高速网络中的数据流信息提取模型 *

高亚东，周明中，丁 伟

(东南大学计算机系，江苏 南京 210096；江苏省计算机网络技术重点实验室)

摘 要：以适当的方式定义流，并对流进行分析，已成为测量和分析网络行为的重要手段之一。本文在特定流定义下，构建了基于高速网络环境数据流信息提取的框架模型，分析和详细介绍了其中的关键问题，通过建立实验系统论证了框架模型的可行性和可靠性，并针对实验系统产生的性能瓶颈提出了相应的改进方案。

关键词：高速网络；流信息提取；散列；快速匹配

1 引言

近十几年来，网络技术的发展日新月异，网络的带宽显著增加，性能不断提高，用户数量和网络应用的规模、种类也在不断增多。然而，在这种急速扩张的背后，各种各样的问题也不断显现，例如网络的安全问题和服务质量问题等。当前，与网络快速发展不相符合的是，对网络运行客观规律的研究还缺乏系统的概念和方法，因而无法为提高网络性能和服务质量提供可量化的依据。

目前，主要通过对网络行为进行测量和分析的方法，去了解网络运行中应用和服务的实际工作状况，为技术的改进提供参考，从而提高网络服务的质量，提高应用的效率和效果。以适当的方式定义流，并对流进行分析，是测量和分析网络行为的重要手段之一。本文将给出流的一种定义方法，并在此基础上探讨数据流信息提取模型的构建方法，为有关高速网络环境中网络行为和网络测量的研究课题提供一个数据整合的工具。

2 流的定义

文献[3]中对数据流的定义：数据流是符合特定的流规范(specification)和超时(timeout)约束的一系列数据包的集合。对于相同的数据包序列，采用不同的流规范和超时约束可以得到不同的流集合。在网络测量中，广泛使用(源地址，宿地址，源端口，宿端口，协议类型)五元组作为流规范来区分不同的流。

本文也以五元组定义流。由于五元组所定义的流是主机对的粒度，而我们即将探讨的数据流信息提取模型也会关注目的主机、网络对、目的网络上的流状况，因此我们仿照五元组定义流的方式，给出适合于这三种情形的流定义方式。我们以(宿地址，宿端口，协议类型)三元组定义目的主机的流；以(源网络地址、宿网络地址、协议类型)三元组定义网络间的流，以(宿网络地址，协议类型)二元组定义目的网络的流。

基于上述定义，我们可以设计模型中使用的流信息数据结构。由于本文主要针对研究网络行为和进行网络测量来定义流，我们更关心的是数据流的测度信息，而不是数据流所承载的应用数据。这些测度信息包括：流的开始时间、流的结束时间、流的包数、数据包 TTL 之和、流的长度等等。这些信息通过提取网络数据报文报头中的信息并进行必要的统计获得。前面所述四种不同粒度的流定义方式中，主机对间的流是粒度最小的方式，在实际网络环境中数量也最多，我们将其流信息数据结构定义为：struct ph_flow{流开始时间；流结束时间；源地址；宿地址；源端口；宿端口；协议类型；流的包数；流的字节数；数据包 TTL 之和}。其它三种情形类似，不再详加叙述。

3 模型构建

根据上一节中给出的四种不同情形流的定义方式，进行数据流信息提取模型的构建。为此，将模型分解为四个子模型，分别对应主机对、网络对、目的主机和目的网络。涉及网络的子模型，通过让用户选择不同的掩码实现对不同网络类型(比如 B 类或 C 类)的支持。分别称这四个子模型为：对等主机、对等网络、目的主机、目的网络。

将这四个子模型都分解为数据处理模块和流存储模块，其整体框架如图 1 所示。

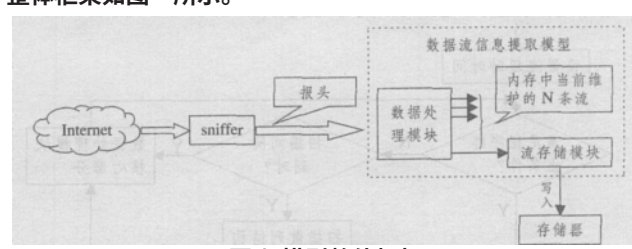

图 1 模型整体框架

这个框架示意图中，数据处理模块根据四种流定义方式之一，以及用户设置的流超时时间，将报头所携带的信息整合进相应的流信息结构；流存储模块将数据处理模块中整合完毕的流(即因超时而结束的流)从内存取出，根据一定的分类策略写入存储器，其实现较容易，不展开讨论。下面我们以四个子模型中流粒度最小、细节最复杂的对等主机模型为例，讨论数据处理模块。

