QuarkTable: Building Compact Forwarding Tables for Programmable Switches on Public Clouds

Jianyuan Lu†, Huaiyi Zhao†, Yu Qi, Yehao Feng, Xinyu Guo, Shengru Li, Enge Song, Xionglie Wei, Biao Lyu, Rong Wen, Shunmin Zhu
Alibaba Cloud
alibaba_cloud_network@alibaba-inc.com

ABSTRACT

Programmable switches have been recently proposed as dataplane solutions for public clouds. However, the conflict of limited on-chip memory and massive forwarding rules in cloud networks hinders the large-scale deployments. We argue that building compact forwarding tables for programmable switches is a viable option to this problem. In this paper, as a first step, we explore the feasibility of compact data structures for VPC routing tables (VRTs) and propose QuarkTable as a solution. The idea of QuarkTable builds upon the existence of redundancy in the prefixes of VRTs, supported by extensive analysis of real-world VRTs collected from six geographically distributed regions of Alibaba Cloud. By cutting the VRT into two partitions and encoding the upper part of the longer prefixes, QuarkTable effectively shortens the length of each entry and reduces the overall memory consumption. We demonstrate the effectiveness of QuarkTable by showing its proximity to the entropy bound of real-world VRTs. Experiments on six VRTs from Alibaba Cloud show practical memory savings up to 33.5% and 30.2% for SRAM and TCAM, respectively.

CCS CONCEPTS

• **Networks** → **Network algorithms**; **Programmable networks**; **Cloud computing**.

KEYWORDS

Programmable Switch, Routing Table, VPC, Public Cloud

ACM Reference Format:
Jianyuan Lu, Huaiyi Zhao, Yu Qi, Yehao Feng, Xinyu Guo, Shengru Li, Enge Song, Xionglie Wei, Biao Lyu, Rong Wen, Shunmin Zhu. 2024. QuarkTable: Building Compact Forwarding Tables for Programmable Switches on Public Clouds. In *The 8th Asia-Pacific Workshop on Networking (APNet 2024), August 3–4, 2024, Sydney, Australia.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3663408.3663415

†Both authors contributed equally to this work.

APNet 2024, August 3–4, 2024, Sydney, Australia

ACM ISBN 979-8-4007-1758-1/24/08
https://doi.org/10.1145/3663408.3663415

1 INTRODUCTION

Programmable ASICs [1, 3, 5, 11, 32] have emerged as alternatives to general-purpose servers or commodity switches in cloud dataplanes [12, 15, 20, 21, 25–27, 37]. These ASICs offer high throughput, low latency, and a balance between performance and flexibility [10, 19, 29]. However, their deployment in large-scale production networks has been hindered due to limited on-chip memory (SRAM, TCAM, etc.) [21, 27, 30]. In public clouds, storing millions of network configurations on a single ASIC processing pipeline is not feasible due to memory constraints.

Facing this problem, several studies have been proposed to extend the memory capability of programmable switches by RDMA [21, 29], on-board off-chip HBM [1], or slow-path DRAM [27]. Although alleviating the memory requirement, such methods will hurt the line-rate processing of programmable ASICs due to extra off-chip memory accesses. For instance, the proposed TEA [21] can only achieve about 9% of the line-rate throughput of programmable ASICs. Moreover, non-ideal traffic patterns (e.g., uniform rather than skewed), may cause a significant increase in off-chip memory accesses, resulting in even lower maximum throughput.

To solve the problem of limited memory while maintaining line-rate processing, we propose creating compact forwarding tables for programmable switches. We aim to reduce the memory required for storing table entries by encoding match fields into more condensed representations. In this paper, as a preliminary research, we choose the VPC routing table (VRT) as our first-step study, since the VRT is one of the largest forwarding tables on the public clouds. Through this study, we want to shed light on the principles of compressing all types of forwarding tables in programmable switches.

