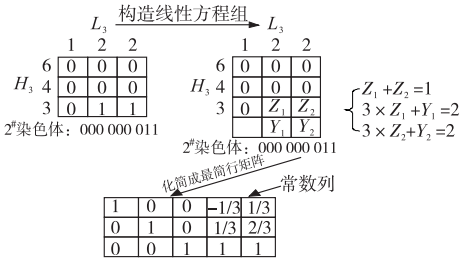

图4　2#染色体线性方程组验证过程

从图 4 可知,2#染色体中所有基因变量所对应的常数(1/3、2/3)属于(0,1),所有非基因变量所对应的常数大于 0。令基因变量为自由变量,非基因变量为非自由变量,则依据上述推论,该染色体可通过线性方程组验证子算法。

综上所述,染色体的可行性验证算法步骤如下:

a)执行初步验证子算法。

b)执行降维子算法。

c)执行初步验证子算法。

d)执行线性方程组判断子算法。

值得注意的是,该算法在执行完第二步后,需要再对染色体进行初步验证,可进一步加快染色体可行性验证过程。例如,降维后的 3#染色体就可以避免执行线性方程组判断子算法,直接由初步验证子算法进行可行性验证。

染色体可行性验证算法的前三个步骤(初步验证—降维—初步验证)仅针对染色体每一个基因组的可行性进行验证,因此,由这三个步骤构成的算法可称之为染色体基因组可行性验证算法,该算法在遗传算法的初始种群生成、交叉、变异操作中有着重要应用。

3.3　适应度函数

本文将适应度函数定义为 $f(A_{n\times m}(t))=c\times f_1(A_{n\times m}(t))-f_2(A_{n\times m}(t))$。其中:$A_{n\times m}(t)$ 为染色体;$f_1(A_{n\times m}(t))$为染色体可行性验证算法的返回值;c 为一个足够大的常数,可取染色体的长度;$f_2(A_{n\times m}(t))$为染色体中"1"基因的个数。

3.4　初始种群生成

为增大初始种群中具备优良基因染色体的比重,本文利用贪心算法局部最优特性可获取具有优良基因种群的特点,在采用随机生成部分种群的基础上,利用重负载优先分配算法、轻负载优先接收算法生产另一部分初始种群。设种群规模为 N(假设 N 为 3 的倍数),则初始种群的生成算法步骤如下:

a)利用重负载优先分配算法生成一个可行染色体 V_0,然后以 V_0 为内点随机生成 $N/3$ 个可通过染色体基因组可行性验证的初始染色体。

b)利用轻负载优先接收算法生成一个可行染色体 V_1,然后以 V_1 为内点随机生成 $N/3$ 个可通过染色体基因组可行性验证的初始染色体。

c)随机生成 $N/3$ 个可通过染色体基因组可行性验证的初始染色体,算法结束。

3.5　交叉算子

为保证交叉运算既能保持上一代染色体的优良基因,同时也能保证交叉运算后的染色体能通过染色体初步验证子算法。本文在传统单点交叉方法[15]的基础上,对杂交点的选择进行了限制——随机选择一个基因组的最后一个基因作为杂交点,令杂交点前的基因不变,杂交点之后的基因在交叉双方之间进行互换。图 5 显示了 1#染色体和 2#染色体之间的交叉运算,交叉点位于第一个基因组的最后一位基因。

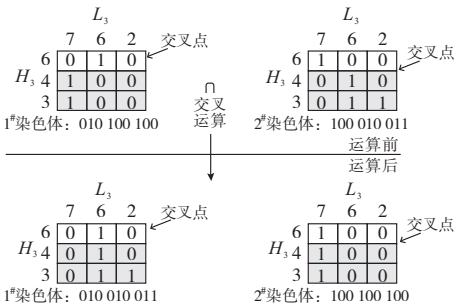

图5　交叉运算过程

3.6　遗传算法描述

借鉴广义遗传算法[16]的进化思想,本算法使种群中的母本在进行交叉和变异后也参与下一代的选择竞争。该算法的终止条件为某一条染色体已经达到最优解或者进化 20 代算法已经稳定或者达到最大进化代数 200 次。该算法描述如下:

