ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.21, No.6, June 2010, pp.1391−1403
doi: 10.3724/SP.J.1001.2010.03522

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

Hits 和 Holds:识别大象流的两种算法*

王 宏+, 龚正虎

(国防科学技术大学 计算机学院,湖南 长沙 410073)

Hits and Holds: Two Algorithms for Identifying the Elephant Flows

WANG Hong+, GONG Zheng-Hu

(School of Computer, National Univisity of Defense Technology, Changsha 410073, China)

+ Corresponding author: E-mail: wh@nudt.edu.cn, http://www.nudt.edu.cn

Wang H, Gong ZH. Hits and Holds: Two algorithms for identifying the large flows. *Journal of Software*, 2010,21(6):1391−1403. http://www.jos.org.cn/1000-9825/3522.htm

Abstract: As networks expanded largely and link speeds grow rapidly, keeping a counter for each flow is too expensive. Estan *et al*. propose two algorithms: sample and hold and multistage filters, to identify large flows, which have obvious shortcomings: sample and hold discard packets randomly while multistage filters simultaneously calculate 5~6 hash functions that is unsuitable for hardware implementation. This paper proposes two algorithms, Hits and Holds, to identify large flows quickly and correctly, which overcome the shortcomings of Estan's algorithms, while effectively reducing false positive and false negative errors.

Key words: collection of network traffic; sampling methodologies; elephant flow; network traffic analysis

摘 要: 随着网络规模的扩大和链路速度的提高,实时采集每条流的流量变得非常困难.Estan 等人提出采集大象流的设想,并提出了识别大象流的算法:Sample and Hold 算法和 Multistage 算法.但这两种算法在实现时存在:Sample and Hold 算法随机丢弃报文,带来采集数据不准确的问题;Multistage 算法需要同时进行 5~6 次访存,无法使用硬件实现的问题.针对上述问题,提出了两种大象流识别算法:Hits 和 Holds 算法.理论和实验结果表明,Hits 和 Holds 算法对网络大象流的误检率和漏检率均优于 Sample and Hold 及 Multistage 算法.

关键词: 流量数据采集;数据采样方法;大象流;流量数据分析
中图法分类号: TP393 文献标识码: A

网络流量采集对于流量计费、带宽规划、路由计算[1]、网络管理和异常流量(如 DoS 攻击)检测[2,3]等网络管理应用是十分必要的[4].然而随着网络带宽的增加和规模的扩大,大规模网络流量采集面临着数据规模庞大和数据到达速度过快[5]的挑战.为了克服上述困难,通常采用硬件实现完成流量采集功能[6,7].当前存在的硬件实现的流量采集方法如 Cisco 的 NetFlow 或者 InMon 的 sFlow 都是基于对数据报文周期性地采样来进行流量采集,这种方法实现存在处理速度缓慢、统计结果不精确、实现代价大、单个流处理访存次数多等缺点.因此,实

* Supported by the National Natural Science Foundation of China under Grant No.90604006 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant No.2008AA01A325 (国家高技术研究发展计划(863)); the National Basic Research Program of China under Grant Nos.2003CB314802, 2009CB320503 (国家重点基础研究发展计划(973))

Received 2008-08-14; Accepted 2008-10-27

现新的效率高、实现代价低、错误率低的硬件流量采集算法是当前研究的热点.

根据 Fang 和 Peterson 对互联网中的流量分析发现[8],流量具有如下特征[9,10]:9%的流发送的数据占到了总流量数据的 90%;相同的流总是具有时间连续性;采用有效的 Hash 算法可以减少流 ID 的 bit 位长,从而减少流 ID 处理时的访存次数以及采用多级过滤器部件可以实现对较大流量的统计,并且忽略那些对网络影响较少的流,从而有效提高统计速度和效率.

数据流模型[11,12]也可以根据不同的时序范围划分成多种子模型,包括界标模型(landmark model)、滑动窗口模型(sliding window model)和快照模型(snapshot model)[13].流计数算法中具有代表性的有 bitmap 算法,文献[4]提出了一系列的 bitmap 算法.面对海量的网络流量数据,多数情况下,网络管理员非常关心网络流量的变化.如果能够自动、实时地发现这些发生变化的流,将可以大大减少网络管理员的注意力.在寻求变化较大的流的算法中,比较有代表性的算法是文献[14,15].其中,文献[14]寻求相对变化,文献[15]利用组合分组测试(combinatorial group test)方法提出了一个寻求各种变化(包括绝对变化、相对变化和方差变化)的框架.为了提高速度,算法放宽了对变化的定义:给阈值ϕ一个误差范围ε,若项的累计值超过所有变化的$\phi+\varepsilon$,则报告,若小于$\phi-\varepsilon$则不报告,介于中间的以误差δ报告.

Estan 提出了采样并保持算法(sample and hold)及多级过滤器算法(multistage)算法[9].当一个数据报文被采样时,如果该数据报文所属的流已经在内存中有一项,则更新该项,如果没有,则创建一个新的项.但是与普通的采样方法不同,对于已经有记录的流,以后每到一个报文就更新该流对应的计数器.该算法的错误率为 $1/M$,M 为算法所需的内存.而传统的随机采样方法的错误率为$1/\sqrt{M}$.

Multistage 算法采用多级过滤器,每个数据报文到达时,更新每级中相应的计数器,并且判断计数器是否超过了阈值.如果每级相应的计数器都超过了阈值,则认为该流为最大流.由于多个流 Hash 到相同的计数器上,可能会误报一些没有超过阈值的流.这种误差是随着过滤器级数的增加而指数减少的,因此误差可以得到很好的控制.后文图 2 给出了该算法的基本思想.

Multistage 算法存在两个不足:1) 算法启动较慢,初期会丢失许多采样报文,因为要等多个 stage 都超过阈值;2) 需要同时进行多个 Hash 运算,无论是使用软件还是硬件都难以实现.具体分析如下:

对于高速网络,用软件实现 Multistage 算法,处理速度远达不到要求,无法实现实时数据采集.因为对于 1Gbps 的网络速度,40 字节报文的到达间隔只有 350ns;10Gbps 的网络,报文到达间隔仅为 35ns.在如此短的时间间隔内,要完成每个报文的多次 Hash 计算几乎不可能.而如果采用硬件实现,级数为 5 的并行 Multistage 需要同时进行 5 次写和 5 次读的操作,无论是 DRAM,SRAM,TCAM 都无法满足要求.因此,Multistage 要实现实时数据采集是很困难的.

1 Hits 和 Holds 算法的思想

为了克服 Multistage 算法难以实现高速网络实时数据采集的问题,本文设计了两种适合实现高速实时数据采集的算法 Hits 和 Holds.

Hits 算法是从 Sample and Hold 思想出发,对于原来直接加入 FlowMemory 中的报文,建立 FlowTable 入口并计数,当计数值超过阈值,则加入到 FlowMemory 中;对于原来以概率 $1-p$ 舍弃的报文,使用 Multistage 计数,如果每一级 stage 均报超过阈值,则将该报文的流标记加入 FlowMemory 中.

