一种高效的 OpenFlow 流表存储与查找实现方法

鄂跃鹏, 陈智*, 葛敬国, 吴玉磊

中国科学院信息工程研究所信息安全国家重点实验室, 北京 100093

* 通信作者. E-mail: chenzhi@iie.ac.cn

收稿日期: 2014-11-03; 接受日期: 2015-07-31; 网络出版日期: 2015-09-22

中国科学院战略性先导专项 (批准号: XDA06010306)、国家重点基础研究发展计划 (973 计划) (批准号: 2012CB315803)、国家自然科学基金青年基金 (批准号: 61303241) 和国家高技术研究发展计划 (863 计划) (批准号: 2013AA013501) 资助项目

摘要 目前 OpenFlow 流表的存储与查找还面临一些挑战, 如流表项匹配字段数量的不断增加、流表规模的爆炸性增长以及实现 OpenFlow 硬件交换机的端口线速处理更加困难. 针对这些问题, 本文提出了一种高效的 OpenFlow 流表存储与查找实现方法. 该方法将 OpenFlow 流表划分成多个规模更小、匹配字段更少的子流表, 进一步根据字段匹配类型将子流表切分成多个字段, 不同字段采用不同的算法和硬件进行存储和查找, 得到字段匹配结果后再结合汇总表查找从而得到子流表的匹配结果. 通过划分流表与切分子流表来压缩流表存储空间, 根据子流表的查找选择和子流表内部查找优化来减少查找过程中硬件资源开销. 通过大量模拟实验表明, 该存储与查找方法能够压缩 50% 左右的存储空间, 且可以有效减少流表项匹配过程中 80% 硬件资源的消耗.

关键词 OpenFlow 流表 存储与查找 子流表 子流表切分 汇总表

1 引言

软件定义网络 (software defined network, SDN) ^[1,2] 是由美国 Stanford 大学提出的一种新型网络创新架构, 具有设备资源虚拟化、接口标准化、开放可编程以及软件灵活控制等特点, 为新型网络体系结构及应用提供了良好的创新平台. OpenFlow ^[3,4] 以基于流表的数据平面和高度开放的控制平面逐渐成为 SDN 最重要的南向接口标准.

目前, OpenFlow 协议还面临很多未解决的问题. 针对 OpenFlow 流表的存储与查找, 存在的挑战主要如下:

1) 流表项匹配字段数量不断增加. OpenFlow v1.0 定义了 12 元组匹配字段. 而 OpenFlow v1.3 则定义了 40 项可匹配字段, 其中包括 13 项交换机必须支持的匹配字段. 未来新型互联网体系结构以及新型网络应用, 将扩展流表项中匹配字段类型和字段数.

2) 流表规模爆炸性增长. 当前网络中仅 IPv4 的 BGP 全局路由表条目已经超过 50 万条. 考虑 OpenFlow 支持不同的控制粒度, 随着网络规模的增大, 流表项条数将会爆炸性增长. OpenFlow v1.1 中引入了多级流表来解决流表规模的可扩展性问题 ^[4], 但多级流表的灵活性超出了传统交换芯片的数据转发处理逻辑, 增加了 OpenFlow 交换机硬件设计的复杂度.

引用格式: 鄂跃鹏, 陈智, 葛敬国, 等. 一种高效的 OpenFlow 流表存储与查找实现方法. 中国科学: 信息科学, 2015, 45: 1280–1288, doi: 10.1360/N112014-00353

3) 实现硬件交换机端口线速处理的难度更大. OpenFlow 通过流表的方式将传统网络协议栈扁平化, 导致转发过程中需要匹配的字段数量大大增加. 目前网络交换机端口可以达到 10, 40 和 100 G/s 的处理速度. 而 100 G/s 的线速处理需要每 6.7 ns 完成一次 512 字节包查找, 这对 OpenFlow 交换机实现大规模流表的线速查找提出了更高的硬件要求.

