基于单向并行多链表的流管理

叶荻秋 程东年 李玉峰
(国家数字交换系统工程技术研究中心 郑州 450002)

摘 要 高速条件下数据访存是流管理的瓶颈,传统表项操作"读-处理-写"模式效率仅为36%,读写转换和读写延时是制约效率的重要因素。针对这个问题,提出了连续读写法处理表项,通过合并读写时延和复用读写转换的方式使表项操作效率超过90%;并在此基础上提出了单向并行多链表法。通过多个链表的交替操作,使连续读写法应用于处理不活动超时流。理论分析和实验表明,单向并行多链表法能够适用于OC-768(40Gbps)链路下管理千万条表项明显优于辅助存储法和双向链表法的OC-192(10Gbps)下百万条表项的管理能力。

关键词 流管理,单向并行多链表,不活动超时流
中图法分类号 TN919.2 **文献标识码** A

NetFlow Based on Parallel Single Multi-linked Lists

YE Di-qiu CHENG Dong-nian LI Yu-feng
(China National Digital System Engineering & Technological R&D Center, Zhengzhou 450002, China)

Abstract In the high speed network, the bottleneck of NetFlow is the memory access. The efficiency of traditional list item operation "read-process-write" mode is only 36%, and the important factors which restrict the efficiency are read and write conversion as well as read and write delay. To solve this problem, continuous reading and writing method was proposed. It makes the efficiency of traditional list item operation increase to 90% or more by merging multi-delay and multiplexing read and write conversion. We proposed the parallel single multi-linked list method to handle with the inactive timeout flows. Continuous reading and writing method is applied to deal with the inactivity timeout flow through multiple list alternating operation. Theoretical analysis and experimental results show that the parallel single multi-linked list method can be applied in the link OC-768(40Gbps) to manage tens of millions of list items, which is significantly superior to the management ability of auxiliary storage method and doubly linked list method in the link OC-192 (10Gbps) to manage millions of list items.

Keywords NetFlow, Parallel single multi-linked lists, Inactivity timeout flow

1 引言

随着互联网技术的飞速发展,网络的规模不断壮大,互联网上的应用越来越多,尤其即时通信和数据共享等业务的发展,导致人们对互联网的依赖也越来越强。与此同时,网络规模的迅速膨胀、新应用层出不穷,导致网络的可管理性和可控性越来越差,因此急需高效可信的网络管控手段[1]。

网络的高速发展和带宽的增大,要求网络设备能够在OC-768(40Gbps)乃至更高速率下同时监控千万量级的流,流管理是完成这一要求的支撑技术。流管理是指在流建立到结束的整个生命周期中,对该流的包进行计数、转发和丢弃的操作[2]。流管理是用户可以设置的一个全局动作,使得满足流建立条件的流量进入流管理。流管理不仅可以监控流,而且是拥塞控制、入侵检测、QoS保证和异常流检测等功能的基础[3]。

在OC-768链路中,包的最高到达速率为 $7.8 * 10^7$ 包/s,对于400兆赫兹的存储器,每个包的平均访存时间仅为5.1个周期,在此期间须查找千万条表项,并删除不活动超时流,因此在短时间内完成数据访存成为流管理的瓶颈。若流表里存有大量的短时流以及不规则流,将导致表项空间爆炸,因此需要及时清理超时流。超时流包括活动超时流和不活动超时流。活动超时流是指从流的开始包算起的持续时间超过某个门限;不活动流指流长时间没有新包到达,距离最后一个包的到达时间超过某个门限。超时门限值需要根据实际网络设定,值过小,容易将一些活动流错误地删除;值过大,系统很可能淹没于SYN包[4]。处理超时流主要有两种方式:包触发和事件触发。处理活动超时流采用包触发方式,将新包到达的时间与起始包的到达时间差值和设定的阈值比较,超过阈值即为超时流,删除该流;否则更新该流所对应的表项。而不活动流无新包到达,无法判断其是否为流超时,采用事件触发将其删除,即定时地查找不活动超时流并将其删除[5]。

