

引用格式:曹作伟,陈晓,倪宏.一种支持快速增量更新的掩码匹配算法[J].微电子学与计算机,2019,36(4):84-88. [CAO Z W, CHEN X, NI H. A mask matching algorithm with fast incremental update support [J]. Microelectronics & Computer, 2019,36(4): 84-88.]

一种支持快速增量更新的掩码匹配算法

曹作伟^{1,2}, 陈晓¹, 倪宏¹

(1 中国科学院 声学研究所 国家网络新媒体工程技术研究中心, 北京 100190;

2 中国科学院大学 电子电气与通信工程学院, 北京 100049)

摘 要: 文章提出一种支持快速增量更新的掩码匹配算法 TernarySort. 该算法使用决策树将掩码匹配规则划分为多个分区, 不同叶节点的规则互不重叠, 以保障规则更新性能. 同时通过选取有效比特位, 减小决策树深度, 并允许叶节点中规则重叠, 减少分区数量, 以提高分类性能. 实验结果表明, 与现有的算法相比, TernarySort 可提升 80% 以上的分类性能, 同时保持快速的更新性能. TernarySort 能够满足 SDN 对掩码匹配算法的要求.

关键词: 软件定义网络; 网包分类; 决策树

中图分类号: TP393.04

文献标识码: A

文章编号: 1000-7180(2019)04-0084-05

A Mask Matching Algorithm with Fast Incremental Update Support

CAO Zuo-wei^{1, 2}, CHEN Xiao¹, NI Hong¹

(1 National Network New Media Engineering Research Center, Institute of Acoustics, Chinese Academy of Sciences, Beijing 100190, China; 2 School of Electronic, Electrical and Communication Engineering, University of Chinese Academy of Sciences, Beijing 10049, China)

Abstract: This paper proposes a mask match algorithm, TernarySort, supporting fast incremental update. The algorithm uses decision trees to partition mask matching rules. To guarantee the update performance, it restricts that rules residing in different leaf nodes do not overlap with each other. At the same time, TernarySort selects effective bits to reduce the depth of decision trees. And it allows rules to overlap in one leaf node, to decrease the number of partitions and improve the classification performance. The experimental results show that, compared to currently used algorithms, TernarySort can improve packet classification performance by more than 80% while maintaining fast rule updating performance. TernarySort meets the requirements of SDN for mask matching algorithms.

Key words: SDN; packet classification; decision tree

1 引言

软件定义网络(Software Defined Networking, SDN)将控制功能与转发功能分离, 通过集中式的控制器以标准化接口对各种网络设备进行管理和配

置. OpenFlow^[1]由斯坦福大学提出, 已成为 SDN 中控制器与网络设备通信的事实标准. 然而 OpenFlow 面临协议扩展问题, 难以满足更多的网络协议、带状态的数据转发等复杂需求. 新的数据平面编程语言, 如 P4^[2], 提出了对任意协议字段的支持. 借

助数据平面可编程技术,交换机无需理解特定的报文协议,新的网络应用的开发不再需要修改交换机代码,新技术、新业务的迭代速度将得到极大的提升。

类似于 OpenFlow, P4 采用了匹配动作表 (Match Action Table, MAT) 作为数据平面的转发模型。在 MAT 模型中,报文处理包括字段解析、流表匹配与动作执行等步骤。流表匹配包括最长前缀匹配、范围匹配和掩码匹配等类型,是影响数据平面转发性能的关键环节之一。流表匹配问题实质上是网包分类问题。在 SDN 环境下,网包分类算法除支持报文的快速分类,还需要支持规则的快速更新,以满足控制器动态修改交换机转发规则的需求。