针对 OpenFlow 流表中存储与查找存在的这些问题, 本文提出了一种高效的 OpenFlow 流表存储与查找实现方法. 主要内容有:

1) 提出了一种高效的 OpenFlow 流表存储与查找实现方法, 该方法能有效压缩流表存储空间, 减少流表项匹配过程中硬件资源的消耗.

2) 结合 SRAM 和 TCAM, 提出该存储与查找方法的一种硬件实现方案, 并对该硬件实现方案进行分析.

3) 基于软件编程模拟 OpenFlow 流表存储与查找方法的硬件实现方案, 通过模拟实验可知该存储与查找方法能够优化 50% 左右的存储空间, 可以有效减少流表项匹配过程中 80% 硬件资源的消耗.

本文第 2 节介绍了 OpenFlow 流表存储与查找的相关研究. 第 3 节主要说明了所提出的流表存储与查找实现方法以及该方法的一种硬件实现方案. 第 4 节主要是硬件实现方案的模拟实验和分析. 第 5 节总结全文.

2 OpenFlow 流表存储与查找研究

现有大多数高速数据包处理器主要是基于 TCAM 或者 Hash 来实现^[5], 但 TCAM 存在成本高、功耗大和表项更新复杂的问题. Meiners 等^[6]提出了 TCAM SPliT 结构, 将一个包含 d 维字段的 TCAM 查找分割成 k 个包含低维字段的 TCAM 查找, 从而采用流水线的方式进行查找. 针对 OpenFlow 流表项包含的 40 维字段, SPliT 结构中 FDD 的维护将面临新的挑战. 刘中金等提出一种 OpenFlow 多级流表结构及其映射算法, 将单一流表映射到多级流表中进行高效存储和查找, 该方法比单一流表的存储方法节省 17%~95% 的 TCAM 资源^[7]. 但是该 TCAM 资源映射方法需要进行多次映射运算, 同时优化结果波动性较大.

ONF (Open Networking Foundation) 在 OpenFlow v1.1 中提出了多级流表结构, 通过分域多级流表查询来压缩流表空间, 在一定程度上减少流表存储空间的消耗. 文献 [8] 描述了一种 TCAM 和 Hash 相结合的 OpenFlow 多级流表实现方式, 但该实现方式对于流表项中匹配字段个数具有很大的依赖性. 而文献 [9] 提出了一种全新的交换芯片设计思路, 从而形成一种全新的 OpenFlow 多级流表架构, 整个处理流程与具体协议无关, 同时交换机的入方向和出方向各有 32 级流表. 整个框架能够达到 OpenFlow 多级流表的灵活性要求, 但所需硬件成本较大, 硬件实现也很复杂.

考虑 OpenFlow 流表项的特性, OpenFlow 流表可划分成 Hash 匹配表和 Linear 匹配表, Linear 表可以匹配任何字段但查找效率低, 当流表中包含大量 wildcard 字段时, 该算法可能查找效率较低^[10]. Curtis 等^[11]通过修改 OpenFlow 协议提出了 DevoFlow 方法, 能够有效减少 OpenFlow 流表项条数, 但不具有可扩展性.

针对 OpenFlow 流表存储优化, 文献 [12] 提出了一种启发式的 OpenFlow 流表存储空间优化算法 (a heuristic storage space optimization algorithm for flow table, H-SOFT). H-SOFT 算法通过分析 OpenFlow 流表结构定义字段之间的共存和冲突关系, 将一个 OpenFlow 流表划分成多个规模更小、匹配字段更少的子流表, 从而降低流表的存储空间.

图 1 OpenFlow 流表存储与查找方法

Figure 1 The storage and lookup method of OpenFlow flow table

基于 H-SOFT 算法, 本文提出一种高效的 OpenFlow 流表存储与查找实现方法, 同时结合 TCAM 和 SRAM 给出该方法的硬件实现方案. 该方法能够很好地解决 OpenFlow 流表项匹配字段的增加以及流表规模增大的问题.

