论文

PMST: 一种面向高性能 IP 查找的双向平衡流水线结构

李彦彪^①, 张大方^{①*}, 黄昆^②, 何大成^①, 曾彬^③^① 湖南大学信息科学与工程学院, 长沙 410082^② 中国科学院计算技术研究所, 北京 100190^③ 中国移动通信集团湖南公司, 长沙 410015

* 通信作者. E-mail: dfzhang@hnu.edu.cn

收稿日期: 2014-09-25; 接受日期: 2015-01-03; 网络出版日期: 2015-06-17

国家重点基础研究发展计划 (973 计划) (批准号: 2012CB315805)、国家自然科学基金 (批准号: 61173167, 61100171) 和湖南省研究生科研创新项目 (批准号: CX2014B150) 资助

摘要 随着网络技术的高速发展及网络应用的日趋多样化, 作为路由器的一项核心技术, IP 查找在吞吐率、存储效率以及更新性能等诸多方面都面临着严峻的挑战. 流水线技术的引入, 使 IP 查找的吞吐率获得了显著提升. 但是, 不平衡的结构不仅会导致低存储效率和高更新开销, 对查找性能以及多流水架构的负载均衡也会产生一定的影响. 而目前针对流水线进行平衡优化的工作, 又会带来一些不容忽视的新问题, 制约了其在 IPv6 或者大规模数据集下的应用. 鉴于此, 本文提出了一种双向平衡的线性流水线结构 — 流水化的多步长拆分特里树 (pipelined multi-bit split Trie, PMST). 通过拆分前缀, 旋转子树以及一系列平衡优化, PMST 仅需要很少的流水级就能获得理想的平衡度, 同时还能实现综合性能的提升. 我们采用真实路由器中的 IPv4/IPv6 数据集以及按一定规则产生的大规模 IPv6 数据集对 PMST 进行了全面的实验评估. 结果表明, 与现有优秀成果相比, PMST 在获得同等理想的平衡度时对流水级的需求下降了 75%~85.7%. 同时, 在流水线延时、单次查找的平均访存、片上存储效率、更新开销以及多流水架构的负载均衡等方面 PMST 都表现出明显的优势. 因此, PMST 具有更高的综合性能和良好的可扩展性, 能更好地适应目前和未来的应用需求.

关键词 包转发 流水线 架构 算法 存储高效 平衡 特里树

1 引言

IP 查找是路由器转发和过滤数据包过程中的一项核心技术, 即根据数据包的目的 IP 地址, 在转发信息表 (forwarding information base, FIB) 中查找与之对应的规则来处理该数据包. 随着网络技术的迅猛发展和广泛应用, 网络规模与日俱增. 在骨干网上, 通信链路的速度已超过 40 Gbps, 一些网络设备制造商 (如 Cisco 等) 也早已在其产品中部署了 100 Gbps 的线卡. 通常情况下, 100 Gbps 的吞吐率需要至少 1.5 亿次/秒的查找速度, 这就对 IP 查找的性能提出了很高的要求. 另一方面, 核心路由器的规模已经超过 400 k¹⁾ 并且还在以平均每年 2.5 万 ~5 万条路由规则的速度增长^[1], 再加上 IPv6

1) BGP Routing Table Analysis Reports [Online]. Available: <http://www.bgp.potaroo.net>.

的发展使 IP 地址的长度从 32 位扩展到了 128 位, 这些都给 IP 查找的存储性能和可扩展性带来了严峻的挑战. 此外, 虚拟化路由器^[2], 软件定义网络²⁾和命名数据网络³⁾等新兴网络技术及未来网络架构的提出与发展也对 IP 查找提出了新的需求.

基于无类型域间路由技术 (CIDR), IP 查找可转化为一个最长前缀匹配 (longest prefix matching, LPM) 问题. 其主流解决方案大致可分为 3 类: (1) 基于三态内容可寻址存储器 (ternary content addressable memory, TCAM) 的解决方案. 这类方案可以在一个时钟周期内完成对所有条目的查询, 具有非常高的性能^[3,4], 但是 TCAM 价格昂贵、能耗较大, 需要维护转发规则的有序性又导致了其更新瓶颈^[5], 这些缺陷使得这类方案难以直接用于大规模路由器或者更新频繁的应用场景^[6]. (2) 基于哈希 (Hash) 的解决方案. 这类方案依托先进的哈希技术, 能实现方便、快捷的查询和维护^[7~9]. 但是由哈希冲突和假阳性带来的问题严重制约了其实际应用. (3) 基于特里树 (Trie) 的解决方案. 这类方案因其结构和算法的灵活多变, 倍受青睐^[10~14]. 虽然已经有很多技术能将特里树压缩得非常紧凑^[11~13], 基于特里树的 IP 查找算法依然需要多次访存来完成一次查找, 导致了其性能瓶颈. 流水线技术的引入, 很好地解决了这个问题. 相关研究成果表明, 将特里树映射为基于静态随机访问存储器 (static random access memory, SRAM) 的流水线结构, 甚至能获得比 TCAM 还高的吞吐率^[15], 且支持快速增量更新^[16,17].

然而, 在流水线结构中, 不平衡的存储分布, 不仅会导致低存储效率和高更新开销, 对查找性能和多流水架构的负载均衡也会产生一定影响. 因此, 平衡优化成为研究这类技术的一个关键问题. Baboescu 等^[18]提出将特里树映射为一个多入口的环形流水线以提高平衡度, 但由于最坏情况下需要扫描流水线两次才能完成一次查找, 其吞吐率下降为平均每时钟周期 0.5 次查询. Kumar 等^[19]为环形流水线设置了多个出口, 虽然成功将吞吐率改进为平均每时钟周期 0.8 次查询, 但多个出口的设计使流水线延时⁴⁾变得波动不稳定, 且处理过的数据包可能出现乱序. 虽然这些非线性的流水线结构具有比较理想的平衡度, 但是带来的新问题也不容忽视.

为构建平衡的线性流水线, Jiang 等^[20]设计了一系列巧妙的算法, 将特里树映射为优化的线性流水线 (optimized liner pipeline, OLP) 结构, 并且可依据子树的划分部署多条流水线以获得更高的吞吐率^[15,21], 或者通过调整部分子树的映射方向进一步提升整体的平衡度^[22]. 但是这些结构需要较多的流水级⁵⁾才能获得理想的平衡度, 这就增大了流水线深度和流水线延时⁶⁾, 也会带来更大的存储开销⁷⁾. 因此, 这些方法难以适用于 IPv6 或者大规模数据集.

为更好的支持 IPv6 和提升存储效率, Hanna 等^[23]通过回溯的方法将特里树中所有与子孙节点具有相同下一跳信息的节点与其子孙节点压缩为一个节点, 再逐层映射为线性流水线. 而 Le 等^[24]则通过设置跳转距离压缩路径的方式来优化 OLP, 去除了大量无用的中间节点. 虽然这两种方法都很好地解决了流水线深度和存储效率的问题以提供对 IPv6 更好的支持, 但是过于紧凑的结构又增加了更新的开销. 此外, Erdem 等^[25]将改进的特里树映射为二维的多向流水线, 不仅提升了吞吐率, 降低了平均延时, 也能很好的支持 IPv6. 但是其非线性的结构依然无法提供稳定延时, 最坏情况下的延时还是很大 (不过其实实验表明出现最坏情况的可能性比较低). 而 Tong 等^[26]更是将流水线划分成不同的层次, 通过不同的结构和技术来进行优化, 其实实验结果表明, 该结构针对现有的 IPv6 数据集具有非常

2) Software-defined networking (SDN) [On line]. Available: <https://www.opennetworking.org>.

