基于局部性原理跳表的内容路由器缓存快速查找机制

潘 恬[1),2)] 黄 韬[1),2)] 张雪贝[1)]

[1)](北京邮电大学网络与交换技术国家重点实验室 北京 100876)

[2)](北京未来网络科技高精尖创新中心 北京 100124)

摘 要 当前的互联网流量模型正迅速从端到端通信发展为内容的传播共享.为了适应这种变化,学术界提出了内容中心网络(CCN)的未来网络架构.在内容中心网络中,终端主机根据内容的名字而不是内容所在主机的IP地址进行通信.内容路由器作为底层基础设施,承载着内容请求和内容响应的高速交换的任务.不同于IP路由器的无状态转发模型,内容路由器的数据平面因具备网络缓存功能而更加复杂.尽管网络缓存可以有效减轻网络拥塞,但这样的设计也将额外的状态附加到网络中间节点上,这在一定程度上破坏了网络设计中的"端到端"原则.这些额外状态给路由器的数据平面造成了性能负担,也使内容路由器数据包转发成为当下热门的研究课题.内容路由器包含转发信息库(FIB)、待定兴趣表(PIT)和内容缓存(CS)三个模块.针对基于名字的最长前缀匹配、名字路由表的状态爆炸、待定兴趣表的频繁更新等问题,目前大量研究工作围绕FIB和PIT开展性能优化.然而,尽管CS也存在潜在的性能问题,却较少有工作对其进行讨论.FIB、PIT和CS在内容路由器中以流水线的形式存在,而流水线的处理速度由最慢的流水段决定.因此,为了提高内容路由器的整体性能,有必要首先确定其性能瓶颈,然后进行针对性的优化.在该文中,为了避免盲目的性能优化,我们首先建立了基于开放排队网络的数学模型.通过定量分析,文章发现CS是整个路由器的性能瓶颈.目前,已有工作采用跳表作为CS的数据结构.然而由于其$O(\log n)$的查找复杂度,经典跳表在处理高速网络流量时依然存在性能问题.受到网络流量中广泛存在的时间局部性和空间局部性的启发,我们提出了局部性原理跳表来提升CS的性能.在新的设计中,考虑到新到达的内容请求可能与之前到达的内容请求共享相同的名字前缀,且所请求的内容块通常处于邻近的存储空间,因此在查找跳表时可以直接从该前缀先前命中的跳表结点附近开始查找.这样可以避免每次都从跳表的头结点开始重复的查找,从而节省大量时间.实验结果表明,局部性原理跳表可以在x86处理器上实现1.796 Mpps的单线程吞吐率,这是经典跳表查找速度的三倍.对于500 B大小的数据包包长来说,CS的单线程吞吐率可以高达7.016 Gbps.

关键词 内容中心网络;内容路由器;排队网络;跳表;局部性;查找算法

中图法分类号 TP393 **DOI号** 10.11897/SP.J.1016.2018.02029

Fast Content Store Search Algorithm via Locality-Aware Skip List in Content Routers

PAN Tian[1),2)] HUANG Tao[1),2)] ZHANG Xue-Bei[1)]

[1)](*State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing* 100876)

[2)](*Beijing Advanced Innovation Center for Future Internet Technology, Beijing* 100124)

Abstract The Internet usage is experiencing a rapid evolution from host-to-host communication to content dissemination. To embrace this trend, content-centric networking (CCN) is proposed as a promising future Internet architecture. In CCN, end hosts communicate based on named content

收稿日期:2016-09-28;在线出版日期:2017-08-13. 本课题得到国家"八六三"高技术研究发展计划项目(2015AA016101)、国家自然科学基金(61501042,61702049)、中国博士后科学基金(2016M590068)和北京市科技新星计划(Z151100000315078)资助.
潘 恬,男,1987年生,博士,讲师,主要研究方向为内容中心网络、软件定义网络、天地一体化网络、高性能可编程网络设备. E-mail: pan@bupt.edu.cn.
黄 韬(通信作者),男,1980年生,博士,教授,主要研究领域为未来网络架构、软件定义网络、内容中心网络. E-mail: htao@bupt.edu.cn.
张雪贝,女,1993年生,硕士研究生,主要研究方向为内容中心网络、天地一体化网络.

rather than IP addresses. Content routers are developed as the underlying infrastructure for carrying the high-speed exchange of content request messages as well as content return messages. Unlike IP router's stateless forwarding model, a content router has a far more sophisticated data plane with the capability of in-network caching. Although the in-network caching allows content requests to be satisfied directly by routers instead of from end hosts to reduce network congestion, it impairs the classic "end-to-end" principle in network design by pushing extra states into the intermediary nodes of the network. These states lay performance burden on content router's data plane and make CCN packet forwarding a hot research topic. There are three major components in a content router, namely, forwarding information base (FIB), pending interest table (PIT) and content store (CS). Due to the performance issues such as longest prefix match in an unbounded name space, expensive memory cost for explosive routing states, frequent update of pending states, lots of recent works focus on the performance issues of FIB and PIT. However, interestingly, although it is quite obvious that CS will also suffer from the performance penalty, it seems to be unintentionally ignored by the research community. Since FIB, PIT and CS are organized into a three-stage pipeline while generally, a pipeline runs only as fast as its slowest stage, to improve the overall performance of the content router, we should identify the performance bottleneck first and then, eliminate the exact bottleneck by a novel design. In this work, to prevent "blind optimization", we first build a mathematical model based on open queuing networks which identifies CS as the performance bottleneck. Currently, the state-of-the-art adopts skip list as CS's underlying data structure. But due to its $O(\log n)$ search complexity, the original skip list has a severe performance issue. Inspired by the extensively existed temporal locality and spatial locality in network content retrieval, we propose locality-aware skip list to improve CS's performance. In our design, since an incoming packet is likely to share a name prefix with previously arrived packets, the search of the packet in a skip list could start directly from near the previously reached node of that prefix. This avoids searches repeatedly starting from the head node in the original skip list thus can save considerable search time. Evaluation shows that our design can achieve 1.796 Mpps single-threaded throughput on a less powerful x86 processor, which is a 3x speedup of the original design. Given the average packet size is 500 B, the single-threaded throughput can reach 7.016 Gbps.

Keywords content-centric networking; content router; queuing networks; skip list; locality; search algorithm

1 引 言

随着科技的进步和网络的普及,信息的发布和共享逐渐成为当今互联网的主要功能.互联网上的流量模型正迅速从端到端的信息通讯传输转变为包括文字、图片和视频等在内的多媒体信息的全球共享.每时每刻大量的信息从一部分终端产生,并通过互联网的传播被另一部分终端访问.用户逐渐不关心也无需关心所请求的内容在网络上存放的位置(where),而只需要关心内容本身(what),获取内容的效率(efficiency)以及内容自身的安全性(security).为了克服早期 IP 网络在信息检索和内容共享方面的低效性,提高早期 IP 网络的内容分发能力,内容分发网络(CDN)[1-2]、对等网络(P2P)[3-4]等基于传统 IP 网络的新技术被提出和使用,来满足当前以内容为中心的网络需求.由于研究和发展以内容为中心的网络架构符合当前互联网发展的迫切需求,全球诸多工业界和学术界的研究小组开始设计信息中心网络(Information-Centric Networking, ICN),旨在通过赋予基本的网络操作原语(primitive)来直接对网络上的内容进行命名和查询,以提高信息的检索和传播效率.在 ICN 中,不同技术方案采用不同的内容寻址方式,可以大致分为基于名字解析寻址(name resolution)和基于名字路由(name-based routing)两大类.在基于名字解析寻址的方法中,内

容的获取分为两个步骤，首先是通过名字找到对应内容的存储位置，然后发送内容请求到该位置将内容获取回来. 而在基于名字路由的方法中，内容请求直接通过内容路由器(content router)转发，内容路由器本身依据请求中的内容名字决定下一跳的转发端口，内容路由器通过类似传统 IP 网络中的分布式路由协议来计算生成名字路由表.

内容中心网络(Content-Centric Networking，CCN)[5-7]是 ICN 中基于名字路由方法中的典型代表. CCN 在根本上改变了数据包的封装结构和寻址方式，数据包的包头不再以 IP 地址作为标识，而是以内容名字作为标识. 当终端向网络请求内容时，它首先向网络发送一个内容请求包，包头包含被请求内容的名字；一旦能够提供这个内容的网络节点收到这个请求包，该节点将立刻把内容封装成数据包原路返回给请求者，从而完成信息的共享. 这个网络节点可以是终端的服务器，也可以是网络中的路由器. CCN 并不关心是哪台设备对内容请求作出了响应，内容存放在哪里，而只关心如何以最小的代价从邻近节点更高效的获取该内容，这是 CNN 和基于 IP 地址的传统网络架构最大的不同. 有别于传统网络通过“打补丁”的方式在覆盖层(overlay)上实现信息的共享(例如 CDN 和 P2P)，CCN 直接在网络层的基础架构上高效的提供这样的能力.