To achieve this goal, we perform comprehensive analysis of the VRTs obtained from six geographically distributed regions of Alibaba Cloud (§ 3). Our statistical analysis revealed three key insights:

- **VRTs exhibit sparsity.** A single VPC routing table contains a limited number of entries, *e.g.*, 80% of the VPCs have fewer than 10 routing entries, which makes traditional large routing table compression methods ineffective.
- **VRTs have longer prefix lengths than backbone routing tables.** The prefix masks of VRTs are scattered at 16-32, whereas the backbone routing tables have few prefix masks at 25-32.
- **VRTs have redundancy in matching keys.** The VPCs on the cloud mainly reuse three private CIDR blocks (*i.e.*, 10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16), resulting in many common prefixes in routing entries. For example, the three most popular prefixes were 10.0.0.0/24, 172.16.0.0/24, and 192.168.0.0/24). Further analysis shows that the upper bits of the prefixes exhibit significant overlapping and redundancy.

Building upon our key insights, we introduce QuarkTable, a compressed data structure for VRT on programmable switches. QuarkTable leverages the redundancy of matching keys to shorten the length of match fields. The idea is to select a cutting position, denoted by k, on the prefixes, and encode the upper k bits of the prefixes (along with the VPC identifiers) into shorter match keys. Prefixes with masks smaller than k remain unaltered. We demonstrate the effectiveness of QuarkTable by establishing its proximity to the entropy bound of real-world VRTs. Our experimental results indicate that QuarkTable can achieve up to 33.5% and 30.2% memory savings for SRAM and TCAM, respectively, thus highlighting its practical utility.

2 BACKGROUND AND MOTIVATION

2.1 VPC routing in public clouds

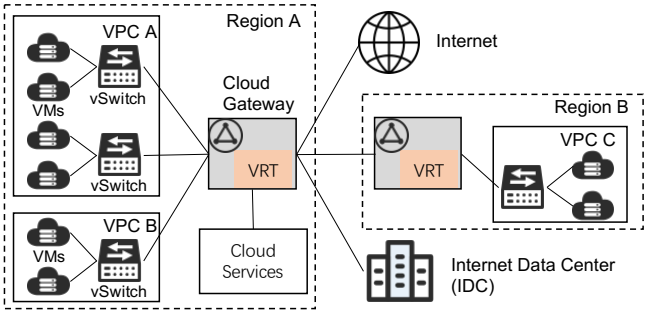

Figure 1: The role of VPC routing.

Virtual private cloud (VPC) is the *de facto* cloud network composition method. Each tenant can have multiple VPCs on the cloud to isolate network addresses and performance. The traffic from a VPC can have diverse forwarding destinations (*e.g.*, Internet, cloud services, cross-region network, *etc.*), therefore each VPC needs (at least) a routing table to tell cloud gateways where the data packets should be forwarded to. VPC Routing Table (VRT) is used to denote routing tables for all the VPCs in a region. Figure 1 shows the role of VPC routing. VM sends data traffic to Cloud Gateway by default for VRT lookup, finding nexthops based on the longest prefix match (LPM) principle. In Table 1, we summarize six typical forwarding scopes of VPC. In general, the north-south traffic (scopes of *cr-peer, Internet, express*) must pass through the cloud gateway for centralized network management and scheduling. By contrast, the east-west traffic (scopes of *local, az-peer, cloud-service*) can be offloaded to the on-host vSwitch to reduce the processing pressure of the cloud gateways. But no matter in which forwarding scope, the cloud gateway must store a full set of VRT entries to prevent breaking the LPM principle.

Table 2 shows an example of VRT. VNI is used to distinguish different VPCs[1]. The lookup process for a VRT is to first match the VNI, then perform LPM-lookup on the prefixes attached to that VNI. For example, the VRT lookup of (VPC-A, 10.0.1.1) returns the result of (cr-peer, VPC-C).

[1]Strictly speaking, VNI is used to distinguish different routing tables and may be larger than the number of VPCs since one VPC can have multiple routing tables.

Table 1: The typical route scopes in VPC.

Scope	Explanation
local	VM to VM within the same VPC.
az-peer	VM to VM in different VPCs across available zones within the same region.
cr-peer	VM to VM in different VPCs across regions.
Internet	VM to Internet access.
cloud-service	VM to services provided by cloud vendors (*e.g.*, data bases, log storage, big-data processing, *etc.*).
express	VM to tenants' private data centers (hybrid cloud).

Table 2: An example of VRTs.