3 OpenFlow 流表存储与查找实现方法

OpenFlow 网络中, 控制器向交换机下发不同的流表项, 从而实现网络的灵活控制. 优化 OpenFlow 流表的存储与查找将有利于交换机性能提升和大规模 OpenFlow 网络部署.

3.1 OpenFlow 流表存储与查找实现方法

H-SOFT 算法从存储空间优化角度考虑, 将 OpenFlow 流表划分成多个子流表进行存储. 本研究基于 H-SOFT 算法得到的多个子流表, 考虑整个流表的查找性能, 提出一种高效的 OpenFlow 流表存储与查找实现方法. 具体方法如图 1 所示. 从图 1 可知, 当数据包到达交换机时, 首先通过数据包字段解析模块得到数据包头部包含的匹配字段信息. 然后数据包进入队列调度模块等待处理, 如果后续的快速匹配模块空闲, 则进入快速匹配模块进行匹配. 快速匹配模块仅包含本规则类中预先设定和匹配次数最多的少量规则. 如果快速匹配模块成功匹配, 则跳过子流表的匹配查找, 直接得到数据包处理结果, 否则进行子流表查找. 在进行子流表查找时, 首先进入子流表选择模块, 通过数据包头部包含的字段信息从 k 个子流表中选择一个或多个需要进行查找匹配的子流表, 各个子流表之间的查找过程是独立的, 并行执行子流表的查找匹配, 将一个或多个子流表输出结果汇总到匹配结果模块, 通过处理得到最终的匹配结果, 从而执行相应的 Actions.

在所提出的 OpenFlow 流表存储与查找实现方法中, 为实现 OpenFlow 流表的高效硬件查找, 本研究对子流表进一步进行切分. 考虑 OpenFlow 流表中每个匹配字段具有的特性, 将匹配字段划分成 4 种不同匹配类型: 线性匹配字段, 如协议号; 精确匹配字段, 如 MAC 地址; 最长前缀匹配字段, 如 IP 地址; 范围匹配字段, 目前只包含端口号, 但端口号并没有明确支持范围匹配, 本文基于已有的 ACL 列表考虑未来 OpenFlow 流表端口号可能的范围匹配扩展, 将端口号划分成范围匹配字段.

针对每个子流表按字段进行切分, 得到多个字段, 每个字段只存储对应字段在该子流表中出现的匹配值. 而字段本身按照不同字段匹配类型采用不同的硬件进行存储查找. 同时一个子流表中可能包含一个或多个同种类型的匹配字段, 也可能不包含某种类型的匹配字段.

在子流表内部划分字段时, 流表项中各个字段之间的相关性也随之消失, 为了保证子流表切分前后数据包匹配结果的一致性, 本研究通过汇总表来存储各个字段之间的相关性, 从而得到正确的子流

图2 子流表内部结构及流表项查找匹配

Figure 2 The structure of sub-flow table and lookup process in it

表匹配结果. 具体的子流表内部结构及查找匹配过程如图2所示. 在图2中, 该子流表中每条流表项包含5个匹配字段, 分别为MAC源地址、MAC目的地址、IP协议号、IP源地址、IP目的地址. 子流表按字段进行切分, 得到5个不同字段. 并行查找各个字段, 根据字段匹配结果得到匹配向量, 并在汇总表中查找该匹配向量, 从而得到数据包在该子流表中的匹配结果. 具体的字段查找和汇总表查找将在存储与查找方法的硬件实现方案部分进行说明.

3.2 存储与查找方法的硬件实现方案

在子流表字段切分过程中, 需要考虑子流表中汇总表的存储空间. 以图2为例, 流表项R2和R4都被扩展成两条, 汇总表规模增大. 这是由于IP字段具有最长前缀匹配特性. 例如IP目的字段值001010即可以匹配001*, 也可以匹配00*, 但是IP目的字段对应的字段只输出匹配结果 e_1 . 同时子流表进行字段切分之后, 不同字段之间存在乘法效应. 例如IP源包含 N 种可能匹配, IP目的包含 M 种可能匹配, 则一条流表项在汇总表中需要被扩展成 $M \times N$ 条. 因此相对于子流表, 汇总表规模可能会爆炸性增长.