由于内容请求可以由 CCN 的网络节点直接作出响应，这势必要求内容路由器自身能够提供高效的内容缓存功能，这与传统 IP 路由器的设计理念存在巨大的差别. 因为传统网络遵循“中间简单，边缘复杂”的“端到端”的设计哲学[8]，IP 路由器只提供尽力而为的转发功能及有限的服务质量保证能力. 而内容路由器增加了内容缓存功能，使得用户可以就近获取内容，进而能够有效减轻网络拥塞，但这也使得路由器数据平面的设计变得异常复杂. 内容路由器高速数据包转发已成为当下热门的研究课题. 内容路由器数据平面包含转发信息库(FIB)、待定兴趣表(PIT)和内容缓存(CS)三个模块. 针对数据包转发过程中遇到的基于名字的最长前缀匹配、名字路由表的状态爆炸、待定兴趣表的频繁更新等问题，目前大量研究工作围绕 FIB 和 PIT 开展性能优化. 然而，尽管相较于在服务器上实现低速的文件缓存系统[9]，在高速路由器上实现类似的设计存在巨大的技术挑战，目前却较少有工作针对 CS 的性能问题进行讨论. 在内容路由器中，FIB、PIT 和 CS 三个模块以流水线的形式存在，而流水线的处理速度取决于最慢的流水段. 因此，为了提高内容路由器的整体性能，有必要首先确定其性能瓶颈，然后进行有针对性的性能优化. 在本文中，为了避免盲目的性能优化，我们首先建立了基于开放排队网络[10]的数学模型，将内容路由器中的每个模块抽象为由一个先入先出队列(queue)和一个服务器(server)构成的对象，并用代表流量转移概率的有向线段将模块们连接起来. 通过定量分析，我们发现 CS 是整个内容路由器的性能瓶颈.

目前，已有的工作[11]采用跳表(skip list)[12]作为 CS 的数据结构. 相比于经典的平衡二叉树数据结构(如 B 树、红黑树、AVL 树等)，跳表具有结构简单、容易实现的优势；相比于哈希表，跳表又具有灵活易扩展的特点. 然而，由于其 $O(\log n)$ 的查找复杂度，经典跳表在处理高速网络流量时依然存在性能问题，尤其是新到达的每一个查找请求都需要从跳表的头结点开始，经过大量的向右和向下遍历操作才能最终定位到目标元素. 受到网络流量中广泛存在的时间局部性和空间局部性的启发，本文提出了局部性原理跳表来提升 CS 的查找性能. 在新的设计中，考虑到新到达的内容请求可能与之前到达的内容请求共享相同的名字前缀，且所请求的内容块通常处于邻近的存储空间，因此在查找跳表的过程中可以直接从该前缀先前命中的跳表结点附近开始查找. 这样的设计可以避免每次都从跳表的头结点开始重复的查找，从而节省大量时间. 实验结果表明，局部性原理跳表可以在 x86 处理器上达到 1.796 Mpps 的单线程吞吐率，这是经典跳表查找速度的三倍. 对于 500 B 大小的数据包来说，CS 的单线程吞吐率可以高达 7.016 Gbps. 我们认为本方案可以为未来内容路由器中内容缓存模块的设计和实现提供一定的参考和依据.

在后续的章节中，我们首先介绍内容中心网络的背景和内容缓存模块的相关工作；然后通过对内容路由器体系结构进行数学建模和量化分析定位内容路由器的性能瓶颈；之后，我们将对真实骨干网流量进行测量，并推测论证内容中心网络流量的局部性特征；基于流量的局部性特征，我们提出局部性原理跳表的设计和实现，并重点讨论局部性原理跳表的查找和更新算法；然后，我们对所提出的数据结构和算法进行实验和性能评估，重点比较内容请求在经典跳表和局部性原理跳表上的查找效率；最后我们总结本文内容并展望后续工作.

《计算机学报》

2 背景及相关工作

2.1 内容中心网络(CCN)

首先简单介绍一下内容中心网络及内容路由器的体系结构.在内容中心网络中,存在两种数据包类型:兴趣包(Interest packet)和数据包(Data packet).用户端发送兴趣包到网络中请求内容,兴趣包的包头上携带有所请求内容的名字,而从内容路由器或终端服务器获取的内容则会被封装成数据包返回.同时,内容会被存储在沿途的内容路由器上,这样后续携带相同名字的内容请求就不必转发到终端服务器上,在内容路由器上就可以直接得到响应.因此,内容路由器的包处理流程比IP路由器更为复杂(如图1所示).内容路由器主要由FIB、PIT和CS三个模块组成.当兴趣包到达内容路由器时,首先查找CS是否存在所请求的内容.如果存在,则路由器会读取相应的内容并将其封装成数据包直接返回,否则兴趣包会进入PIT查找是否存在相同的内容名字.如果存在,说明之前已经向上游路由器发起了对该内容的请求,且所请求的内容尚未返回,那么此时就不需要重复向上游网络转发兴趣包,只需要在PIT中新增该兴趣包的到达接口信息.如果PIT中不存在该内容名字,那么除了要在PIT中添加以该名字为键值的记录外,还需要将兴趣包通过FIB继续向上游网络转发.当包含所请求内容的数据包从上游网络返回到当前路由器时,路由器首先会查找PIT中的记录,如果能找到之前兴趣包留下的到达接口信息,就将数据包通过对这些接口的多播操作(multicast)转发到下游网络,同时还会在CS中依据一定的缓存策略(cache policy)对内容进行缓存.如果PIT中找不到相应记录,则直接丢弃该数据包.通过以上对内容路由器流量处理流程的分析,我

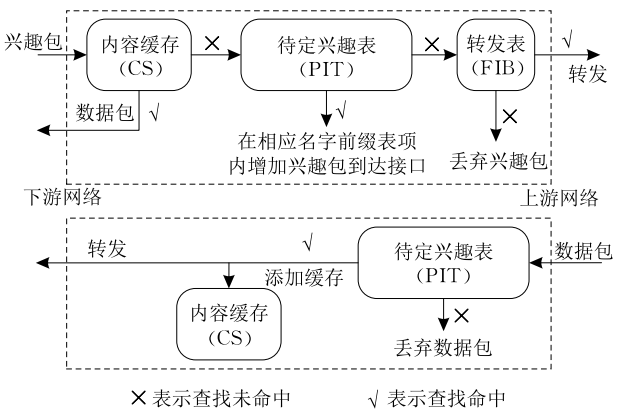

图1 内容路由器中的数据包处理流程

们可以看出,数据包总是与对应的兴趣包沿着相反的路径穿过沿途的内容路由器.

2.2 FIB和PIT的性能优化

设计内容路由器FIB的主要技术挑战在于在巨大的名字空间中实现无长度限制的基于字符串的最长前缀匹配.针对该技术挑战的主要研究工作包括:文献[13]提出了通过GPU实现线速名字匹配的方法.为了更好地利用GPU的高并行度,提出了基于字符串树的存储对齐状态转移数组作为FIB的数据结构.该方法可以取得63.52 Mpps的查找吞吐率和100 μs的查找时延.文献[14]利用两级布隆过滤器数组实现基于内容名字的FIB,其中第一级布隆过滤器数组确定名字的长度,第二级布隆过滤器数组对从第一级中筛选出来的确定长度的名字进行查找.该方法可以在x86服务器上取得37 Mpps的查找吞吐率.文献[15]提出了基于确定最坏情况哈希表实现的FIB,并取得了20 Gbps的吞吐率.文献[16]采用Patrica树对FIB进行压缩并将压缩后的FIB放入快速存储器中.该方法能够基于SRAM和DRAM分别实现284 Gbps和62 Gbps的吞吐率.

设计内容路由器PIT的主要技术挑战在于同时满足网络上高速的查找和更新请求[17].文献[18]设计了面向PIT表项的固定长度索引结构,将PIT的存储开销压缩至37~245 MB,并取得了100 Gbps的查找速度.文献[19]提出了基于开放寻址d-left哈希表的PIT设计,能够在存储1 M表项的情况下达到10 Gbps的吞吐率.文献[20]提出了基于布隆过滤器的PIT设计,并将PIT的存储开销压缩了63%.

有趣的是,尽管CS也有成为内容路由器潜在性能瓶颈的可能性,目前却较少有研究对CS的性能开展讨论.根据我们的调研,文献[21]提出了基于DRAM/SSD的层次化CS设计,在现有硬件上达到了10 Gbps的查找吞吐率.显然,相比于研究者宣称的FIB和PIT所达到的处理性能,CS的性能优化工作还任重而道远.