3) Named Data Networking (NDN). [On line]. Available: <http://www.named-data.net>.

4) 流水线延时即一次查询从进入流水线到从流水线输出查询结果经过的时钟周期数.

5) 基于 SRAM 的流水线结构通常是由多个独立的 SRAM 模块串联而成, 这些独立的 SRAM 模块即流水级.

6) 流水线深度即流水线包含流水级的总数, 在线性流水线中, 流水线的延时稳定为流水线的深度.

7) 基于 SRAM 的流水线其存储开销由最大流水级决定, 即最大流水级的存储开销与流水线深度的乘积.

Idx.	Prefix	Next hop
P ₁	0	N ₁
P ₂	1 1	N ₃
P ₃	0 0 1	N ₂
P ₄	0 1 1	N ₆
P ₅	1 1 0	N ₅
P ₆	1 1 1 1	N ₆
P ₇	0 1 1 0 0	N ₇
P ₈	1 0 1 0 0	N ₂
P ₉	0 1 1 1 0 1	N ₄
P ₁₀	0 0 0 1 1 1	N ₅
P ₁₁	0 1 1 0 0 0 1	N ₄
P ₁₂	1 1 1 0 0 0 1	N ₁
P ₁₃	1 1 1 0 0 1 0	N ₇

图 1 转发信息表
Figure 1 FIB

图 2 单步长特里树
Figure 2 Uni-bit Trie

图 3 多步长特里树

Figure 3 Multi-bit Trie

高的存储效率,但是并未讨论对 IPv4 或者大规模的 IPv6 数据集的支持情况 (目前真实世界的 IPv6 数据集规模还非常小).

此外,近年来,国内外研究学者针对基于特里树的流水线结构还提出了很多优化技术,比如通过缓存技术来提高吞吐率 [16,21,22,25]、借助多区间搜索树 [27,28] 或者更灵活的分层技术 [29] 来提升平衡度和存储效率,以及针对负载均衡 [30,15] 和能耗 [31] 等方面的优化等. 或者是针对某些特定的应用场景 (比如虚拟化路由器 [32]) 并结合数据和结构的特点来优化流水线的部署 [33,34]. 总的来说,他们各有优劣,但受结构限制,都难以实现综合性能的全面提升.

本文着眼于结构优化, 以提升综合性能. 我们首先以前缀拆分和子树旋转为基础, 提出了一种存储高效、平衡的双向线性流水线结构 — 流水线化的多步长拆分特里树 (PMST), 然后定义了流水线的平衡度参数并设计了一系列平衡优化算法, 使 PMST 仅需很少的流水级即能获得理想的平衡度. 最后, 我们采用真实路由器中的 IPv4/IPv6 数据集以及按一定规则^[35]生成的大规模 IPv6 数据集对 PMST 的综合性能进行了实验评估和简要分析. 结果表明, PMST 不仅保留了线性流水线能提供稳定延时和保持数据包顺序等基本特性, 同时还具有更高的综合性能和良好的可扩展性.

2 背景及相关工作介绍

如图 2 所示, 根据转发信息表 (如图 1 所示) 构建的单步长⁸⁾特里树就是一棵二叉树, 在其上进行 IP 查找只需要依次读入目的 IP 地址的每一位, 获得从根节点到某个叶子节点的一条路径, 此路径上包含的最长前缀所对应的下一跳信息即为所求. 为减少每次查询的访存次数, Srinivasan 等^[10]提出了可控前缀扩展算法来构建多步长特里树 (图 3), 其核心思想是以极小化存储开销为目标, 通过动态规划来计算最优步长数组, 再通过叶推算法将所有前缀信息推到叶子节点以压缩存储并简化查询.

将特里树的每一层映射为一个流水级即可构建简单的线性流水线。由于针对不同流水级 (SRAM 模块) 的访存可同时进行, 故多个查询请求产生的访存互相重叠, IP 查找的吞吐率将提升为平均每时钟周期完成一次查询。Basu 等^[16]基于不同流水级的并行访存设计了一种高效的更新机制 (write

8) 步长即在特里树上进行 IP 查找时每一步读入目的地址的比特数.

bubble), 通过线下更新产生一系列 (流水级, 偏移位置, 数值) 三元组, 再将针对不同流水级的三元组合并为一个称为 bubble 的结构. 然后利用查询的间隙将产生的 bubble 逐个部署到流水线上, 完成线上更新. 此外, 考虑到最大流水级将成为更新和存储的瓶颈, 他们还改进了构建多步长特里树时计算最优步长数组的动态规划算法, 以极小化最大级为目标, 提出了 MinMax 算法来构建流水化的多步长特里树 (pipelined multi-bit Trie, PMT) (如图 4 所示).

另一方面, 将单步长特里树映射为指定级数的线性流水线时, 为获得尽可能高的平衡度, Jiang 等^[20]先选定一个初始步长将单步长特里树划分为多棵子树, 然后将所有子树的根节点映射到流水线的第一级以保证稳定的延时 (出口设置在最后一级), 同时确保具有父子关系的节点不被映射到同一级, 且父在前子在后. 在逐级映射的过程中, 采用贪心策略使每一级所包含的存储块⁹⁾总数尽量接近于理想平均值. 按照此法构建的结构被称为优化的线性流水线 (OLP).

以 OLP 为基础, 进一步将所有子树划分为多个子集, 为每一个子集建立一条单独的流水线, 即可构建并行多流水的架构^[21] (parallel optimized liner pipeline, POLP, 如图 5 所示) 来提升吞吐率. 若指定流水线的数目, 则在划分子树的过程中需尽量均衡各子集的大小 (存储块总数) 以平衡各流水线间的负载. 此外, 若选取部分子树进行旋转, 并将这些子树的根节点映射到最后一级, 则可构建一个双向流水线结构^[22] (bidirectional optimized liner pipeline, BiOLP, 如图 6 所示). 只要选择合适的子树进行旋转, BiOLP 将会比 OLP 和 POLP 具有更理想的平衡度. OLP, POLP 和 BiOLP 都需要较多的流水级才能获得较理想的平衡度, 因而难以适用于 IPv6 或者大规模数据集.

3 流水化的多步长拆分特里树

为实现使用更少的流水级来构建存储高效且平衡的线性流水线, 以获得对 IPv6 和大规模数据集更好的支持, 本文提出了一种存储平衡且负载均衡的双向线性流水线结构 — PMST. 核心思想是先拆分前缀构建拆分特里树以提升存储和更新效率, 然后通过子树旋转和联合最优步长计算将多步长拆分特里树映射为一个双向流水线结构, 并设计了一系列优化算法来提升其平衡度.

3.1 拆分前缀建单步长拆分特里树

所谓拆分前缀, 即选定一个拆分位置 (实验表明, 选择中部附近的位置进行拆分效果最好), 将 IP 前缀沿该位置切开, 产生两个新的更短的前缀. 如 10001, 在位置 3 可拆分为 100 和 01. 如图 7 所示, 选定一个拆分位置 Pos 后, 在进行前缀拆分前, 我们先根据前缀长度将 FIB 横向划分为两个子集 FIB₁ 和 FIB₂, 使得 FIB 中所有长度大于 Pos 的前缀都在 FIB₂ 中, 而 FIB₁ 则保存着所有长度不大于 Pos 的前缀. 于是, 在 FIB 中进行最长前缀匹配 (LPM), 就可转化为对 FIB₁ 和 FIB₂ 的独立并行操作. 若在 FIB₂ 中匹配成功则匹配到的规则即为所求, 否则在 FIB₁ 中匹配到的规则为所求. 若都匹配失败, 则说明在 FIB 中匹配失败.

