空间限制下的三维大流量常数时间检测

张适可 zsk9026@gmail.com
吴嫣然 polynomia.wyr@gmail.com
从文艳 2908499531@qq.com
李珉超 marshallee413lmc@sina.com

摘要—大流量 (Heavy Hitter) 对于网络而言是比较大的挑战。不均衡的流量会导致一部分线路拥堵，造成网络资源的浪费；另外，网络攻击如 DDoS 必然伴随大流量的特性。如果可以对大流量进行判定，就能采取适当的均衡算法，优化网络中的流量，或者一定程度上预防网络攻击。因此，对大流量的检测尤为重要。然而，交换机中有限的内存限制了对大量 ip 项目的监视，甚至事实上远远达不到监视每一个 ip 的要求，并且实时的检测要求极短的更新时间。我们在这个项目中使用 O(1) 更新时间算法和 O($\frac{H}{\varepsilon}$) 内存算法，其中 H 代表某一个等级的大小。我们的创新点在于：(1). 对时间算法和空间算法进行了一个结合 (2). 基于一维和二维算法创造性地提出了在三维条件下的算法。

I. 背景介绍

大流量源 (Heavy Hitter) 是数据流和网络检测研究中的常用术语，在数据流方面，大流量源指频繁出现的数据项 [1]，在网络检测中，大流量源通常被认为是发出的数据包超过一定阈值的流 [2]，在参考文献 [3] 中，出于检测网络攻击的目的，作者为大流量源做出了一个更广泛的定义，将大流量源定义为一个实体，这个实体的某部分行为在以包数量、字节数或者连接数等为单位计算时，积累超过一定程度，大流量源可以对应于一条流，一个连接，也可以是流或连接的汇聚。

很多大规模的网络攻击都具备大流量源的特性，其中就包括蠕虫扩散以及拒绝服务攻击（DDoS）攻击。当蠕虫开始扩散时，网络中的一个主机感染了蠕虫，该主机会试图传播该蠕虫来感染其他主机，因此会试图建立大量连接。以 Slammer 蠕虫为例，它可以在 1s 内完成 26000 次扫描，这样就会在路由器端看到大量的数据包具有相同源地址。发生拒绝服务攻击时，由于三次握手不完全，可以观察到 SYN 和 ACK 数据包数量之间的巨大差异 [4] 将有大量连接具有相同的目的地址，在路由端就能看到有大量数据包有相同的目的地址。

监测大流量源是数据流模式中的关键问题，也是许多网络，数据库和数据挖掘应用程序的基础。实际应用中，网络管理者通常希望确定哪个 IP 地址正在发送或接收最多的流量，以检测异常活动或优化性能。IP 域有一个非常普遍的层次结构。二进制 IP 地址和前缀可以很容易地应用于表示其他应用程序中的变量，在跟踪大流量时忽略这一点会失去很多有用的信息，所以 Hierarchical Heavy Hitters(HHHs) 的概念应运而生，并广泛使用于在 IP 网络的研究中。

通常情况下，由于网络流量过于庞大，将相关数据存储在内存中的策略是不可行的。相反，我们可以使用流式算法来实时计算（近似）统计数据，顺序访问数据，并在宇宙大小和流长度中使用空间次线性。

吞吐量是在流式算法处理的关键指标。目前，100 Gbps 的网络传输已经成为现实。400 Gbps 被认为是下一个里程碑 [5]。对于如此高的吞吐量需求，有大量的流数据需要处理，而且需要严格的实时数据处理约束。

分层域可以有大量的聚合点。例如，IPv4 域有 2^{32} 个 IP 地址和 2^{32} -1 个前缀。从计算上来说，监视如此大量的项目有以下几方面的问题：

- 每次更新时需要对层次结构进行大量操作：每个新活动都会导致其所有层次结构级别的聚合点更新。以 IPv4 为例，每个与 IP 地址关联的新活动的发生都会更新其层次结构中的所有 31 个前缀。因此，至少需要 32 次内存访问和相应的计算。
- 为层次结构维护大量数据：32 位 IPv4 域具有 $2^{32}-1$ 个聚合点。传统的按项目状态计数器的解决方案为每个项目保留一个计数器。因此，对于 IPv4 域，其内存占用可以轻易地超过几千兆字节。这么大的内存消耗只能用大容量的外部存储（如主存储器）来处理。该方案易受到外部存储设备的低带宽的影响，难以实现高吞吐量。