2.3 CS的数据结构(跳表)

目前,软件内容路由器NFD(NDN Forwarding Daemon)[11]基于跳表[12]实现内容路由器内容缓存数据结构.相比于经典的平衡二叉树数据结构(如B树、红黑树、AVL树等),跳表具有结构简单、容易实现的优势;相比于哈希表,跳表又具有灵活易扩展且不受装载率(load factor)影响的特点.一个跳表是由多层的有序链表构建而成.其中,最底层的链表包

含了已经排好序的所有待查找元素.在跳表中,除了最底层的链表,每一层链表均包含下面一层链表中所有元素的一个有序子集,因此,在跳表中进行元素查找时,上层链表可以看作是下层链表的一个“快车道”.具体而言,元素的查找过程首先从跳表的头结点(跳表中顶层链表的头结点)开始.查找时,不断向右遍历本层有序链表,直到遇到当前结点的键值大于或等于目标元素键值的情况.若当前结点的键值等于目标元素的键值,则查找成功并返回;否则,将回退到当前结点同一层的前一个结点,并从该结点进入下一层的链表继续上述向右遍历的操作.上述“向右-向下”的循环过程一直持续到完成最底层链表的遍历查找.若在最底层的链表中依然没有找到目标元素,则返回查找失败.图 2 显示了经典跳表的查找过程.查找键值为 26 的目标元素时,首先从第 4 层(最顶层)链表的头结点开始,依次经过第 4 层链表键值为 6 的结点、第 3 层链表键值为 6 的结点、第 3 层链表键值为 25 的结点、第 2 层链表键值为 25 的结点、第 1 层(最底层)链表键值为 25 的结点、第 1 层链表键值为 26 的结点最终查找成功.算法 1 展示了经典跳表查找算法的细节.其中 list−>level 代表跳表的层数;x−>forward[i]代表第 i 层链表中结点 x 的前向指针(指向有序链表中 x 的下一个结点的指针).根据文献[12]所述,跳表查找操作的计算复杂度为 $O(\log n)$.也就是说对于一个包含 1M 个元素的内容缓存来说,平均至少需要完成 10 次遍历操作才能最终定位到目标元素.这样的计算复杂度在处理高速网络流量时显然存在较大的性能问题.

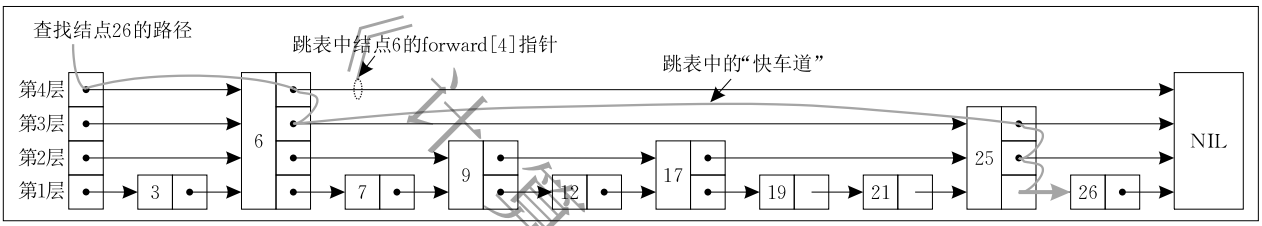

图 2　经典跳表的查找过程

算法 1.　经典跳表查找算法.

输入:经典跳表数据结构 *list*,查找元素的键值 *search_key*

输出:若成功返回键值为 *search_key* 的元素,否则返回失败

```
search(list,search_key)
  x := list->header
  //循环不变量: x->key <search_key
FOR i := list->level DOWNTO 1 DO
  WHILE x->forward[i]->key <search_key DO
    x := x->forward[i]
  //x->key<search_key≤x->forward[1]->key
  x := x->forward[1]
IF x->key=search_key THEN RETURN x
ELSE RETURN failure
```

图 3 显示了经典跳表的插入过程.大致来说,插入过程可分为三个阶段.第一个阶段为搜索过程,即确定插入元素在跳表中的具体位置(即插在哪两个跳表结点之间).该过程与跳表的查找算法基本一致,区别是插入过程中还使用了一个 update 数组在搜索时中顺带记录下了未来新结点在每层有序链表中的插入位置.第二个阶段为新结点的创建过程.在创造新结点时,首先通过随机数生成器确定新结点的层数,若该层数大于跳表原来的最大层数,则需要更新跳表的最大层数和 update 数组.然后根据新结点的层数为其分配内存空间并完成初始化.第三个阶段

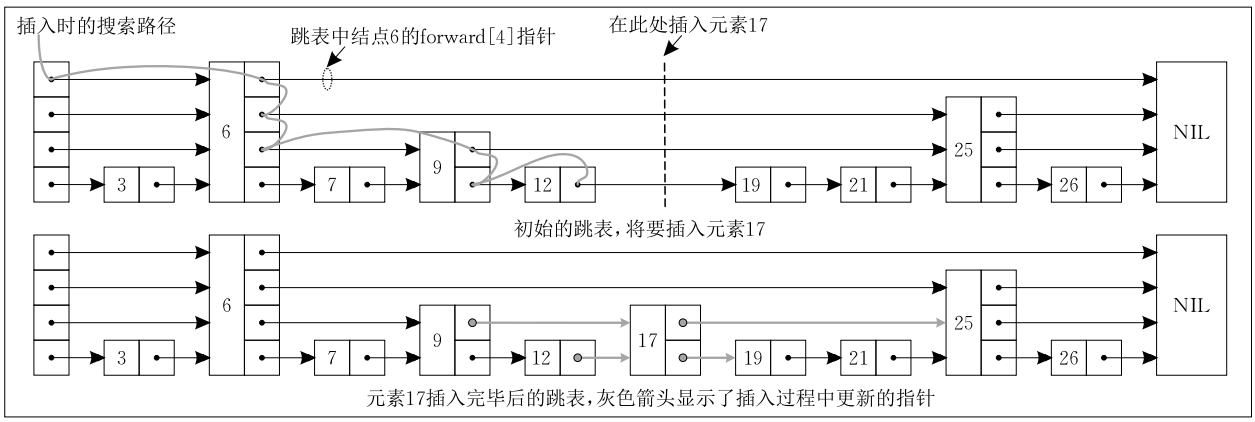

图 3　经典跳表的插入过程

为插入时中通过修改跳表内部指针将新结点融入到跳表数据结构中.算法 2 展示了经典跳表插入过程的三个阶段.跳表插入操作的计算复杂度为 $O(\log n)$,主要由第一阶段决定.

算法 2. 经典跳表插入算法.

输入:经典跳表数据结构 *list*,插入元素键值 *search_key*
输出:将该元素成功插入跳表

```
insert(list, search_key)
  x := list->header
FOR i := list->level DOWNTO 1 DO
    WHILE x->forward[i]->key<search_key DO
      x := x->forward[i]
    update[i]=x
  x := x->forward[1]
IF x->key≠search_key THEN
    new_level := random_level()
    IF new_level>list->level THEN
    FOR i := list->level+1 TO new_level DO
        update[i]:= list->header
      list->level := new_level
    x := alloc_node(new_level,search_key)
    FOR i := 1 TO new_level DO
      x->forward[i]:=update[i]->forward[i]
      update[i]->forward[i]:=x
```

3 内容路由器数学模型

3.1 基本假设

图 1 中的内容路由器包括 ingress 和 egress 两个转发方向.在每个方向上,兴趣包或数据包都需要穿过一条流水线,并在流水线上的各个模块中接受处理.当包离开某个模块时,将有选择性的以一定概率进入后续模块.当某个模块处理能力不足以应对到达的突发流量时,将有可能发生流水线阻塞或者丢包的现象.显然,图 1 中的内容路由器体系结构可以用排队论中的数学工具进行建模分析.我们首先给出以下假设前提:(1)内容路由器中的每一个模块(例如 FIB)均可以被抽象为由一个先入先出(FIFO)的队列(queue)和一个服务器(server)构成的对象,其中队列用于包缓存,服务器用于包处理;(2)包的到达过程可以被视为一个泊松过程(Poisson process);(3)每个服务器上的包处理时间服从负指数分布(negative exponential distribution);(4)内容路由器中的每个模块可以看作是一个 $M/M/1$ 排队系统;(5)兴趣包或数据包在任意两个模块间的转移概率是稳定的,因此我们可以使用开放排队网络(即 Jackson 网络)[10]对内容路由器进行理论分析.图 4 显示了基于开放排队网络的内容路由器数学模型.其中,CS 和 PIT 在 ingress 和 egress 两个方向上都有排队,而 FIB 只在 ingress 方向上存在排队.表 1 显示了数学建模过程中所用到的数学符号及相关说明.