传统的网包分类算法大多只关注报文分类速度,难以满足快速更新的需求。基于决策树的算法如 HiCuts^[3]、SmartSplit^[4]等,按空间将规则划分为单个决策树,分类性能高,然而由于规则重叠,同一条规则会重复出现于叶子节点,更新一条规则需要同时处理多个节点,影响更新性能。开源 OpenFlow 交换机 Open vSwitch^[5] 采用改进的 TSS (Tuple Space Search)^[6] 算法,按照字段掩码长度对规则进行划分,分区内使用哈希搜索,同一分区内规则互不重叠,TSS 算法可支持快速更新,然而 TSS 算法的分区数量过多,影响了报文的分类性能。在 TSS 算法基础上,TupleMerge^[7] 算法根据协议字段,去除不重要的比特(比如 IP 地址的末尾比特),以降低分区数量,提高分类速度,然而该算法要求交换机具备协议字段的先验知识,不能直接应用于可编程数据平面。PartitionSort^[8] 算法采用最大独立集的方式对规则进行划分,要求同一分区内的规则满足可排序、不重叠,对于最长前缀匹配和范围匹配规则,该算法在更新速度与查找速度上取得了良好的平衡。然而,由于掩码匹配规则无法排序,不能直接应用 PartitionSort 算法。

针对上述问题,本文提出一种支持快速增量更新的掩码匹配算法 TernarySort,选取有效比特将规则划分为决策树,在避免分区内生成重复规则的同时,减少分区数量,实现高速的分类性能与快速的更新性能。

2 TernarySort 算法

2.1 问题描述

报文字段 f 是 w 个报文比特构成的自然数集合 $[0, 2^{w-1}]$,规则 r 是从 d 个字段的子集到某个动作的映射关系,即 $r = (s_1 \subseteq f_1 \times s_2 \subseteq f_2 \times \cdots \times s_d \subseteq f_d$

$\rightarrow$ decision)。若报文 $p = (p_1, p_2, \cdots, p_d)$ 满足 $p_1 \in s_1 \wedge p_2 \in s_2 \wedge \cdots \wedge p_d \in s_d$,称 p 匹配 r 。 n 个规则按照优先级排列,组成一个规则集合 $[r_1, r_2, \cdots, r_n]$ 。报文分类即找到优先级最高的匹配规则。

根据规则在字段上的定义,将规则分为范围匹配、最长前缀匹配和掩码匹配等类型。范围匹配规则由区间定义 $s = [l, h]$;最长前缀匹配规则由前缀定义 $s = \{0, 1\}^{w_p} * w - w_p$,其中 $0 \leq w_p \leq w$ 为前缀长度, $*$ 表示同时匹配 0 和 1;与最长前缀匹配规则一样,掩码匹配规则由 0/1/ $*$ 比特构成,不同的是,0/1/ $*$ 可以按照任意顺序排列。

2.2 算法思想

TernarySort 算法结合决策树的规则划分方式与避免重复规则的分区思想。决策树作为掩码匹配规则的查询结构,可实现报文的快速分类,在决策树内,按照比特划分规则;一个决策树作为一个分区,分区内没有重复的规则,规则被划分为多个分区,规则的插入和删除只涉及单个分区的节点,可实现规则的快速更新。如表 1 所示,与现有的网包分类算法相比较,TernarySort 算法支持掩码匹配规则,并能够实现高速的报文分类与快速的规则更新。

表 1 TernarySort 算法与其他算法的比较

算法	SmartSplit	TSS	PartitionSort	TernarySort
范围匹配	支持	不支持	支持	不支持
最长前缀匹配	支持	支持	支持	支持
掩码匹配	不支持	支持	不支持	支持
分类性能	很高	较慢	高	高
更新性能	很慢	快	快	快
内存占用	很高	低	低	低

2.3 用于掩码匹配的决策树

本小节将介绍用于掩码匹配的决策树结构。选取比特划分规则,非叶节点表示用于划分的比特,叶节点存储掩码匹配规则。对于同一分区内,任意两个不同叶节点中的规则,存在可以区分的比特,即规则不会重复出现于不同的叶节点。同一叶节点中的规则允许重叠。如表 2 所示的掩码匹配规则示例中, $\{r_1, r_2, r_3, r_4, r_5\}$ 共同构成图 1 所示的决策树,“ $b < 1 >$ ”表示将规则按照第 1 位比特的值进行划分。 r_6 由于无法与其他规则区分,将划分到其他决策树中。