Multistage 使用多级过滤器的思想,实际上是为了解决 Hash 运算中的冲突问题.如果能够解决 Hash 冲突问题, Multistage 无须使用多级过滤器.Holds 算法设计了一种解决冲突问题的 FlowTable,使用一级过滤器,实现报文的高速采样.图 1 说明了 Hits 和 Holds 算法的思想.

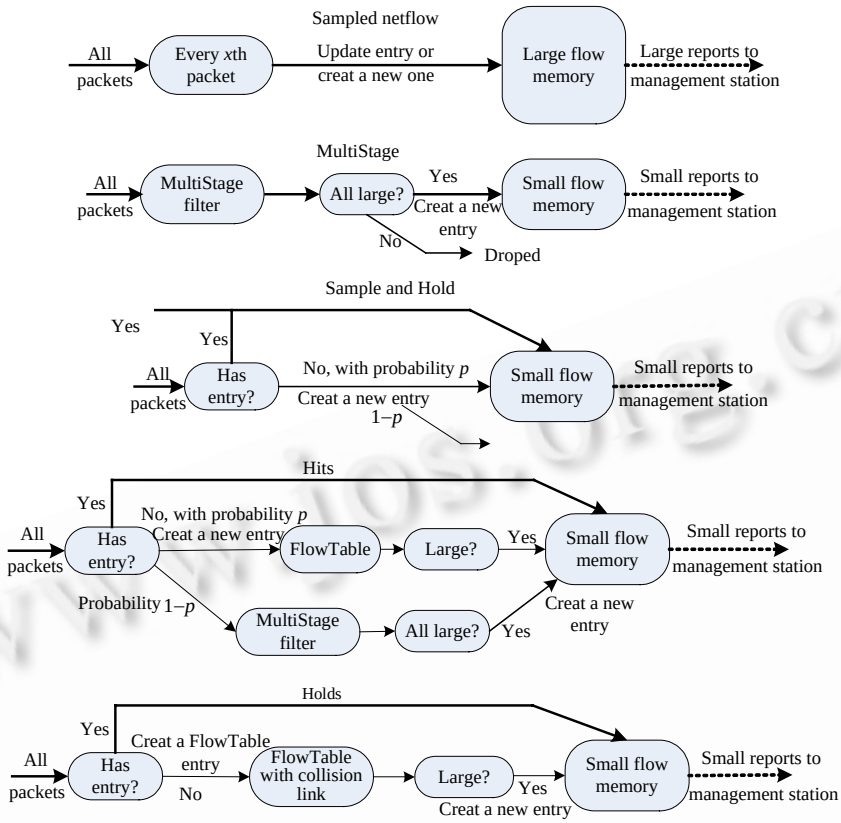


Fig.1 Basic ideas of Hits and Holds compared with other algorithms

图 1 Hits 和 Holds 算法思想与其他算法的比较

2 Hits 算法

本节详细描述 Hits(hit streams)算法,并针对 Hits 算法的正确性和性能进行初步分析.

Hits 算法使用 FlowTable 而非 stage,其目的是:1) 避免同时多次进行 Hash 运算;2) 减少误检率(false positives).计算 Hash,必然存在 Hash 冲突,可能导致一些小流累加成大流,而 FlowTable 的误检率为 0,从而减少误检率.

Hits 为减少误检率和过滤器的级数,设计了分阶段定期筛选 FlowTable 和各 Stage 的 ItemTable 的方法.对发展势头较好的流标志尽早加入 FlowMemory 中,对落选流中发展势头不好的流计数器清零.图 2 说明了 Hits 算法处理报文流的过程.

Hits 克服了 Multistage 算法的不足:1) 解决了 Multistage 算法初始阶段丢弃报文较多的问题.算法开始时 Hits 通过将流表直接加入 Flow Table,所以在算法初始阶段不会丢弃很多报文;2) 解决了难以实现实时采集的问题.Hits 通过增加 FlowTable,减少了处理每个报文流 ID 的访存次数,有效地提高了硬件处理的速度,我们将在下一节详细分析 Hits 的正确性和处理效率.

图 3 是 Hits 的形式描述.

Hits 流量采集算法的处理过程如下:

1) 当一个数据报文被采样时,如果该数据报文所属的流已经在内存中有一项,则更新;如果没有,则以概率 $1-p$ 进入 Hits_Multistage 判断过程,如果 Hits_Multistage 的各 stage 均超过阈值,则创建一个新的项.Hits 初始

FlowTable 的创建过程与 Sample and Hold 相同,概率 p 初值为 1.与 Sample and Hold 不同的是,概率 p 随 FlowTable entry 所占整个表的比例变化而变化,所占比例越大,p 越小.p 逐渐减小,直至为 0.对于已经有记录的流,以后每到一个报文,就更新该流对应的计数器.