另外，我们还结合了流式逼近算法。假设单个控制实体子网 IP 是 021.132.145.*，其中 * 是通配符。控制实体有可能在这组 IP 地址之间均匀地分配业务量，使得在地址组 021.132.145.* 中的任意单个 IP 地址都不是 heavy hitter。但是，网络管理员可能想知道子网中所有 IP 地址的流量总和是否超过了指定的阈值。

可以进一步扩展概念以考虑多维分层数据。例如，可以跟踪路由器级别的源地址与目的 IP 地址对之间的流量。在这种情况下，网络管理员可能想要知道源 IP 地址和目标子网之间，源子网和目的 IP 地址之间，或两个子网之间是否存在大流量攻击。之前的工作中已经研究了 HHHs 的一维和多维层次结构 [4][6][7][8]

II. 相关工作

大流量检测的课题在上世纪 80 年代的时候被提出。以 [12][13] 为代表的两篇文章以及其他相关的文章，在这一课题上的研究，为后来的工作提供了诸多的引导和启发。后来，[14][15][16][17] 中都提出了大流量检测在规定的时间窗口中的应用。这些文章的贡献使得我们的检测能够对来自某一个来源的流量突变的情况做出反应，这更加符合工程上的需求。

在 2003 年，[18] 首次提出了分级大流量检测 (Hierarchy Heavy Hitter, 简写为 HHH)，它所提出的一维分级情况下的分级大流量检测近似算法为之后的分级大流量检测的研究进步奠定了基础。以下文章都应用了一维分级的大流量监测，[19] 中的空间复杂度为 $O(H^2/\epsilon)$；[20] 中的空间复杂度降到了 $O(Hlog(\epsilon N)/\epsilon)$，而它的时间复杂度为 $O(-Hlog\epsilon)$。其中，H 为分级的大小，ϵ 为错误率。另外的文章，如 [21] 中的算法更新数据结构所用的时间复杂度为 $O(logn)$，查询的时间复杂度为 $O(\epsilon^{-1}logn)$，空间复杂度为 $O(\epsilon^{-1}logn)$。其中 n 为数据结构所要维护的长度，epsilon 为错误率。

还有一些文章，如 [27] 给出了在特定硬件条件下的高性能方案，其给出了在 FPGA 上深度流水线（deep pipeline）的架构中的高性能的结果。但是这种架构对寄存器数量和硬件性能有很强的依赖。我们希望我们的工作能够不依赖于硬件的性能和平台，实现一定程度上的硬件设备的鲁棒性。

另外的文章中，如 [22] 中参考了隐马尔可夫模型（hidden Markov Models）[23]，贝叶斯网络（Bayesian networks）[24] 以及其他的相关文章 [25][26]，通过建立特定的概率模型，提出了以马尔可夫链为模型的大流量检测的算法。实验证明了以条件概率而非频率维护的数据结构无论在稠密或者稀疏的数据下都最为高效，但是单次的时间复杂度为 $O(logn)$，其中 n 为堆所维护的长度。[22] 中同样使用了一维的分级大流量检测，同时文章通过实验指出哈希函数的使用在分级检测上能获得最高的性能。更进一步地，[28] 中提出了将一维分级大流量检测拓展至多维分级大流量检测的思想，并给出了二维情况下的实现方法。本文将空间复杂度降到了 $O(H/\epsilon)$，时间复杂度为 $O(Hlog(\epsilon N))$。

III. 我们的工作

A. 名词定义及解释

借鉴前人的工作，我们分等级定义 IP 的域。不失一般性，本工作中只考虑了 IPv4 的结构，而 IPv6 可以类推。

完全 IP 我们定义等 IP 等级的最底层为完全 IP，这些 IP 是指向明确的定点 IP 地址，而不是一个局域网。比如 10.162.40.65 就是一个完全 IP，而 10.162.40.* 就不是完全 IP，并且它作为局域网产生（包含）了 10.162.40.65。依此类推，10.162.*.* 产生了 10.162.40.*。可以说一个完全 IP 是由它的任意一个前缀 IP 局域网产生的。由此我们定义 H 为 IP 等级的大小。对于一个 IPv4 的 ip，在一维中大小为 5，二维中大小为 9。在每一个等级中取一个例子为：

大流量维度对于一个一级维度的大流量检测，我们只检测源 IP 中的大流量，被表示为 <src>；而对于二级维度的大流量检测，我们检测某一对源 IP 和目的 IP 中交流的大流量，被表示为 <src , dst>；在三级维度中，我们检测某源 IP，源端口和目的 IP 之间产生的大流量，被表示为 <src, port, dst>。在本文中我们先讨论对于一级维度的大流量检测，再考虑在三级维度中的大流量检测。