在所提出的硬件实现方案中, 需要考虑子流表中不同字段的具体硬件存储与查找以及汇总表存储优化. 针对线性匹配字段, 子流表不采用硬件进行查找处理, 直接返回该字段值, 即input等于output; 精确匹配字段采用SRAM进行直接比较, 相对于原字段, 返回位数更少的匹配结果; 而端口号字段目前并不考虑范围匹配的扩展, 因此现有的范围匹配字段同样不采用硬件进行处理, 直接返回端口号的值. 最长前缀匹配字段中不同值之间存在覆盖包含关系, 该字段的某一个值可能出现多种匹配, 考虑汇总表的存储空间优化, 流表项中最长前缀匹配字段在汇总表中直接存储对应值, 即最长前缀匹配字段不进行字段存储, 到达数据包的头字段在子流表中进行字段匹配时, 直接返回对应的字段值, 然后形成匹配向量进入汇总表中查找. 因此子流表中的一条流表项在经过子流表切分后只形成一条汇总表流表项, 同时汇总表中每条流表项所需的存储空间减少. 而汇总表采用TCAM实现查找. 具体实现如图3所示. 由图3可知, 子流表被切分后形成多个字段和一个汇总表, 而多个字段中只有精确匹配字段对应的字段进行硬件存储查找. 交换机内部具有多个固定容量的SRAM和TCAM硬件模块. 每个子流表根据自身规模对应一个或多个SRAM与TCAM, 同时逻辑上仍属于同一个子流表. 未来该方法的硬件实现方案将会针对子流表中字段的硬件查找匹配进行更多研究.

图 3 流表存储与查找方法的硬件实现方案

Figure 3 The hardware implementation of proposed method

4 实验和分析

为了评估所提出的 OpenFlow 流表存储与查找实现方法, 本文通过 python 编程模拟实现该方法的硬件实现方案, 并进行流表存储效率以及流表查找时硬件资源开销的相关实验. 同时还对 OpenFlow 流表存储与查找实现方法进行了分析. 所有模拟实验都是基于 Window7 系统进行的, 其中硬件配置为 Intel Core i7 2.9 GHz 双核 CPU 以及 4 GB 的内存.

目前 OpenFlow 还没有完全支持 1.3 标准的商业交换机, 同时外界很难获知 Google, Facebook 等服务提供商在其数据中心部署的 OpenFlow 流表信息. 本研究则通过 NetFlow 按照 1000:1 的比例进行采样, 得到一段时间内的数据包统计信息, 转换包头字段信息生成不同的流表项用于相关实验. 生成的流表中匹配字段有 Inport, Mac_src, Mac_dst, Eth_Type, IP_Proto, IPv4_src, IPv4_dst, IPv6_src, IPv6_dst, TCP_src, TCP_dst, UDP_src, UDP_dst, ICMPv4_type, ICMPv4_code, ICMPv6_type, ICMPv6_code.

4.1 OpenFlow 流表存储与查找实现方法的分析

设集合 $T = R_1, R_2, \dots, R_i, \dots, R_N$ 表示流表中包含的 N 条流表项, 其中每条流表项包含 d 维匹配字段, 用集合 $D = F_1, F_2, \dots, F_d$ 表示, 每个字段所占用的 bit 数用 $W = w_1, w_2, \dots, w_d$ 表示. 因而对于单一流表而言, 整个流表所需存储空间为

$$\text{Space}_0 = N \sum_{j=1}^d w_j. \quad (1)$$

如果只考虑单个匹配字段, 该字段中某些匹配值会重复出现, 比如多个不同的主机访问同一个 MAC 地址. 对于单个匹配字段 $F_i (1 \leq i \leq d)$, 其中互不重复的匹配值数目定义为 $\text{uniq}(F_i)$, 其中 $\text{uniq}(F_i) <$