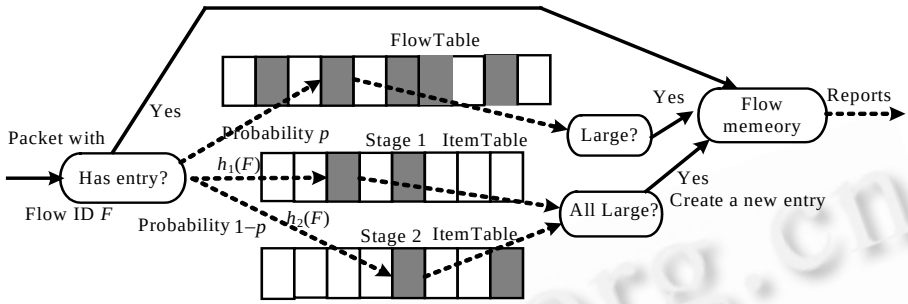


Fig.2 Idea of Hits algorithm

图 2 Hits 算法

```
Procedure Hits_Multistage(FlowID Fid,Size s,StageNum d,
                          Threshold T)
    {   int ngt=0; fsnum=0;
        for (int k=1; k<=d, k++){
            Stage_addNewItem(k,Fid,s,Hi(Fid),fsnum);
            if (fsnum>T) ngt++;}
        if (ngt==d) FlowMemoryAddNewFlow(Fid)}
Procedure CheckEntryofFlowTable(Threshold T,StageNum d,
                                Int R1,Int R0);
    {   FlowEntry i=0; Counter fsnum=0;
        For (i in FlowTable){
            fsnum=GetFlowCounter(i);
            if (fsnum>R1) FlowMemoryAddNewFlow(i);
            if (fsnum<R0) ClearCountInFlowTable(i);}}
Procedure CheckEntryofItemTable(Threshold T,StageNum d,
                                Int R1,Int R0);
    {   ItemEntry i=0; Counter fsnum=0;
        For (k=1, k<=d, k++)
            For (i in stage k ItemTable){
                fsnum=GetItemCounter(i,k);
                if (fsnum>R) FlowMemoryAddNewFlow(Fid);
        if (fsnum<R0) ClearCountInItemTable(i,k);}}
Procedure ReadNewFlow(FlowID Fid,Size s,Probability p,
                      StageNum d,Threshold T)
    {   int i=0;
        if (i=Has_Entry_in_FlowMemeory(Fid)){
            FlowMemeoryFlowAdd(Fid,s,i);
        } else {
            create new entry in FlowTable with
            probability p;
    Hits_Multistage(Fid,s,d,T) with probability 1−p;}}
Procedure PhaseCheck(int Phase,param α,param β);
    If (Phase<=TPhase){
      R=α*T*Phase/TPhase; /*Caculate Interval
                            Threshold R*/
      R0=β*T*(Phase−1)/TPhase;
      CheckEntryofFlowTable(T,d,R,R0);
          CheckEntryofItemTable(T,d,R,R0);}}
Main()
    {Constant TPhase=5; d=2; α=1.5, β=0.5,
     Time TimeInterval=MeasureInterval/TPhase;
     While (1) {
        int Phase=1;
        while (Phase*TimeInterval<MeasureInterval) {
            while (ComeNewFlow)
                ReadNewFlow(Fid,s,p,d,T);
            If (DiffTime>TimeInterval){
            PhaseCheck(Phase,α,β);
            Phase++;}
        p=1−(FlowTableEntrySize/TotalSize);}}
     }
```

Fig.3 Descriptions of Hits

图 3 Hits 算法描述

2) 算法下半部分是一个具有较少级数的多级过滤器算法.如果级数可以降为 3 甚至 2,多级过滤就可以使用串行方式实现.每一级过滤器都是一个计数器的列表,使用流 ID 的 hash 函数进行访问.在测量周期开始前,整个表都被清零.当流 ID 到达时,提取出报文长度,然后计算相应流 ID 的 hash 值,将报文长度乘以采样率累加到对应的计数器上.由于同一个流将 hash 到相同的计数器上,当一个流的数据量超过了预定的阈值 T,对应的计数器就会超过阈值.但是,由于流的数目远远超过了计数器的数目,很多流就会 hash 到相同的计数器上,可能很多 hash 到相同计数器的很小的流加起来超过了 T.这样会导致误检率增加:(1) 很小的流会 hash 到最大流的计数器上,导致误报;(2) 很多很小的流会 hash 到同一个计数器上,累计结果超过 T,导致误报.

我们采用 4 种措施来减少误检率:(1) 对 FlowTable 和 ItemTable 的 Entry 进行周期性筛选,设定筛选阈值 R 和 R_0,$R_0<<R$,将 FlowTable 中计数器中数值没有超过 R_0 的入口项进行删除,ItemTable 中计数器中数值超过 R 的

入口项 ID 加入到 FlowTable 中,小于 R_0 的入口项删除;(2) 在 FlowTable 有入口的流不再进入 Hits_Multistage 过程;(3) 进入 Hits_Multistage 过程的概率随 Entry 个数在 FlowTable 中的增加而减小;(4) Hits_Multistage 计数采用保守方法[9].

3 Hits 算法性能分析

本节将从算法的空间、时间复杂度和误检率对算法进行分析.假设链路带宽为 1GB/s,并且假设有 10^6 条流在链路中,我们要发现其中发送速率超过链路速度 0.1%,即 1MB/s 的流,并且假设测量周期为 10s,则阈值 T 设为 10MB.Multistage 的级数设为 4,每一级计数器数量为 10^4 (ItemTable 的 entry 数量),FlowTable 使用 10^4(最大流数的 10 倍)个计数器,stage 级数设为 2,每一级计数器数量与 Multistage 的相同,为 10^4.首先我们分析 MultiStage 的效果,然后再分析 Hits 的情况.

对于 MultiStage 算法,一条发送速率为 100KB/s 的流通过所有过滤器的概率,即误检率(false positive).引起这条流误报的原因是其他的流 hash 到了同一个计数器上,并且在测量周期内累加的数目超过了 10MB.这样,其他流就必须要贡献出 10MB−100KB×10=9MB 的流量.最多有 10^6/100 个这样的流,需要(10000−1)/9=1111 个计数器,该流通过一级过滤器的概率为 1111/10000=11.11%,如果采用三级过滤器,通过率为 1.3%;若采用四级过滤器,则通过的概率为 1.523×10^{-4}.

对于 Hits 算法,最多有 10^6/100 个速率为 100KB/s 的流,如果这些流全部进入 FlowTable,则 Hits 的误检率为 0%(因为 FlowTable 误检率为 0%).对于进入 Hits_Multistage 的流,分析由于采用了减少误检率第(1)项措施,计算较为复杂.一级过滤器通过概率分析如下:设在整个测量周期内共进行 5 次阶段检查,第 1 次检查没有被删除的概率(为计算方便,设β=1,i 为当前检查周期,$R_0=(i-1)\times T/h$)为(10000−1)/((2−1)×10000),第 2 次为(10000−1)/((4−1)×10000),…,第 5 次为(10000−1)/((10−1)×10000).通过这 5 次检查的概率为所有概率相乘,一级过滤器通过概率为 0.1052%,二级过滤器通过概率为 1.106×10^{-6}.所以,Hits 的误检率要小于 MultiStage.由于只采用了二级过滤器,使得算法较 Multistage 容易实现.

而将最大流的流量分离后,仅仅有少部分流量会送到多级过滤器进行分析.因此,不仅使用多级过滤器的次数大大减少,访问内存的次数也减少.对最大流进行单独处理,可以更为精确地记录流的大小,同时也提高了检测算法的准确性.在面对大量流的时候,内存的消耗是可以预测的.当新报文到来时,访存次数也大为减少,大部分情况下访存为 1 次,访存最多为 5 次.

下面对算法性能进行形式化分析.符号定义见表 1.

Table 1 Notations

表 **1** 符号说明

Symbol	Meaning	Symbol	Meaning
p	The probability for sampling a byte	s	The size of a flow (in bytes)
C	The link capacity in a measurement interval	Q	The number of bytes actually counted for a flow
c	The number of bytes actually counted for a flow	T	The threshold for large flows
m	The ratio of threshold in link capacity	d	The depth of the fiLter (the number of stages)
b	The number of buckets in a stage	k	The stage strength is the ratio of the threshold and The average size of a counter.
h	The stage number in a measurement interval	R, R_0	The threshold be selected or eliminate
α	Factor of enlarge threshold	β	Factor of reduce threshold
n	The number of active flows	i	The current stage

由定义 $b=k\times C/T$,即 $k=T\times b/C$.

引理 **1**[9]. 一条大小为 s 的流通过一级过滤器的概率 $p_s \le \dfrac{T}{k(T-s)}$.

定理 **1**. 设在 FlowTable 中的超额采样因子为 Q,则 Hits 中一条在测量周期内超过阈值 T 的流 F 不被采样的概率小于$(1-Q/m)^{mC}\approx e^{-Q}$.