表项在存储器中采用哈希表的结构组织,而哈希表结构

到稿日期:2012-11-14 返修日期:2013-02-23 本文受国家高技术研究发展计划(863计划)(2011AA01A103)资助。
叶荻秋(1987—),男,硕士生,主要研究方向为网络安全,E-mail:yediqiu@126.com;程东年(1957—),男,教授,主要研究方向为宽带信息网络体系结构、网络安全协议和网络性能分析技术;李玉峰(1976—),男,讲师,主要研究方向为宽带路由器设计。

本身无法快速高效处理大量数据和不活动超时流，通常通过增加额外的数据结构或存储单元来提高性能，基本的方法有快速辅助存储法和双向链表法。快速辅助存储法的基本思想是在原来的大容量存储器外增加速度较快的 SRAM，将少量的常用信息存放在 SRAM 中，使大量的操作在 SRAM 中进行，避免频繁读写速度较慢的大容量存储器。文献[6,7]将统计的包数和字节数存放在 SRAM 中，只有在 SYN 包和 FIN/RST 包到达时才去读写大容量存储器，将占绝大多数的中间包在 SRAM 中操作。文献[8]在 SRAM 中存储流活跃数组，减少了不活动超时流的处理时间。随着网络链路带宽的急剧增加，SRAM 的容量已经很难满足存储表项数的需求。双向链表法的基本思想是用双向链表将表项按最后一个包的到达时间进行排序，不活动流表项在链表的前端，达到快速处理不活动超时流的目的。双向链表法适合软件处理，在硬件上难以实现，且处理过程复杂，因此速度上受到极大的限制。

快速辅助存储法和双向链表法经过多次改进，仅仅能够达到 OC-192(10Gbps)链路下百万条表项的处理能力，存储器访存效率低下，远不能满足 OC-768 链路下千万条表项的处理要求，因此 OC-768 链路下流管理需要全新的思路来解决。本文主要研究流管理的模型，提出连续读写法解决包触发下表项建立、更新和删除操作，单项并行多链表法维护不活动超时流，提高了存储器访存效率，进一步从理论上分析了该方法的高效性，并且通过实验证明了其可靠性。

快速辅助存储法和双向链表法都是研究表项之间的关系，本文将研究表项的处理过程来提高处理效率。前人只是简单地根据芯片手册逻辑时序访存数据，对数据访存方面的研究几乎没有。本文在芯片手册提供的时序的基础上，设计出巧妙的数据访存方法，并提出连续读写法和单向并行多链表法。

本文的主要工作包括：

1. 研究现有的流表操作，发现所有的操作可以归纳为“读-处理-写”的模型，并利用此模型分析已有方法，发现访存效率低下是制约流管理性能的主要原因；

2. 提出连续读写法代替现有的“读-处理-写”的模型，使基本表项操作的效率超过 90%；

3. 在连续读写法的基础上提出单向并行多链表法用以处理不活动超时流，使平均每表项的处理时间降低为 2 个多周期，使效率超过 90%。

2 基于单向并行多链表的流管理

本章首先介绍流管理流程，提出流管理处理过程的一般模型，根据此模型对已有方法进行分析，针对包触发下表项建立、更新和删除操作，提出一种新的连续读写法；在此基础上，针对不活动超时流的处理，提出了一种高效的单项并行多链表法。

2.1 模型建立

流管理一般包括：头域提取模块、哈希模块、表项建立模块、表项更新模块、表项删除模块和不活动超时流处理模块[9]。头域提取模块负责提取链路过来的包信息(包括五元组、服务类型、TCP 控制字段、字节数)，并且打上时间戳；哈希模块负责将五元组哈希映射生成流标识，哈希可能将多个不同的五元组映射到同一个地址，这不可避免地存在碰撞和冲突，用多级哈希的方法来减少和尽最大努力避免冲突[10]；表项建立模块负责处理 SYN 包，读出对应位置的表项，判断其是否为空，如果为空则将流信息写入该流表项；表项更新模块负责处理中间数据包，读出对应表项，判断五元组是否匹配，如果匹配则进行更新处理，并将处理完的数据写回原地址；删除表项模块负责处理 FIN/RST 包，读出对应表项，判断五元组是否匹配，匹配则将原表项的使能写为无效；不活动超时流处理模块负责定时读出流表项，判定该表项是否超时，超时则将原表项的使能写为无效。头域提取模块和哈希模块为前端预处理模块。表项建立模块、表项更新模块、表项删除模块的处理过程时序如图 1 所示。每条流处理过程需要调用表项建立模块和表项删除模块各一次，多次调用表项更新模块。