由于 FIB₁ 中所有前缀的长度都不大于 Pos, 故只需要取待查地址的前 Pos 位进行 LPM 即可. 而 FIB₂ 中所有前缀的长度都大于 Pos, 故我们先在位置 Pos 对其中所有前缀进行拆分, 然后将后半部相同的前缀合并为一个条目, 且只保留后半部作为新的前缀, 而他们的前半部以及对应的下一跳信息则插入到该新前缀对应的哈希表中, 这样便得到了 FIB'₂ (如图 7 所示). 为便于讨论, 我们称 FIB'₂ 中的前缀为新前缀, FIB₂ 中的前缀为原始前缀. 因此, 若将待查地址 Addr 沿位置 Pos 拆分为 Addr₁ 和

9) 存储块是 Trie 节点的基本组成单元, 一个叶推后的 S -步长的节点包含 2^S 个存储块.

图 4 5 级 PMT
Figure 4 5-stage PMT

图 5 2-流水 5 级 POLP
Figure 5 5-stage POLP with 2 pipelines

图 6 2-流水 5 级 BiOLP
Figure 6 5-stage BiOLP with 2 pipelines

Addr₂, 则 Addr 在 FIB₂ 中的 LPM 可转化为 Addr₂ 在 FIB'₂ 中的 LPM. 匹配到一个新前缀后, 再以 Addr₁ 作为关键字进入对应的哈希表中进行哈希即可获得最终匹配的原始前缀以及对应的下一跳信息 (如图 8 所示).

接下来, 我们分别根据 FIB₁ 和 FIB'₂ 建单步长特里树, 得到 Tree₁ 和 Tree₂, 再将所有信息推到叶子节点 (在 Tree₁ 上进行叶推与普通特里树一样, 而在 Tree₂ 上进行叶推以及做前缀扩展还需要进行哈希表的融合). 然后用一个虚拟根节点保存拆分位置以及指向两棵子树的指针, 则完成了单步长拆分特里树 (uni-bit split Trie, UST, 如图 8 所示) 的构建. 由于下一跳信息可能非常复杂 (比如多下一跳^[36]等), 而片上存储容量有限, 故我们将所有的下一跳信息以数组和哈希表的形式保存在片外存储

图 7 选择拆分位置 3, 将图 1 所示的 FIB 拆分为 FIB₁ 和 FIB₂, 再分别建单步长特里树
Figure 7 The FIB shown in Figure 1 is split into FIB₁ and FIB₂ at position 3, two uni-bit Tries are then built on them

图 8 单步长拆分特里树 (UST) 及其对应的查找算法
Figure 8 The uni-bit split Trie and its corresponding lookup algorithm

中, 并采用多哈希技术 [7] 来维护片外哈希表, 以提供快速稳定的哈希. 此外, 考虑到片外访存可能会成为 IP 查找的性能瓶颈 [9], 我们通过哈希函数和并行访存模块的设置以牺牲部分片外空间为代价来降低哈希表的密度, 尽量保证一次并行访存即可完成片外哈希.

算法 1 描述了在拆分特里树上进行 IP 查找的过程: 先拆分待查地址, 在两棵子树上进行并行查找, 得到两个索引值, 然后分别从对应的下一跳信息表和哈希表中获取相应的下一跳信息, 再确定最终结果. 如图 8 所示, 我们先把待查地址 0110001 沿位置 3 拆分为 011 和 0001, 然后并行地在 Tree₁ 和 Tree₂ 上进行查找, 得到 P₄ 和 G₅. 根据 P₄ 我们可从下一跳信息表中读取到 N₆. 另一方面, 根据 G₅

图 9 以图 8 所示的 UST 为基础, 通过 3 种算法分别计算联合最优步长后构建的 PMST

Figure 9 Based on the UST shown in Figure 8, three PMSTs are constructed according to different algorithms. (a) From UST to PMST; (b) Algorithm 2; (c) Algorithm 4; (d) Algorithm 6

我们可确定一个片外哈希表, 然后以 011 作为关键字在该哈希表中进行哈希, 可得 N_4 . 因为在 $Tree_2$ 上能成功查找到一个下一跳信息 N_4 , 故 N_4 即为所求. 再举一例, 若查找 0111101, 虽然 1101 在 $Tree_2$ 上能匹配到 G_1 , 但是以 011 作为关键字在对应的哈希表中会哈希失败, 于是 011 在 $Tree_1$ 上的查找结果 N_6 才是所求.

3.2 联合最优步长计算及流水化的多步长拆分特里树

构建好 UST 后, 我们再以极小化最大级为目标, 通过计算两棵子树的最优步长数组来构建多步长拆分特里树 (MST), 然后逐层映射为 PMST. 如图 9(a) 所示, 假设 UST 的两棵子树分别为 $Tree_1$ 和 $Tree_2$, 则 PMST 的某一级可能来自 $Tree_1$ 或者 $Tree_2$ 的某几层, 也可能是由 $Tree_1$ 的某几层和 $Tree_2$ 的某几层联合构成 PMST 的某一级. 因此, 不能单独考虑 UST 两棵子树的最优步长, 而是需要采用联合最优步长计算的方法来求最优步长.

假设需要从 UST 建 H -级 PMST, 而 UST 两棵子树的树高分别为 H_1 和 H_2 , 且他们每一层的节点总数分别保存在数组 $Nodes_1[]$ 和 $Nodes_2[]$ 中. 令 $Opt[i][j][k]$ 表示将 $Tree_1$ 的前 j 层和 $Tree_2$ 的前 k 层转化为 PMST 的前 i 级在最理想 (极小化最大级) 情况下最大一级所包含的存储块总数. 则计算联合最优步长需先通过模型 (1) 求解 Opt .

$$\begin{cases} Units[i][j] = j > i ? 2^{j-i} : 0, & Opt[0][0][0] = 0, \\ Opt[i][j][k] = \min_{\substack{1 \leq s \leq j, \\ 1 \leq t \leq k, \\ s+t \leq j+k}} \max \begin{cases} Opt[i-1][s][t], \\ Nodes_1[s] \times Units[s][j] + Nodes_2[t] \times Units[t][k]. \end{cases} \end{cases} \quad (1)$$

算法 2 描述了计算联合最优步长的过程, 即先通过一个三维动态规划求解模型 (1), 然后再通过对应的回溯算法 (算法 3) 完成两个最优步长数组的计算. 若根据图 8 所示的 UST 建 3-级 PMST, 通过算法 2 可计算得到两个步长数组 $\{1, 0, 2\}$, $\{3, 1, 0\}$, 据此可完成从 UST 到 PMST 的转化 (如图 9(b), 此时的最大级包含 10 个存储块).

