AHashFlow 实验方案

赵宗义

2019 年 11 月 13 日

目录

1 基本定义

1.1 HashFlow

原始的 HashFlow 方案，在数据平面记录完整的 Flow ID，没有控制平面，并且没有不活跃大流的退出机制。

1.2 DHashFlow (HashFlow with Digest)

在数据平面记录 Flow ID 的 digest 而不是完整的 Flow ID，同时增加控制平面来记录完整的 Flow ID，但是并没有引入不活跃大流的退出机制。

1.3 AHashFlow (Augmented HashFlow)

改良后的 HashFlow，在数据平面记录 Flow ID 的 digest 而不是完整的 Flow ID，因此包含控制平面来记录完整的 Flow ID，此外还增加了不活跃大流的退出机制。

2 测量指标

假设我们重放的数据包序列中包含 m 个数据包和 n 个数据流，则相关的测量指标可以定义如下：

2.1 数据包的不测量率 (NMR, No Measurement Rate)

假设 m_1 是没有被我们的测量算法测量到数据包的数目，具体而言就是在辅助表中发生冲突的时候被丢弃的辅助表表项的计数值的总和，则数据包的不测量率可定义如下：

$$NMR = \frac{m_1}{m}$$

2.2 流覆盖率（FSC）

假设网络测量算法记录了完整的流标识符的数目为 n_1，则流覆盖率可定义如下：

$$FSC = \frac{n_1}{n}$$

2.3 流数目估计相对误差（RE）

假设网络测量算法估计的网络中数据流的数目为 n_2，则流数目估计相对误差可定义如下：

$$RE = \frac{|n - n_2|}{n}$$

由于本文所考察的网络测量算法可以通过数据结构的使用情况估计没有被明确记录的数据流的数目，因此算法所估计的数据流数目和算法所明确记录的数据流的数目并不完全相等。

2.4 数据流大小估计的平均相对误差（ARE）

对于原数据包序列中的任意一个数据流 f_i，假设它的真实大小为 m_i，而网络测量算法所返回的该数据流的大小为 m_i，如果该数据流没有被明确定义则默认为 $m_i = 0$，则数据流大小估计的平均相对误差可定义如下：

$$ARE = \frac{1}{n} \sum \frac{|m_i - m_i'|}{m_i}$$

2.5 被测数据流大小估计的平均相对误差 (ARE, Average Relative Error)

假设我们的测量算法记录的流分别为 $f_1, f_2, ..., f_k$，且对于任意一个数据流 f_i $(1 \leq i \leq k)$，它的实际大小为 m_i，而我们的算法测得其大小为 m_i'，则被测数据流的平均相对误差可定义为：

$$ARE = \frac{1}{k} \sum \frac{|m_i - m_i'|}{m_i}$$

2.6 heavy hitter 检测的 F1 Score

已经定义 heavy hitter 的阈值 ‘γ‘，假设原数据包序列中 heavy hitter 的数目为 c_1，网络测量算法所返回的 heavy hitter 的数目为 c_2，网络测量算

法所返回的 heavy hitter 中真实的 heavy hitter 的数目为 c，则 heavy hitter 检测的召回率为 $RR = \frac{c}{c_1}$，准确率为 $PR = \frac{c}{c_2}$，因此 heavy hitter 检测的 F1 Score 可定义如下：

$$F1\ Score = \frac{2 \cdot PR \cdot RR}{PR + RR}$$

2.7 heavy hitter 检测的平均相对误差（ARE）

如上，假设原数据包序列中共有 n 个 heavy hitter，其中 heavy hitter f_i 的大小为 m_i，而网络测量算法报告的 f_i 的大小为 m_i，如果 f_i 不包含在网络测量算法报告的 heavy hitter 中（包括 f_i 已经被网络测量算法记录但是没有被识别成为 heavy hitter 的情况），则默认 $m_i = 0$，则 heavy hitter 检测的平均相对误差可定义如下：

$$ARE = \frac{1}{n'} \sum \frac{|m_i - m_i'|}{m_i}$$

2.8 控制平面的包负荷比 (Packet Load Rate, PLR)