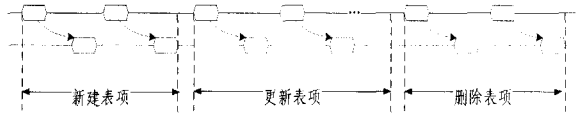

图 1 流管理主要模块时序示意图

在流管理的各个模块中，头域提取模块和哈希模块较为简单，能够适应高速处理的要求，而表项建立模块、表项更新模块、表项删除模块和不活动超时流处理模块制约了流管理的速率。根据上文分析，表项建立模块、表项更新模块、表项删除模块和不活动超时流处理模块本质上完全一致，都是“读-处理-写”的模型，本文的单向并行多链表法就是基于这种模型提出来的。“读-处理-写”的模型如图 2 所示[11]，首先下达读命令，经 t_{delay} 后数据到达数据线，然后根据不同的要求处理，处理完成后，再将数据重新写入相应的位置。整个过程的目的是读写数据，因此有效时间为 $2t_{data}$，而 $2t_{delay}+t_{cmp}$ 为损耗时间。完成一次完整的操作时间为：

$$T=2t_{delay}+2t_{data}+t_{cmp} \tag{1}$$

存储器访问效率如式(2)所示，典型 η 的值为 36%。存储器访问效率为：

$$\eta=\frac{2t_{data}}{2t_{delay}+2t_{data}+t_{cmp}} \tag{2}$$

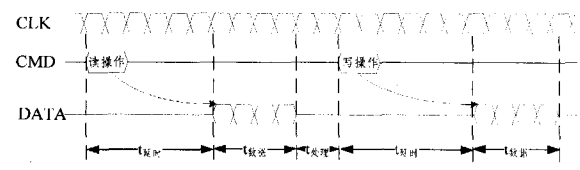

图 2 基本表项操作

辅助存储法通过使用 SRAM 降低表项更新模块的处理时间，并且表项更新模块的调用超过 90%，这样平均处理时间得到降低，但存储器访存效率没有任何提高。双向链表法时间消耗主要有以下 3 个方面：首先，双向链表法是按最后一个包的到达时间排序，因此更新涉及多个表项的维护，导致成倍地增加时间消耗；其次，器件存在固有的读写延时 t_{delay}，即数据和命令之间的时延；最后，表项处理需要时间 t_{cmp}，数据

读出来经处理后才能写入，流表操作比较复杂，需要多个周期才能完成处理[12]。双向链表法增加了平均处理时间，并且存储器访存效率不变。本文研究如何提高存储器访存效率及流管理性能。

2.2 连续读写法

已有方法的制约因素主要有3点：多表项操作、读写延时和表项处理时间消耗。连续读写法是针对这3点提出来的。

制约因素1：多表项操作

双向链表每次更新涉及多个表项操作，这是制约双向链表法的最主要的原因。双向链表有两点作用：将有用表项连接起来，避免处理不活动超时流时读到空表项，否则无法判断有效数据的地址，必须将地址通读一遍；将表项按最后到达报文时间排序，则超时流表项就在链表的前部，便于处理不活动超时流。表1是从网络节点获取的流表项信息[13]，从表中观察到不活动流的数量远远高于活动流的数量，活动流非常少甚至都不足1%。即使设置超时时间从1分钟提高到30分钟，活动流的数量也仅仅提高了1倍。通过表1，我们发现超时流占流表的绝大部分。随着网络性能的提高、流媒体和P2P的广泛应用，当前网络中的UDP应用越来越多，UDP流的数量是TCP流的2.6～3倍[6]，网络中的流持续时间大部分为秒级，且TCP大部分可以通过FIN/RST包正常结束，而UDP没有结束包，只有通过不活动超时来删除，所以最终流表中的不活动流表项的数量远远高于活动流表项的数量，即正常情况下整个流表中大部分为不活动超时流，需要通过遍历将其清除；相反，活动流相对不活动流的比例过高，通常可以判断为DDOS攻击。排序并没有多大意义，那么采用链表仅需将有效表项连接起来，因此考虑采用单向链表，放弃复杂的双向链表。