算法 2 SplitMinMax ($H, \text{Stride}_1, \text{Stride}_2$)

```

1: for all  $i, j, k$  such that  $i \in [1, H], j \in [1, H_1], k \in [1, H_2]$  do
2:   for all  $s, t$  such that  $s \in [1, j], t \in [1, k], s + t < j + k$  do
3:      $\text{Cur} \leftarrow \text{Nodes}_1[s] \times \text{Units}[s][j]$ 
4:        $+ \text{Nodes}_2[t] \times \text{Units}[t][k];$ 
5:      $\text{Max} \leftarrow \max(\text{Cur}, \text{Opt}[i-1][s][t]);$ 
6:     if  $\text{Max} < \text{Opt}[i][j][k]$  then
7:        $\text{Opt}[i][j][k] \leftarrow \text{Max};$ 
8:     end if
9:   end for
10: end for
11: CalStrideMM( $H, \text{Stride}_1, \text{Stride}_2$ );

```

算法 3 CalStrideMM ($H, \text{Stride}_1, \text{Stride}_2$)

```

1:  $J = H_1; K = H_2;$ 
2: for  $i = H$  to 1 do
3:   for all  $s, t$  such that  $s \in [1, J], t \in [1, K], s + t < J + K$  do
4:      $\text{Cur} \leftarrow \text{Nodes}_1[s] \times \text{Units}[s][J]$ 
5:        $+ \text{Nodes}_2[t] \times \text{Units}[t][K];$ 
6:      $\text{Max} \leftarrow \max(\text{Cur}, \text{Opt}[i-1][s][t]);$ 
7:     if  $\text{Max} = \text{Opt}[i][J][K]$  then
8:        $jj \leftarrow s; kk \leftarrow t;$ 
9:     end if
10:   end for
11:    $\text{Stride}_1[i] \leftarrow (J - jj); \text{Stride}_2[i] \leftarrow (K - kk);$ 
12:    $J \leftarrow jj; K \leftarrow kk;$ 
13: end for

```

算法 4 SplitRevMM ($H, \text{Stride}_1, \text{Stride}_2$)

```

1: for all  $i, j, k$  such that  $i \in [1, H], j \in [1, H_1], k \in [1, H_2]$  do
2:   for all  $s, t$  such that  $s \in [1, j], t \in [1, k], s + t < j + k$  do
3:      $\text{Cur} \leftarrow \text{Nodes}_1[s] \times \text{Units}[s][j]$ 
4:        $+ \text{Nodes}_2[H_2 - k] \times \text{Units}[t][k];$ 
5:      $\text{Max} \leftarrow \max(\text{Cur}, \text{Opt}[i-1][s][t]);$ 
6:     if  $\text{Max} < \text{Opt}[i][j][k]$  then
7:        $\text{Opt}[i][j][k] \leftarrow \text{Max};$ 
8:     end if
9:   end for
10: end for
11: CalStrideREV( $H, \text{Stride}_1, \text{Stride}_2$ );

```

算法 5 CalStrideREV ($H, \text{Stride}_1, \text{Stride}_2$)

```

1:  $J \leftarrow H_1; K \leftarrow H_2;$ 
2: for  $i = H$  to 1 do
3:   for all  $s, t$  such that  $s \in [1, J], t \in [1, K], s + t < J + K$  do
4:      $\text{Cur} \leftarrow \text{Nodes}_1[s] \times \text{Units}[s][J]$ 
5:        $+ \text{Nodes}_2[H_2 - K] \times \text{Units}[t][K];$ 
6:      $\text{Max} \leftarrow \max(\text{Cur}, \text{Opt}[i-1][s][t]);$ 
7:     if  $\text{Max} = \text{Opt}[i][J][K]$  then
8:        $jj \leftarrow s; kk \leftarrow t;$ 
9:     end if
10:   end for
11:    $\text{Stride}_1[i] \leftarrow (J - jj); \text{Stride}_2[i] \leftarrow (K - kk);$ 
12:    $J \leftarrow jj; K \leftarrow kk;$ 
13: end for

```

3.3 旋转子树打造平衡基点

借鉴 BiOLP^[22] 旋转优化的思路, 我们先旋转 UST 的任一棵子树, 让其头尾颠倒与另一棵子树反向排列以获得一个更平衡的计算基点, 再采用动态规划来计算联合最优步长. 不失一般性, 我们假定旋转的是 Tree_2 , 令 $\text{Opt}[i][j][k]$ 表示将 Tree_1 的前 j 层和 Tree_2 的后 k 层转化为 PMST 的前 i 级在最理想 (极小化最大级) 的情况下的最大级所包含的存储块总数. 于是, 可将模型 (1) 和算法 2 分别改进为模型 (2) 和算法 4 (算法 5 为其对应的回溯算法).

$$\begin{cases} \text{Units}[i][j] = j > i ? 2^{j-i} : 0, & \text{Opt}[0][0][0] = 0, \\ \text{Opt}[i][j][k] = \min_{\substack{1 \leq s \leq j, \\ 1 \leq t \leq k, \\ s+t < j+k}} \max \begin{cases} \text{Opt}[i-1][s][t], \\ \text{Nodes}_1[s] \times \text{Units}[s][j] + \text{Nodes}_2[H_2 - k] \times \text{Units}[t][k]. \end{cases} \end{cases} \quad (2)$$

依然考虑从图 8 所示的 UST 建 3-级 PMST, 通过算法 4 也可计算得到两个步长数组 $\{3, 0, 0\}$, $\{0, 1, 3\}$, 据此构建的 PMST 如图 9(c) 所示, 其平衡程度相对于图 9(b) 来说要更好一点, 且此时最大

级仅包含 8 个存储块.

3.4 平衡度的形式化描述及优化策略

若将流水线每一级所包含的存储块总数视为一个序列, 则可通过该序列的均方差与平均值的比值来刻画其离散程度. 这个比值 (即序列的离散化系数) 越小, 说明流水线越平衡. 另一方面, 我们优化流水线平衡度的一个重要原因是为降低更新开销, 而 Write Bubble 机制^[16]的使用, 使更新开销主要受制于流水线的宽度, 而非深度. 故我们以离散化系数与流水线深度的比值作为平衡度参数 (balance parameter, BP) 来表征流水线的存储平衡程度 (假设共有 H 个流水级, 每一级包含的存储块总数都保存在数组 MemList 中, 且 $\overline{\text{MemList}} = \frac{\sum_1^H \text{MemList}[i]}{H}$):

$$\text{BP} = \frac{\sqrt{\sum_1^H (\text{MemList}[i] - \overline{\text{MemList}})^2}}{\overline{\text{MemList}} \times H}. \quad (3)$$

BP 越小说明流水线结构越平衡. 于是, 在指定目标流水线深度后, 我们以极小化最大级作为第一级目标, 同时以极小化 BP 作为第二级目标来计算联合最优步长. 考虑到求解联合最优步长的核心过程是动态规划, 本质上来说是一个递推的过程, 而 (3) 式的计算是线性复杂度, 且不适合递推 (需要开方), 故我们转而考虑 BP^2 , 并作相应变形如下:

$$\begin{aligned} \text{BP}^2 &= \frac{\sum_1^H (\text{MemList}[i] - \overline{\text{MemList}})^2}{(\overline{\text{MemList}} \times H)^2} \\ &= \frac{\sum_1^H \text{MemList}[i]^2 - 2 \times H \times \overline{\text{MemList}} \times \sum_1^H \text{MemList}[i] + H \times \overline{\text{MemList}}^2}{\overline{\text{MemList}}^2 \times H^2} \\ &= \frac{\sum_1^H \text{MemList}[i]^2 - H \times \overline{\text{MemList}}^2}{\overline{\text{MemList}}^2 \times H^2} \\ &= \frac{\sum_1^H \text{MemList}[i]^2}{(\sum_1^H \text{MemList}[i])^2} - \frac{1}{H}. \end{aligned} \quad (4)$$

在 (4) 式中, $\sum_1^H \text{MemList}[i]^2$ 和 $\sum_1^H \text{MemList}[i]$ 都非常适合做递推计算, 于是我们以极小化 BP^2 作为第二级优化目标, 通过 (4) 式来评估当前的平衡度, 将算法 4 (算法 2 也可进行类似的改进) 改进成算法 6 (算法 7 即为其对应的回溯算法).