证明:由定义有 $T=C\times m$,则在测量周期内超过阈值 T 的流最多有 $1/m$ 条,在 FlowTable 中总共有 Q/m 入口,

每个字节不进入 FlowTable 的概率为$(1-Q/m)$,流 F 总共发送了超过 $T=C\times m$ 流量,则流 F 不被采样的概率等于$(1-Q/m)^{mC}\approx e^{-Q}$. □

定理**2**. 设每个 stage 使用的 Hash 函数是独立的,则一条大小为$s<T(1-1/k)$的流通过(被误报)Hits_Multistage 的概率:

$$p_s \le \left(\prod_{i=1}^{h-1}\frac{T}{k(T-\beta\times i\times s)}\right)\times\left(\frac{T}{k(T-s)}\right)^d.$$

证明:设一条流的大小为 s,由引理 1 可知,该流通过一级过滤器的概率为 $p_s \le \frac{T}{k(T-s)}$,要通过 d 级过滤器,概率为 $\left(\frac{T}{k(T-s)}\right)^d$.

对于测量周期内每一次阶段检查,筛选阈值设为 $R_0=\beta\times T\times i/h$,则每一阶段不被筛落的概率为

$$\frac{T/h}{k(T-\beta\times i\times s)/h}=\frac{T}{k(T-\beta\times i\times s)}.$$

多阶段不被筛落的概率为 $\prod_{i=1}^{h-1}\frac{T}{k(T-\beta\times i\times s)}$,所以, $p_s \le \left(\prod_{i=1}^{h-1}\frac{T}{k(T-\beta\times i\times s)}\right)\times\left(\frac{T}{k(T-s)}\right)^d$.

定理 **3**. Hits 的内存消耗为 $O(QC/T+bd)$.

证明:FlowTable 的内存需求为 $O(QC/T)$,Hits_Multistage 的内存需求为 $O(bd)$,综合起来为 $O(QC/T+bd)$. □

定理 **4**. Hits 的每一个报文访存次数为 1 或者 $1+\log_{10}n$.

证明:由于使用了 FlowTable,多数情况下只需 1 次查表,而如果进入 Hits_Multistage,则访存增加为 $1+\log_{10}n$.

定理 **5**. 在测量周期内,被 Hits 记录的流个数的数学期望为

$$E(n_f)\le T/C+\max\left(T/C,n\left(\frac{n}{kn-b}\right)^d\right)+n\left(\frac{n}{kn-b}\right)^d.$$

证明:设 s_i 为各种大小流量的一个流的长度,n_i 为这种长度的流的个数,则 $h_i=n_i\times s_i/C$ 为这种长度的流占总流量的比例,有 $\sum n_i=n,\sum h_i=1$,由引理 1 可知, $E(n_f)=n_i\times p_{s_i}\le n_i\times\min(1,(T/(k\times T-s_i))^d)$,由数学期望的线性特征可知, $E(n_f)=\sum E(n_{if})$.

下面我们将流量根据其长度分为 3 组:第 1 组为大于 T 的流量,第 3 组为小于 C/n 的流量,第 2 组为介于两组之间的流量,则

$$E(n_f)=\sum E(n_{if})=\sum_{S_i>T}E(n_{if})+\sum_{\frac{C}{n}\le S_i\le T}E(n_{if})+\sum_{S_i<\frac{C}{n}}E(n_{if})$$

$$\le\sum_{S_i>T}\frac{h_iC}{s_i}+\sum_{\frac{C}{n}\le S_i\le T}\frac{h_iC}{s_i}\left(\frac{T}{k(T-s_i)}\right)^d+n\left(\frac{T}{k\left(T-\frac{C}{n}\right)}\right)^d\le\frac{C}{T}+\sum_{\frac{C}{n}\le S_i\le T}\frac{h_iC}{s_i}\left(\frac{T}{k(T-s_i)}\right)^d+n\left(\frac{T}{k\left(T-\frac{C}{n}\right)}\right)^d.$$

求函数 $f(x)=\frac{1}{x}\left(\frac{T}{T-x}\right)^d$ 在区间 $\left[\frac{C}{n},T\right]$ 的极值,令 $f'(x)=-\frac{1}{x^2}\left(\frac{1}{T-x}\right)^d+\frac{1}{x}\frac{d}{(T-x)^{d+1}}=0$,得 $x=\frac{T}{d+1}$ 时取最小值,所以其两端点值的较大者为最大值. $C\left(\frac{T}{k}\right)^d f(T)=\frac{C}{T},C\left(\frac{T}{k}\right)^d f\left(\frac{C}{n}\right)=n\left(\frac{n}{kn-b}\right)^d$.据此,得

$$E(n_f)\le T/C+\max\left(T/C,n\left(\frac{n}{kn-b}\right)^d\right)+n\left(\frac{n}{kn-b}\right)^d.$$

4 Holds 算法

Holds 算法为解决 Hits 算法中的 Hash 冲突问题,同时避免使用多级过滤器而设计.在 Holds 算法中,对 Hits 算法中的 FlowTable 中有 Hash 冲突的入口建立 Hash 冲突链表,链表中记录保存流的 ID 号和计数器.对于计数器中超过阈值的流标志号加入 FlowMemeory 中.去除 Hits 算法中的多级过滤方法.图 4 说明了 Holds 算法处理报文流的过程.

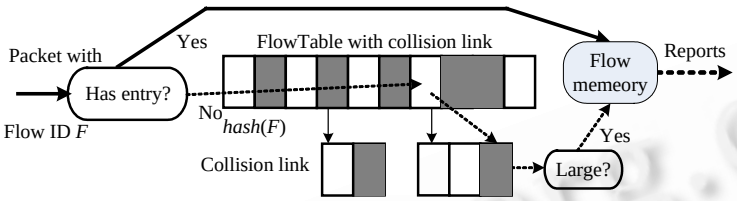


Fig.4 Idea of Holds algorithm

图 4 Holds 算法

Holds 算法的高效实现除 FlowTable 大小以外,一个好的能够实现均匀分布的 Hash 函数也非常重要.Holds 算法解决了 Hash 冲突问题,又避免了同时多次计算 Hash 的问题.

图 5 是 Holds 的形式描述.

```
Algorithm Holds(FlowID Fid,Size s,Probability p,
                StageNum d,Threshold T)
{    int i=0;
if (i=Has_Entry_in_FlowMemeory(Fid)){
          FlowMemoryFlowAdd(Fid,s,i);
} else {
i=HashMapping(Fid);
case FlowTable(i).flownum
0: {Create new entry in flowTable
        FlowTable(i).flownum=1;
      FlowTableFlowAdd(Fid,s,i);}
1: {  FlowTable(i).flownum++;
FlowTableFlowAdd(Fid,s,i);}
>1: {    j=FindFlowInCollisionLink(Fid,s,i);
If (j==True){  FlowTableFlowAdd(Fid,s,i);}
else {
 create new Collision link of i in FlowTable;
FlowTableFlowAdd(Fid,s,i);
FlowTable(i).flownum++;}}
```

Fig.5 Descriptions of Holds

图 5 Holds 算法描述

5 Holds 算法分析

定义 **1**(**Hash** 冲突). 对于任意 x_1,x_2,如果 $f(x_1)=f(x_2)$,则称 x_1,x_2 对于 Hash 函数 $f(x)$存在冲突.