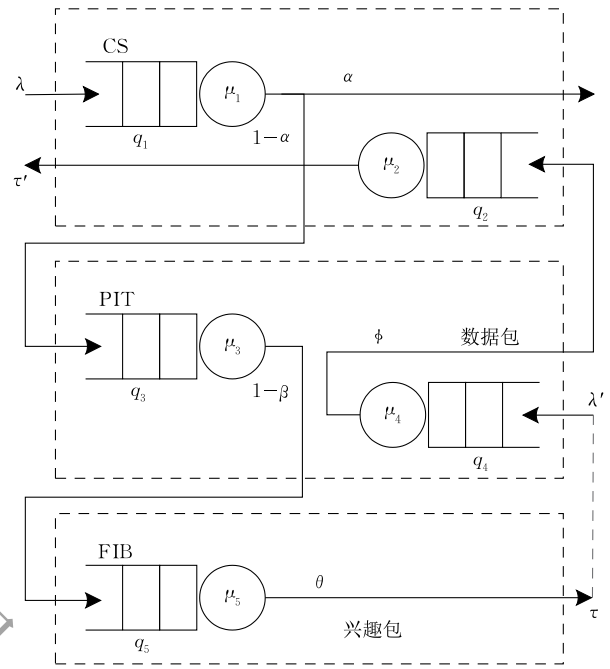

图 4 基于开放排队网络对内容路由器进行数学建模

表 1 内容路由器数学建模符号说明

符号	说明
λ	兴趣包的平均到达率
T_i	第 i 个排队子系统的平均吞吐率
μ_i	第 i 个服务器的平均服务率
α	兴趣包查找内容缓存(CS)的成功率
β	兴趣包查找待定兴趣表(PIT)的成功率
θ	兴趣包查找转发信息库(FIB)的成功率
τ	兴趣包的平均离开率
λ'	数据包的平均到达率
ϕ	数据包选择性写入内容缓存(CS)的概率
τ'	数据包的平均离开率
ρ_i	第 i 个服务器的利用率(忙碌时间占空比)

3.2 模型建立

根据 Jackson 网络的理论,当整个系统处于稳定状态时,每一个队列上的平均包到达率应该等于平均包离开率.假设从外部进入系统的兴趣包的平均到达率为 λ,那么在系统处于稳定状态时,对于 ingress 方向上的多个队列我们有以下定量关系

$$\begin{cases} T_1 = \lambda \\ T_3 = (1-\alpha)T_1 \\ T_5 = (1-\beta)T_3 \\ \tau = \theta T_5 \end{cases} \tag{1}$$

计算机学报

同样的，在内容路由器的 egress 方向上也存在以下定量关系

$$\begin{cases} T_4 = \lambda' \\ T_2 = \tau' = \phi T_4 \end{cases} \tag{2}$$

正如之前所述，当不考虑丢包时，内容路由器内部存在“流平衡”的现象，即 ingress 方向上朝上游转发的兴趣包总能与 egress 方向上从上游返回的数据包一一对应. 因此在系统处于稳定状态时，ingress 方向上的兴趣包离开率应该与 egress 方向上的数据包到达率大致相同，即我们有

$$\tau \approx \lambda' \tag{3}$$

根据式(1)、式(2)和式(3)，我们可以推导得到在系统处于稳定状态时每个队列处理包的平均吞吐率(即包到达率)如下

$$\begin{cases} T_1 = \lambda \\ T_2 = \phi\theta(1-\beta)(1-\alpha)\lambda \\ T_3 = (1-\alpha)\lambda \\ T_4 = \theta(1-\beta)(1-\alpha)\lambda \\ T_5 = (1-\beta)(1-\alpha)\lambda \end{cases} \tag{4}$$

3.3　性能瓶颈分析

根据排队论，每个服务器的利用率可以用任务的到达速率除以服务器自身的处理速率得到，且服务器的利用率不能够大于 1，否则会出现服务器过载及队列溢出的现象，即

$$\rho_i = \frac{T_i}{\mu_i} \leq 1 \tag{5}$$

当我们将式(4)代入到式(5)中可得到

$$\begin{cases} \lambda \leq \mu_1 \\ \lambda \leq \dfrac{\mu_2}{\phi\theta(1-\beta)(1-\alpha)} \\ \lambda \leq \dfrac{\mu_3}{(1-\alpha)} \\ \lambda \leq \dfrac{\mu_4}{\theta(1-\beta)(1-\alpha)} \\ \lambda \leq \dfrac{\mu_5}{(1-\beta)(1-\alpha)} \end{cases} \tag{6}$$

从式(6)中我们可以看到，整个内容路由器的最大吞吐率(即兴趣包的最大到达率)是由每个模块的自身处理能力(即 μ_i)和模块之间的流量转移概率(例如 α 和 β)共同决定的. 式(6)也可以等价的变换为以下公式，即内容路由器的最大吞吐率是由以下五个代数式的最小值决定

$$\max(T_1) = \max(\lambda) = \min\left\{\mu_1, \frac{\mu_2}{\phi\theta(1-\beta)(1-\alpha)}, \frac{\mu_3}{(1-\alpha)}, \frac{\mu_4}{\theta(1-\beta)(1-\alpha)}, \frac{\mu_5}{(1-\beta)(1-\alpha)}\right\} \tag{7}$$

显然，在式(7)中，流量在模块间的转移概率均小于 1，因此，ingress 方向上的 CS 模块必然是整个内容路由器的性能瓶颈，除非其处理能力(即 μ_1)足够高. 换句话说，当 μ_1 较小时，流水线上的 CS 模块内部会发生拥塞，而流水线上的其它模块，如 FIB 和 PIT，则会处于相对空闲的状态. 当且仅当式(6)中的五个不等式的等号同时成立时，流水线才会处于没有气泡的满负荷工作状态.

4　网络流量模型

考虑到目前较难获取内容中心网络的真实流量，且内容中心网络与 IP 网络的流量特征存在一定的共性，在这里我们对真实的 IP 骨干网流量进行测量，然后利用 IP 网络的相关流量特征推测论证内容中心网络的流量特征. 这些流量特征将有助于我们优化 CS 的数据结构. 我们从 CAIDA 上下载了 2014.3.20 当天的三段持续时间为 20 min 的 OC-192 骨干网流量，分别标记为 Chicago. A、Chicago. B 和 Sanjose. A. 它们的链路利用率分别为 14.646%、43.448% 和 37.883%.

4.1　时间局部性

时间局部性特征广泛存在于网络层(L3)和传输层(L4)的流量中[22]. 举例来说，在传输层中，良好的时间局部性通常意味着当前到达的数据包，预示着未来可能会到达与当前数据包属于同一个流(flow)的其它数据包. 测量结果显示，CAIDA 上获取的三段流量均具有良好的时间局部性. 尽管 Chicago. A、Chicago. B 和 Sanjose. A 的并发流数目大概在 0.6 M、1.3 M 和 2.5 M 的数量级(见图 5)，但根据图 6 结果显示，使用一个仅仅能够包含 4 K 条表项，采用直接映射(direct-mapped)方式组织的较小缓存就可以基

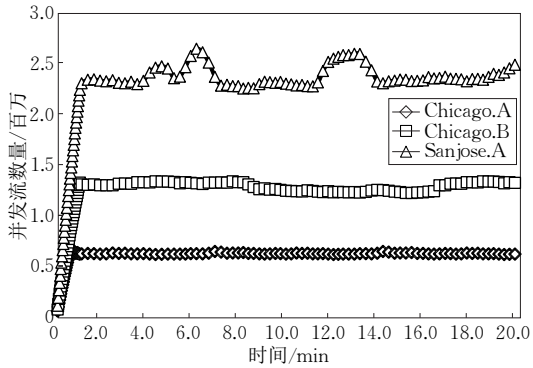

图 5　三个流量随时间变化的并发流数目

《计算机学报》

于三段骨干网流量分别取得高达 0.69、0.68 和 0.61 的缓存命中率.通常来说,我们习惯于引入快速缓存来提升具有良好时间局部性的流量的处理速度[23].

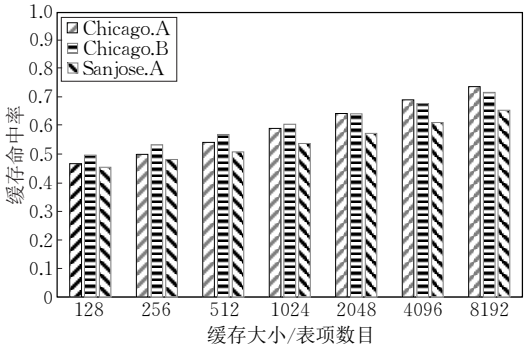

图 6 缓存命中率与缓存大小的关系

4.2 空间局部性

在计算机网络中,一个完整的文件(包含连续存储的数据)是以离散的数据包的形式传输的.在这里,我们定义的空间局部性是指当前到达的数据包预示着在不久的将来大概率到达另外一个封装有后续邻近数据的数据包.对于传输层(L4)流量来说,空间局部性的好坏程度可以用 TCP 的序列号来测度.假设一个 TCP 流中陆续到达的数据包的序列号为 seq_1 到 seq_n,那么我们可以定义这个流的数据包乱序率为 $Probability(seq_i > seq_{i+1})$,其中 i 属于 $[1,n-1]$.显然,流量的数据包乱序率越低,流量的空间局部性就越好.图 7 展示了三个流量的乱序率累积分布曲线.可以看到,三个流量中的数据包乱序率均非常小,属于同一个流的数据包大概率顺次到达,即骨干网流量具有良好的空间局部性.