表1 网络节点netflow输出

	活动流数量/超时时间设置	不活动流数量/超时时间设置
1	7367/30min	520150/15s
2	6455/30min	643521/15s
3	5355/30min	473572/15s
4	2138/1min	406356/15s
5	4245/1min	601844/15s
6	3362/1min	409685/15s

然而，采用单向链表会带来额外的查找开销问题。当要删除其中的一条表项时，需要将其前面的表项指向其下一跳，因为是单向链表，只知道它的下一跳，并不知道其前面表项的地址，如果要查找，需要对链表从头遍历，这将导致巨大的消耗。因此本文删除表项时只将有效位置空，并不修改其指针，这样在链表中会出现空表项，将这种空表项在处理不活动超时流时一起处理。空表项只是推迟了处理时间，总处理时间会略微缩减，因为如果在已删除表项上重新建立新的表项，链表的表项维护也就不需要了。

制约因素2：读写延时

读写延时是存储器访存的固有特性，研究存储器的读写时序发现，如图3所示的连续读或连续写中[11]，多个读或写过程共用一个 t_{delay}，此时数据线几乎达到满负荷，从而大大提高读写效率。当数据包到达链路时，改变原有的一包一处理的方式，将数据包积累一定数量后一并处理。

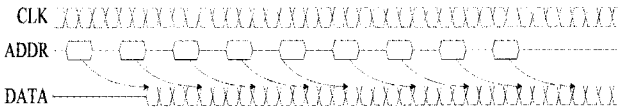

图3 连续读写操作

制约因素3：表项处理时间消耗

连续读写法不仅解决了读写延时问题，而且解决了读写转换问题。这种方法复用读数据时间，可同时读取并流水处理多个表项数据，当读出一定数量的表项时，最初读出的数据已经输出到数据线上，并且已经处理完毕，转换进行写操作，此后的写过程可以同时处理后续读出的数据。利用数据处理和读写操作复用来压缩 t_{cmp}，可使整个读写过程中数据线的利用率接近100%。

假设连续处理 k 条表项，其每条表项平均处理时间为：

$$T'=\frac{2kt_{data}+2t_{delay}}{k} \tag{3}$$

存储器访存效率为：

$$\eta'=\frac{2kt_{data}}{2kt_{data}+2t_{delay}} \tag{4}$$

显然当 k 较大时平均处理时间接近理论上限 $2t_{data}$，效率接近1，实际运用中 k 取值超过10，η 就能达91%。连续读写法和前面普通表项处理的效率比如式(5)所示，k 越大，效率比越高，连续读写法本质上合并了读写延时，消除了处理时间，因此 t_{delay}、t_{cmp} 越大时，效率比越高。

$$\frac{\eta'}{\eta}=\frac{2kt_{data}+2kt_{delay}+kt_{cmpd}}{2kt_{data}+2t_{delay}} \tag{5}$$

2.3 单向并行多链表法

连续读写法提高了包触发下的表项操作效率，利用此方法处理不活动超时流发现，传统的链表操作只有读出当前表项的信息才能知道下一跳表项的地址，这种方式显然有悖于连续读写的思想。为了达到连续读写的目的，需要将中间的读写延时填满，以达到数据线的高效利用，为此引入多链表交替的方式来实现。多链表在存储中的组织方式如图3所示，多个链表将散落的表项连接起来，而非原先的一条链表。如图4所示：以3链表为例，首先对链表1的第1个地址进行读操作，这个数据需要2个周期后才能到数据线上，然后分别读取链表2和链表3的第1个数据，当链表3的读操作结束以后，链表1的数据(包括下一跳地址)已经读了出来，可以继续读取链表1的下一个地址的数据，以此循环类推，可以连续读取大量数据，提高时间利用率，写操作同样如此。具体算法如图5所示。链表数目根据读写时延确定，只需将读写时延填充满即可。链表中已删除的表项和超时不活动流表项一起处理，因为链表数据是随着链表连续读取，只需保存上一有效表项地址即可，这样就能很容易地维护已删除的表项。