定义 **2**(**Hash** 值冲突深度 ***h***). 对于集合 C,有 $C_1=\{x|\forall x_1,x_2\in C,f(x_1)=f(x_2)\}$,则该 Hash 值的冲突深度定义为

$$h_{f(x_1)}=|C_1| \tag{1}$$

Hash 值的冲突深度越深,则该 Hash 值所对应的报文流越多,确定该 Hash 值所属报文的平均时间就越长,访问存储器的次数也越多,可能造成的拥塞程度越大,实现线速处理并保证不丢包需要的报文缓冲区也越大.

定义 **3**(**Hash** 函数的冲突比例σ). 对于集合 C,有 $C_1=\{x|\forall x\in C,\exists y\in C,$使得 $f(x)=f(y)\}$,则称该 Hash 函数在集合 C 的冲突比例为

$$\sigma=|C_1|/|C| \tag{2}$$

定义 **4**(**Hash** 算法平均访存次数α). 设对于集合 C 中的任意元素 x_i,通过 Hash 函数 $f(x)$访问得到流表表项的访问次数为 $a(x_i)$,则有

$$\alpha=\sum_{i=0}^{|C|-1}a(x_i)/|C| \tag{3}$$

Hash 算法的平均访存次数越多,确定报文所属流的平均时间越长,Hash 算法的整体性能也就越低.

定理 **6**. 设 Hash 函数的冲突比例为σ,Hash 值冲突深度 h,则对于流集合 C,Holds 算法的平均访存次数α为

$$\alpha=\left(\frac{h}{2}-1\right)\sigma+1.$$

证明:设 C 中某一条流访问流表次数为 $a(x_i)$,访存深度为 h_i,则由定义有 $\alpha=\sum_{i=0}^{|C|-1}a(x_i)/|C|=\sum_{x_i\in C}a(x_i)h_i$.

设有 j 条流访问流表时发生了冲突,C 中共有 n 条流,则

$$\alpha=\left(\sum_{x_i\in C}a(x_i)h_i\right)\Big/|C|=((n-j)+\sum_{x_i\in J}a(x_i)h_i)/n=\left((n-j)+\frac{1}{2}jh\right)\Big/n.$$

由定义有 $j/n=\sigma$,得 $\alpha=\left(\frac{h}{2}-1\right)\sigma+1$. □

由于使用冲突链表,FlowTable 中都准确记录了流 ID 号,所以 Holds 算法的误检率为 0.同时,Holds 避免了 Multistage 算法需同时计算多个 Hash 带来的存储访问冲突的问题.由此带来的问题是,Hash 冲突增加了访存次数.Holds 算法在 Hash 冲突比例和冲突深度较低的情况下,算法优越性很明显.

6 Hits 和 Holds 与其他算法理论比较

Hits 和 Holds 与其他算法的比较结果见表 2.M 为保存 Hash 表项中能够保存流 ID 的数目,我们定义最大流为网络中那些流量为 zC 的流.表中第 1 行显示,Holds 算法的误检率为 0,Hits 算法与 Multistage 算法错误率大致相等.表中第 2 行是报文访存次数比较.由于 Hits 算法采用了一个高速缓存用来保存当前被统计的流,大部分情况下只需访存一次,这种结果要远好于 Multistage 算法,接近 Sample and Hold 算法;Holds 算法访存次数与 Hash 冲突率相关.采用 Sample 算法的访存次数同采样率的设置有关.表中第 3 行显示了算法实现时需要的内存比较,Hits 算法内存需求略高于 Multistage 算法(sample,sample and hold,Multistage 算法参数计算源自文献[9]).

Table 2 Hits and Holds theoretical comparison with other algorithms

表 2 Hits 和 Holds 与其他常用采样算法的比较

Algorithm	Hits	Holds	Sample and hold	Sample	Multistage
Relative error of a flow with zC in size	$\left(\prod_{i=1}^{h-1}\frac{T}{k(T-\beta\times i\times s)}\right)\times\left(\frac{T}{k(T-s)}\right)^{d}$	0	$\frac{\sqrt{2}}{Mz}$	$\frac{1}{\sqrt{Mz}}$	$\frac{1+10l\log_{10}(n)}{Mz}$
Memory accesses	1 or $1+\log_{10}(n)$	$\left(\frac{h}{2}-1\right)\sigma+1$	1	$\frac{1}{x}=\frac{M}{C}$	$1+\log_{10}(n)$
Memory bound	$QC/T+bd$	$bh/2$	M	$>>M$	db

7 Hits 和 Holds 算法评估

我们在 Windows 和 Linux 系统上分别实现了算法 Hits 和 Holds.在本节主要使用 AMP-0 和 PSC-0 等网络真实数据流量数据[16]对 Sample and Hold,Multistage,Hits,Holds 的有效性进行评估,并进行分析比较.表 3 列出了这些 traces 的基本情况.对 Hits 和 Holds 的评估从以下几方面进行:

1) Sample and Hold,Multistage,Hits,Holds 获取流量的效果;
2) 参数设置对 4 种算法的影响;
3) 4 种算法的误检率、漏检率的比较.

Table 3 Basic characteristic of testing traces

表 3 测试 Trace 的基本流量特性统计

Trace	Formats	# of packets	# TCP packets	Fraction of TCP packets (%)	# of flows	Last time (s)	Traces time
AMP-0	TSH	469 444	329 189	70.1	10 867	90	2006.4.30
PSC-0	TSH	2 304 261	1 785 447	77.5	58 468	89	2006.4.30

7.1 Hits,Holds算法功能有效性评估

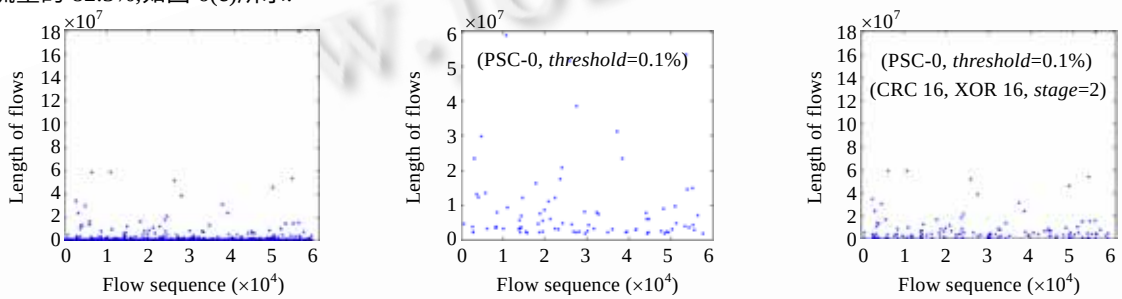
图 6 列出了使用测试数据 PSC-0,不同采样算法所获得的采样大象流的数据,大象流阈值使用实时流量的 0.1%,总流量为 1.4148e+009Byte,PSC-0 流的总数为 58 468,如图 6(a)所示.