VNI	Prefix	Scope	Nexthop
VPC-A	10.0.1.0/24	local	null
VPC-A	192.168.1.0/24	az-peer	VPC-B
VPC-A	10.0.1.0/25	cr-peer	VPC-C
VPC-B	192.168.2.0/24	express	IDC-A
VPC-B	192.168.3.0/25	cloud-service	Service-A

Table 3: The size of VRTs (# of routing entries) and estimated TCAM and SRAM resources consumption (bits) on programmable ASICs. (time: 2023/06/06 08:00)

Region	A	B	C	D	E	F	ASIC Pipeline
Size of VRT	1.16M	1.04M	342K	897K	787K	177K	
TCAM	74M	66M	21M	57M	50M	11M	~7M
TCAM(ALPM)	2.7M	2.4M	803K	2.1M	1.8M	415K	~7M
SRAM(ALPM)	212M	178M	58M	154M	135M	29M	~120M

2.2 The size of VPC routing tables

The public clouds can serve hundreds of thousands of tenants (or VPCs), therefore, the VRT usually contain a considerable amount of routing entries. We obtained six VRT snapshots from six regions in Alibaba Cloud, shown in Table 3. We can see that the VRTs of a cloud region can have over 1M routing entries. If all the routing entries are stored in programmable ASICs, it is estimated to use >70Mbits TCAM by pure TCAM-based LPM algorithm, or 2.7Mbits TCAM and >200Mbits SRAM by ALPM algorithm[2] [34]. However, according to latest stats of programmable ASICs, the TCAM and SRAM on each ASIC pipeline are only about 7Mbit and 120Mbit respectively [21, 26, 27].

To make the cloud gateways based on programmable ASICs possible, the solution in the industry is to split VRT entries to several gateway clusters (splitting by VPCs), *i.e.*, a cluster only stores a subset of VRT entries. But this solution will decrease the utilization of programmable switches, because the Tbps-level throughput of programmable ASICs cannot be fully utilized by serving a smaller set of VPCs. Additional to throughput waste, the VRT splitting method also faces the table overflow threat on the cloud. Since the routing entries can be dynamically added by tenants, once a tenant adds a large number of routes, there is a risk of table overflow of VRTs in these gateway clusters.

Towards the above problems, our goal is to build compact VRTs for programmable switches, so that more routing entries can be installed on programmable ASICs, thus increasing the throughput utilization and lowering the risk of table overflow.

[2]ALPM trades off SRAM for TCAM by splitting a prefix tree to many subtrees and storing the path to subtrees in TCAM and the subtrees themselves in SRAM.

2.3 Related Work and Design Space

Admittedly, there are extensive literatures on routing table compression. **1)Entry Aggregation** [16, 23, 33] exploits the possibility of using another, smaller set of entries while preserving routing semantics. However, these methods take advantageous of the prefix structure in a large FIB (e.g., > 500K entries). As for VRT which is a collection of small routing tables from all the VPCs, the compressibility is not high. **2)Bitmap Compression** [8, 13, 17, 35] uses compact bitmap representations for trie nodes. They are suitable for CPU memory hierarchy and not designed for modern programmable ASICs. **3)Direct Acycle Graphs (DAG)** [22, 28] generalizes the trie structure to DAGs to further save the memory of repeated subtrees in the trie. They are too complicated to be implemented on hardware. **4)Hash and BloomFilter** [14, 36] uses additional compact probabilistic data structures in the cache to guide the lookup process. However, the overall memory consumption does not decrease. **5)Trie Merging** [18, 24, 31] combines the tries of several large FIBs into a single trie to save memory in the trie structure at the cost of increased memory consumption per trie node. VRT consists of a large number of routing tables, which would make the memory overhead of the trie nodes unacceptable.

In summary, no existing work solves the problem of VRT compression in programmable switches. Considering the hardware characteristics, the design space consists of three dimensions. *1) Reducing the number of entries.* Following the entry aggregation approaches, novel algorithms may be devised to shrink the length of VRT, while they inevitably introduce tradeoffs between memory consumption and update performance. *2) Bit compression of table entries and table organizations.* If the tables can be smartly organized and the number of memory bits for each entry can be smaller, the overall memory consumption will be reduced. *3) Efficient table implementations.* Efficient implementation of common packet processing operations like LPM or exact match (EM) can reduce the resource consumption on hardware. Though some proprietary algorithms like ALPM[34] has been proposed to trade off SRAM for TCAM in LPM, it is witnessed that the memory occupancy is approaching its limit and not future-proof. Through comprehensive analysis of the VRTs in real world, we share insights on the *second* dimension, the possibility of bit-level compression for each entry, which is less touched by existing studies.