在数据处理模块，我们采用散列结构来维护内存中的流：

1、开设一定大小的静态数组作为散列表，静态数组的元素是流结构。

2、对于不同的关键码，通过散列函数的计算，可能得到同

* 基金项目：国家 973 计划课题(2003CB314803)

一散列地址，这就会产生冲突。冲突太多会降低匹配效率，所以构造散列函数时要使关键码经过散列函数的计算，比较均匀地映射到散列表地址集合，以减少冲突。但是关键码集合通常比散列表地址集合大得多，这种情况下冲突是不可避免的。我们在发生散列冲突时，通过构造二叉搜索树来处理冲突，方法是在表示流结构的内存单元中添加两个指针域，用于指向两棵子树。

3、在对等主机子模型中，我们以新流的源、宿地址之和与已有流的源、宿地址之和的大小关系，来决定新流插入到已有流的哪棵子树中。由于此时只考虑了五元组中的两元，而且使用了加法，造成不同流的源、宿地址之和依然可能相同，所以比较结果是小于时插入到左子树中，而大于和等于时则插入右子树中。

于是，静态数组的元素成为树根，而其子树上的元素在程序运行过程中进行动态插入和删除，所需内存空间也进行动态分配和回收。匹配时先通过散列函数得到直接匹配位置，再在相应的二叉搜索树中搜索。

图 2 显示了上述散列方法的散列表结构。

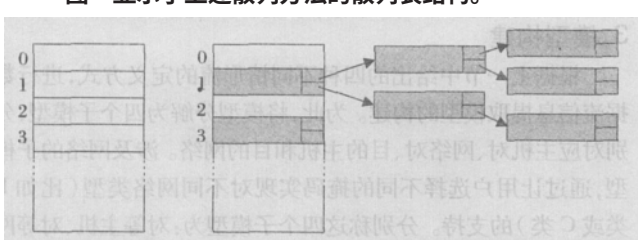

图 2(a) 空散列表　　图 2(b) 使用中发生冲突的散列表

我们通过图 3 和图 4 给出对等主机模型数据处理模块的框架和核心部分概要流程。

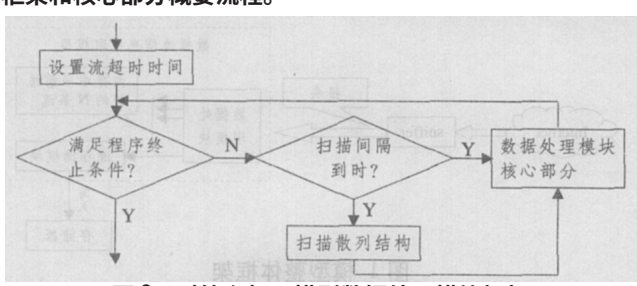

图 3 对等主机子模型数据处理模块框架

图 3 是数据处理模块的框架，由用户设置流超时时间（timeout时间），然后即进入循环。在循环中调用如图 4 所示的核心部分，并且在每个扫描间隔时间内扫描一次内存中的散列结构。

图 4 显示了数据处理模块核心部分的概要流程。程序首先输入一个报头，调用散列函数以报头中的特征信息为关键码计算散列值，然后根据散列值直接定位到相应散列表项。如果该散列表项是空的，则直接根据报头中的信息在该散列表项建立新流。如果该散列表项非空，则说明存在节点数至少是 1 的二叉搜索树。此时，计算报头中源、宿地址之和，与当前流的源、宿地址之和进行比较。如果不相等，则转入二叉搜索树当前节点的左子树或右子树 如果相等，则进行五元组比较。在进行五元组比较时，如果五元组不匹配，则转入右子树继续搜索 如果五元组匹配，则检查当前流是否超时，超时则取出流进行存储并将报文整合成新流存入二叉搜索树的当前位置，不超时则直接根据报头信息更新流信息。如果报文已经搜索到叶节点但仍未找到匹配的流，则说明内存中不存在和当前报文匹配的流，此时报文整合成新流，插入到二叉搜索树中。