N_i . 对于字段 $F_i(1 \leq i \leq d)$, 存储位数为 $w_i(1 \leq i \leq d)$. 经过字段模块查找后返回存储位数为 $w'_i(w'_i \leq w_i)$ 的结果, 从而降低汇总表中每条流表项所占的 TCAM 存储空间. 例如 48 bit 的 MAC 源地址, 经过精确匹配之后返回 16 bit 的匹配值. 同时减少汇总表的存储空间. 假设 OpenFlow 流表被划分成 K 个子表. 则所提出的流表存储与查找实现方法中流表的存储空间为

$$\text{Space}_K = \sum_{i=1}^K (S_i + T_i) = \sum_{i=1}^K \left(\sum_{j=i}^e \left(w_i \times \text{uniq}(F_j) + N_i \times \sum_{x=1}^{d_i} w'_x \right) \right), \quad (2)$$

其中 S_i 表示流表占用的 SRAM 存储空间, T_i 表示流表占用的 TCAM 存储空间, F_i 表示需要进行 SRAM 存储与查找的字段, N_i 表示第 i 个子表的流表项规模, d_i 表示第 i 个子表中包含的匹配字段, w'_x 表示汇总表中每个字段存储的 bit 位数. 其中 $N = \sum_{i=1}^K N_i, 1 \leq d_i \leq d$, 因此存储空间的优化效率为

$$\text{Opt} = \text{Space}_K / \text{Space}_0 = \sum_{i=1}^K (S_i + T_i) / N \sum_{j=1}^d w_j, \quad (3)$$

Opt 值与子流表结构以及所有流表项在子流表中的分布有关. 当所有流表项平均分布到子流表中, 不同子流表中平均每条流表项所需存储空间约为单一流表中相对应值的 $p\%$, 则 $\text{Opt} = p$. 基于本文的流表存储与查找方法, 在每个子流表内部, 数据包经过 SRAM 和 TCAM 查找后得到子流表匹配结果. 同时通过子流表选择, 每个数据包只进入一个或某几个子流表进行匹配. 相对于基准 OpenFlow 流表, 每个数据包的一次查找匹配过程消耗的硬件资源减少, 同时通过子表选择匹配模块进行数据包的匹配调度, 当两个数据包查找时所需子流表相互独立, 则可以并行处理这两个数据包的查找, 从而提高查找效率. 同时可以针对各种不同的试验场景或者不同的新型网络协议来划分不同的子流表. 交换机也可以设计不同的子流表, 而新的子流表的加入, 并不影响其他子流表的匹配结果. 从而使得交换机具有更好的可扩展性.

4.2 硬件存储效率分析

所提出的 OpenFlow 流表存储与查找实现方法, 通过将 OpenFlow 流表划分成多个子流表, 再对子表进行字段划分及汇总表存储优化, 从而压缩流表存储空间. 实验中, 设置不同的流表规模, 流表项条数分别为 3000, 5000, 10000, 50000 和 100000 条, 计算该存储与查找实现方法中所需存储空间与基准 OpenFlow 流表所需存储空间的比值, 其中基准 OpenFlow 流表是指所有流表项存储在同一个流表中. 实验结果如图 4 所示. 从图 4 可以看出, 在所提出的方法中, 针对由网络流信息产生的不同规模的流表, 流表的硬件存储压缩效率稳定在 50% 左右. 由于实验数据中涉及 IPv4 的流表项较多, 同时包含 IPv4 字段的子流表中每条流表项所占存储空间约为单一流表中每条流表项所占存储空间的 50%, 因此本实验中该方法的流表存储空间压缩效率约为 50%.