一维		二维	
HHH_0	10.162.40.65	HHH_0	10.162.40.65 , 192.168.1.108
HHH_1	10.162.40.*	HHH_1	10.162.40.*, 192.168.1.108
HHH_2	10.62.*.*	HHH_2	10.162.40.*, 192.168.1.*
HHH_3	10.*.*.*	HHH_3	10.162.40.*, 192.168.*.*
HHH_4	*.*.*.*	HHH_4	10.162.*.*, 192.168.*.*
HHH_5	None	HHH_5	10.162.*.*, 192.*.*.*
HHH_6	None	HHH_6	10.*.*.*, 192.*.*.*
HHH_7	None	HHH_7	10.*.*.*, *.*.*.*
HHH_8	None	HHH_8	*.*.*.*, *.*.*.*

IP 偏序定义一个 IP 偏序“$\succ$”，如果某一 ip q 是 p 的前缀，那么 q $\succ$ p。反之，q $\prec$ p。比如 p=<10.162.40.*> $\prec$ q=<10.162.*.*>，这个可以按照级别来判定，级别越高的在这个偏序中排位越高。

IP 直接前驱当某一个 ip（ip_1）可以由另外一个 ip(ip_2) 推广一步得到，则 ip_2 是 ip_1 的直接前驱。比如一个 10.162.40.* 是 10.162.40.65 的直接前驱，而 10.162.*.* 不是。对于二维 IP 来说 P=<10.162.40.*>，<192.168.1.108> 是 p=<10.162.40.65>, <192.168.1.108> 的直接前驱，而 Q=<10.162.40.*>, <192.168.1.*> 不是。

大流量 (Hierarchical Heavy hitter) 给定一个阈值 θ，对于总数为 N 的 IP 地址，如果一个完全 IP 的地址 p 的频率 f(p) $\geq \theta$ N，则认为这个 ip 是一个大流量发出源。

条件大流量 (Exact Hierarchical Heavy hitter) 如果这个 IP 成为大流量的原因不是因为它有一个子地址是大流量，那么它被称为条件大流量，这也是我们需要计算得到的。比如，一个 <10.162.40.65, 1002> 代表了这个 IP 地址为 10.162.40.65 发来的包有 1002 个，而统计结果又表明存在 <10.162.40.*, 1010>，如果阈值为 1000，那么本来 10.162.40.65 和 10.162.40.* 都会被计入大流量。但是条件大流量说明 10.162.40.* 的真实流量为 8(1010-1002)，所以 10.162.40.* 并不在输出的范畴之内。

近似大流量 (Approximate Hierarchical Heavy hitter) 考虑到求精确的解所需的时间复杂度非常高，定义一个组合 $(\delta, \varepsilon, \theta)$，如果

$$Pr[f^*(p) - f(p) \leq \varepsilon N] \geq 1 - \delta \tag{1}$$

$$Pr[f(q) < \theta N] \geq 1 - \delta \qquad \forall q \notin HHH \tag{2}$$

，则认为算法满足准确率要求。

B. 算法

我们的工作主要是基于 [10] 提出的随机算法和 [28] 提出的内存优化算法。随机算法的时间复杂度是 O(1)，空间的复杂度是 O($\frac{H}{\varepsilon}$)，接下来会分别介绍并给出时间、空间复杂度的证明。

1) 空间算法: 我们的空间算法通过设置计数器的数量确定了空间上限。一个（ip , ip 增量）是它的输入，其中 ip 增量是恒大于 0 的。当到来一个新的 ip 的时候，它通过哈希算法定位一组计数器，如果在定位的这组计数器中，当前 ip 的计数器存在，那么就可以直接

加上 ip 增量；反之，如果这组计数器不存在，那么从这些计数器中选取值最小的计数器踢出，把当前 ip 的计数器加上所踢出的计数器的值。Berinde et al.[11] 证明了用该算法所得到的误差上界：

$$\forall \ i \quad |f^*(i) - f(i)| \ \leq \ \frac{N}{m} \tag{3}$$

其中 $f^*(i)$ 代表频率的真值，而 $f(i)$ 代表频率的孤寂值，N 为 ip 的数量，m 为计数器的数量。这也就说明了给定 m 个计数器，频率的估计误差不会超过 $\frac{N}{m}$，空间复杂度为 $O(mH)$，H 为等级的大小。