在 P4 交换机版的实现方案中，我们需要将特定的数据包从数据平面发送到控制平面以维护流标识符和指纹之间的映射关系，因此控制平面的负荷比可定义如下：

$$PLR = \frac{m_1}{m}$$

其中 m_1 是控制平面处理的数据包的数目。

2.9 控制平面的流负荷比 (Flow Load Rate, FLR)

假设在 AHashFlow（或 DHashFlow）的运行过程中，控制平面一共收到了 m_1 个数据包，而交换机处理的数据包中一共包含了 n 个数据流，则控制平面的流负荷比可以定义如下：

$$FLR = \frac{m_1}{n}$$

3 实验设计

3.1 Exp84491 $\surd$

- Use the simulator of HashFlow implemented in python.

- Set the memory size to be 1 MB, so it can accommodate around 55K flow records.
- Select 10 trace files from each of the four traces.
- Initiate 50K flows from each trace file.
- Increase the depth of HashFlow from 1 to 4.
- Count the packets processed by the simulator, i.e., the number of original packets plus the resubmitted packets.

3.2 Exp84492 $\surd$

- Select a file from the CAIDA trace (equinix-nyc.dirA.20180315-125910.UTC.anon.pcap), extract the first 2.5 million packets, classify the TCP/UDP packets into flows, and then calculate the average size as well as the maximum size of the flows.
- Select a file from the HGC trace (20080415000.pcap), extract the first 2.5 million packets, classify the TCP/UDP packets into flows, and then calculate the average size as well as the maximum size of the flows.
- Calculate the average as well as maximum size of a file from China Telecom trace (nfcapd.201601022000).
- Calculate the average as well as maximum size of a file from Tsinghua campus trace (20140206-6).

3.3 Exp84493 $\surd$

- Set the memory size to be 1MB.
- Increase the number of flows from 10K to 100K, in the step size of 10K.
- Use a trace file from CAIDA and HGC respectively.
- Use four versions of HashFlow. In the versions the number of buckets in the ancillary table is 0.25×, 0.5×, 1.0×, and 2× respectively of the number of buckets in main table.

- Calculate the average relative error for flow size estimation.

3.4 Exp84494 √

- Randomly select a file from the traces of ChinaTelecom, HGC, Tsinghua and CAIDA respectively.
- Extract 5 million packets from each trace file.
- Record the number of distinct flows in each trace file. Note that all the packets with the same source IP address, destination IP address, source port, destination port, and protocol belong to the same flow. The flow ID is the five tuple (srcip, dstip, srcport, dstport, protocol).
- Map the flow ID of each flow to a digest using a given hash function.
- Record the number of distinct digest for each trace file.
- Compare the number of distinct flows and the number of distinct digests for each trace file.

3.5 Exp84495

本实验的实验参数设置如下：

- 实验方案为 HashFlow, DHashFlow 和 AHashFlow
- 将内存容量设为 1MB
- 使用一个 CAIDA 的 trace 文件
- 进行 10 组实验，将重放的数据包的数目以 50 万的步长从 50 万增加到 500 万
- 将 heavy hitter 的阈值设为一个固定值 10，即包含 10 个及 10 个以上数据包的流为一个 heavy hitter
- 测量的指标包括 heavy hitters 测量的 F1 Score 和平均相对误差，数据包的不测量率以及控制平面的包负荷比

在此实验中可以预见的效果是随着数据包数目的增加，HashFlow 和 DHashFlow 的性能逐渐降低，其中 HashFlow 的性能降低尤为明显，而 AHashFlow 的性能却保持在一个比较稳定的状态，说明不活跃大流的退出机制能够发挥良好的作用。此外，AHashFlow 的控制平面包负荷比会明显高于 DHashFlow，但是这是情理之中的，因为这从侧面证明 AHashFlow 能够比 DHashFlow 测量更多的流。