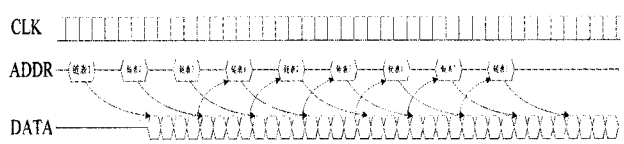

图4 遍历不活动超时流表项

单向并行多链表处理不活动超时流算法

```
1. 输入:L[1],L[2],…L[k]//输入链表
2.       head[1],head[2],…,head[k]//输入链表头指针
3.       cycle//处理周期数
4. 输出:queue//超时表项队列
5.    for i=1 to k do //初始化
6.      last[i]=head[i];//last 为最后一个保留表项地址
7.      L[i]. e=head[i]. next;//寻找下一个保留表项时,e 保存中间
                地址
8.    end
9.    for j=1 to cycle do
10.      for i=1 to k do
11.        if isdelete(L[i]. e)//处理已删除表项
12.          do_del(L[i]. e);//删除表项
13.          L[i]. e=L[i]. next. e;//下一待处理项指针
14.        elsif istimeout[L(i). e]//处理不活动超时流表项
15.          push(L[i]. e);//超时表项送超时队列
16.          do_timeout(L[i]. e);//删除不活动流表项
17.          L[i]. e=L[i]. next. e;//下一街处理项指针
18.        elxe
19.          last[i]=L[i]. e//跟新为最后一个保留表项地址
20.          L[i]. e=L[i]. next. e;//下一待处理项指针
21.      end
22.    end
23. end
```

图 5　不活动超时流的单向并行多链表读过程算法

3　性能分析及仿真

上一节介绍了连续读写法和单向并行多链表法，本节将利用实际测量的数据对其性能进行分析，根据处理方式不同分为两部分讨论：基本包触发操作和删除不活动超时流操作，并且利用网络中的实际数据验证方法的可靠性。

3.1　基本包触发操作分析

以太网最小包长为 64 字节，OC-768 链路最高速率为 $7.8*10^7$ 包/s。在基本操作中，选用较为常见的 4 突发读写，利用图 3 所示的 3 链表分析，12 个包为一组。

12 个包完成读写操作需要 56 个周期，平均每个包需要 4.67个周期处理，将 SDRAM 的频率设置为 400MHz，每秒可以处理 $8.5*10^7$ 包，处理速度满足链路速率的需求。而实际网络中平均包长为 614 字节，正常情况下网络平均包到达率仅为 $8.1*10^6$ 包/s，远远低于我们的处理能力。

3.2　删除不活动超时流操作分析

假设以两种情况进行分析，由于删除表项操作没有修改指针和维护链表，遍历不仅需要从链表删除不活动流，还包括删除前面已删除但没有修改指针的表项，但为了叙述方便，我们仍将其称为删除不活动流。由链表的知识可知，活动流表项越集中，消耗越小，理想情况下所有的不活动流表项和活动流表项分别集中在一起，最坏情况下所有的活动流表项在链表中均匀分布，本文均按最坏情况考虑。记 α 为不活动超时流和活动流的数量比。

(a)$\alpha=10$

假定有 22 个流表项、20 个不活动流表项和 2 个活动流表项，其中 2 个活动流表项分布在流表项的两端。由单向并行多链表法可知，22 个表项的维护需要 48 个读周期和 8 个写周期，即平均每个表项需要 2.55 个周期。

(b)$\alpha=1$

假定有 20 个流表项、10 个不活动流表项和 10 个活动流表项，活动流表项与不活动流表项交替分布，对其维护需 44 个周期的读操作和 24 个周期的写操作，平均每个表项需要 3.4 个周期。