还是考虑从图 8 所示的 UST 建 3- 级 PMST, 通过算法 6 同样可计算得到两个步长数组 $\{2, 0, 1\}$, $\{1, 2, 1\}$, 据其构建的 PMST 如图 9(d) 所示, 此时是完全平衡的状态, 每一级都包含 8 个存储块.

3.5 基于 PMST 的双向多流水架构

为充分利用输入带宽¹⁰⁾, 我们根据输入带宽为 PMST 部署多条主流线, 其中每条主流线都包括两条方向相反的子流水线, 由完整的 MST 映射而成, 且所有主流线结构完全相同并共享片外存储. 于是, PMST 的多流水架构可提供 3 个层次的并行, 一方面可以利用输入带宽提供数据并行, 另一方面各主流线又能对陆续抵达的查询请求进行流水化的处理, 此外一个待查地址被拆分成两部分可在两条子流水线上进行同步并行处理. 与大多数流水线架构一样, 前缀缓存的使用可进一步提升 PMST 的吞吐率. 图 10 展示了一个 3- 流水的 $n+1$ 级 PMST 架构.

10) 即同时进入流水线架构进行并行处理的数据包的个数.

算法 6 SplitRevOptMM ($H, \text{Stride}_1, \text{Stride}_2$)

```

1: for all  $i, j, k$  such that  $i \in [1, H], j \in [1, H_1], k \in [1, H_2]$  do
2:   for all  $s, t$  such that  $s \in [1, j], t \in [1, k], s + t < j + k$  do
3:      $\text{Cur} \leftarrow \text{Nodes}_1[s] \times \text{Units}[s][j]$ 
4:        $+ \text{Nodes}_2[H_2 - k] \times \text{Units}[t][k];$ 
5:      $\text{sum2} \leftarrow \text{Sum2}[i - 1][j][k] + \text{Cur}^2;$ 
6:      $\text{sum} \leftarrow \text{Sum}[i - 1][j][k] + \text{Cur};$ 
7:      $\text{BP} \leftarrow \text{sum2}/\text{sum}^2 - 1/i;$ 
8:     if  $\text{Max} \leq \text{Opt}[i][j][k]$  then
9:       if  $\text{Max} < \text{Opt}[i][j][k]$  or  $\text{BP} < \text{OptBP}$  then
10:         $\text{OptBP} \leftarrow \text{BP};$ 
11:         $\text{Sum2}[i][j][k] \leftarrow \text{sum2};$ 
12:         $\text{Sum}[i][j][k] \leftarrow \text{sum};$ 
13:      end if
14:       $\text{Opt}[i][j][k] \leftarrow \text{Max};$ 
15:    end if
16:  end for
17: end for
18: CalStrideROM( $H, \text{Sum}, \text{Sum2}, \text{Stride}_1, \text{Stride}_2$ );

```

算法 7 CalStrideROM ($H, \text{Stride}_1, \text{Stride}_2$)

```

1:  $J \leftarrow H_1; K \leftarrow H_2;$ 
2: for  $i = H$  to 1 do
3:   for all  $s, t$  such that  $s \in [1, J], t \in [1, K], s + t < J + K$  do
4:      $\text{Cur} \leftarrow \text{Nodes}_1[s] \times \text{Units}[s][J]$ 
5:        $+ \text{Nodes}_2[H_2 - K] \times \text{Units}[t][K];$ 
6:      $\text{Max} \leftarrow \max(\text{Cur}, \text{Opt}[i - 1][s][t]);$ 
7:      $\text{sum2} \leftarrow \text{Sum2}[i - 1][J][K] + \text{Cur}^2;$ 
8:      $\text{sum} \leftarrow \text{Sum}[i - 1][J][K] + \text{Cur};$ 
9:     if  $\text{Max} = \text{Opt}[i][J][K]$  and  $\text{sum2} = \text{Sum2}[i][J][K]$  and  $\text{sum} = \text{Sum}[i][J][K]$  then
10:       $jj = s; kk = t;$ 
11:    end if
12:  end for
13:   $\text{Stride}_1[i] \leftarrow (J - jj); \text{Stride}_2[i] \leftarrow (K - kk);$ 
14:   $J \leftarrow jj; K \leftarrow kk;$ 
15: end for

```

图 10 3-流水的 $n+1$ 级 PMST 架构**Figure 10** The architecture of a $(n+1)$ -stage PMST with 3 main pipelines**表 1** 实验数据集**Table 1** Experimental data

Router	Collect time	Total prefixes			Total updates		
		IPv4	IPv6	IPv6g	IPv4	IPv6	IPv6g
rrc11	2013-01-01-0759	3201093	70165	3201093	1177425	207287	1177425
rrc12	2013-01-01-0759	6679012	233191	6679012	4049260	1260126	4049260
rrc13	2013-01-01-0759	3926093	22826	3926093	2025239	107244	2025239
rrc14	2013-01-01-0759	2646590	78553	2646590	1388217	247228	1388217

当一个待查地址 Addr 准备进入某条主流流水线时, 我们先根据拆分位置将其拆分为 Addr_1 和 Addr_2 , 分别发送到两条子流水线. 再进一步根据 MST 两棵子树的最优步长数组, 将 Addr_1 和 Addr_2 分离成连续的地址片段存入相应的等待队列中. 然后, 分别从 logic_{10} 和 logic_{2n} 启动本次查询. 此后,

除启动单元外, 每一个逻辑单元负责从上一级接受信息, 然后按一定的规则解析出一个位置信息, 若已经查询成功或者该位置信息指向其他流水级, 则将原始信息继续传递给下一级. 否则, 从对应的地址队列中取出一个地址片段与该位置信息一起计算出本流水级的一个具体位置, 从中读出新的信息再传递给下一级, 直至进入结束单元 (logic_{1n} 和 logic_{20}). 最后, 根据 logic_{1n} 解析出的索引值从下一跳信息表中读取相应的信息. 同时, 根据 logic_{20} 解析出的索引值确定一个哈希表, 然后以 Addr_1 作为关键字进行哈希, 若哈希成功, 则哈希到的下一跳信息即作为最终结果输出, 否则输出从下一跳信息表中读取的信息作为最终结果.

4 实验评估与结果分析

4.1 实验环境与配置

本文采用 C++ 作为主要的编程语言, 首先实现了 PMT (通过 MinMax^[16] 算法构建), POLP^[21] 和 BiOLP^[22], 然后根据是否旋转子树以及是否针对 BP 进行优化, 分别采用了 4 种不同的最优步长计算算法来实现 PMST. 所有流水线架构均部署在 DELL T620 服务器 (操作系统为 Ubuntu 12.04, CPU 为 Intel Xeon E5-2630, 6 核 12 线程, 主频 2.30 GHz) 上, 其中 15 M 缓存即为片上 SRAM, 而片外存储即 16 G 内存.

表 1 列出了实验中用到的所有数据集 (包括 FIB 和一天的路由更新, 原始前缀未去重合并). 其中, IPv4 和小规模的 IPv6 数据集均来自真实网络¹¹⁾. 此外, 我们还按照一定规则^[35] 生成了大规模的 IPv6 数据集 (后缀为 v6g).