3.6 Exp84496

本实验的实验参数设置如下：

- 实验方案为 AHashFlow 和 DHashFlow
- 将内存容量设为 1MB
- 从 CAIDA 和 HGC 的的 trace 中分别选取一个文件
- 进行 10 组实验，将重放的数据包的数目以 50 万的步长从 50 万增加到 500 万
- 测量的指标包括流覆盖率，数据流大小估计的平均相对误差，被测数据流大小估计的平均相对误差，以及控制平面的流负荷比

在此实验中，我们预期的实验结果是 AHashFlow 的这四项指标都保持相对稳定的状态，其中被测数据流大小估计的相对误差要明显小于数据流大小估计的平均相对误差，而控制平面的流负荷比接近 2.0 甚至小于 2.0；AHashFlow 的各项性能指标应该都要优于 DHashFlow，尤其是 DHashFlow 的控制平面流负荷比应该会高于 AHashFlow 的控制平面流负荷比，说明在 DHashFlow 中不活跃大浪占据了宝贵的内存空间，导致其它的流更频繁地被替换。

3.7 Exp84497

本实验的实验参数设置如下：

- 实验方案为 AHashFlow 和 DHashFlow
- 将内存容量设为 1MB
- 从 CAIDA 和 HGC 的 trace 中分别选取一个文件

- 仅进行 1 组实验，将重放的数据包的数目设为 500 万
- 将 heavy hitter 的阈值以 5 的步长从 5 增加到 50
- 测量的指标包括 heavy hitter 检测的平均相对误差以及 F1 Score

在此实验中我们的预期实验结果是随着阈值的增加，heavy hitter 检测的 F1 Score 和平均相对误差都有明显的改善，说明 AHashFlow 确实有得大流的测量，同时 AHashFlow 的性能总是优于 DHashFlow。

附录

A AHashFlow 方案介绍

AHashFlow 的数据结构如图1所示，AHashFlow 的数据结构包括三个表，分别是 M 表、A 表和 B 表，其中 M 表中的第一个表项包括一个 4 字节的 digest 域和一个 4 字节的 count 域，A 表中每个表项包括一个 1 字节的 digest 域和一个 1 字节的 count 域，B 表中每个表项包含一个 4 个节的 count 域。

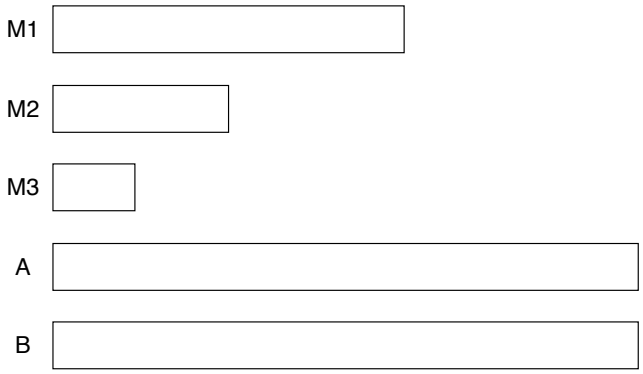

图 1: AHashFlow 的数据结构：包括一个 M 表，一个 A 表以及一个 B 表，其中 M 表可分成若干小表

建议的内存分配方案是使得 M 表、A 表和 B 表有相同数目的表项；M 表可以分成若干个小表，其中后一个小表的表项数目是前一个小表的表项数目的一半。

Algorithm 1 $promote(idx, digest, cnt)$

1: $export(\mathbf{A}[idx].digest, \mathbf{A}[idx].cnt)$
2: $\mathbf{A}[idx] \leftarrow [digest, cnt]$

Algorithm 2 $reset(idx)$

1: $export(\mathbf{A}[idx].digest, \mathbf{A}[idx].cnt)$
2: $\mathbf{A}[idx].cnt \leftarrow 0$