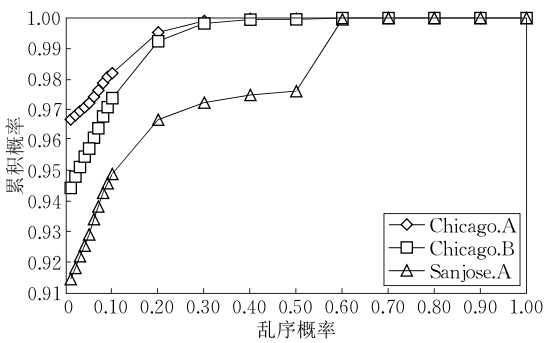

图 7 三个流量乱序概率的累积分布函数(CDF)

4.3 内容中心网络的流量特征

如前所述,在内容中心网络中,内容的名字通常由名字前缀(name prefix)和内容块序号(segment ID)两部分组成.类似于 IP 网络流量,内容中心网络中的名字前缀也大概率具备时间局部性和空间局部性的特征.举例来说,因为网络上的文件通常都是按照一定的次序连续存储或被连续访问的,因此当前到达内容路由器的请求内容名字为"google.com/maps/china/1"的兴趣包往往预示着在不久的将来还将出现请求内容名字为"google.com/maps/china/2"的兴趣包.在内容的获取方式上,内容中心网络和 IP 网络存在显著的不同:在 CCN 中内容请求可以在内容路由器上直接得到满足,而在 IP 网络中,内容请求只能在终端节点上被满足.CCN 特殊的内容获取方式将会在一定程度上影响网络的流量模型.对于一台特定的内容路由器来说,由于网络缓存的存在,连续到达的两个具有相同名字前缀的兴趣包有可能包含间隔相差很大的内容块序号,这是因为一部分兴趣包有可能在下游路由器的内容缓存中发生命中而直接被"吸收"了.另外,与 IP 网络类似,由于网络中存在包重传的现象,当前到达的兴趣包有可能包含比之前到达的具有相同名字前缀的兴趣包更小的内容块序号.以上两种边界情况均需要在算法设计时考虑到.

5 局部性原理跳表

5.1 核心思想

在经典跳表中,当每次有新的查找请求到来时,我们都会从跳表的头结点开始进行查找.然而,内容中心网络流量存在一定的空间局部性,即陆续到来的多个具有相同名字前缀的查找请求有可能包含邻近的内容块序号;另一方面,跳表本身是有序的.基于这种考虑,我们可以在跳表查找过程中记录下每次查找成功的结点位置,当有相同名字前缀的后续查找请求到达时,如果后续查找请求查找的是邻近的内容块,那么我们就可以从最近查找成功的跳表结点附近开始查找,而不需要重复的从跳表的头结点位置开始查找,从而有效降低跳表的查找开销.图 8 中展示了一个具体的例子,当搜索键值为 26 的跳表结点时,我们可以直接从跳表结点 17(最近查找的键值为 19 的跳表结点的附近结点)开始查找,而不是从跳表的头结点开始查找.

5.2 数据结构

为了实现上述局部性原理跳表的思想,我们需要引入一个额外的数据结构以记录每一个名字前缀下的查找请求最近访问的跳表结点信息.在经典跳表的数据结构中,每一个跳表结点包含一个{key,value}的映射配对和一组指向前向跳表结点(即链表中的后继结点)的指针 forward[].为了实现局部

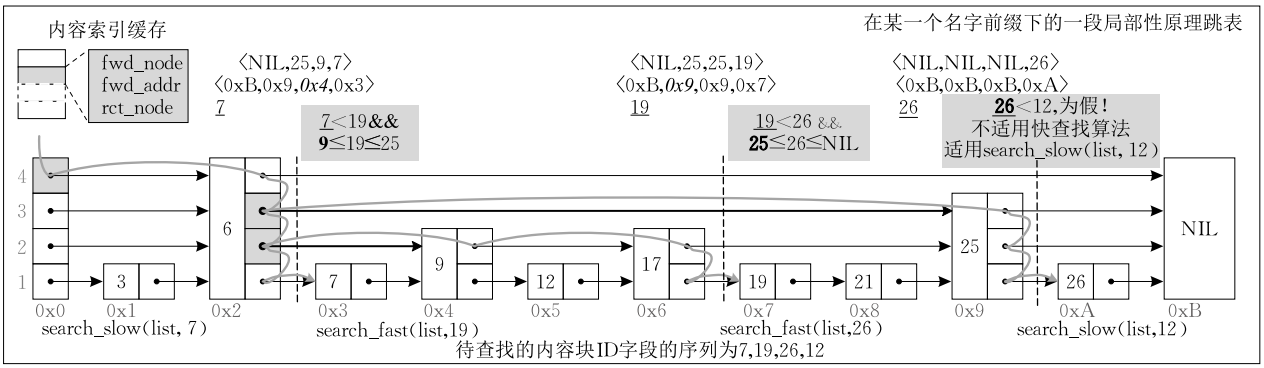

图 8　基于局部性原理跳表的查找过程

性原理跳表，我们在经典跳表的数据结构中再增加以下一些变量：对于一个最大层数为 L 的跳表 sk 来说，当我们在 sk 中查找键值为 t 的结点时，首先定义数组 fwd_node[i]（i 属于[1,L]），使得不等式 x->key<t≤fwd_node[i]=x->forward[i]->key 对于任意 i 属于[1,L]均成立（其中 x 是跳表中的某个结点，且使得上述不等式成立）. 此外，我们还定义 fwd_addr[i]为键值是 fwd_node[i]的结点的地址. 举例来说，在图 8 中，当查找元素的键值为 7 时，fwd_node[]的值为{NIL,25,9,7}，fwd_addr[]的值为{0xB,0x9,0x4,0x3}. 值得说明的是，fwd_node[]和 fwd_addr[]的值均可在查找键值为 7 的元素的过程中顺带计算得到，并不需要增加太多额外的计算负担. 事实上，对于某一个相同名字前缀下顺次到达的多个内容查找请求来说，如果我们能够在完成当前查找请求的同时记录下 fwd_node[]和 fwd_addr[]的数值，那么在相同名字前缀下的后续查找请求到达时，可以不必从头开始遍历查找跳表，而是可以从最近查找成功的跳表结点附近（即 fwd_addr[]数组中的某一个指针）开始继续向右或向下遍历查找，从而提高查找的效率. 也就是说，fwd_node[]和 fwd_addr[]中存储了在跳表中高效的处理后继查找请求的“路径指引信息”. 为了存储不同名字前缀下的“路径指引信息”，我们引入了内容索引缓存这一额外的数据结构. 内容索引缓存以名字前缀作为查找键值，表项中包含指针指向结点（键值）fwd_node[]、指针指向结点地址 fwd_addr[]和最近访问结点（键值）rct_node. 表 2 展示了内容索引缓存的数据结构. 考虑到网络流量存在较好的空间局部性，我们可以使用较小的内容索引缓存来取得较高的缓存命中率，尽量降低额外的存储开销. 当新的内容查找请求到达时，我们首先使用从兴趣包包头提取的名字前缀查找内容索引缓存，以获得该名字前缀下的查找请求最近访问的跳表结点信息.

表 2　内容索引缓存数据结构

名字前缀	指针指向结点	指针指向结点地址	最近访问结点
fnlab. net/	〈NIL,25,9,7〉	〈0xB,0x9,0x4,0x3〉	7
bupt. com/	〈NIL,36,8,3〉	〈0xB,0xA,0xB,0xE〉	3
…	…	…	…

5.3　算法描述

5.3.1　使用内容索引缓存加速查找

那么，如何利用内容索引缓存实现跳表中元素的快速查找呢？这里的关键在于从当前名字前缀的 fwd_addr[]数组中选择一个最优的指针进行后续内容请求的查找. 那么，如何定义最优的指针呢？我们认为从最优的指针出发，能够通过最少的“向右-向下”遍历操作找到目标结点. 举例来说，在图 8 中，当完成键值为 7 的结点的查找后，内容索引缓存中该名字前缀下 fwd_addr[]的值被更新为{0xB,0x9,0x4,0x3}. 当查找下一个键值为 19 的结点时，我们从当前 fwd_addr[]中的任意一个指针出发，均能够成功完成查找操作. 但是从不同的指针出发将带来不同的查找开销. 如果我们从指针 0x3 出发，那么查找路径为 7→9→12→17→19；如果我们从指针 0xB 出发，那么查找路径为 6→6→9→17→17→19；如果我们从指针 0x4 出发，那么查找路径为 9→17→17→19. 显然，从指针 0x4 出发的查找路径更优. 根据归纳总结，我们发现，当在跳表中查找键值为 t 的结点时，最优指针 fwd_addr[i]中 i 的取值取决于以下不等式：fwd_node[i]≤t≤fwd_node[i+1]. 例如，当查找下一个键值为 19 的结点时，fwd_node[2]=9≤19≤fwd_node[2+1]=25，因此 fwd_addr[2]=0x4 是此时的最优指针. 当确定 fwd_addr[]中的最优指针之后，我们就可以从该指针出发继续向右或向下遍历跳表查找后续目标结点. 当一个目标结点查找成功之后，我们也需要在内容索引缓存的相应表项

《计算机学报》

中更新 fwd_node[]、fwd_addr[]和 rct_node 的信息.算法 3 中展示了详细的局部性原理跳表快查找算法.在算法步骤中,cache[search_key]可以分解为两个步骤:首先,通过 extract_prefix()从内容请求中完整的内容名字 search_key 中提取名字前缀;然后通过哈希计算将该名字前缀映射为内容索引缓存上的偏移地址,从而实现对内容索引缓存的访问.