对 Sample and hold 算法,我们采用的实现方式是:对流 ID 进行 Hash 运算,如果在 FlowTable 中没有表项(entry),则加入 FlowTable 中;如果在 FlowTable 中已有表项,则丢弃该报文.Sample and hold 采样到的大象流数量是 61,占总流数的 0.1%.采样流量是 731 738 589,占总流量的 51.9%,如图 6(b)所示.

对 Multistage 算法,我们使用两级过滤器,Hash 算法使用 CRC16 和 XOR16.Multistage 采样到的大象流数量是 159,占总流数的 0.27%.采样流量是 1.1004e+009,占总流量的 77.8%,如图 6(c)所示.

对 Hits 算法,我们的实现方式是:对流 ID 进行 Hash 运算,如果在 FlowTable 中没有表项,则加入 FlowTable 中;如果在 FlowTable 中已有表项,则进行第 2 次 Hash 运算,进入 Multistage 过程.Hits 算法使用一级过滤器,Hash 算法使用 CRC16 和 XOR16.采样的大象流数量是 129,占总流数的 0.22%.采样流量 1.1852e+009,占总流量的 83.7%,如图 6(d)所示.

Holds 算法使用 CRC16 作 Hash.采样的大象流数量是 111,占总流数的 0.19%,采样流量 1.1647e+009,占总流量的 82.3%,如图 6(e)所示.



(a) All flows (PSC-0)
(a) PSC-0 全部流量

(b) Elephant flows found by sample and hold
(b) Sample and hold 算法采样

(c) Elephant flows found by Multistage
(c) Multistage 算法采样

(d) Elephant flows found by Hits
(d) Hits 算法采样

(e) Elephant flows found by Holds
(e) Holds 算法采样

Fig.6 Effects using Hits and Holds sample data PSC-0 compared with other algorithms.

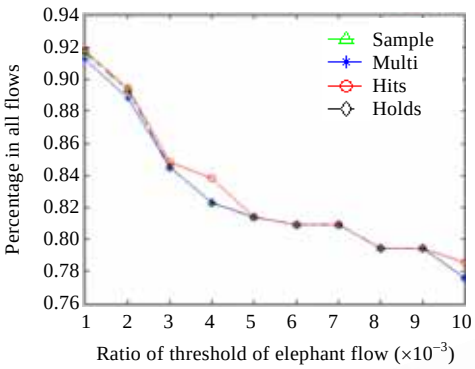
图 6 Hits 和 Holds 算法与其他算法采样 PSC-0 数据效果比较

以上结果表明,Hits 算法在减少了一级过滤器的情况下取得的结果优于 Multistage 算法,Holds 算法在减少了一级过滤器的情况下对大象流筛选的结果优于 Multistage 算法.使用 Sample and hold 算法采样 PSC-0 漏检率较高.

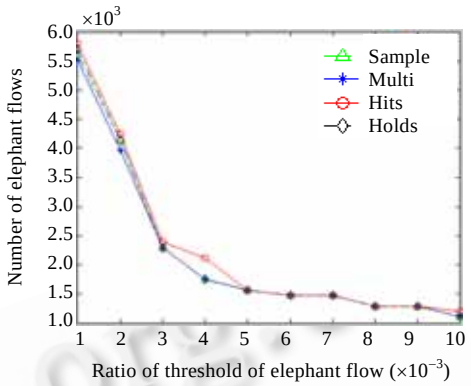
7.2 大象流阈值设置对Hits和Holds的及其他算法的影响

大象流阈值设置与大象流数量及占总流量比例之间存在很强的相关性.大象流阈值设置越大,采样到的大

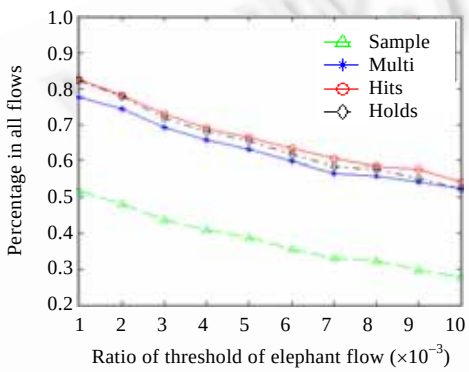
象流数量越小,采样的流量占总流量的比例也越小.图 7 列出了使用测试数据 AMP-0 和 PSC-0 测得的大象流阈值设置与大象流数量及占总流量比例之间的关系图.



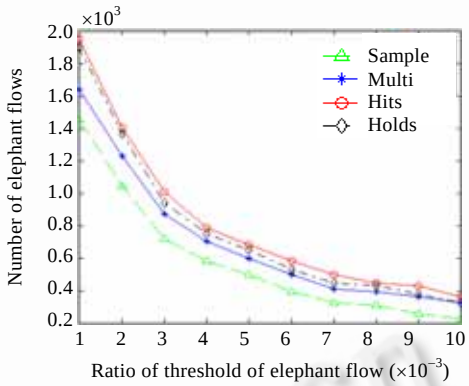
(a) Percentage of elephant in all flows with diff algorithms (AMP-0)
(a) AMP-0 大象流占全部流量的比例



(b) Numbers of elephant flows with diff algorithms (AMP-0)
(b) 获取 AMP-0 的大象流数量



(c) Percentage of elephant in all flows with diff algorithms (PSC-0)
(c) PSC-0 大象流占全部流量的比例