4.2 平衡度

我们首先针对 rrc12 的三个 FIB, 选择不同的拆分位置, 采用 4 种算法来构建指定级数 (v4 下为 4 级, v6 和 v6g 下为 8 级) 的 PMST, 并统计不同情况下的平衡度参数 (BP). 这 4 种算法都是以极小化最大级为一级优化目标, 其中基础算法 (即算法 2) 简称为 MM, 只旋转子树的算法 (即算法 4) 简称 REV, 只针对 BP 进行优化的算法简称为 OPT, 先旋转子树再针对 BP 进行优化的算法 (即算法 6) 简称为 ROM.

如图 11 所示, 有些情况下, 旋转子树 (REV, ROM) 利于平衡度的提升, 而有些情况下不旋转 (MM, OPT) 反而更好. 相比之下, 针对 BP 进行优化 (OPT, ROM) 效果要稳定得多, 即使在最坏情况下也只是和优化前 (MM, REV) 一样. 实际上, 针对 rrc12 的 3 个 FIB, 采用 OPT 构建 PMST 比采用 MM 在平衡度上分别能获得 0 ~ 72%, 33% ~ 80% 和 0 ~ 77% 的提升 (BP 越小平衡度越好). 而采用 ROM 相比于 REV 在平衡度上的优化也可达 83%, 69% 和 50%. 另一方面, ROM 与 OPT 在不同的拆分位置对平衡度的优化各有优劣. 但是针对 rrc12 的 3 个 FIB, 当拆分位置分别为 15, 32 和 36 时, 采用 ROM 构建的 PMST 能获得最优的平衡度参数 0.01, 0.03 和 0.02, 且均要优于其他算法. 虽然旋转子树不是一个绝对优化的策略, 但在大多数情况下 ROM 算法确实能提供最大程度的平衡度优化. 因此, 在后续实验和讨论中, 除特别说明外, PMST 均采用 ROM 构建, 且我们针对不同类型的 FIB (v4, v6, v6g) 选定的拆分位置分别为 15, 32 和 36.

图 12 刻画了 PMT, POLP, BiLOP 以及 PMST 在不同级数下的平衡度. 如图 12(a) 所示, PMT 和 PMST 的 BP 变化曲线都是先出现一次小波动然后逐步上升. 这是因为他们是通过调整步长数组来提

11) RIPE Network Coordination Centre [On line]. Available: <http://www.ripe.net>.

图 11 不同拆分位置以及不同优化算法对 PMST 平衡度的影响

Figure 11 How the balance parameter is influenced by split position and optimizing algorithm. (a) rrc12_v4; (b) rrc12_v6; (c) rrc12_v6g

图 12 4 种流水线架构在不同级数下的平衡度

Figure 12 The balance parameter of 4 pipeline architectures with different stages. (a) rrc12_v4; (b) rrc12_v6; (c) rrc12_v6g

升平衡度的, 当级数增加到一定程度后, 步长可调整的空间就越小 (因为树高是固定的), 很难再借此来改善存储分布, BP 曲线便开始上扬. 此外, 当级数较小的时候, PMST 两棵子树的互补增加了步长调整的灵活性, 因而能获得更好的平衡度. 但是, 其两棵子树的初始高度都远低于拆分前, 故步长调整的灵活性也要下降得更快, 于是其 BP 曲线的上扬点也就可能出现得更早. 而 POLP 和 BiOLP 本质上都是通过调整节点的映射位置来提升平衡度的, 级数越大调整的灵活性也就越大. 但根据其映

图 13 3 种多流水线架构在不同输入带宽下的吞吐率

Figure 13 The throughput of 3 multi-pipeline architectures under different configurations of input width. (a) rrc12_v4; (b) rrc12_v6; (c) rrc12_v6g

射规则, 第一级主要由初始步长和旋转因子确定, 可调整的空间较小. 因此, 当级数增大到一定程度后, POLP 和 BiOLP 的平衡度参数会稳定在最优值附近且缓慢上升.

为降低流水线延时和提高存储效率, 在保证一定平衡度的前提下, 流水线级数应越小越好. 若设定 BP 的阈值为 0.05, 当一个流水线结构满足 $BP \leq 0.05$, 则认为它已经获得了比较理想的平衡度. 如图 12 所示, PMT 无法满足要求, PMST 在级数较小时即能获得理想平衡度, 而 POLP 和 BiOLP 则需要级数较大时才能满足要求. 实际上, 对于 rrc12_v4, PMST 仅需 4 个流水线级就能获得理想的平衡度, 而 POLP 和 BiOLP 则需要至少 16 个流水线级. 对于 IPv6 的 FIB (包括 v6 和 v6g), 8 级 PMST 已经能满足要求. 而 POLP 和 BiOLP 在数据规模较小时 (图 12(b)) 分别需要 56 和 52 级才能满足平衡度要求, 即使数据规模的增大有利于平衡度的调整, 他们也需要至少 42 个流水线级才能获得理想的平衡度 (图 12(c)). 总的来说, 在保证理想平衡度的前提下, PMST 对流水线级的需求相对于 POLP 和 BiOLP 下降了 75% ~ 85.7%. 且这种优势在 IPv6 下表现得更明显.

4.3 查找性能

针对 rrc12 的 3 个 FIB, 我们先以最小级数来构建满足平衡度要求 ($BP \leq 0.05$) 的 PMST, POLP 和 BiOLP. 然后根据前缀信息随机产生不同规模的数据包流量, 测试这 3 种结构对应的多流水线架构在不同输入带宽下的吞吐率 (平均每时钟周期处理的数据包总数, packet per cycle, PPC). 如图 13 所示, 处理小流量 (数据包总数为 1 K) 时, PMST 和 POLP、BiOLP 的平均吞吐率都要远低于输入带宽, 且 PMST 因为级数更小表现出更高的性能. 而处理任务的增多逐步掩盖了级数对吞吐率的影响, 因此, 在处理大流量 (数据包总数为 10 M) 时, 3 种架构的吞吐率都非常接近于输入带宽. 总的来说, PMST 针对 IPv4 和 IPv6 分别能获得 1% ~ 5% 和 3% ~ 21% 性能提升, 且带宽越大优势越明显.

为比较 3 种架构的综合查找性能, 图 14 还给出了他们在延时、负载和访存上的对比. 如图 14(a) 所示, POLP 和 BiOLP 的延时基本差不多, 而 PMST 针对 3 种不同类型的 FIB 分别能使流水线延时

图 14 3 种流水线结构的综合查找性能对比

Figure 14 Comparison of comprehensive lookup performance among 3 architectures. (a) Pipeline latency; (b) traffic load of 8-pipeline architecture; (c) average memory accesses per lookup

降低 75%, 84.6% 和 81%. 从图 14(b) 可看出, 在以 8-流水架构处理 10M 流量时, PMST 的 8 条主流水线各承载 12.5% 的流量, 负载完全均衡. 而 BiOLP 针对两种不同类型的 FIB, 各流水线负载在 3.2% ~ 21.9% 和 8.4% ~ 16.7% 的范围内波动, 且其不均衡的负载主要体现在正向流水线 (1 ~ 4) 和反向流水线 (5 ~ 8) 之间. 这是因为 BiOLP 是以存储平衡为优化目标来选择子树进行旋转的, 虽然获得了比 POLP 更好的平衡度 (如图 12 所示), 却无法兼顾负载在两个方向上的均衡. 而 POLP 各流水线间的负载要更加均衡 (针对两个 FIB 的负载波动范围缩小至 12.42% ~ 12.61% 和 12.48% ~ 12.53%). 但比起 PMST 还是稍逊一筹. 事实上, PMST 首先根据输入带宽将同时进入流水线的多个数据包均摊到各条主流水线上, 以实现主流水间的负载均衡. 然后, 每条主流水线的两条反向子流水线完全同步地处理一个地址前后两部分的查找, 子流水线的负载同样也是完全均衡的. 这也是 PMST 与 BiOLP 在本质上的区别. 此外, 由于片上访存的情况直接关系到查找过程的能耗^[31], 我们还对比了 3 种架构在处理大流量过程中平均每次查询产生的片上访存次数. 如图 14(c) 所示, 相比于 POLP 和 BiOLP, PMST 能减少 37% ~ 79% 的平均访存, 且这种优势针对 IPv6, 特别是大规模数据集, 表现得更加明显.