报文分类时,按照决策树的比特顺序查找匹配规则。若叶节点中存在匹配该报文的规则,则返回优先级最高的规则,否则继续查找其他决策树。

表 2 掩码匹配规则示例

规则	掩码	优先级
r_1	100	6
r_2	11 *	5
r_3	01 *	4
r_4	* 01	3
r_5	* 00	2
r_6	* * *	1

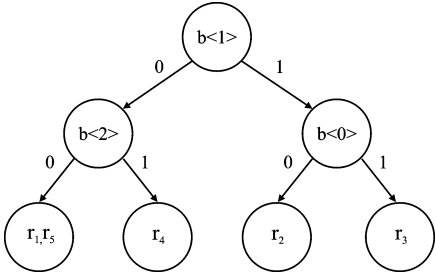


图 1 决策树示意图

插入规则 r 时,按照决策树的比特顺序查找叶节点,若 r 在非叶节点比特上的值为通配符,说明该规则与多个叶节点中的规则重叠,不能插入该决策树;若找到了叶节点,与节点中规则集合 R 比较,若存在可区分的比特,则插入新的非叶节点,以区分 R 与 r ;若 R 与 r 无法区分,并且 R 中的规则数量未达到阈值 Thr_∞ (比如 10),则令 $R = R \cup \{r\}$;若 R 中的规则数量等于阈值,则不能插入. 通过阈值 Thr_∞ 允许节点中的规则重叠,可降低决策树深度,提高分类性能. 若规则 r 无法插入决策树,则尝试插入其他决策树. 若规则 r 无法插入现有决策树,则构建新的决策树.

删除规则 r 时,从所属叶节点中移除 r ,若所属叶节点中的规则为空,则删除叶节点及其父节点,并用兄弟节点代替其父节点的位置.

决策树的时间性能与深度线性相关,令 n 为决策树中的规则数量, d 为决策树的实际深度, l 为掩码匹配规则中的字段长度和, 必然有 $\log_2(n/\text{Thr}_\infty) \leq d \leq l$. 存在正实数 $M = l$ 与 $n_0 = 2$, 对所有的 $n \geq n_0$, 均满足 $|d| \leq M \lceil \log_2(n/\text{Thr}_\infty) \rceil$. 故 $d = O(\log n)$, 即单个决策树的查询、插入和删除性能为 $O(\log n)$. 决策树所需的节点数量为 $O(n)$, 空间性能为 $O(n)$.

2.4 离线分区算法

当重叠的规则数量较多时,规则将被划分为多个分区,考虑到报文分类性能与分区数量直接相关,为提高分类性能,应尽可能减少分区数量. 对此,采用有效比特对规则进行划分,所谓“有效比特”,即能够区分最多规则的比特,应满足两点要求,其一,规

则在该比特的通配符比例应尽可能低,以容纳更多的规则,其二,该比特的 0/1 分布应尽可能均匀,以更好地区分规则.

令 N 为当前待划分的规则数量, N_*^i 表示第 i 位为通配符的规则数量, 则第 i 位的通配符比例为:

$$P_*^i = \frac{N_*^i}{N} \tag{1}$$

第 i 位为 0 的比例表示为 P_0^i , 第 i 位为 1 的比例表示为 P_1^i , 则第 i 位的 0/1 分布情况可表示为:

$$H^i = -\frac{P_0^i}{1-P_*^i} \log \frac{P_0^i}{1-P_*^i} - \frac{P_1^i}{1-P_*^i} \log \frac{P_1^i}{1-P_*^i} \tag{2}$$

将二者相加作为第 i 位的权重:

$$\omega_i = H^i + (1 - P_*^i) \tag{3}$$

选择权重最高的比特作为有效比特.

以图 1 为例, 给定规则集合 $\{r_1, r_2, r_3, r_4, r_5, r_6\}$, 计算每个比特的权重, 选取第 1 个比特划分规则, 得到 $\{r_1, r_4, r_5\}$ 、 $\{r_2, r_3\}$ 与 $\{r_6\}$, $\{r_6\}$ 的第 1 位比特为通配符, 将划分到新的决策树. 第二步划分 $\{r_1, r_4, r_5\}$ 和 $\{r_2, r_3\}$, 计算比特权重分别选择第 2 位与第 0 位作为有效比特, 前者被划分为 $\{r_1, r_5\}$ 与 $\{r_4\}$, 后者被划分为 $\{r_2\}$ 与 $\{r_3\}$, 最终得到图 1 所示的决策树.