我们的算法对每一个等级设置一个计数器，然后按照上述规则对每一个到来的 ip 按照条件大流量的规则进行更新。

2) 时间算法: Ran Ben Basat et al. 在 [10] 中提出了 O(1) 的更新时间算法。算法规定一个特定的 V 值，对于一个给定的 ip 地址，随机产生一个数 $k \in [0, V]$，如果这个数小于等于 H，则把该 ip 纳入计算，否则，忽略这个 ip。也就是说这个算法取得流量中的一个样本，并对流量的型进行估计。对于任意一个没有被忽略的 ip，不再更新每一等级的 ip 地址。比如假设该 ip 为 <10.162.40.65>，我们会只随机根据包含与非包含的规则更新某一个等级，如 <10.162.*.*>，而不是更新全部的 5 个等级，因此时间复杂度为 O(1)。[10] 证明了该算法的准确性和误差范围，实现见 [10]。接下来将会介绍我们在维度上作出的拓展。

3) 3D 算法: **包含与非包含关系** 多维的条件大流量检测必须要考虑包含和非包含的关系。对于二维的更新，假设有 {<a,1>, 10} ,{<a,2>, 5}, {<b,1>, 8} ,{<b,2>, 7} 四个 ip。考虑 <*,*> 级别的流量，假设阈值为 10。我们有：

$$< a, * >=< a, 1 > + < a, 2 > - < a, 1 >= 5$$

$$< b, * >=< b, 1 > + < b, 2 > -none = 15$$

$$< *, 1 >=< a, 1 > + < b, 1 > - < a, 1 >= 8$$

$$< *, 2 >=< a, 2 > + < b, 2 > -none = 12$$

值得注意的是这里 <b,*> 和 <*,2> 的值都超过了 10，所以都是大流量源。再考虑 <*,*> 的流量检测，这时候如果把 {<a,1>, 10} ，{<a,2>, 5} ,{<b,1>, 8}, {<b,2>, 7} 加起来再去掉 {<b,*>, 15} 和，{<*, 2>, 12} 和 {<a,1>, 10} 三个，则 {<b,2>, 7} 被算了两次。需要再减掉这部分。对于一维和二维的实现可参考 [10]，接下来介绍三维的具体实现方式。

3D 实现 三维是指源维度、目标维度和端口维度。图1是一个三维组织结构。图中的节点 <S,D,P> 分别代表三个维度的值，每一个节点代表一个 ip，不同等级的 ip 的具体程度不同，最底层的节点为完全节点。连线代表了两个节点之间直接前驱的关系。图1设置了两个源节点，两个目的节点和两个端口。三维与二维的不同在于对于一个 <S,D,P> 节点，<S,*,*>，<*,D,*> 和 <*,*,P> 会将 <S,D,P> 计算三次。最后需要将该节点的部分加回。

三维的条件频率计算公式为：

$$F_{ip} = \sum_{ip_b \in HHH_0} f(ip_b) - \sum_{ip \in HHH} f(ip) + \sum_{\forall q \in glb(ip_1, ip_2)} f(q) + \sum_{\forall q' \in glb(ip_1, ip_2, ip_3)} f(q) \tag{4}$$

再考虑到三维空间下的上下界问题，有

$$F_{max} = \sum_{ip_b \in HHH_0} f(ip_b) - \sum_{ip \in HHH} f_{min}(ip) + \sum_{\forall q \in glb(ip_1, ip_2)} f_{max}(q) + \sum_{\forall q' \in glb(ip_1, ip_2, ip_3)} f_{max}(q) \tag{5}$$

$$F_{min} = \sum_{ip_b \in HHH_0} f(ip_b) - \sum_{ip \in HHH} f_{max}(ip) + \sum_{\forall q \in glb(ip_1, ip_2)} f_{min}(q) + \sum_{\forall q' \in glb(ip_1, ip_2, ip_3)} f_{min}(q) \tag{6}$$

接下来是三维常数时间检测的伪代码：

Algorithm 1 Three Dimension algorithm

Input: DataFlow, θ, δ, N
Output: output result

```
function UPDATE(index)
    d = randomGenerate(0,V)

    if d<H then /* 选择一个等级更新 O(1)*/
        Prefix p=Counters[index]&Counters[d].mask
        Counters[d].INCREMENT(p)
function MAIN
    HH = φ
    sum = ∈_{ip∈HHH_0, ip∈DataFlow} f(ip)
    /*DC=DoubleCount, TC=TripleCount*/
    for level l = HHH_0 to HHH_k do
        for ip ∈ level l do
            f(ip) = sum - HHH(ip,level) +DC(ip, level)
            f(ip) = f(ip) + TC(ip, level)
            f(ip) = f(ip) + 2Z_{1-δ} ∛(NV)
            if f(ip) ≥ θ N then
                HH = HH ∪ {f(ip)}
                output(f(ip), f_min(ip), f_max(ip))

return HH
```