a)按照初始种群生成算法生成种群规模为 N(N 为 3 的倍数)的初始种群,放入集合 SET 中,并计算每一条染色体的适

应度,若某一条染色体的适应度已经达到最优解,则返回具有最优解的染色体,转步骤 m)。

b)SET1: = ∅,S: =0。

c)保存 SET 中适应度最大的染色体为 Z。

d)从 SET 中随机抽取 2 个染色体 X、Y 执行交叉运算,得到 X'、Y'。

e)计算 X'、Y'的适应度,若某一条染色体已经达到最优解,则返回具有最优解的染色体,转步骤 m),否则从 X、Y、X'、Y'中选择两条适应度最大的作为 X'、Y'。

f)对 X'、Y'执行变异运算,得到 2 个新染色体 X''、Y''。

g)计算 X''、Y''的适应度,若某一条染色体已经达到最优解,则返回具有最优解的染色体,转步骤 m),否则从 X'、Y'、X''、Y''中选择两条适应度最大的放入下一代种群 SET1 中。

h)SET: = SET - $\{X, Y\}$;如果 SET≠∅,转步骤 c)。

i)选择 SET1 中一条适应度最大的染色体,保存为 Z',若 Z 和 Z'相同,则 S ++;否则令 S: =0。

j)若 S =20,返回 Z',转步骤 m)。

k)SET: = SET1。

l)重复 200 次步骤 c) ~ k)的操作,最后返回适应度最大的染色体。

m)结束。

4 实验测试

实验测试分成两个部分,第一个部分是在没有蜜网应用背景下,测试遗传算法和贪心算法在降低额外通信开销上的优劣;第二个部分则结合蜜网应用环境,对这两种算法进行测试。

4.1 测试 1

用算法 A 表示遗传算法,该算法所使用参数:种群规模 N 为 950,交叉概率为 0.75,变异概率为 0.15。用算法 B 表示重负载优先分配算法,用算法 C 表示轻负载优先分配算法。所有算法均在同一台计算机上运行,该计算机具体配置为:WinXP SP3 操作系统,Pentium(R) Dual-Core 2.79 GHz处理器,2 GB内存。表 1 显示了算法 A、B、C 随着染色体长度的增加,三种算法在平均分配次数上的比较。在测试过程中,对于一特定长度的染色体,系统将随机产生 300 对过载和轻载向量,使每一对向量的染色体长度等于指定长度,算法 A、B 和 C 将逐一对每一对向量进行计算,然后分别对这 300 对的计算结果取平均值。

表 1 三种算法平均分配次数比较

		染色体规模($n \times m$)								
		2×2	3×3	3×4	4×4	4×5	5×5	5×6	6×6	5×8
平均分配次数	A	2.10	3.28	3.38	4.35	4.59	5.88	6.01	6.02	5.94
	B	2.26	3.52	3.62	4.89	4.96	6.12	6.16	6.03	6.01
	C	2.25	3.46	3.58	4.90	4.93	6.13	6.15	6.13	6.03

从表 1 可知,当染色体长度小于某一规模(36)时,算法 A 具有比算法 B、C 更少的平均分配次数;当大于某一规模时,算法 A 的计算结果与算法 B 和 C 相当。并且在实验过程中,针对每一对随机生成的过载和轻载向量,算法 A 计算得到的分配方案均优于或相当于算法 B 和 C 计算得到的分配方案。

4.2 测试 2

依据图 1 所示,在实验室搭建了一个测试环境,采用 Honeyd 充当低交互蜜罐。该测试环境中共包含六台 Honeyd 主机,各个 Honeyd 主机的配置相同(Ubuntu 操作系统,512 MB 内存,Pentium4 2.4 GHz处理器),共同组成一个局域网。初始状态时,每个 Honeyd 的配置如表 2 所示。

表 2 Honeyd 配置

Honeyd	模拟地址空间	虚拟主机服务
1#	192.168.1.0/24	Web 服务
2#	192.168.2.0/24	Web 服务
3#	192.168.3.0/24	Telent 服务
4#	192.168.4.0/24	Web 服务
5#	192.168.5.0/24	Telent 服务
6#	192.168.6.0/24	Web 服务