基于有效比特的概念, 提出划分规则的离线分区算法, 如表 3 所示. 算法通过不断迭代, 将规则划分为决策树. 第 5 行中, 当规则无法继续划分时, 比如所有比特都被选择, 或是剩余比特都相同, 则选择优先级最高的前 Thr_∞ 个规则生成叶子节点, 返回剩余规则. 对于未能划分的规则, 继续使用该算法生成新的决策树, 直到将所有规则完成划分.

令规则数量为 n , 每个规则需要经过 $\Omega(\log n)$ 次划分, 每一次划分需要计算 $O(l)$ 个比特的权重, 故而算法 1 的时间性能为 $\Omega(l \cdot n \cdot \log n)$. 最坏的情况下, 当所有规则的所有比特都相同时, 第 i 个规则需要经过 i 次划分, 算法 1 的时间性能为 $O(l \cdot n^2)$.

利用算法 1 只需少量的分区即可完成规则划分. 为进一步提高报文分类性能, 采用类似于 Open vSwitch 对 TSS 的优化, 记录每个分区的最高优先级, 将分区按照最高优先级进行排序. 在报文分类时, 按照最高优先级降序搜索分区, 若匹配规则的优先级大于下一个分区的最高优先级, 则停止搜索, 否则继续搜索下个分区. 分区 T 内的规则优先级采用最大堆 $H(T)$ 维护, 其插入和删除时间性能均为 O

(logn),空间性能为 $O(n)$ 。

表 3 规则分区算法

算法 1 规则分区算法 Partition(S,node)
Input:规则集合 S,规则所属节点 node Output:未划分的规则集合 R
1. if $ S \leq \text{Thr}_{\text{co}}$
2. 将 node 作为叶子节点,存入 S 中的规则,返回 $\emptyset$
3. for each 未选择的比特 in S
4. 计算各比特的通配符比例 P_i^* 与熵 H^i
5. if 无法继续划分
6. 将 node 作为叶子节点,存入 S 中优先级最高的前 Thr_{co} 则,返回剩余规则
7. 选择权重最大的比特 i,记录在节点 node 中
8. let $R = \{S \text{ 中第 } i \text{ 位为通配符的规则}\}$
9. let $S_0 = \{S \text{ 中第 } i \text{ 位为 } 0 \text{ 的规则}\}$, node0=新的节点
10. let $S_1 = \{S \text{ 中第 } i \text{ 位为 } 1 \text{ 的规则}\}$, node1=新的节点
11. 返回 $R \cup \text{Partition}(S_0, \text{node0}) \cup \text{Partition}(S_1, \text{node1})$

2.5 在线分区算法

在线分区算法中,由于规则被随机插入和删除,无法预先获取规则的比特信息,不能直接应用算法 1. 为降低报文分类时的查找分区的平均数量,插入规则 r 时,按照最高优先级降序依次尝试插入各个分区,使规则尽量聚集在优先级最高的分区中. 若所有分区都不能容纳 r ,则创建新的分区. 将规则 r 插入分区 T 后,同时插入 $H(T)$ 内,若最高优先级发生变化,则重新排序所有分区.

决策树比特的选取受到规则插入顺序的影响,由于规则的添加顺序随机,可能导致决策树不平衡,影响分类性能. 对此,当分区内的规则数量达到阈值 Thr_{re} (比如 10) 时,运用算法 1,重新调整决策树. 由于阈值 Thr_{re} 较小,运行算法 1 的时间不会太长.

此外,规则的随机插入,还可能导致通配符比例低的规则被通配符比例高的规则绑定的情况. 若第一个插入的规则为 $r_6 = \langle * * * \rangle$,无论第二个规则为何,都将与 r_6 位于同一叶节点,无法再与其他规则区分. 对此,采用 Jaccard 相似度作为重叠规则的准入条件. 定义规则 A 和规则 B 的 Jaccard 相似度为:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

(4)

对于掩码匹配规则,定义 $|A \cap B|$ 为相同的非通配符比特数量:

$$|A \cap B| = \text{pop_count}((m(A) \& m(B)) \& (k(A) \wedge k(B)))$$

(5)

式中, $m(A)$ 表示规则 A 的掩码,有 $m(r_1) = \langle 111 \rangle$, 而 $m(r_6) = \langle 000 \rangle$; $k(A)$ 表示规则 A 的值,如 $k(r_1) = \langle 100 \rangle$, 而 $k(r_6) = \langle 000 \rangle$,

$\text{pop_count}(b)$ 表示比特向量 $O(n)$ 中 1 的个数. 定义 $O(m \log n)$ 为非通配符比特的数量:

$$|A \cup B| = \text{pop_count}(m(A) \mid m(B))$$

(6)

通过(4)式可以计算出 $J(r_1, r_6) = 0$, 而 $J(r_1, r_5) = 2/3$. 设置适当的相似度阈值 Thr_{sim} (比如 0.1), 即可避免高通配符比例规则绑定的情况.