4.4 存储效率和更新性能

最后, 我们通过比较存储开销和更新产生的 bubble 总数来说明 PMST 在片上存储效率和更新性能上的优势. 如图 15 所示, 针对 3 种类型的 FIB, PMST 最多仅需要 0.2 MB, 0.4 MB 和 55.2 MB 的片上存储, 相比于 POLP 和 BiOLP, 在片上存储效率上分别能获得 75.8% ~ 98.1%, 32.8% ~ 61.4% 和 93.9% ~ 96.0% 的提升. 特别是对 IPv6 的大规模的 FIB, 即使是维护 8-流水的 PMST (即 8 个同样的副本), 这种片上存储优势依然能维持在 51.0% ~ 68.2%. 而其片外存储开销在最坏情况下分别增加了 41.6 倍, 8.9 倍和 123.9 倍. 考虑到片上存储相对于片外存储的紧缺程度 (15 MB 缓存与 16 GB 主存), 这样的置换还是很值得.

图 16 给出了 POLP, BiOLP 和 PMST 处理一天的路由更新所产生的 bubble 总数. 如图 16(b), 针

图 15 3 种流水线结构的存储开销

Figure 15 Memory cost (in MBytes) of 3 architectures. (a) IPv4; (b) IPv6; (c) IPv6g

图 16 3 种流水线结构的更新开销

Figure 16 Update overhead (total number of produced bubbles) of 3 architectures. (a) IPv4; (b) IPv6; (c) IPv6g

对小规模的 IPv6 数据, 3 种结构的更新开销相差无几, 互有优劣. 而对于 IPv4 或者是大规模的 IPv6 数据, PMST 的优势就非常明显了. 如图 16(a) 和 16(c) 所示, PMST 分别减少了 71.7% ~ 98.4% 和 47.8% ~ 87.2% 的更新开销. 另一方面, POLP 和 BiOLP 还可能导致部分更新失效. 这是因为当级数确定后, 为提升平衡度, POLP 和 BiOLP 可能将某些子树拉得很“长”(即从根节点到叶子节点跨越了数目远超其树高的流水级). 当需要在这些子树上插入前缀时, 新增的叶子节点又必须沿着流水线的方向往后映射, 而级数是有限的, 这就导致了某些新增节点找不到合适的位置映射而成为失效节点. 所

会有产生失效节点的更新我们都称为失效更新. 从图 16(c) 可看出, 针对大规模的 IPv6 数据, POLP 和 BiOLP 产生的失效更新甚至高达 72.5% 和 39.3%, 这就需要非常频繁的重构才能保证查找的正确性. 相比之下, PMST 基于最优步长数组的映射机制从根本上避免了这一更新隐患.

5 结论

基于前缀拆分和子树旋转, 本文首先提出了一种双向的流水线结构 —PMST, 然后定义了流水线的 BP 来形式化描述流水线的存储平衡程度, 并据此对 PMST 进行了更深入的优化. 实验结果表明, 相比于前人的优秀工作, PMST 在获得同等理想的平衡度时对流水级需求下降了 75% ~ 85.7%, 这也进一步使 PMST 在保持高吞吐率的前提下能有效降低流水线延时, 且在查找过程中减少了 37% ~ 79% 平均片上访问, 也有利于能耗的降低. 而多流水架构的负载完全均衡, 又使 PMST 能更充分的利用输入带宽来进一步提升吞吐率. 不过, PMST 牺牲了片外空间来换取更高效率的片上存储. 实验结果表明, PMST 能减少至少 32.8% 的片上存储需求, 针对大规模的 IPv6 数据其存储效率的提升甚至能达到 93.9% ~ 96.0%. 在更新开销方面, PMST 在 IPv4 和大规模 IPv6 数据下都表现出了明显的优势, 分别能获得至少 71.7% 和 47.8% 的优化. 因此, PMST 不仅具有更高的综合性能, 而且其存储、更新优势在大规模 IPv6 数据集下表项得更为明显, 也说明其具备更好的可扩展性, 能更好地适应未来的应用需求.

参考文献

- 1 Chao H J, Liu B. IP Address Lookup. Hoboken: John Wiley and Sons, Inc., 2006. 25–76
- 2 Chowdhury N M, Boutaba R. A survey of network virtualization. *Comput Netw*, 2010, 54: 862–876
- 3 Wang G, Tzeng N F. TCAM-based forwarding engine with minimum independent prefix set (mips) for fast updating. In: *Proceedings of IEEE International Conference on Communications*, Istanbul, 2006. 103–109
- 4 Akhbarizadeh M J, Nourani M, Panigrahy R, et al. A TCAM-based parallel architecture for high-speed packet forwarding. *IEEE Trans Comput*, 2007, 56: 58–72
- 5 Shah D, Gupta P. Fast updating algorithms for TCAMS. *IEEE Micro*, 2001, 21: 36–47
- 6 Luo L Y, Xie G G, Xie Y K, et al. A hybrid IP lookup architecture with fast updates. In: *Proceedings of INFOCOM*, Orlando, 2012. 2435–2443
- 7 Broder A, Mitzenmacher M. Using multiple hash functions to improve IP lookups. In: *Proceedings of 20th Annual Joint Conference of the IEEE Computer and Communications Societies*, Anchorage, 2001. 1454–1463
- 8 Song H Y, Hao F, Kodialam M, et al. IPv6 lookups using distributed and load balanced bloom filters for 100 Gbps core router line cards. In: *Proceedings of INFOCOM*, Rio de Janeiro, 2009. 2518–2526
- 9 Huang Z, Peir J K, Chen S G. Approximately-perfect hashing: improving network throughput through efficient off-chip routing table lookup. In: *Proceedings of INFOCOM*, Shanghai, 2011. 311–315
- 10 Srinivasan V, Varghese G. Fast address lookups using controlled prefix expansion. *ACM Trans Comput Syst*, 1999, 17: 1–40
- 11 Eatherton W, Varghese G, Dittia Z. Tree bitmap: hardware/software IP lookups with incremental updates. *ACM SIGCOMM Comput Commun Rev*, 2004, 34: 97–122
- 12 Huang K, Xie G G, Li Y B, et al. Offset addressing approach to memory-efficient IP address lookup. In: *Proceedings of INFOCOM*, Shanghai, 2011. 306–310
- 13 Song H Y, Kodialam M, Hao F, et al. Scalable IP lookups using shape graphs. In: *Proceedings of 17th IEEE International Conference on Network Protocols*, Princeton, 2009. 73–82
- 14 Uzmi Z A, Nebel M, Tariq A, et al. SMALTA: practical and near-optimal FIB aggregation. In: *Proceedings of the 7th Conference on Emerging Networking Experiments and Technologies*, New York, 2011. 29–36