(d) Numbers of elephant flows with diff algorithms (PSC-0)
(d) 获取 PSC-0 的大象流数量

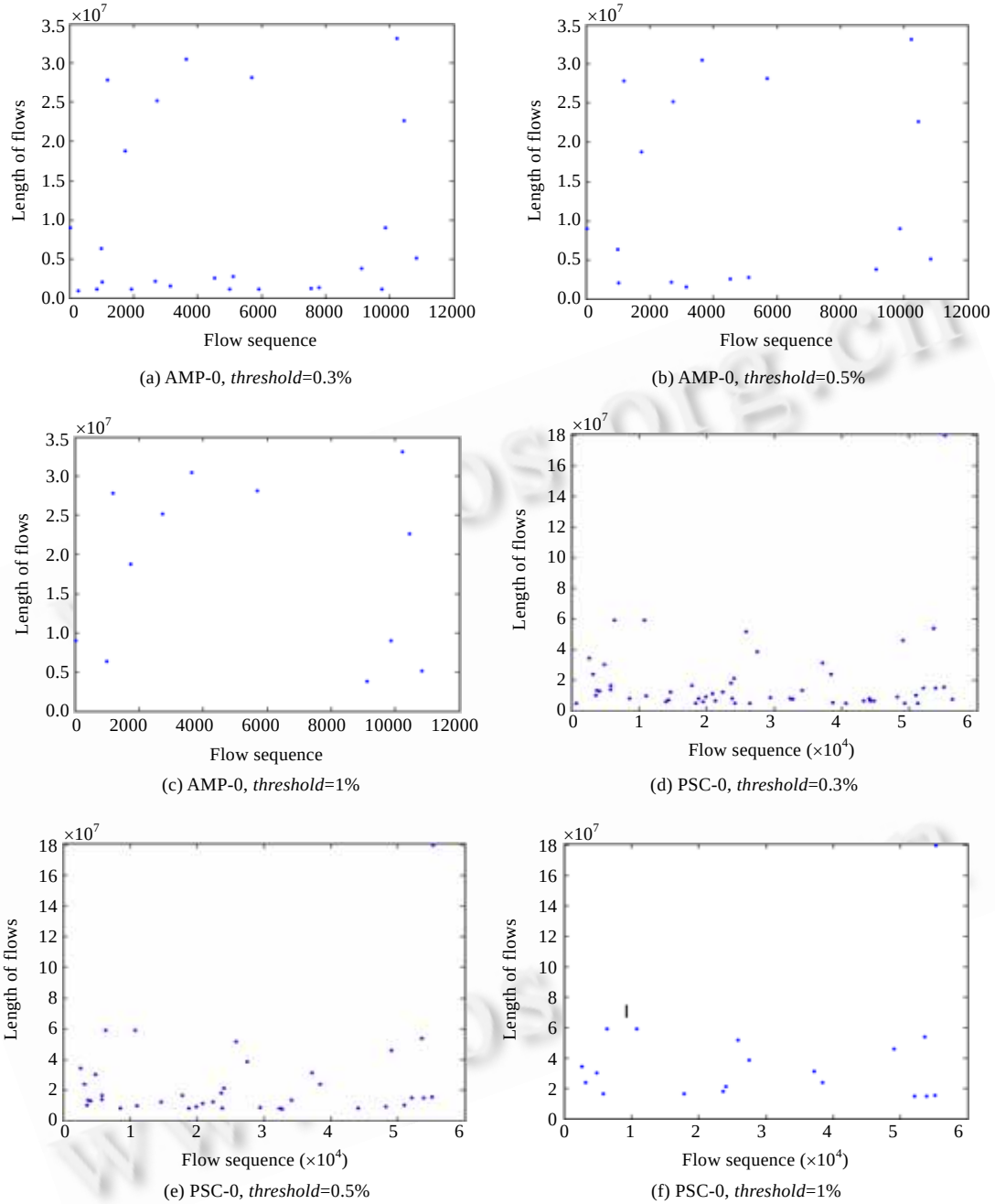
Fig.7 The influence of elephant flows thresholds setting to algorithms

图 7 大象流阈值设置对算法的影响

图 8 显示不同大象流阈值设置对流采样结果的影响.

图 8(a)~图 8(c)使用 AMP-0 数据.图 8(a):T=0.3%,采集到 25 条流,占总流数的 0.23%,采集流量 238 186 401,占总流量的 84.5%.图 8(b):T=0.5%,采集到 17 条流,占总流数的 0.16%,采集流量 229 455 208,占总流量的 81.5%.图 8(c):T=1%,采集到 12 条流,占总流数的 0.11%,采集流量 218 663 963,占总流量的 77.6%.

图 8(d)~图 8(f)使用 PSC-0 数据.图 8(d):T=0.3%,采集到 55 条流,占总流数的 0.094%,采集流量 1.0164e+009,占总流量的 71.8%.图 8(e):T=0.5%,采集到 38 条流,占总流数的 0.064%,采集流量 926 767 238,占总流量的 65.5%.图 8(f):T=1%,采集到 19 条流,占总流数的 0.032%,采集流量 741 234 225,占总流量的 52.4%.



(a) AMP-0, *threshold*=0.3%　(b) AMP-0, *threshold*=0.5%

(c) AMP-0, *threshold*=1%　(d) PSC-0, *threshold*=0.3%

(e) PSC-0, *threshold*=0.5%　(f) PSC-0, *threshold*=1%

Fig.8　Relations between elephant flows thresholds setting and sampled results (Holds)

图 8　大象流阈值设置与采集结果的关系(Holds 算法)

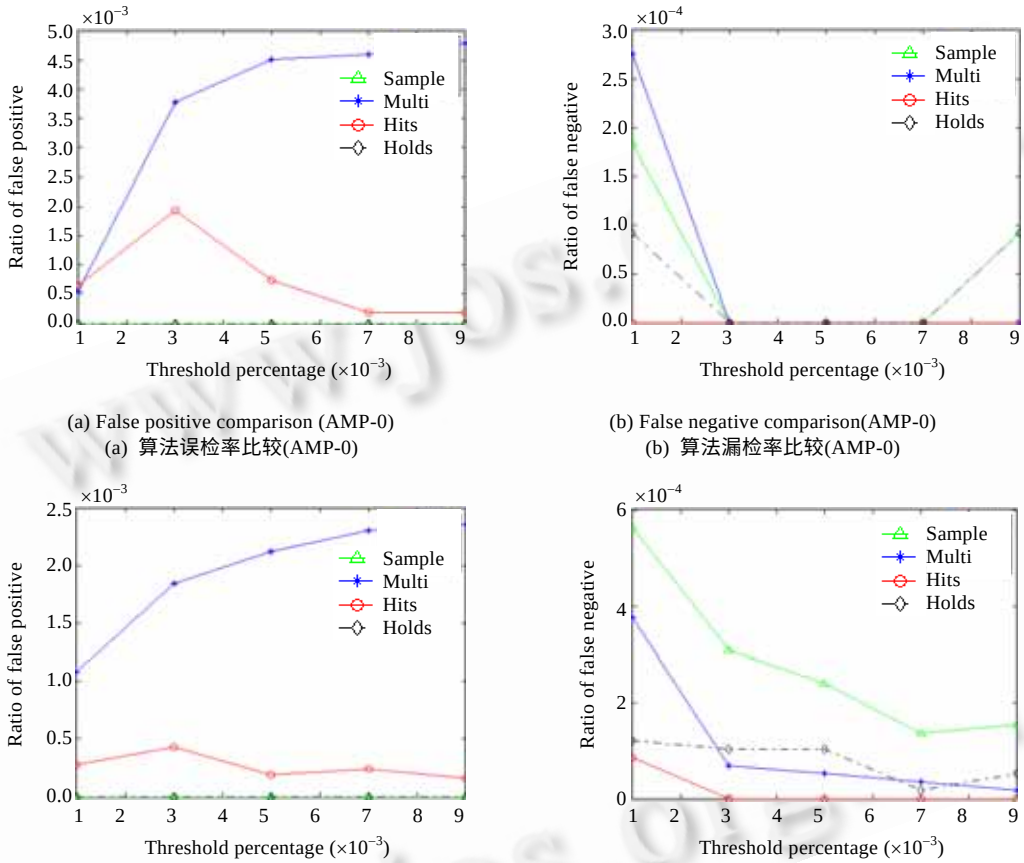
7.3　4种算法的误检率、漏检率的比较

使用 AMP-0,PSC-0 数据对算法的误检率、漏检率的比较.图 9 显示了算法的误检率、漏检率的比较结果.