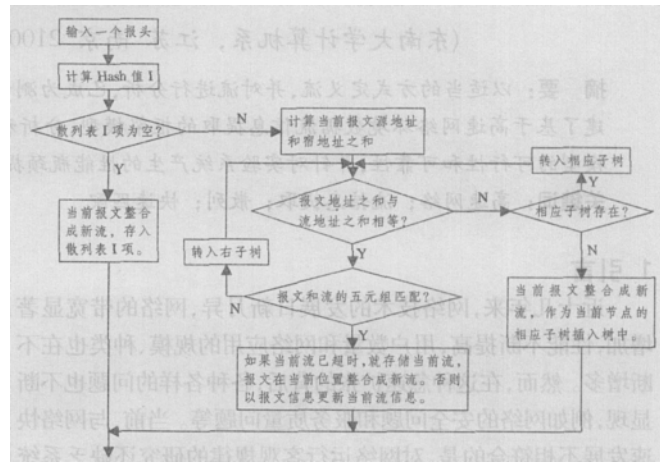

图 4 对等主机子模型数据处理模块核心部分概要流程

4 关键问题阐述

4.1 报头和流的快速匹配

在 IPv4 网络报文中，源、宿地址字段都是 32 比特，源、宿端口字段都是 16 比特，五元组的样本空间非常庞大，即使不考虑协议类型字段 16 比特的取值，样本空间仍将达到。显然，由于内存资源的限制，我们无法为每一个可能出现的流都预先开设一个流结构单元的内存空间。在实际的网络运行环境中，流的数量通常在一定时间范围内保持相对稳定。我们使用五元组定义流，采用 64 秒超时机制，对 CERNET 华东（北）地区主干进行观测，目前正常情况下网络最繁忙时主干上有 100 多万条流。这种情况下，在内存中开设 200 万个流结构单元，并且引入淘汰机制处理可能出现的流数量超出预定义空间的情形，就足以维护这些流。对 CERNET 华东（北）地区主干的统计表明，超过 40% 的流是只有一个包的流，即单包流。此外，包数较多的流在研究网络行为时特别值得关注。所以在选择要淘汰的流时，优先淘汰单包流，避免淘汰多包流，从而尽可能减少淘汰流所带来的负面影响。

然而，我们必须考虑性能问题。假设内存中正在维护的流的数量为 N，而且存在与到达报文匹配的流，如果我们使用静态数组或线性链表维护这 N 个流，那么为一个到达的报文寻找匹配的流平均需要 N/2 次匹配。而 N 值通常在几万、几十万以上，顺序匹配极其耗时，根本无法满足高速网络在线实时处理的要求。

正是基于上述原因，我们在模型中使用散列技术（二叉搜索树处理散列冲突）来实现报文和流的快速匹配。在性能上，我们设散列表已使用的表项数和当前正在维护的流数之比是 M：N。显然，M 的值除了理想状况下和 N 的值相等外，通常会因为存在散列冲突而小于 N 的值。M 棵树，平均每棵树有 N/M 个流。由于二叉搜索树在理想情况下是二叉平衡搜索树，在最坏

情况下退化为线性链表,所以其搜索效率介于二叉平衡搜索树和线性链表之间。对于网络上一个到达的报文,假设内存中存在与其匹配的流，则经过散列函数的计算得到散列地址后,在二叉平衡搜索树中的匹配次数不超过[log(N/M+1)];在线性链表中 /的平均匹配次数是(N/M) /2。设 N 的值是 100 万,M 的值是 10 万,上述方法的平均匹配次数不超过 5,显著优于顺序匹配时的平均 5 万次匹配。

4.2 散列结构维护机制

在程序运行过程中,不断有新流产生,同时也有很多流超时结束。如果这些超时的流不及时从内存取出存储,就会影响到匹配的效率。更严重的是,内存中的流会越积越多,最终导致系统内存耗尽,进程死亡。因此,在数据处理模块中,我们设计了 3 种对内存中散列结构进行维护的机制：