对于规则删除,为快速定位规则所在分区,维护映射 $M(R)$ 记录规则插入的分区. 在删除规则时,除了从决策树中删除规则,还应从 $H(T)$ 与 $M(R)$ 内移除对应的记录,并根据需要重新排序分区. 若分区 T 内不再含有其他规则,删除分区 T . $M(R)$ 可采用平衡二叉树实现,其插入和删除时间性能为 $O(\log n)$,空间性能为 $O(n)$.

令分区数量为 m ,则报文的分类时间与规则的插入时间为 $O(m \log n)$,规则的删除时间与 m 无关,为 $O(\log n)$,算法的空间性能为 $O(n)$.

3 实验与分析

为评估算法有效性,本文实现了 TernarySort 算法,并与 TSS 算法进行了比较,后者采用 Sor-rachai 的代码实现.

3.1 实验环境

实验使用的规则由 ClassBench^[9]生成,使用 12 个种子文件(5 个 ACL,5 个防火墙 (FW),2 个 IP-chain (IPC)),生成 1 k、2 k、4 k、8 k、16 k、32 k、64 k 共 7 种不同大小的规则集合,每种大小生成 5 个规则,总共测试 420 个规则集合. 这些规则由五元组定义,其中 IP 地址使用最长前缀表示,端口号使用范围表示,协议号使用精确匹配表示. 为支持这类规则,在 TernarySort 算法的实现中,对于无法使用掩码匹配表示端口范围的规则,仅使用 IP 地址的比特用于规则划分.

实验包括三类,离线分类、在线分类、在线更新. 离线分类使用所有规则生成静态的数据结构,对 1M 个报文进行分类. 在线分类,按照随机顺序插入规则,用生成的数据结构对 1M 个报文进行分类. 报文数据由 ClassBench 根据规则生成. 在线更新测试中,使用一半的规则作为初始规则插入,然后随机运行 500 k 次插入和 500 k 次删除,统计规则的更新时间.

实验运行的软件环境为 Linux 4.12.14 64 位系统,硬件环境为 Intel Core i5-4570 @ 3.20 GHz,4 核心,4 GB DDR3 内存.

3.2 与 TSS 算法的比较

使用在线分类测试比较 TernarySort 算法与

TSS 算法的分类性能. 用 TSS 的报文分类时间除以 TernarySort, 对 420 个比例求平均值, TernarySort 的分类速度比 TSS 快 88%. 具体到 60 个 64 k 规则集合, TernarySort 的分类速度比 TSS 快 126%. 图 2 展示了二者在不同规则集合上的分类时间. 通过求分类时间的倒数, 可得到算法的吞吐速率, TernarySort 可以实现 1.79 Mpps 的平均吞吐, 而 TSS 只有 0.99 Mpps 的平均吞吐.

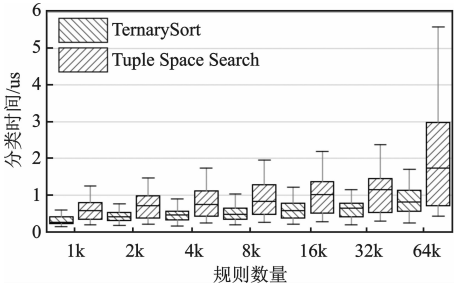


图 2 TernarySort 算法与 TSS 算法的分类性能比较

TernarySort 的分类速度能够超过 TSS 的分类速度, 原因在于前者所需的分区数量远少于后者. 如图 3 所示, TSS 所需的分区数量平均为 TernarySort 的 11 倍, 报文查询的分区数量平均为 TernarySort 的 7 倍.