4.3 流表查找时硬件资源开销分析

针对存储与查找方法的硬件实现. 考虑流表查找时所需执行的硬件匹配次数, 计算流表查找过程中硬件匹配开销减少的比例 Reduce_rate . 其中 $\text{Reduce_rate} = 1 - (H_{\text{opt}}/H_{\text{ori}})$, H_{opt} 表示基于该方法查找时所需执行的硬件匹配 bit 位数, H_{ori} 表示基于基准 OpenFlow 流表所需执行的硬件匹配 bit 位数. 实验中, 分别设置 30 万和 100 万个数据包到达 OpenFlow 交换机, 在不同规模的流表中分别进行查找 ($N = 3000, 5000, 10000, 50000$ 和 100000). 当到达数据包完成查找时, 计算各个查找过程中的

图 4 存储与查找方法中流表存储空间的压缩

Figure 4 The compression of storage space in proposed method

图 5 存储与查找实现方法中 Reduce_rate 的变化

Figure 5 The vary of Reduce_rate in proposed method

Reduce_rate. 同时针对固定的流表项 ($N = 10000$), 分别设置 10 万、20 万、30 万、50 万和 100 万个数据包到达 OpenFlow 交换机, 计算各个查找过程中的 Reduce_rate. 实验结果如图 5 所示. 在文献 [10] 中, 作者将 OpenFlow 流表可划分成 Hash 匹配表和 Linear 匹配表, Hash 表处理精确匹配而 Linear 表可以匹配任何字段. 数据包先进入 Hash 表中匹配查找, 当匹配成功时返回结果, 如果不成功则进入 Linear 表进行查找匹配. 该算法只是将整个流表划分成两个匹配表, 存储空间并没有进行优化. 而当整个流表规模较大且包含 wildcard 字段的表项较多时, 如果 Hash 表命中率不高, 对 Linear 表的查找会降低整个算法的查找效率. 因此, 所提出的算法相对于该算法在存储空间上有一定优化, 在查找过程具有更好的可扩展性.

从图 5(a) 可以看出, 在不同的流表规模下, Reduce_rate 的值稳定在 85% 左右. 当到达数据包的规模为 30 万和 100 万时, Reduce_rate 的变化曲线基本重合. 这是因为实验中的流表项数据由网络流生成, 不同规模的流表存储结构类似. 同时由图 5(b) 可知, 当流表规模固定, 到达数据包规模不断增大时, Reduce_rate 的值稳定在 85% 左右.

5 总结

本文主要提出了一种高效的 OpenFlow 流表存储与查找方法. 首先基于 H-SOFT 算法将高维稀

疏的 OpenFlow 流表划分成多个规模更小, 匹配字段更少的 OpenFlow 子流表. 进一步考虑匹配字段本身的特性, 将匹配字段划分成: 线性匹配字段、精确匹配字段、最长前缀匹配字段和范围匹配字段. 对子流表进一步进行切分, 得到多个字段和一个汇总表. 通过划分多个子流表以及切分每个子流表, 从而压缩流表存储空间. 通过 SRAM 和 TCAM 结合的方式实现该存储与查找方法的硬件实现. 实验结果表明, 该存储与查找方法能够优化 50% 左右的存储空间, 同时能够有效减少流表项匹配过程中 80% 硬件资源的消耗.

在此基础上, 下一步工作将考虑该存储与查找实现方法的优化以及完成该方法在存储和查找效率上与现有相关算法的对比, 同时考虑该方法如何实现 OpenFlow 流表项的快速准确更新.