IV. 算法实现

A. 算法测试

我们用 C 语言实现了 1D 和 2D 两个版本的程序，使用的编译器为 Apple LLVM，版本号为 8.1.0。实验环

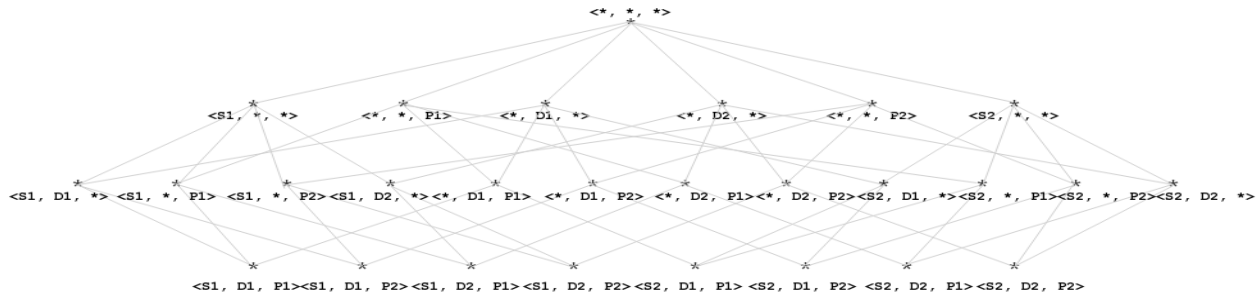

图 1. 3D 框图

Algorithm 2 HHH

Input: ip, level
Output: output result

```
1: function Main
2:     for ip ∈ HH do
3:         result=result+Counters[ip]
4:     return result
```

Algorithm 3 DC

Input: ip, level
Output: output result

```
1: function Main
2:     L=level
3:     result=0
4:     for ip_1 in level L-1 and ip_1 in HH do
5:         for ip_2 in level L-1 and ip_2 in HH do
6:             if ip_1 and ip_2 are not independent then result = result+HH[ChildOf(ip_1,ip_2)]
7:     return result
```

Algorithm 4 TC

Input: ip, level
Output: output result

```
1: function Main
2:     L=level
3:     result=0
4:     for ip_1 in level L-1 and ip_1 in HH do
5:         for ip_2 in level L-1 and ip_2 in HH do
6:             for ip_3 in level L-1 and ip_3 in HH do
7:                 if ip_1 and ip_2 and ip_3 are not independent then result = result+HH[ChildOf(ip_1,ip_2)]
8:     return result
```

境的系统是 macOS Sierra(10.12.6)，处理器为 2.7 GHz Intel Core i7，内存为 16GB 2133 MHz LPDDR3。我们分别对两个版本的程序进行了测试，测试使用的数据集来自 CAIDA（Center for Applied Internet Data Analysis），是芝加哥 08 年的网络数据。测试基于不同的计数器数量，不同的 ip 数量，还有不同的阈值。

B. 实验结果

counter 数	阈值	越界个数	误判数	准确率
1500	200000	5	3	90.36%
2000	200000	5	3	90.36%
3000	200000	5	3	90.36%
3000	100000	5	3	86.42%

表 I
1D 算法实验结果

counter 数	阈值	越界个数	误判数	准确率
1500	200000	26	22	62.79%
2000	200000	26	22	62.79%
3000	100000	27	23	76.23%
4000	100000	27	23	75.23%

表 II
2D 算法实验结果

表I和表II展示了在不同计数器数量和阈值的情况下 1D 和 2D 算法各自的实验结果，实验测试集的 ip 数量均为 2×10^8。其中，越界个数是指超过了设定误差范围却被判定为大流量攻击的 ip，误判个数是指本应该是大流量攻击者却没有被计算在内的 ip 数量，而准确率的计算则是指最终的实验结果中真正为大流量攻击者的 ip 占总结果的百分比。

图2和图3展示了在相同阈值下不同计数器数量对结果正确率和运行时间比的影响。运行时间比是指暴力搜索算法运行时间与随机算法运行时间的比率。实验的测试集数量均为 2×10^7，其中图2和图3的阈值也均设定为为 1^5。