AHashFlow 对数据包的处理流程如算法3所示，当一个数据包到达时，提取它的流标识符（默认为它的五元组（源 IP 地址，目的 IP 地址，源端口号，目的端口号，协议类型)），并将流标识符映射成为一个 32 位的 $digest_1$ 和一个 8 位的 $digest_2$。

之后，我们根据这个数据包的流标识符依次将这个数据包映射到 M 表的 d 个位置。在这 d 个位置上，如果我们找到了一个空表项，那么我们就将 $digest_1$ 写入该表项中并将其计数值置为 1，同时将数据包的流标识符导出到控制层，之后结束对数据包的处理。如果我们在 M 表中找到了一个 digest 域为 $digest_1$ 的表项，那么我们只需要简单地将这个表项的计数值增加 1 便可以完成对这个数据包的处理。

反之，如果我们在这 d 个位置都不能找到一个空表项或者一个 digest 域的值为 $digest_1$ 的表项，那么我们就记录这 d 个表项中的最大计数值 max 和最小计数值 min，以及与之相对应的索引 idx_{max} 和 idx_{min}。随后，我们进一步查询 A 表和 B 表。我们首先使用另一个哈希函数 g 将数据包的流标识符映射到一个索引值 idx。由于 A 表和 B 表具有相同数目的表项，所以我们建议在 A 表和 B 表中均使用索引值 idx。首先如果 A 表中对应的表项为空，则我们将该表项的内容设为 $[digest_2, 1]$，于是便可以结束对数据包的处理。但是如果该表项不为空且它的 digest 和 $digest_2$ 相同，那么我们就在 A 表中找到了一个匹配的表项，因此我们首先把这个表项的计数值加 1 并记录在一个临时变量 $temp$ 中。如果 $temp > min$，则说明 A 表中缓存的数据流比 min 对应的数据流更大，因此按照优先记录大流的原则，我们使用 $promote$ 函数将这个流提回 M 表中。因为我们在 M 表中并不记录数据流的完整流标识符，因此我们需要将这个数据包的流标识符导出到控制层。最后我们再将 A 表中的这个表项置为空。如果 $temp \leq min$，则我们需要将 $temp$ 写回到该表项的计数值中，实际上是将这个计数值加 1。

如果该表项的计数值不为空且它的 digest 和 $digest_2$ 不相同，说明新到达的数据包和 A 表中的相应表项发生冲突，那么我们首先将 A 表中相应表项的计数值和 B 表中 idx 对应的表项的计数值相加并存入 $temp$ 中，然后再将 A 表中 idx 对应的表项置为 $[digest_2, 1]$。随后我们比较 $temp$ 和 max 的大小。因为 M 表和 B 表具有相同的表项数目，而且 M 表优先记录大流，而被 B 表记录的数据包一般来自小流，已知在正常的网络中绝大多数的数据包都来自大流，因此 M 表中表项的计数值应该远大于 B 表中表项的记数值。于是，如果 $temp > max$，那么我们就断定测量算法的性能已经下降到一个需要引起注意的程度。而测量算法的性能下降的一个最重要的原因就是主表中表项已经被不活跃的大流占据，从而使得很多其它的流不能有效地在 M 表中积累。这里我们怀疑 max 对应的流是一个不活跃的大流，因此我们引入不活跃大流退出机制，将 M 表中 idx_{max} 对应的表项进行重置，同时将 idx 对应的 B 表的表项置为 0。反之，如果 $temp <= max$，则我们只需要简单将 $temp$ 的值写入 idx 对应的 B 表表项中，相当于将该表项加 1。