- 15 Jiang W R, Wang Q B, Prasanna V K. Beyond TCAMs: an SRAM-based parallel multi-pipeline architecture for terabit IP lookup. In: Proceedings of the 27th Conference on Computer Communications, Phoenix, 2008. 1786–1794
- 16 Basu A, Narlikar G. Fast incremental updates for pipelined forwarding engines. *IEEE/ACM Trans Netw*, 2005, 13: 690–703
- 17 Chang Y K, Liu Y C, Kuo F C. A pipelined IP forwarding engine with fast update. In: Proceedings of International Conference on Advanced Information Networking and Applications, Bradford, 2009. 263–269
- 18 Baboescu F, Tullsen D M, Rosu G, et al. A tree based router search engine architecture with single port memories. In: Proceedings of 32nd International Symposium on Computer Architecture, 2005. 123–133
- 19 Kumar S, Becchi M, Crowley P, et al. CAMP: fast and efficient IP lookup architecture. In: Proceedings of the ACM/IEEE Symposium on Architecture for Networking and Communications Systems, New York, 2006. 51–60
- 20 Jiang W R, Prasanna V K. A memory-balanced liner pipeline architecture for Trie-based IP lookup. In: Proceedings of IEEE Symposium on High-Performance Interconnects, Stanford, 2007. 83–90
- 21 Jiang W R, Prasanna V K. Sequence-preserving parallel IP lookup using multiple sram-based pipelines. *J Parall Distr Comput*, 2009, 69: 778–789
- 22 Le H, Jiang W R, Prasanna V K. A sram-based architecture for Trie-based IP lookup using FPGA. In: Proceedings of 16th International Symposium on Field-Programmable Custom Computing Machines, Palo Alto, 2008. 33–42
- 23 Hanna M, Cho S, Melhem R. A novel scalable IPv6 lookup scheme using compressed pipelined Tries. In: NETWORKING 2011. Berlin: Springer, 2011. 406–419
- 24 Le H, Jiang W R, Prasanna V K. Memory-efficient IPv4/v6 lookup on FPGAs using distance-bounded path compression. In: Proceedings of IEEE 19th Annual International Symposium on Field-Programmable Custom Computing Machines, Salt Lake City, 2011. 242–249
- 25 Erdem O, Bazlamaçci C F. High-performance IP lookup engine with compact clustered Trie search. *Comput J*, 2012, 55: 1447–1466
- 26 Tong D, Yang Y E, Prasanna V K. A memory efficient IPv6 lookup engine on FPGA. In: Proceedings of International Conference on Reconfigurable Computing and FPGAs, Cancun, 2012. 1–6
- 27 Qu Y, Prasanna V K. High-performance pipelined architecture for tree-based IP lookup engine on FPGA. In: Proceedings of IEEE 27th International Parallel and Distributed Processing Symposium Workshops & PhD Forum, Cambridge, 2013. 114–123
- 28 Yang Y H E, Qu Y, Haria S, et al. Architecture and performance models for scalable IP lookup engines on FPGA. In: Proceedings of IEEE 14th International Conference on High Performance Switching and Routing, Taipei, 2013. 156–163
- 29 Chang Y, Kuo F, Guo H, et al. Layered trees: most specific prefix based pipelined design for on-chip IP address lookups. *IEEE Trans Comput*, 2013, 99: 1–14
- 30 Wu Y, Nong G. A pipeline ip lookup architecture with random duplicate allocation. In: Proceedings of 21st International Conference on Computer Communications and Networks, Munich, 2012. 1–7
- 31 Jiang W R, Prasanna V K. Architecture-aware data structure optimization for green IP lookup. In: Proceedings of International Conference on Proceedings of High Performance Switching and Routing, Richardson, 2010. 113–118
- 32 Luo L Y, Xie G G, Salamatian K, et al. A Trie merging approach with incremental updates for virtual routers. In: Proceedings of INFOCOM, Turin, 2013. 1222–1230
- 33 Matousek J, Skacan M, Korenek J. Memory efficient IP lookup in 100 Gbps networks. In: Proceedings of 23rd International Conference on Field Programmable Logic, Porto, 2013. 1–8
- 34 Erdem O, Carus A, Le H. Large-scale SRAM-based IP lookup architectures using compact Trie search structures. *Comput Electr Eng*, 2014, 40: 1186–1198
- 35 Wang M, Deering S, Hain T, et al. Non-random generator for IPv6 tables. In: Proceedings of IEEE Symposium on High Performance Interconnects, 2004. 35–40
- 36 Chen W P, Zhang X M, Zhang J H, et al. Research on multi next hop rip. In: Proceedings of International Forum on Information Technology and Applications, Chengdu, 2009. 16–19

PMST: a balanced bidirectional pipeline architecture to high-performance IP lookup

LI YanBiao¹, ZHANG DaFang^{1*}, HUANG Kun², HE DaCheng¹ & ZENG Bin³

¹ College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China;

² Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China;

³ China Mobile Group Hunan Co., Ltd, Changsha 410015, China

*E-mail: dfzhang@hnu.edu.cn

Abstract With the rapid development of network technology and the growing diversity of network applications, IP lookup, as a core function of routers, has been confronted with serious challenges, ranging from throughput, memory efficiency and update performance. Benefiting from the pipeline technology, the throughput of IP lookup has been improved significantly. However, unbalanced structure will not only lead to low memory efficiency and high update overhead, but also influence the lookup performance and the load balance of multi-pipeline architectures. While existing solutions to balance optimizing would bring some indispensable new problems, impending their applications in IPv6 or large-scale routing tables. In view of this, this paper proposes a balanced bidirectional pipeline architecture—pipelined multi-bit split tree (PMST). After splitting prefixes, reversing a sub-tree and a series of optimizations, PMST can achieve a desirable balance degree at the cost of only a few stages. What is more, PMST also realizes significant improvements on comprehensive performance. Large IPv6 routing data sets generated in some manner, as well as real-world IPv4/v6 routing data sets, are used to evaluate the performance of PMST comprehensively. According to the experimental results, when achieving a desirable balance degree, the number of stages cost in PMST decreases by 75% ~ 85.7% in comparison to existing solutions. Meanwhile, the superiorities of PMST on pipeline latency, average memory accesses per lookup, on-chip memory efficiency, update overhead and the load balance of multi-pipeline architecture are also significant. Therefore, PMST has higher comprehensive performance and better scalability, which can better meet current and future demands.

Keywords packet forwarding, pipeline, architecture, algorithm, memory-efficient, balance, Trie

LI YanBiao was born in 1986. He is a Ph.D. candidate in Hunan University. His main research interests include IP lookup, high-performance computing and future internet architecture.

ZHANG DaFang was born in 1959. He is a Professor and Ph.D. supervisor in Hunan University. His main research interests include dependable systems/networks, network security and software engineering.

HUANG Kun received his Ph.D. and M.S. degrees in computer science from Hunan University, China, in 2009 and 2004, respectively. He is currently an assistant professor at Institute of Computing Technology, Chinese Academy of Sciences. His research interests focus on packet processing and future Internet architecture.

HE DaCheng was born in 1987. He is a Ph.D. student in Hunan University. His main research interests include IP lookup and software defined network.