图 9(a)、图 9(c)分别为使用 AMP-0 和 PSC-0 数据测得的算法误检率的比较.由图可知,Multistage 的误检率较高,原因是 Multistage 使用 Hash 函数,存在多条流映射到同一个入口,多条小流相加称为大流的情况.在实验中,我们使用两级过滤器(filterstage).Hits 算法的误检率比 Multistage 算法的误检率低,但高于 Sample and Hold

算法和 Holds 算法,原因也是使用 Hash 函数的问题.但 Hits 算法的使用了准确记录流量大小的流表(flowtable),所以误检率有所减少.Sample and Hold 算法和 Holds 算法能准确判断流的大小,所以算法误检率为 0.

图 9(b)、图 9(d)分别为使用 AMP-0 和 PSC-0 数据测得的算法误检率的比较.由图可知,Sample and Hold 算法的漏检率较高,原因是 Sample and Hold 算法随机丢弃报文,造成一些大象流报文被丢弃而造成漏检.其他 3 种算法的漏检率都相对较低,Hits 算法则表现更为出色.



(a) False positive comparison (AMP-0)
(a) 算法误检率比较(AMP-0)

(b) False negative comparison(AMP-0)
(b) 算法漏检率比较(AMP-0)

(c) False positive comparison (PSC-0)
(c) 算法误检率比较(PSC-0)

(d) False negative comparison (PSC-0)
(d) 算法漏检率比较(PSC-0)

Fig.9 Ratio of false positive and false negative of hits and holds compared with other algorithms

图 9 hits and holds 算法与其他算法的误检率、漏检率的比较

8 结 论

本文针对 Sample and Hold 算法随机丢弃报文,带来采集数据不准确的问题和 Multistage 算法需要同时进行多次(5~6)次访存,无法使用硬件实现的问题,提出了两种大象流识别算法:Hits 和 Holds 算法.本文首先介绍了 Hits 和 Holds 算法的基本思想,对两种算法进行了详细描述,并对算法的有效性进行了理论分析,最后使用网络实际流量数据对算法进行了评估,并与 Sample and Hold 及 Multistage 进行了比较.理论和实验结果表明,Hits 和 Holds 算法对网络大象流的误检率和漏检率均优于 Sample and Hold 及 Multistage 算法.

References:

[1] Liu YP, Gong ZH, Zhu PD. Research on the bottleneck area of optimal BGP route selection. Journal of Software, 2005,16(5): 946−959 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/16/946.htm

[2] Wang H, Gong ZH, Guan Q, Wang BS. Detection network anomalies based on packet and flow analysis. In: Bi J, Gyires T, eds. Proc. of the 7th Int'l Conf. on Networking. Cancun: IEEE Computer Society, 2008. 497−502.

[3] Agrawal S, Naidu KVM, Rastogi R. Diagnosing link-level anomalies using passive probes. In: Proc. of the IEEE INFOCOM. 2007. http://www.stanford.edu/~shipra/Publications/anomaly.pdf

[4] Estan C, Varghese G, Fisk M. Bitmap algorithms for counting active flows on high speed links. In: Proc. of the 3rd ACM SIGCOMM Conf. on Internet. 2003. http://woozle.org/~mfisk/papers/countingdistinct.pdf

[5] Yang LL, Michailidis G. Sampled based estimation of network traffic flow characteristics. In: Proc. of the IEEE INFOCOM 2007. http://www.stat.lsa.umich.edu/~gmichail/infocom_07.pdf

[6] Krishnamurthy B, Sen S. Sketch-Based change detection: Methods, evaluation, and applications. In: Proc. of the ACM SIGCOMM. 2003. http://www.imconf.net/imc-2003/papers/p305-krishnamurthy11111.pdf

[7] Duffield N, Lund C, Thorup M. Estimating flow distributions from sampled flow statistics. In: Proc. of the ACM SIGCOMM. 2003. http://www.research.att.com/~duffield/papers/DLT03-lengths.pdf

[8] Fang W, Peterson L. Inter-As traffic patterns and their implications. In: Proc. of the IEEE GLOBECOM. 1999. http://www.nlanr.net/Papers/inter-AS.pdf

[9] Estan C, Varghese G. New directions in traffic measurement and accounting. In: Proc. of the ACM SIGCOMM. 2002. http://conferences.sigcomm.org/sigcomm/2002/papers/traffmeas.pdf

[10] Huang L, Nguyen XL, Garofalakis M, Hellerstein J. Communication-Efficient online detection of network-wide anomalies. In: Proc. of the IEEE INFOCOM. 2007. http://bnrg.cs.berkeley.edu/~adj/publications/paper-files/pca_infocom2007.pdf

[11] Jin CQ, Qian WN, Zhou AY. Analysis and management of streaming data: A survey. Journal of Software, 2004,15(8):1172−1181 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/15/1172.htm

[12] Baiocchi A, Vacirca F. TCP fluid modeling with a variable capacity bottleneck link. In: Proc. of the IEEE INFOCOM. 2007. http://infocom.uniroma1.it/~vacirca/papers/vacirca/infocom07.pdf

[13] Chen AY, Cao J, Bu T. Network tomography: Identifiability and fourier domain estimation. In: Proc. of the IEEE INFOCOM. 2007. http://arxiv.org/pdf/0712.3618

[14] Charikar M, Chen K, Farach-Colton M. Finding frequent items in data streams. In: Proc. of the Symp. on Principles of Database System. 2002. http://www.cs.rutgers.edu/~farach/pubs/FrequentStream.pdf

[15] Cormode G, Muthukrishnan S. What's new: Finding significant differences in network data streams. In: Proc. of the IEEE INFOCOM. 2004. http://www.cs.rutgers.edu/~muthu/676879419.pdf

[16] National Laboratory for Applied Network Research. http://www.nlanr.net

附中文参考文献:

[1] 刘亚萍,龚正虎,朱培栋.BGP 最优路径选择中的瓶颈区域的研究.软件学报,2004,15(8):946−959. http://www.jos.org.cn/1000-9825/15/946.htm

[11] 金澈清,钱卫宁,周傲英.流数据分析与管理综述.软件学报, 2004,15(8):1172−1181. http://www.jos.org.cn/1000-9825/15/1172.htm

王宏(1964－),男,湖南长沙人,副研究员,主要研究领域为计算机网络协议软件,高性能计算机网络,网络管理,网络流量测量与分析.

龚正虎(1942－),男,教授,博士生导师,主要研究领域为计算机网络体系结构,高性能计算机网络,网络协议软件,网络管理.