C. 算法应用

由于现有的一些不提供 DDos 防护的 VPS（虚拟专用服务器）提供商会对遭受到大流量攻击的主机持有

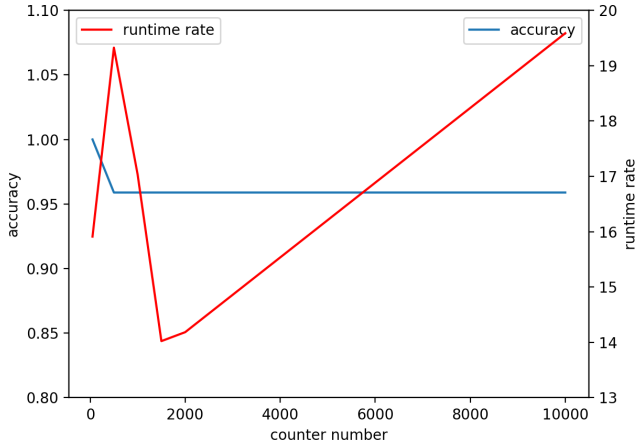

图 2. 1D 算法实验结果

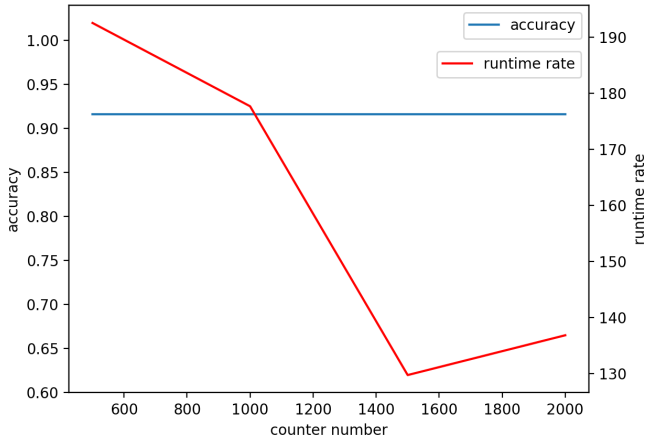

图 3. 2D 算法实验结果

者进行封号处理，但是对于个人用户来说，DDos 防护的价格无疑是高昂的，因此我们将大流量检测算法运用在个人服务器上以便及时发现针对个人的大流量攻击，使得所有者能够快速做出处理（如删除正在遭受攻击的 VPS 等），避免不必要的损失。

图4说明了在网站服务器上检测程序的运作流程。首先将访问者和网站服务器间的数据包进行定时收集，再启动检测程序对收集到的包进行分析，检测是否存在大流量攻击（大流量攻击的阈值由管理员设定），若存在大流量攻击，则启动通知程序，将信息发送给管理员，管理员便可依据通知对服务器做出相应的处理。

整个检测流程我们使用 shell 脚本来控制，可以很轻松地部署到其他网站服务器上，并且支持“热部署”，即可以在 Web 程序正在运行时部署。运行时占用的内存也非常小，仅为实验用机器内存的 7%。图5展示了在

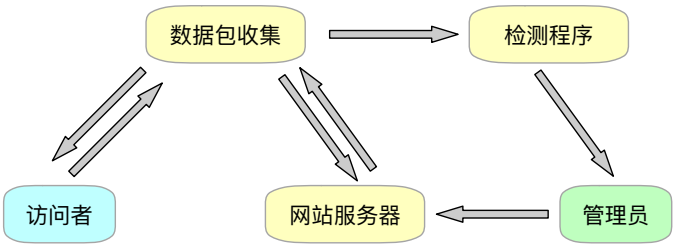

图 4. 检测流程图

服务器上部署检测程序前后的 CPU 占用情况。

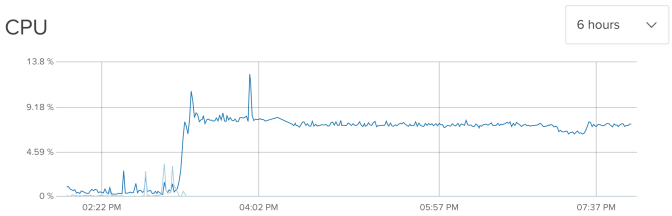

图 5. 部署前后 CPU 占用情况

1) 网站服务器: 本次所使用的网站服务器为 Digitocean 的虚拟专用服务器，机房位于纽约，使用 KVM 虚拟化架构，2 核 CPU，内存大小为 2G，硬盘为 40G 的固态硬盘。系统环境为 64 位 Ubuntu16.04.3。