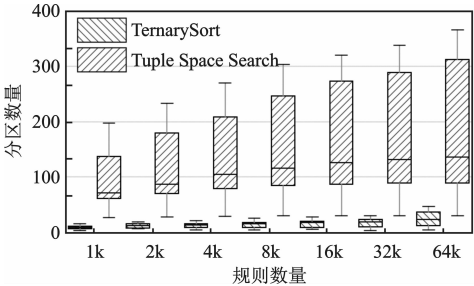


图 3 TernarySort 算法与 TSS 算法的分区数量比较

使用在线更新测试比较 TernarySort 算法与 TSS 算法的更新性能. TernarySort 的平均规则更新时间为 1.07 μ s, 最大更新时间为 1.89 μ s. 对于大多数应用而言, 这样的规则更新性能已经足够. 图 4 展示了 TernarySort 与 TSS 在不同规则集合上的平均更新时间. TernarySort 的规则更新时间平均比 TSS 多 83%.

在内存占用方面, TernarySort 与 TSS 所需的内存都为 $O(n)$, TernarySort 所需内存平均较 TSS 多 38%.

3.3 在线分区算法与离线分区算法的比较

离线分类测试结果显示, TernarySort 的离线分区算法在分类性能方面要明显优于在线分区算法. 如图 5 所示, 在线算法的分类时间平均比离线算法多 125%. 在线算法中规则插入的顺序随机, 对比特的选则相对盲目. 对比二者的分区数量, 在线算法

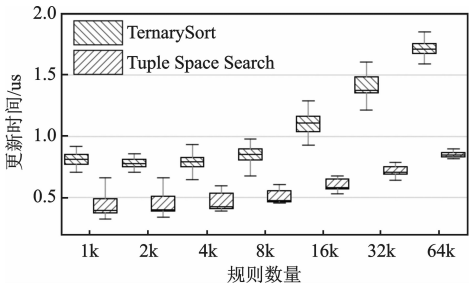


图 4 TernarySort 算法与 TSS 算法的更新性能比较
平均比离线算法多 106%。

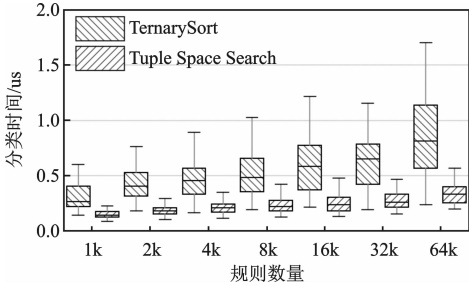


图 5 TernarySort 在线分区算法与
离线分区算法的分类性能比较

图 6 展示了在线算法与离线算法的平均构建时间, 二者相差在 1 个量级以上. 在线算法可以在 0.1 s 完成 64 k 个规则的插入, 而离线算法则需要 2 s 以上. 对报文分类性能要求很高、更新性能要求较低的场景下, 采用离线分区算法会更好一些.

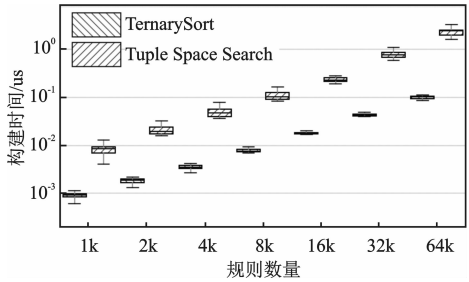


图 6 TernarySort 在线分区算法与
离线分区算法的构建时间比较

4 结束语

此前的研究提出了可以同时满足高速分类和快速更新需求的网包分类算法. 然而, 这些研究并不能应用于可编程数据平面的掩码匹配. 针对这个问题, 本文提出了一种应用于支持快速增量更新的掩码匹配算法 TernarySort, 基于有效比特的计算, 将掩码匹配规则划分为多个决策树, 同时提供高速的分类

(下转第 92 页)

的复位措施. 确保主机在枚举键鼠异常时 CH9350 可以通过 CH9317 复位 CH317Q, 达到自动重新枚举的目的.