1、在为报文寻找匹配流的过程中,对于遇到的每一条不匹配的流,均检查其是否超时。如果超时,就将流从二叉搜索树中删除,转入存储模块进行存储。

2、当报文寻找到匹配流时,先检查这个流是否超时。如果超时,就将流转入存储模块进行存储,报文则在原匹配流的位置整合成新流。

3、内存散列结构的定时扫描机制,每过一段时间就扫描整个散列结构,将其中所有超时的流取出进行存储。

其中，第 1 种和第 3 种机制需要对二叉搜索树结构进行动态调整,并对内存空间进行动态回收。如果每隔一段较短的时间就对散列结构进行一次扫描,则可以不使用第 1 种机制,从而使匹配过程尽可能简单。图 3 和图 4 中就只使用了后两种机制。

5 进一步改进

为进一步提高效率,可以在两方面进行改进：一方面,构造更适用于数据流整合的散列函数，使散列地址分布更均匀,提高散列表的利用率；另一方面,对前一节中设计的散列结构继续改进。前者不是本文讨论的内容,这里针对第二种情况提出改进方法。

模型中散列结构在处理冲突时使用二叉搜索树,它通常不平衡,而且存在使二叉搜索树退化为线性链表的极端情况。因此,较为可行的办法是增加二叉搜索树平衡算法,每次对内存中散列结构进行扫描时,有选择地对一些二叉搜索树做平衡化操作,但这必然会增加散列结构的扫描时间。因此,我们必须加大 sniffe和模型间的报头暂存缓冲区容量,使其足以容纳在扫描时间内采集到的报头。我们之所以不对所有的树进行平衡化,是为了在改进匹配效率的同时,降低平衡化操作带来的额外开销,从而改进系统整体性能。

二叉树的平衡算法在有关数据结构的书籍中都有介绍,本文不再赘述,但在处理时需要注意这里的二叉搜索树的树根是静态散列表的表项,涉及树根的调整只能通过结构变量赋值实现,而树根以外节点的调整只需修改指针。下面我们介绍如何选择要进行平衡的二叉搜索树。

我们设置两个整型变量：当前二叉搜索树的节点数 k 和高度 h，在进行内存散列结构扫描的过程中获取当前二叉搜索树的信息。这两个值在扫描过程中,每遇到一棵二叉搜索树就从零开始累加。这些累加操作在扫描过程中附带进行,对系统的影响微乎其微。此外,我们还设置整型常量 H 和 D,其中 H 用来表示需进行平衡的二叉搜索树最小高度,D/100 则用来表示需进行平衡的二叉搜索树的节点最高使用率。当二叉搜索树的高度 h 大于 H,并且二叉搜索树的节点数 k 小于(2^{h+1}-1) D/100 时,对该二叉搜索树做平衡化操作。我们在加入平衡算法后的实验系统中将 H 取为 4,D 取为 60，实现只对那些对系统性能有较大影响的非平衡二叉搜索树做平衡化操作,从而尽可能降低了平衡化操作对系统造成的额外负载,并使系统性能得到改进。

6 总结

本文探讨了高速网络环境中,在线实时提取数据流基本信息的模型构造方法,并对其中的关键问题进行了阐述。我们在 CERNET 的 2.5G 主干上使用该模型做了在线实时处理的实验。在当前的网络环境下,该模型能够适应实时处理的要求,具备较高的效率。同时我们也发现依然存在一些问题,主要是散列值的分布仍然不是很均匀,程序运行过程中会出现少量节点数在 100 以上的二叉搜索树(内存中流数量和使用的散列表项数之比约是 4) 。出现这种现象的原因在于关键码的分布的不均匀性传递到散列值上。希望有好的散列函数构造方法克服这一问题。

参考文献：

[1] 喻中旭,吴建平,徐恪.IP 分类技术研究.电子学报,2001.2.

[2] 陈刚,鲍剑洋,丁颖.对网络行为进行测量和分析.计算机世界网, URL:http://www.ccw.com.cn/applic/tech/htm2003/20030313_15WNH.asp.

[3] Ryu B, Cheney D, Braun H.W. Internet Flow Characterization: Adaptive Timeout Strategy and Statistical Modeling [J].In Workshop on Passive and Active Measurement(PAM),2001.4.

[4] Pankaj Gupta and Nick McKeown, Packet Classification on Multiple Fields,URL:http:// tiny-tera.stanford.edu/~nickm/papers/Sigcomm99.pdf.