网站使用基于 Python 的 Web 框架——Flask 搭建。Flask 是一个非常灵活和稳定的 Web 框架，没有绑定诸如数据查询或者表单处理等功能库。我们选用 gunicorn 作为 Web 服务器网关接口（WSGI），并且用 gevernt 库来配合 gunicorn 对高并发的请求做处理。gevent 使用协程来并行地运行程序，它可以轻松处理数百个并发访问请求。

我们还选择了 Nginx 作为网站的 HTTP 服务器和反向代理服务器，即以 nginx 作为前端，为 WSGI 提供反向代理服务。Nginx 的配置比较简单灵活，占用的系统资源更少，能支持更多的并发连接。并且在必要时可以作为负载均衡及缓存服务使用。在 Nginx 中，我们采用了 8080 端口作为外界 Internet 访问的端口，监听本地 Flask 的 5000 端口。

2) 数据包收集: 由于在服务器中我们需要进行实时的抓包检测，因此 tcpdump 是一个很好的工具。tcpdump 是系统提供的一种网络监视和进行抓包的工具，它可以将网络中传输的数据包的头部完全截获下来提供分析，并且它可以指定端口和协议，以及需要的内容。

我们在服务器上设置每隔 20 秒进行抓包，抓包的内容为 8080 端口的 tcp 协议的数据包头，并存储到 *tip.pcap* 中。由于初次获得的顺序还需要经过 tcpdump 的转储才能转换为其他程序可读的格式，因此还需要运行 tcpdump 将文件转储，才能将获取到的头部存储为可利用的格式。

3) 检测程序: 对于个人网站服务器来说，数据包的一端是固定的，即本机的 ip，因此我们使用 1D 算法对 ip 进行分析。所以上一步收集到的数据头部还需要进行处理。我们利用 python 中的正则表达式库提取出数据包的 ip，输入到检测程序中，检测程序便会根据算法输出是否有大流量攻击，若存在大流量攻击，则输出攻击者的 ip 地址。

4) 通知程序: 在这一步我们需要读入检测程序输出的文件，如果读到了存在攻击者，那么就发邮件通知管理员正在遭受攻击，并将攻击者的 ip 地址发送给管理员。在这里我们使用的是 python 自带的 smtplib 和 email 库来发送邮件通知。SMTP 即简单邮件传输协议，它是一组用于从源地址到目的地址传输邮件的规范，通过它来控制邮件的中转方式。发送邮件的邮箱必须先经过 SMTP 认证才可以使用该脚本进行通知。

V. 结论

在本文中我们讨论了大流量攻击检测的算法，并在降低空间和时间复杂度的前提下，提出了从源维度、目标维度和端口维度进行检测的算法。并对一维和二维的检测算法进行了实验，实验结果见表2表3和图I图II。从图中我们可以看出，计数器的数量并不影响结果的正确性，计数器的数量只对时间有一定的影响，并且可以看出随机算法所用的时间要远远小于暴力搜索算法所需的时间。从表中我们可以看到，阈值对算法的正确率有很大的影响，而 1D 算法的正确率又显著地比 2D 算法要高一些。

除此之外，我们还将 1D 算法运用到了个人网站服务器上，用较小的 CPU 占用达到了很好的检测大流量的效果，可以使个人用户及时知晓攻击，并采取相应措施避免损失扩大。我们的检测程序具有检测速度快，可以热部署，并能及时有效通知管理员等优点，但是也还有一些不足之处。比如对大流量攻击阈值的设定需要根据用户服务器的情况进行调试，目前尚没有可以可视化运行的程序方便没有计算机基础的人操作等。目前没有针对大流量攻击很好的解决方法，对于个人用户来说，做好备份，在遭受攻击时及时下线服务器是个不错的选择。

参考文献

[1] Charikar M, Chen K, Colton M F. *Finding frequent items in data streams*. In: Proceedings of the 29th International Colloquium on Automata, Languages and Programming, Springer-Verlag (2002)

[2] Demaine E, Ortiz A L, Munro J I. *Frequency estimation of Internet packet streams with limited space*. In: The 10th Annual European Symposium on Algorithms, Springer-Verlag (2002)

[3] Zhang Y, Singh S, Sen S, et al. *Online identification of hierarchical heavy hitters: algorithms, evaluation and applications*. In: The 4th ACM SIGCOMM Conference on Internet Measurement, Taormina (2004)

[4] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava. *Finding hierarchical heavy hitters in data streams*. In: Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29, ser. VLDB03. VLDB Endowment, pp.464–475 (2003)

[5] M. Attig and G. Brebner. *400 gb/s programmable packet parsing on a single FPGA*. In: Seventh ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS) (2011)