以下我们描述 $promote$ 和 $reset$ 两个算法即算法1和2的算法操作及其逻辑。首先在 $promote$ 中，我们的参数包括一个索引值 idx，新数据流的流标识符的 digest 以及对应的计数值。我们在主表中找到 idx 对应的表项并将其导出到控制层，然后将新数据流的 digest 和计数值写入到 idx 对应的主表表项中。而在 $reset$ 中，我们的参数是主表中待重置表项的索引值 idx，因此我们找到 idx 对应的主表表项并将其导出到控制层。此后，我们将该表项的计数值置为 0，但是并不会对它的 digest 进行更改（即并不会将该表项的 digest 域也置为 0），这样的方案设计主要是出于两个方面的原因的考虑。首先如果我们将 idx 对应的表项都置为空，那么 M 表中就可能出现一个数据流存放在多个表项中的情况，比如我们在图1中将 $M1$ 中的一个表项 $M1[x]$ 置为空，另一个流 f 会被映射到 $M1$ 和 $M2$ 中的两个表项 $M1[x]$ 和 $M2[y]$ 且我们已经为 f 在 $M2[y]$ 维护了一个流表项，那么 f 的下一个数据包到达的时候我们检测到 $M1[x]$ 为空，因此还会在 $M1[x]$ 为它建立一个流表项，这样我们在 M 表中就为 f 维护了重复的流表项，造成了内存资源的浪费。如果我们只是将 $M1[x]$ 的计数值置为 0 而保留它的 digest 就可以完美避开这个问题。此外，因为主表中被重置的流是一个大流，并且我们不能完全确定它是一个不活跃的大流，相反，它有可能是一个非常活跃的大流，因此我们仍然为它在主表中保留一个表项，如果它真的是一个活跃大

流的话它就可以迅速在这个表项中聚集起来，避免了在 A 表中积累最后被 *promote* 的过程，可以有效减少测量误差。

Algorithm 3 $processPacket(p)$

1: $min \leftarrow \infty, max \leftarrow 0, idx_{min} \leftarrow -1, idx_{max} \leftarrow -1$
2: $digest_1 \leftarrow H_1(p.flowID), digest_2 \leftarrow H_2(p.flowID)$
3: **for** $i = 1$ to d **do**
4: $idx \leftarrow h_i(p.flowID)$
5: **if** $\mathbf{M}[idx] == [0, 0]$ **then**
6: $\mathbf{M}[idx] \leftarrow (digest_1, 1)$, $export(p.flowID)$ **return**
7: **else if** $\mathbf{M}[idx].key == digest_1$ **then**
8: Increment $\mathbf{M}[idx].cnt$ by 1 **return**
9: **else**
10: **if** $\mathbf{M}[idx].count < min$ **then**
11: $min \leftarrow \mathbf{M}[idx].cnt, idx_{min} \leftarrow idx$
12: **end if**
13: **if** $\mathbf{M}[idx].count > max$ **then**
14: $max \leftarrow \mathbf{M}[idx].cnt, idx_{max} \leftarrow idx$
15: **end if**
16: **end if**
17: **end for**
18: $idx \leftarrow g(p.flowID)$
19: **if** $\mathbf{A}[idx] == [0, 0]$ **then**
20: $\mathbf{A}[idx] \leftarrow [digest_2, 1]$
21: **else if** $\mathbf{A}[idx].key == digest_2$ **then**
22: $temp \leftarrow \mathbf{A}[idx].cnt + 1$
23: **if** $temp > min$ **then**
24: $export(p.flowID)$, $\mathbf{A}[idx] \leftarrow [0, 0]$
25: $promote(idx_{min}, digest1, temp)$
26: **else**
27: $\mathbf{A}[idx].count = temp$
28: **end if**
29: **else**
30: $temp \leftarrow \mathbf{B}[idx].cnt + \mathbf{A}[idx].cnt$, $\mathbf{A}[idx] \leftarrow [digest_2, 1]$
31: **if** $temp > max$ **then**
32: $reset(idx_{max})$, $\mathbf{B}[idx] \leftarrow 0$
33: **else**
34: $\mathbf{B}[idx] \leftarrow temp$
35: **end if**
36: **end if**