算法 3. 局部性原理跳表快查找算法.

输入:局部性原理跳表 *list*,查找元素的键值 *search_key*

输出:若成功返回键值为 *search_key* 的元素,否则返回失败

```
search_fast(list, search_key)
  //在跳表中找到最近访问结点的最优层级
  opt_level := list→level
FOR i := 1 UPTO list→level−1 DO
  IF cache[search_key]→fwd_node[i]≤search_key
    AND search_key≤cache[search_key]→fwd_
    node[i+1] THEN
    opt_level := i BREAK
  //从最近访问结点的最优层级开始进行查找
x := cache[search_key]→fwd_addr[opt_level]
FOR i := opt_level DOWNTO 1 DO
    WHILE x→forward[i]→key ⟨search_key DO
      x :=x→forward[i]
    //缓存本次查找过程中的前向指针信息
    cache[search_key]→fwd_node[i]:=x
                    →forward[i]→key
    cache[search_key]→fwd_addr[i]:=x
                    →forward[i]
  x := x→forward[1]
IF x→key=search_key THEN
    //缓存本次查找访问到的结点信息
    cache[search_key]→rct_node:=x→key RETURN x
ELSE
    cache[search_key]→rct_node:=NIL RETURN false
```

5.3.2 使用快查找算法的条件

算法 3 总是按照键值升序在跳表中遍历结点.而在同一个名字前缀下,当后继查找元素的内容块序号小于前驱查找元素的内容块序号时(由包重传导致),算法 3 将失效.在这种情况下,我们只能从头开始重新遍历跳表.为了与算法 3 中提到的快查找进行区分对比,我们将这个过程称为慢查找.根据图 7 的流量测量统计结果,发生包乱序的概率还是相对较低的.也就是说,相对较低频率的慢查找算法调用不会对局部性原理跳表的整体查找性能产生过大的影响.算法 4 显示了调用快查找算法或慢查找算法的条件,即内容索引缓存中最近查找结点的键值(rct_node)是否大于当前到达查找请求中目标结点的键值.若大于,则调用慢查找算法;否则调用快查找算法.算法 5 显示了慢查找算法的具体细节.事实上,慢查找算法(算法 5)和经典跳表的查找算法(算法 1)基本相同.区别仅仅在于在慢查找算法的执行过程中,还需要顺带维护 fwd_node[]、fwd_addr[]和 rct_node 的信息,为后续到达的查找请求做好准备.

算法 4. 局部性原理跳表条件查找算法.

输入:局部性原理跳表 *list*,查找元素的键值 *search_key*

输出:若成功返回键值为 *search_key* 的元素,否则返回失败

```
conditional_search(list, search_key)
IF cache[search_key]→rct_node≠NIL AND cache
        [search_key]→rct_node≤search_key THEN
  result := search_fast(list, search_key)
ELSE
  result := search_slow(list, search_key)
RETURN result
```

算法 5. 局部性原理跳表慢查找算法.

输入:局部性原理跳表 *list*,查找元素的键值 *search_key*

输出:若成功返回键值为 *search_key* 的元素,否则返回失败

```
search_slow(list, search_key)
  x := list→header
FOR i := list→level DOWNTO 1 DO
    WHILE x→forward[i]→key <search_key DO
      x := x→forward[i]
    //缓存本次查找过程中的前向指针信息
    cache[search_key]→fwd_node[i]:= x
                    →forward[i]→key
    cache[search_key]→fwd_addr[i]:= x
                    →forward[i]
  x := x→forward[1]
IF x→key=search_key THEN
    //缓存本次查找访问到的结点信息
    cache[search_key]→rct_node := x
                    →key RETURN x
ELSE
    cache[search_key]→rct_node:=NIL RETURN failure
```

5.3.3 插入算法

在局部性原理跳表中插入一个新结点时将会引起跳表和内容索引缓存的缓存一致性问题(cache coherence issue).为了解决该问题,在当前的设计中,每当跳表中插入或删除新结点时,我们会在内容索引缓存中找到对应名字前缀的表项,然后将该表项设置为无效(invalidated).下次有该名字前缀下的查找请求到达时,在查询内容索引缓存失败之后,将从头开始遍历查找跳表.

《计算机学报》

算法 6.　局部性原理跳表的插入算法.

```
输入:局部性原理跳表 list,插入元素键值 search_key
输出:将该元素成功插入跳表,作废索引缓存中的无效表项
insert_and_invalidate(list,search_key)
  x := list→header
FOR i := list→level DOWNTO 1 DO
    WHILE x→forward[i]→key<search_key DO
      x := x→forward[i]
    update[i]=x
  x := x→forward[1]
IF x→key≠search_key THEN
    new_level := random_level()
    IF new_level >list→level THEN
    FOR i := list→level+1 TO new_level DO
        update[i] := list→header
      list→level := new_level
    x := alloc_node(new_level, search_key)
    FOR i := 1 TO new_level DO
      x→forward[i] := update[i]→forward[i]
      update[i]→forward[i] := x
    //将内容索引缓存中以 search_key 为键值的表项作废
    invalidate(cache[search_key])
```

6　实验与性能评测

6.1　实验方法

6.1.1　流量的产生

由于获取内容中心网络真实流量难度较高，在本文中我们使用四个人工合成的流量(Rand、Seq、Seq+Int(m)、Seq+Int(m)+Bk(n))进行实验. 每个合成流量都包含 1 000 000 个数据包和 10 000 个不同的名字前缀. 每个数据包请求的内容名字包含类似“[url]/[id]”的格式，其中名字前缀 url 是从 Alexa 网站上统计得到的前 1M 个用户访问量最大的域名中选取的. 而每个名字前缀 url 下生成的数据包个数符合偏离度(Zipf skew)为 0.5 的 Zipf 分布. 偏离度反映了不同名字前缀下数据包个数的差异程度，如果偏离度为 0，则 Zipf 分布将会蜕化为均匀分布. 在流量 Rand 中，相同名字前缀下到达的数据包的 id 编号以随机的方式出现；在流量 Seq 中，相同名字前缀下先后到达的数据包的 id 编号以升序出现，即 $id_{i+1}-id_i=1$；在流量 $Seq+Int(m)$ 中，相同名字前缀下先后到达的数据包的 id 编号以升序出现，且先后到达的两个数据包的 id 编号之差不总为 1 且不大于 m，即 $id_{i+1}-id_i\leq m$(这类测试用例是用来模拟在 CCN 网络中由于下游路由器具备缓存能力而导致在某一台上游路由器上可能出现的内容数据块 id 查找间隔的情况)；在流量 Seq+Int(m)+Bk(n)中，除了上述提到的内容数据块查找间隔的问题，我们在流量中还加入了网络重传的可能性，即 $0\leq id_i-id_{i+1}\leq m$ 将以 n 的概率出现在人工合成流量中. 显然，在上述四种人工合成流量中，Rand 的空间局部性最差；Seq+Int(m)+Bk(n)次之；Seq+Int(m)具有较好的空间局部性；Seq 具有最好的空间局部性.

6.1.2　软硬件参数

我们使用一台 x86 架构的笔记本作为实验平台. 该笔记本包含主频为 2.2 GHz 的 i5-5200u 处理器、3 MB 的 L3 缓存、8 GB 的 DDR3 DRAM，并运行内核版本号为 4.2.6 的 Linux 操作系统. 为了方便性能对比，我们分别实现了经典跳表和局部性原理跳表. 两种跳表均容纳有 1M 的元素，跳表最大层数均设置为 20 层. 为了降低实现复杂度，在设计内容名字前缀的数据结构时，我们并没有采用额外的名字编码技术，而是直接使用字符数组 char[]存储内容名字前缀，并使用 strcmp()函数在跳表遍历时进行键值比较. 我们基于比较简单的直接映射(direct-mapped)方式构建内容索引缓存. 默认的缓存大小设置为包含 64 K 的表项. 当对内容索引缓存进行查找时，我们采用 djb2[24] 作为哈希函数将名字前缀字符串映射为一个整数，用来表示内容索引缓存上的偏移地址. 具体而言，djb2 的原理是对一个初始值按照预先定义的算法反复计算 m 次得到最终的哈希值，而 m 为字符串的长度. 此外，我们使用 GNU profiler (gprof)工具对程序性能进行分析. 分析结果将指引程序性能的进一步优化.