由此可见,每一个 Honeyd 模拟了 254 台虚拟主机(0 和 255 不进行模拟),每一台虚拟主机根据部署要求向外提供 Web 服务或 Telent 服务。

在 Honeyd 中,当攻击者试图访问某 Web 服务时,将产生一个连接;试图访问 Telnet 服务时,将在生成一个连接的同时启动一个进程,同攻击者进行交互。若大量用户同时访问某 Web 服务或 Telenet 服务时,将使 Honeyd 主机的可用带宽、内存等资源迅速减少。因此,该测试利用一台物理主机对 1# Honeyd 主机、2#Honeyd 主机中部分虚拟主机模拟的 Web 服务进行拒绝服务攻击,对3#Honeyd主机中部分虚拟主机模拟的 Telnet 服务进行拒绝服务攻击,从而使这三个 Honeyd 主机产生负载过重,触发蜜网进行负载均衡。在保证触发蜜网动态负载均衡初始条件一致的前提下,先后测试了遗传算法和贪心算法解决蜜网动态负载均衡问题的优劣。图 6 显示了测试遗传算法在解决蜜网动态负载均衡中各 Honeyd 主机的负载状况。

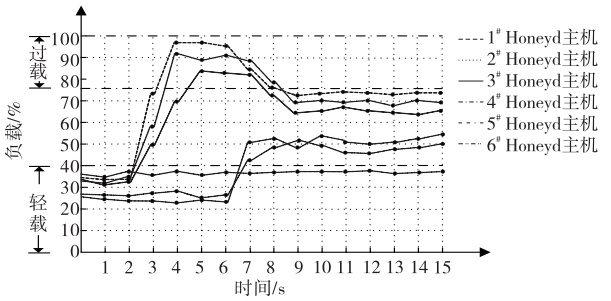

图6 遗传算法解决蜜网动态负载均衡

从图 6 分析可知,在第 3 s 时,攻击者开始进行拒绝服务攻击,1#Honeyd 主机、2#Honeyd 主机、3#Honeyd 主机负载开始急剧上升,其他 Honeyd 主机负载基本稳定,经过一定时间的累积判断,负载均衡中心开始进行负载均衡。根据遗传算法,过载节点开始向 4#Honeyd 主机、5#Honeyd 主机进行负载迁移,同时,负载迁出节点负载开始下降,负载迁入节点负载开始上升,最终趋于稳定状态。其中,6#Honeyd 主机虽然处于轻载状态,但并未参加负载均衡过程。图 7 显示了测试贪心算法(重负载优先分配算法)在解决蜜网动态负载均衡中各 Honeyd 主机负载状况。

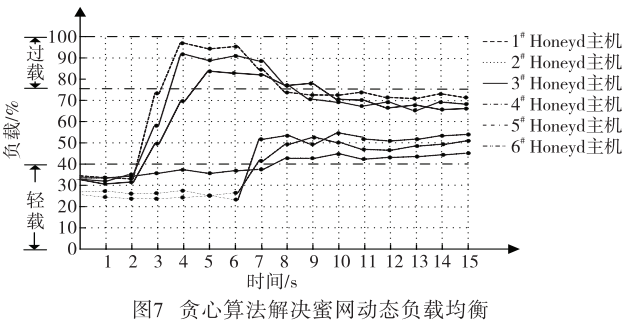

图7 贪心算法解决蜜网动态负载均衡

图 7 显示的触发蜜网进行动态负载均衡的初始条件与图 6 基本一致,但其负载分配方案不同,在贪心算法指导下,6# Honeyd 主机参与了负载均衡过程。

图 8 显示了上述实验中这两种算法在蜜网动态负载均衡过程中产生的额外通信开销。该图 Y 轴表示动态负载均衡产生的网络收发速率,由于在测试环境中,总的网络流量基本等于攻击者产生的网络流量和动态负载均衡产生的网络流量之和,因此,蜜网动态负载均衡所产生的网络收发速率基本等于测试环境中所有主机的网络收发速率减去攻击者主机的网络收发速率。图 8 中曲线与 X 轴之间的区域便可以描述为动态负载均衡所产生的通信开销,其中曲线较为平缓的部分是由于代理周期性向负载均衡中心汇报信息产生的。由图可知,遗传算法产生更少的通信开销。