参考文献

- 1 Zuo Q Y, Chen M, Zhao G S, et al. Research on OpenFlow-based SDN technologies. *J Softw*, 2013, 24: 1078–1097 [左青云, 陈鸣, 赵广松, 等. 基于 OpenFlow 的 SDN 技术研究. *软件学报*, 2013, 24: 1078–1097]
- 2 Open Networking Foundation. Software-Defined Networking: The New Norm for Networks. ONF White Paper, 2012
- 3 Mckeown N, Anderson T, Balakrishnan H, et al. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Comput Commun Rev*, 2008, 38: 69–74
- 4 Open Networking Foundation. OpenFlow Switch Specification. Version 1.3.0. <https://www.opennetworking.org/>. 2012
- 5 Jiang W, Prasanna V K. Scalable packet classification on FPGA. *IEEE Trans Very Large Scale Integr (VLSI) Syst*, 2012, 20: 1668–1680
- 6 Meiners C R, Liu A X, Torng E, et al. Split: optimizing space, power, and throughput for tcam-based classification. In: *Proceedings of the Seventh ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, Brooklyn, 2011. 200–210
- 7 Liu Z J, Li Y, Su L, et al. TCAM-efficient flow table mapping scheme for OpenFlow multiple-table pipelines. *J Tsinghua Univ (Sci & Technol)*, 2014, 54: 437–442 [刘中金, 李勇, 苏厉, 等. TCAM 存储高效的 OpenFlow 多级流表映射机制. *清华大学学报 (自然科学版)*, 2014, 54: 437–442]
- 8 Ferkouss O E, Snaiki I, Mounaouar O, et al. A 100Gig network processor platform for openflow. In: *Proceedings of 7th International Conference on Network and Service Management (CNSM)*, Paris, 2011. 1–4
- 9 Bosshart P, Gibb G, Kim H S, et al. Forwarding metamorphosis: fast programmable match-action processing in hardware for SDN. In: *Proceedings of the ACM SIGCOMM*, Hong Kong, 2013. 99–110
- 10 Tanyingyong V, Hidell M, Sjödin P. Improving pc-based openflow switching performance. In: *IEEE 13th International Conference on High Performance Switching and Routing (HPSR)*, San Diego, 2012. 52–58
- 11 Curtis A R, Mogul J C, Tourrilhes J, et al. DevoFlow: scaling flow management for high-performance networks. In: *Proceedings of the SIGCOMM*, Toronto, 2011. 254–265
- 12 Ge J G, Chen Z, Wu Y L, et al. H-SOFT: a heuristic storage space optimisation algorithm for flow table of OpenFlow. *Concurr Comput Pract Exper*, 2015, 27: 3497–3509

An efficient implementation of storage and lookup for flow tables in OpenFlow

E YuePeng, CHEN Zhi*, GE JingGuo & WU YuLei

State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China

*E-mail: chen_zhi@iie.ac.cn

Abstract The storage and lookup of OpenFlow flow tables still faces many challenges such as the increasing number of fields in flow inlets, the explosive growth of flow scales, and difficulties in achieving wired-speed processing of ports in OpenFlow switches. To solve these problems, this paper presents an efficient storage and lookup method for flow tables in OpenFlow. This method divides flow tables into multiple sub-flow tables of smaller scale and fewer matching fields. Based on the different types of matching fields, sub-flow tables are split into multiple slices, which can be handled by different algorithms and hardware. The matching result of sub-flow tables can be obtained using the matching results of slices in sub-flow and summary tables. By splitting the flow and sub-flow tables, this method compresses the storage space of the flow table in OpenFlow switches. By optimizing of sub-flow table selection and lookup processes, this method reduces the cost of hardware resources when executing the lookup process. Extensive simulation experiments show that the proposed method can compress storage space by about 50% and can reduce hardware resource consumption to 80% when executing matching processes.

Keywords OpenFlow flow table, storage and lookup, sub-flow table, split sub-flow table, summary table

E YuePeng was born in 1976. He received a Ph.D. degree in computer software and theory at the Graduate University of the Chinese Academy of Sciences, China, in 2012. His research interests include future network architecture and network testbeds.

CHEN Zhi was born in 1989. He received an M.S. degree in computer application technology at the Chinese Academy of Sciences, China, in 2014. His main research interest is software-defined networks.

GE JingGuo was born in 1973. He received a Ph.D. degree in computer system architecture from the Institute of Computing Technology at the Chinese Academy of Sciences in 2003. His research focuses on future network architectures and protocols, wireless networks, and mobile computing.

WU YuLei was born in 1983. He received a Ph.D. degree in computing and mathematics in computer science from the University of Bradford, UK, in 2010. His research focuses on future network architectures and protocols, wireless networks, and mobile computing.