6.2　性能分析

6.2.1　查找开销

表 3 展示了经典跳表和局部性原理跳表的平均查找开销. 在这里，我们使用查找跳表时向右和向下两种操作出现的次数来定义跳表的查找开销. 也就是说，查找开销的大小仅与查找算法本身有关，与机器性能无关. 我们可以看到，由于测试流量具备一定的空间局部性，局部性原理跳表可以利用这一特征大大降低平均查找开销. 当然，考虑到路由器自带缓存和网络重传将或多或少影响到流量的空间局部性，因此对于局部性原理跳表的实验结果来说，Seq+Int(5)和 Seq+Int(5)+Bk(0.1)的平均查找开销比 Seq 更大. 而在同样使用 Seq 的情况下，局部性原理跳表的平均查找开销几乎是经典跳表的十分之一. 我们注意到，对于经典跳表来说，平均的向下查找次数永远是 20. 这是因为每次查找都是从第 20 层开始，在第 1 层以查找成功或失败结束.

《计算机学报》

表 3 两种算法的平均查找开销(包括向右和向下)对比

流量类型	经典跳表		局部性原理跳表	
	向右	向下	向右	向下
Rand	20.072	20.000	12.486	13.639
Seq	20.848	20.000	2.246	3.221
Seq+Int(5)	19.395	20.000	3.997	4.879
Seq+Int(5)+Bk(0.1)	19.736	20.000	5.680	6.302

6.2.2 查找时延和吞吐率

表 4 展示了经典跳表和局部性原理跳表在实验所用 x86 机器上的平均数据包查找时延.我们可以看到,对于流量 Rand 来说,局部性原理跳表相比经典跳表并没有更大的优势,这是因为随机到达的内容块序号将会导致较慢的 search_slow()函数的频繁调用.对于局部性最好的流量 Seq 来说,局部性原理跳表相比经典跳表能够取得 3 倍的加速比.当然,上述在真实物理机器上取得的加速比并没有表 3 中算法取得的接近 10 倍的加速比那么显著,这里可能的原因有以下两个:(1)局部性原理跳表引入了内容索引缓存数据结构,对该数据结构的操作将会引起额外的性能开销;(2)基于 x86 架构的通用处理器拥有比较大的片上缓存(如 3MB 的 L3 缓存),其本身也能充分挖掘流量潜在的局部性,这将大大削弱局部性原理跳表带来的优势.当我们在诸如 ASIC 或 FPGA 等更高并行度的硬件上实现 CS 时,局部性原理跳表将有望取得更高的性能加速比.

表 4 两种算法的平均查找时延的对比

流量类型	经典跳表/ns	局部性原理跳表/ns
Rand	2819.895	2641.094
Seq	1822.657	556.629
Seq+Int(5)	2199.895	975.904
Seq+Int(5)+Bk(0.1)	1970.067	1085.113

表 5 显示了两种数据结构的平均查找吞吐率对比.在 i5-5200u 的 x86 处理器上,局部性原理跳表可以取得 1.796 Mpps 的单线程查找吞吐率,大大超过经典跳表 0.548 Mpps 的单线程查找吞吐率.假设数据包的平均包长为 500 B,则理论上局部性原理跳表可以取得 7.016 Gbps 的单线程处理能力.

表 5 两种算法的平均查找吞吐率的对比

流量类型	经典跳表/Mpps	局部性原理跳表/Mpps
Rand	0.354	0.378
Seq	0.548	1.796
Seq+Int(5)	0.454	1.024
Seq+Int(5)+Bk(0.1)	0.507	0.921

6.2.3 插入耗时

图 9 显示了当插入数据包个数不同时,局部性原理跳表与经典跳表插入过程所消耗的时间对比.由于插入算法并没有针对局部性原理跳表的插入流程作过多的细节优化,而仅仅对内容索引缓存中无效表项发起作废操作以维护内容索引缓存与跳表间的数据一致性.因此局部性原理跳表的插入算法会比经典跳表的插入算法(算法 2)效率略低.两者性能上的差距主要体现在维护内容索引缓存一致性时的额外写缓存开销.

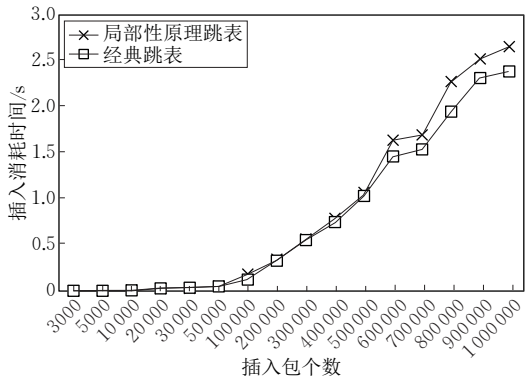

图 9 局部性原理跳表与经典跳表插入性能比较

6.2.4 基于 gprof 的性能分析

为了更好的对程序性能进行优化,我们使用 GNU profiler (gprof)对局部性原理跳表处理流量 Seq 时的程序执行过程进行剖析.性能分析的结果如表 6 所示.其中,耗时占比表示不同函数消耗程序执行总时间的比例;耗时总量表示不同函数执行的总时间;调用次数表示不同函数执行的次数;时延表示不同函数的单次执行时间.从表 6 中我们可以看到,search_fast()函数的时延非常小.而其它辅助函数,例如 search_slow()、hash()、extract_prefx()的频繁调用将会影响程序的性能.为了使这些辅助函数执行的更快或者执行的次数更少,在未来的工作中,我们可以通过设计更快的哈希函数、使用名字编码技术或者提高内容索引缓存命中率等手段进一步提高局部性原理跳表程序的执行效率.

表 6 使用 gprof 对局部性原理跳表查找算法进行性能分析

函数名	耗时占比/%	耗时总量/s	调用次数	时延/μs
search_fast	43.6	0.24	949312	0.25
hash	16.4	0.09	1000000	0.09
search_slow	10.9	0.06	50688	1.09
extract_prefix	9.1	0.05	1000000	0.05
others	20.0	0.11	未统计	未统计

6.2.5 参数灵敏度评估

我们对内容索引缓存大小、最大查找间隔(Int)和最大回退概率(Bk)这三个重要参数进行灵敏度

分析，以评估它们对跳表查找性能的影响程度.

图 10 显示了查找吞吐率和内容索引缓存大小之间的关系. 可以看到，当测试流量为 Rand 时，查找吞吐率并不会随着内容索引缓存大小的变化而剧烈变化. 而当测试流量的局部性较好时，更大的内容索引缓存将有助于提高程序的性能.

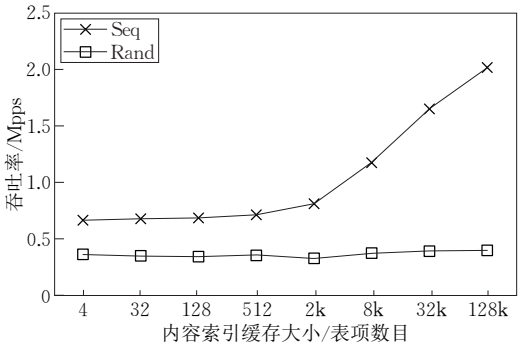

图 10　吞吐率和内容索引缓存大小的关系

图 11 显示了查找吞吐率和最大查找间隔(Int)之间的关系. 可以看到，当最大查找间隔增大时，查找吞吐率将有所降低. 这是因为当最大查找间隔变大时，虽然并不需要反复从跳表头结点开始查找，但每完成一次跳表查找过程将会触发更多的向右和向下遍历操作，进而消耗更多的查找时间.

图 12 显示了查找吞吐率和最大回退概率(Bk)之间的关系. 可以看出，当最大回退概率变大时，查找吞吐率将有所降低. 根据算法 4 中提到的启用快查找算法和慢查找算法的条件可知，当回退概率变大时，慢查找算法 search_slow()将会以更大频率被调用，进而降低整个查找过程的效率.

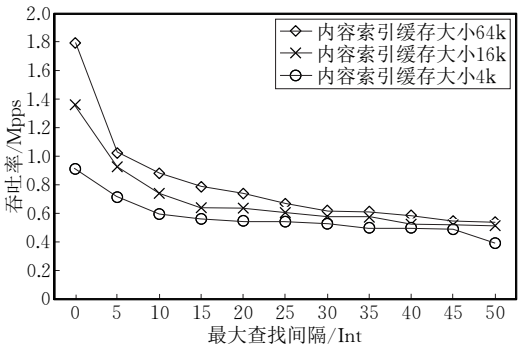

图 11　吞吐率和最大查找间隔(Int)的关系

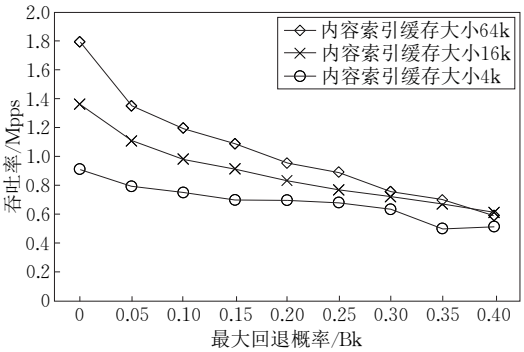

图 12　吞吐率和最大回退概率(Bk)的关系

7　总结与展望

在本文中，我们首先基于开放排队网络的数学模型对内容路由器的体系结构进行定量分析. 通过模型分析，我们发现内容缓存模块(CS)是内容路由器真正的性能瓶颈. 然后，我们利用网络流量中广泛存在的时间局部性和空间局部性特征，设计了局部性原理跳表作为内容路由器中内容缓存模块的底层数据结构，并分别实现了基于局部性原理跳表的查找和更新算法. 在新的设计中，考虑到新到达的兴趣包可能与之前的兴趣包共享相同的名字前缀，且所请求的内容块通常处于邻近的存储空间，因此在查找跳表的过程中可以直接从该前缀先前命中的跳表结点附近开始查找. 这样的设计可以避免每次查找都从跳表头结点开始的重复开销，从而保证查找过程的高效性. 实验结果表明，局部性原理跳表可以在 x86 处理器上实现 1.796 Mpps 的单线程吞吐率，这是经典跳表查找速度的三倍. 对于 500 B 大小的数据包来说，CS 的单线程吞吐率可以高达 7.016 Gbps. 我们认为本方案可以为未来内容路由器中内容缓存的设计提供一定的参考和依据.