[6] J. Hershberger, N. Shrivastava, S. Suri, and C. D. T´oth. *Space complexity of hierarchical heavy hitters in multi-dimensional data streams*. In: Proc. of the Twenty-Fourth ACM Symp. on Principles of Database Systems, pp. 338-347 (2005)

[7] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava. *Diamond in the rough: Finding hierarchical heavy hitters in multi-dimensional data*. In: Proc. Of the 2004 ACM SIGMOD Intl. Conf. on Management of Data, pp. 155-166 (2004)

[8] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava. *Finding hierarchical heavy hitters in streaming data*. In: ACM Trans. Knowl. Discov. Data, 1:2:1-2:48 (2008)

[9] C. Estan, S. Savage, and G. Varghese. *Automatically inferring patterns of resource consumption in network traffic*. In: Proc. of the 20

[10] Ran Ben Basat, Gil Einziger, Roy Friedman, *Constant Time Updates in Hierarchical Heavy Hitters*. In *ACM SIGCOMM,2017* 127-140

[11] R. Berinde, P. Indyk, G. Cormode, and M. J. Strauss, *Space-optimal heavy hitters with strong error bound*. ACM Trans. Database Syst.,35:26:1-26:28,October 2010.

[12] Misra, J. and Gries, D: *Finding repeated elements*. Science of Computer Programming 2, 143¨C152 (1982)

[13] Boyer, B. and Moore, J.: *A fast majority vote algorithm*. In: Tech. Rep. ICSCA-CMP-32, Institute for Computer Science, University of Texas (1981)

[14] Lee, L. and Ting, H.: *A simpler and more efficient deterministic scheme for finding frequent items over sliding windows*. In: ACM Principles of Database Systems (2006)

[15] Cormode, G., Korn, F. and Tirthapura, S.: *Time decaying aggregates in out-of-order streams*. In: ACM Principles of Database Systems (2008)

[16] Tantono, F.I., Manerikar, N. and Palpanas, T.: *Efficiently discovering recent frequent items in data streams*. In: SSDBM, pp. 222¨C239 (2008)

[17] Dallachiesa, M. and Palpanas, T.: *Identifying streaming frequent items in ad hoc time windows*. Data Knowl. Eng. 87, 66¨C90 (2013)

[18] Graham Cormode, Flip Korn, S. Muthukrishnan, and Divesh Srivastava: *Finding Hierarchical Heavy Hitters in Data Streams*. In VLDB. 464¨C475. (2003)

[19] Yuan Lin and Hongyan Liu.: Separator: *Sifting Hierarchical Heavy Hitters Accurately from Data Streams*. In ADMA (ADMA). 170¨C182 (2007)

[20] Graham Cormode, Flip Korn, S. Muthukrishnan, and Divesh Srivastava: *Finding Hierarchical Heavy Hitters in Streaming Data*. ACM Trans. Knowl. Discov. Data 1, 4 (Feb. 2008), 2:1¨C2:48. (2008)

[21] Kasper Green Larsen, Jelani Nelsony, Huy L. Nguyenz and Mikkel Thorup :*Heavy Hitter Via Cluster preserving Clustering*, (2016)

[22] Katsiaryna Mirylenka, Graham Cormode, Themis Palpanas and Divesh Srivastava:Conditional Heavy Hitters: *Detecting Interesting Correlations in Data Streams*. In The VLDB Journal. 395¨C414 (2015)

[23] Baum, L.E. and Petrie, T.: *Statistical Inference for Probabilistic Functions of Finite State Markov Chains*. The Annals of Mathematical Statistics 37(6), 1554¨C1563 (1966)

[24] Pearl, J.: *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann Publishers Inc. (1988)

[25] Letchner, J., Re, C., Balazinska, M. and Philipose, M.: *Approximation trade-offs in markovian stream processing: An empirical study*. In: ICDE, pp. 936¨C939 (2010)

[26] Wang, P., Wang, H. and Wang, W.: *Finding semantics in time series*. In: ACM SIGMOD International Conference on Management of Data, pp. 385¨C396 (2011)

[27] Da Tong and Viktor Prasanna: *High throughput hierarchical heavy hitter detection in data stream*. In HiPC. (2015)

[28] M. Mitzenmacher, T. Steinkey, and J. Thalerz: *Hierarchy Heavy Hitter with the Space Saving Algorithm*. In ALENEX '12 Proceedings of the Meeting on Algorithm Engineering and Expermiments, 160-174. (2012)