对上述两种情况分析知，α 值越大，表项中大部分只需进行读操作，则平均每个表项的处理时间越短，一般情况下只需 2 个多周期；α 值越小，平均每个表项的处理时间越长，当 $\alpha=1$ 时，平均每个表项的处理时间达到最大为 3.4 个周期。

将不活动流超时时间设置为 15s，对于长度为 64 字节的包，需要消耗 14s 处理基本的包触发操作，剩下的时间处理不活动超时流，即使当 $\alpha=1$ 时，仍然可以处理多达 1.1 亿条表项。当然要处理千万或上亿条表项需要更大的存储，否则哈希冲突的加剧会严重影响性能。

3.3　性能比较

表 2 是对双向链表法和辅助存储法实际测试的结果，而单向并行多链表法的数据是最坏情况下的理论计算值，实际测试的结果会优于表中的值[14]。尽管如此，还是可以看出单向并行多链表法较双向链表法和辅助存储法有较大的提高，尤其比双向链表法提高了 1 个数量级，主要原因是单向并行多链表每个包的处理时间为 4.67 个周期，而双向链表需要 48 个周期。辅助存储法不涉及链表，因此遍历不活动超时流需要逐条查找，并在期间访问到大量空表项，访存效率低下；虽然辅助存储法查找时可以多条表项并行查找，但是由于哈希的大量冗余会大大降低存储访存效率，冗余过低又会使哈希冲突加剧，从而致使整体的性能下降。

表 2　性能比较

	链路速率	处理能力	表项容量
双向链表法	2.8Gbps	$7.8*10^6$ 包/s	$1.2*10^7$
辅助存储法	10 Gbps	$2*10^7$ 包/s	$2.5*10^7$
单向并行多链表法	40 Gbps	$7.8*10^7$ 包/s	$1.1*10^8$

3.4　仿真验证

上一小节从速率方面分析了单项并行多链表的高效性，这一小节将通过实验验证该方法的可靠性。利用 MAWI 工作组在链路中截得的 dump 数据包进行测试，主要针对 TCP 应用[15]。流表管理的可靠性主要包括超时表项能够及时删除和不该删除的表项不删除。删除不该删除的表项会产生不可逆的结果，所以删除表项的可靠性是评价检验方法最为重要的指标[12,13]。为了科学分析，首先进行如下定义：

真阳性(true positives 即 TP)：在当前时间段的活动流；

真阴性(true negatives 即 TN)：在当前时间段通过 RST/FIN 包结束的流以及清理的超时流；

假阳性(false positives 即 FP)：在下一时间段清理的超时活动流；

假阴性(false negatives 即 FN):在当前时间段被提前清理的活动流;

假阳率(FPR):该删除而没有被删除的流占活动流的比率;

$$FPR=\frac{FP}{FP+TN} \tag{6}$$

假阴率(FNR):不该删除而被删除的流占删除流的比率;

$$FNR=\frac{FN}{TP+FN} \tag{7}$$

正常流包括:SYN 包、中间包和 FIN/RST 包,但根据宽带网的特性,并不是所有包都走同一条路径,因此网络中单个节点不能捕获流的每个包。在链路传输过程中丢失 SYN 包,该条流不被管理;丢失中间包所受影响最小,仅仅导致包数和流数统计不准;但丢失 FIN/RST 包则不可避免地产生不活动超时流。图 6 为链路统计和分析的仿真结果。

图 6(a)为链路的平均活动流(TP)在设置不同超时门限时的变化图;图 6(b)为被清理的不活动流(TN)在设置不同超时门限时的变化图,发现 TP 和 TN 相差一个数量级;在图 6(c)中,FPR 随着设置超时门限的增加而逐渐减少,最终稳定在 10%以内,而实际网络中平均有 85%的流的所有数据包沿同一路径,实验与理论一致;在图 6(d)中,FNR 随着设置超时门限的增加而减小,除在 10s 以内时性能较差外,其余都能确保在 2%以内。