在后续的工作中，我们将利用多核处理器对已有工作进行并行化，基于多线程实现局部性原理跳表，探索线程之间的同步和互斥机制，进一步提高 CS 以及整个内容路由器的处理性能.

致　谢　审稿专家对论文提出了宝贵意见，在此表示衷心的感谢！

参　考　文　献

[1] Choi J, Han J, Cho E, et al. A survey on content-oriented networking for efficient content delivery. IEEE Communications Magazine, 2011, 49(3): 121-127

[2] Saroiu S, Gummadi K P, Dunn R J, et al. An analysis of internet content delivery systems. ACM SIGOPS Operating Systems Review, 2002, 36(SI): 315-327

[3] Liu Y, Guo Y, Liang C. A survey on peer-to-peer video streaming systems. Peer-to-Peer Networking and Applications, 2008, 1(1): 18-28

[4] Tran D A, Hua K A, Do T T. A peer-to-peer architecture for media streaming. IEEE Journal on Selected Areas in Communications, 2004, 22(1): 121-133

[5] Jacobson V, Smetters D K, Thornton J D, et al. Networking named content//Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies—CoNEXT 2009. Rome, Italy, 2009: 1-12

[6] Zhang L, Afanasyev A, Burke J, et al. Named data networking. ACM SIGCOMM Computer Communication Review, 2014, 44(3): 66-73

[7] Zhang L, Estrin D, Burke J, et al. Named data networking (NDN) project. Relatório Técnico NDN-0001, Xerox Palo Alto Research Center-PARC, 2010

[8] Saltzer J H, Reed D P, Clark D D. End-to-end arguments in system design. ACM Transactions on Computer Systems, 1984, 2(4): 277-288

[9] McKusick M K, Joy W N, Leffler S J, Fabry R S. A fast file system for UNIX. ACM Transactions on Computer Systems, 1984, 2(3): 181-197

[10] Jackson J R. Jobshop-like queueing systems. Management Science, 1963, 10(1): 131-142

[11] Afanasyev A, Shi J, Zhang B, et al. NFD developer's guide. Department of Computing Science, University of California, Los Angeles, USA, Technology Report: NDN-0021, 2014

[12] Pugh W. Skip lists: A probabilistic alternative to balanced trees. Communications of the ACM, 1990, 33(6): 668-676

[13] Wang Y, Zu Y, Zhang T, et al. Wire speed name lookup: A GPU-based approach//Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation—NSDI 2013. Lombard, USA, 2013: 199-212

[14] Wang Y, Pan T, Mi Z, et al. Namefilter: Achieving fast name lookup with low memory cost via applying two-stage bloom filters//Proceedings of the 32nd IEEE International Conference on Computer Communications—INFOCOM 2013. Turin, Italy, 2013: 95-99

[15] So W, Narayanan A, Oran D. Named data networking on a router: Fast and DoS-resistant forwarding with hash tables//Proceedings of the 9th ACM/IEEE Symposium on Architectures for Networking and Communications Systems—ANCS 2013. San Jose, USA, 2013: 215-226

[16] Song T, Yuan H, Crowley P, Zhang B. Scalable name-based packet forwarding: From millions to billions//Proceedings of the 2nd International Conference on Information-centric Networking—ICN 2015. San Francisco, USA, 2015: 19-28

[17] Dai H, Liu B, Chen Y, Wang Y. On pending interest table in named data networking//Proceedings of the 8th ACM/IEEE Symposium on Architectures for Networking and Communications Systems—ANCS 2012. Austin, USA, 2012: 211-222

[18] Yuan H, Crowley P. Scalable pending interest table design: From principles to practice//Proceedings of the 33rd IEEE International Conference on Computer Communications—INFOCOM 2014. Toronto, Canada, 2014: 2049-2057

[19] Varvello M, Perino D, Linguaglossa L. On the design and implementation of a wire-speed pending interest table//Proceedings of the 32nd IEEE International Conference on Computer Communications Workshops—INFOCOM WKSHPS2013. Turin, Italy, 2013: 369-374

[20] You W, Mathieu B, Truong P, et al. Dipit: A distributed bloom-filter based pit table for ccn nodes//Proceedings of the 21st International Conference on Computer Communication Networks—ICCCN 2012. Munich, Germany, 2012: 1-7

[21] Mansilha R B, Saino L, Barcellos M P, et al. Hierarchical content stores in high-speed ICN routers: Emulation and prototype implementation//Proceedings of the 2nd International Conference on Information-centric Networking—ICN 2015. San Francisco, USA, 2015: 59-68

[22] Almeida V, Bestavros A, Crovella M, De Oliveira A. Characterizing reference locality in the WWW//Proceedings of the 4th International Conference on Parallel and Distributed Information Systems. Miami, USA, 1996: 92-103

[23] Pan T, Guo X, Zhang C, et al. Tracking millions of flows in high speed networks for application identification//Proceedings of the 31st IEEE International Conference on Computer Communications—INFOCOM 2012. Orlando, USA, 2012: 1647-1655

[24] Roussev V, Richard G G, Marziale L. Multi-resolution similarity hashing. Digital Investigation, 2007, 4: 105-113

PAN Tian, born in 1987, Ph. D., lecturer. His research interests include content-centric networking, software-defined networking, satellite networking, high-speed programmable network system, etc.

HUANG Tao, born in 1980, Ph. D., professor. His research interests include future network architecture, software-defined networking, content-centric networking, etc.

ZHANG Xue-Bei, born in 1993, M. S. candidate. Her research interests include content-centric networking, satellite networking, etc.

《计算机学报》

Background

In this work, we focus on the performance issues of content router's forwarding plane. Content routers are the underlying infrastructure of the content-centric networking (CCN), which is proposed as a promising future Internet architecture. In contrast to conventional IP-based, host-oriented Internet architecture, CCN emphasizes content by making it directly addressable and routable and end hosts communicate based on named content instead of IP addresses. Actually, in CCN, the network content can be replicated anywhere due to the in-network caching capability provided by content routers. Although this architectural innovation gains advantages of optimal content dissemination, it does impair the classic "end-to-end" principle by pushing additional states into the intermediary nodes of the network. Compared with the stateless IP forwarding model, CCN packet forwarding is far more sophisticated. There are three major components in a content router, namely, forwarding information base (FIB), pending interest table (PIT) and content store (CS). Due to the performance issues in content routers such as longest prefix match in an unbounded name space, expensive memory cost for explosive routing states, frequent update of pending states, lots of recent works focus on the performance issues of FIB and PIT. However, CS seems to be unintentionally ignored by the research community. Since FIB, PIT and CS are organized into a three-stage pipeline while generally, a pipeline runs only as fast as its slowest stage, to improve the performance of the entire content router, we should identify the performance bottleneck first and then, eliminate such a bottleneck by a novel design. In this work, to prevent "blind optimization", we first build a mathematical model which identifies CS as content router's performance bottleneck. Inspired by the extensively existed temporal and spatial locality in content retrieval, we propose locality-aware skip list to improve CS's performance. In our design, since an incoming packet is likely to share a prefix with previous packets, the search of the packet in a skip list could start directly from near the previously reached node of that prefix. This avoids searches repeatedly starting from the head node in the original skip list thus can save considerable search time. Extensive evaluation shows that our design can achieve a 3x speed up of the original design on an x86 machine. The mathematical model as well as the initial proposal of CS design can potentially shift research interests from previously widely-exploited FIB and PIT to CS, which is unintentionally ignored, but the actual performance bottleneck.

This research is supported by the National High Technology Research and Development Program of China (863 Program) (No. 2015AA016101), the National Natural Science Foundation of China (Nos. 61501042, 61702049), the China Postdoctoral Science Foundation (No. 2016M590068), and the Beijing Nova Program (No. Z151100000315078).

《计算机学报》