5 结束语

针对 USB 传输距离有局限性问题, 本文研究提出了一种基于 CH317Q 的 USB 长线传输设计方案, 同时对长线传输鼠标键盘不识别问题进行了分析, 给出了解决方案. 上述长线传输系统经过实际运行, 工作稳定、可靠, 可以推广到对传输距离要求高的工业控制、医疗设备及武器系统中实现智能仪器与计算机之间的远程 USB 通信.

参考文献:

[1] 陈喜春. 基于 CY7C68013A 芯片的 USB 键盘的设计

(上接第 88 页)

性能和快速的更新性能. 在此基础上, 本文通过计算规则的相似性, 允许部分规则重叠, 以减少分区数量, 进一步提高分类性能. 本文基于实际流量规则, 对所提方法进行了验证. 实验结果表明, 与现有网包分类算法相比, 所提 TernarySort 算法可提升 80% 以上的分类性能, 同时保持快速的规则更新性能. 本文的研究表明, TernarySort 算法可以满足 SDN 对掩码匹配算法的性能要求.

在未来工作中, 作者将针对多核环境的同步问题, 进一步优化 TernarySort 算法, 并在可编程数据平面中实现.

参考文献:

[1] MCKEOWN N, AANDERSON T, BALAKRISHNAN H, et al. OpenFlow: enabling innovation in campus networks [J]. ACM SIGCOMM Computer Communication Review, 2008, 38(2): 69-74.

[2] BOSSHART P, DALY D, GIBB G, et al. P4: Programming Protocol-independent Packet Processors[J]. ACM SIGCOMM Computer Communication Review, 2014, 44(3): 87-95.

[3] GUPTA P, MCKEOWN N. Packet classification using hierarchical intelligent cuttings[A]. Hot Interconnects VII[C]. 1999: 34-41.

[4] HE P, XIE G, SALAMATION K, et al. Meta-algorithms for Software-Based Packet Classification[A]. 2014 IEEE 22nd International Conference on Network Protocols[C]. 2014: 308-319.

[J]. 山西电子技术, 2012 (1): 6-7.

[2] 杨红军, 洪三其. 基于 FPGA 的高标清数字光端机的研究[J]. 制造业自动化, 2015 (10): 117-120.

[3] 刘泽西, 程晶晶, 孔力. 基于 USB2. 0 接口的高速实时数据采集系统[J]. 测控技术, 2007, 26(2): 34-37.

[4] 马伟. 计算机 USB 系统原理及其主/从机设计[M]. 北京: 北京航空航天大学出版社, 2004. 1

[5] 刘荣. 圈圈教你玩 USB. [M]. 北京: 北京航空航天大学出版社(第 2 版), 2013.

作者简介:

许小兰 女, (1994-), 硕士研究生. 研究方向为嵌入式计算机系统应用. E-mail: 13227818958@163. com.

李 灏 男, (1978-), 研究员, 硕士生导师. 研究方向为嵌入式计算机系统应用.

[5] PFAFF B, PETTIT J, KOPPNEN T, et al. The Design and Implementation of Open vSwitch[A]. 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15) [C]. Oakland, CA: USENIX Association, 2015: 117-130.

[6] SRINIVASAN V, SURI S, VARGHESE G. Packet Classification Using Tuple Space Search[A]. Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication [C]. New York, NY, USA: ACM, 1999: 135-146.

[7] DALY J, TORNG E. TupleMerge: Building Online Packet Classifiers by Omitting Bits[A]. 2017 26th International Conference on Computer Communication and Networks (ICCCN) [C]. 2017: 1-10.

[8] YINGCHAREONTHAWORNCHAI S, DALY J, LIU A X, et al. A Sorted-Partitioning Approach to Fast and Scalable Dynamic Packet Classification[J]. IEEE/ACM Transactions on Networking, 2018: 1-14.

[9] TAYLOR D E, TURNER J S. Classbench: A packet classification benchmark[J]. IEEE/ACM Transactions on Networking (TON), 2007, 15(3): 499-511.

作者简介:

曹作伟 男, (1991-), 博士研究生. 研究方向为软件定义网络、可编程数据平面.

陈 晓(通讯作者) 男, (1964-), 硕士, 研究员. 研究方向为网络通信、新媒体技术等. E-mail: xxchen@dsp. ac. cn.

倪 宏 男, (1964-), 硕士, 研究员. 研究方向为网络通信、新媒体技术等.