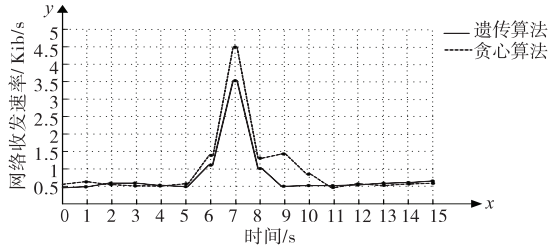

图8　额外通信开销比较

综合上述测试 1 和 2 可知,本文所提出的遗传算法较贪心算法在解决动态负载均衡问题时可获得更优的负载分配方案,并且在解决蜜网动态负载均衡问题时,将产生更少的通信开销。

5　结束语

本文针对蜜网动态负载均衡中产生额外通信开销的问题,结合动态负载均衡理论和蜜网特点,设计和实现了利用遗传算法解决该问题的新方法。通过实验证明,该方法可进一步降低蜜网动态负载均衡中产生的额外通信开销,具备一定的实用性。本文提出的基于最小通信开销的数学模型以及在此基础上的遗传算法,对于解决其他应用系统中的动态负载均衡问题具有一定的借鉴意义。

参考文献:

[1] Honeynet Project. Know your enemy: learning about security threats [M]. 2nd ed. Boston: Addison-Wesley Professional Publishers, 2004.

[2] 江森林. 蜜罐软件 Honeyd 的研究[D]. 无锡:江南大学,2007.

[3] 李世颖,胡荣贵,满毅. 网络目标环境模拟中的主机模型[J]. 计算机工程,2010,36(17):291-293.

[4] 李世颖,胡荣贵,司秀华. 操作系统指纹模拟的设计与实现[J]. 电子工程学院学报,2009,28(4):56-60.

[5] 谷裕,胡荣贵,司秀华. 基于 Honeyd 的大规模网络仿真任务调度策略研究[J]. 电子工程学院学报,2010,29(4):61- 65.

[6] PROVOS N, HOLZ T. 虚拟蜜罐:从僵尸网络追踪到入侵检测[M]. 张浩军,李景峰,等译. 北京:中国水利水电出版社,2011.

[7] 刘岩,韩承德. 负载均衡静态分析方法及其在网络路由分析分配中的应用[J]. 计算机学报,1997,20(9):790-798.

[8] 杨际祥,谭国真,王荣生. 并行与分布式计算动态负载均衡策略综述[J]. 电子学报,2010,38(5):1122-1130.

[9] 董丽丽. 基于 Linux 集群负载均衡算法的分析与研究[D]. 西安:西安建筑科技大学,2009.

[10] DEVINE K D, BOMAN E G, HEAPHY R T, *et al*. New challenges in dynamic load balancing [J]. Applied Numerical Mathematics, 2005,52(2-3):133-152.

[11] 刘振英,方滨兴,胡铭曾,等. 一个用于工作站网络的动态负载平衡算法[J]. 小型微型计算机系统,2001,21(6):651-653.

[12] PARDINES I, RIVER F F. Minimizing the load redistribution cost in cluster architectures[C]//Proc of the 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing. 2004:326-331.

[13] 王玥,蔡皖东,段琪. 一种自适应动态负载均衡算法[J]. 计算机工程与应用,2006,42(21):121-123.

[14] 岳嵚,冯珊. 遗传算法的计算性能的统计分析[J]. 计算机学报,2009,32(12):2389-2392.

[15] 范青武,王普,张会清,等. 遗传算法交叉算子的实质分析[J]. 北京工业大学学报,2010,36(10):1328-1336.

[16] 贺晓丽. 一种用于任务调度的广义遗传算法[J]. 计算机工程,2010,36(17):184-186.