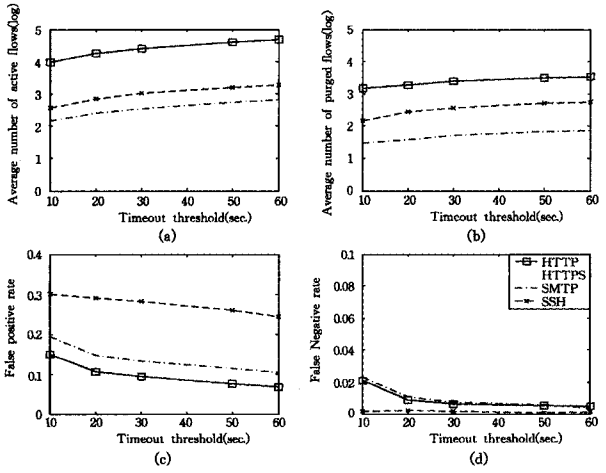

图 6 链路流状态仿真

结束语 读写时延和读写转换是影响流表项处理效率的关键因素。本文基于现有的存储器的读写特性提出了单向并行多链表法。通过理论分析证明了单向表项多链表法的优越性和高效性,并且通过仿真实验验证了其可靠性。单向并行多链表法相对于现有的流管理方法在处理速率上有显著的提高,能够满足 OC-768 链路逐包流管理的基本要求。同时单向并行多链表还可以应用到与读写操作相关的研究和技术实践之中,甚至可以结合辅助存储法继续提高性能。

本文的不足之处是,由于条件限制,没有在骨干网进行链路实验。虽然在理论上证明了单向并行多链表的性能,但其离实际应用还有距离。下一步需要将单向并行多链表运用到实际环境中,对其性能进行更加全面的测试,以期进行进一步的完善。

参 考 文 献

[1] 亓亚烜. 多域网包分类算法研究[D]. 北京:清华大学,2011

[2] Nam G, Patankar P, Kesidis G, et al. Mass Purging of Stale TCP Flows in Per-flow Monitoring Systems[C]// Computer Communications and Networks, ICCCN 2009, Proceedings of 18th Internatonal Conference on Date of Conference. 2009:1-6

[3] Estan C, Keys K, Moore D, et al. Building a Better NetFlow[C]// Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications, SIGCOMM'04. 2004:245-256

[4] Sadre R. The effects of DDoS attacks on flow monitoring applications [C]// Network Operations and Management Symposium (NOMS). IEEE, 2012:269-277

[5] Yoon S, Kim B, Oh J, et al. Session Management Architecture for Implementing an FPGA-based Stateful Intrusion Detection System[C]// Proceedings of the 8th WSEAS International Conference on Applied Computer Science(ACS'08). 2008:31-36

[6] Kumar P R, Deepamala N. Design for implementing NetFlow using existing session tables in devices like stateful inspection firewalls and Load balancers. 2010:210-213

[7] Tsai W-Y, Huang Nen-fu, Hung H-W. A Lock-Controlled Session Table Partitioning Scheme with Dynamic Resource Balancing for Multi-Core Architecture[C]// IEEE International Conference on Communications(ICC). 2011:1-5

[8] Nam G, Patankar P, Lim S-H, et al. Clock-like Flow Replacement Schemes for Resilient Flow Monitoring[C]// 29th IEEE International Conference on Distributed Computing Systems, ICDCS'09. 2009:129-136

[9] Koch R. Towards Next-Generation Intrusion Detection [C]// 3rd International Conference on Cyber Conflict(ICCC). 2011: 62-69

[10] Kanizo Y, Hay D, Keslassy I. Optimal Fast Hashing[C]// Publication in the IEEE INFOCOM 2009 proceedings. 2009

[11] 三星:DDR2 SDRAM 操作时序规范[OL]. www. SolidPDF. com. 2012. 5

[12] Yoon S, Kim B, Oh J. High-Performance Stateful Intrusion Detection System[C]// International Conference on Computational Intelligence and Security. 2006:574-579

[13] Sekar V, Reiter M K, Willinger W, et al. cSamp: A system for networkwide flow monitoring[C]// Proc. 5th USENIX NSDI. 2008

[14] Duffield N, Lund C, Thorup M. Estimating flow distributions from sampled flow statistics [C] // Proc. ACM SIGCOMM. 2003:325-336

[15] MAWI: WIDE traffic archive[OL]. http://tracer. csl. sony. co. jp/