3 ANALYSIS OF VRTS

In this section, we perform extensive data mining on six real-world IPv4 VRTs from Alibaba Cloud to analyze their unique characteristics, especially the difference to traditional backbone FIBs of BGP routers. These characteristics are utilized in the design of QuarkTable in the next section.

3.1 Sparsity of Routing Entries for Each VPC

One straightforward difference of VRTs compared with backbone FIBs is that VRTs are actually a collection of small VPC routing tables. The number of entries in a VPC routing table is usually restricted by cloud vendors, *e.g.*, 200 entries in a single table (the restriction can be relaxed with permissions). Figure 2 shows the cumulative distribution of the number of entries each VPC has. It can be observed that most VPCs have very few entries. About

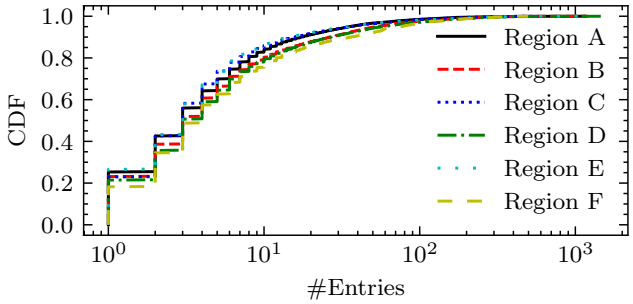

Figure 2: Cumulative distribution of the number of entries each VPC has. Note that X-axis is log-scale.

80% of the VPCs have fewer than 10 entries in all the six regions. Less than 5% of the VPCs have over 100 entries. The maximum number of entries one VPC has is 1509. Such sparsity of VRTs makes previous FIB compression methods ineffective, since they usually leverage the redundancy of a large and full prefix trees [8, 16–18, 23, 24, 31, 33, 35].

3.2 Distribution of Prefix Masks

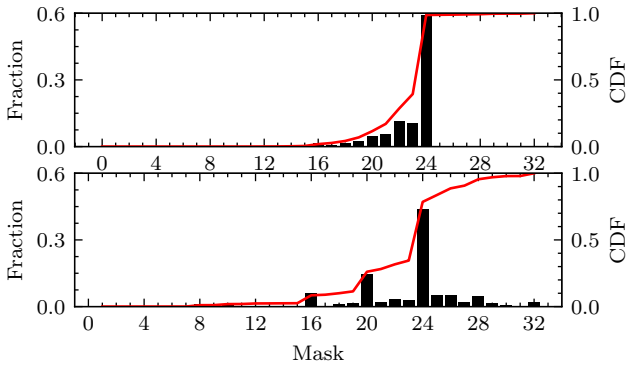

Figure 3: Distribution of prefix masks for backbone FIB (up) and VRT (down). Black bars indicate the fraction of entries (left Y-axis). Red lines show the CDF (right Y-axis).

The distribution of masks (prefix lengths) in VRT also differs from the backbone FIBs. Figure 3 compares the mask distribution of a VRT (from Region A) with a backbone FIB[3]. Masks in the backbone FIB are mainly centered at 16 ~ 24 with very few exceptions. However, masks of a VRT are scattered at 16 ~ 32, with peaks in 16,20 and 24. This indicates that manual routing configurations (by tenants) in public clouds show diversity in masks, mainly at lower 16 bits.

Interestingly, tenants have a tendency to use the "/24" mask in their routing entries, which coincide with the popular "/24" entries in backbone FIBs. This may come from the fact that many subnets are organized in human-friendly "/24" blocks since the IPs in a "/24" subnet are human readable and 255 hosts are usually sufficient for most cases. Moreover, though there are some entries with masks of

[3]FIB is collected from RouteView [6] on May 1, 2023.

8, 10, 12, the total fraction is insignificant. For example, the number of entries with a mask ≤ 12 is $\sim 2\%$. This gives us the possibility that divide the VRT into two disjoint partitions where entries with masks larger than a threshold are classified into a majority partition and other entries into a minor partition (§ 4). The minor partition with smaller masks contains only a small fraction of total entries.

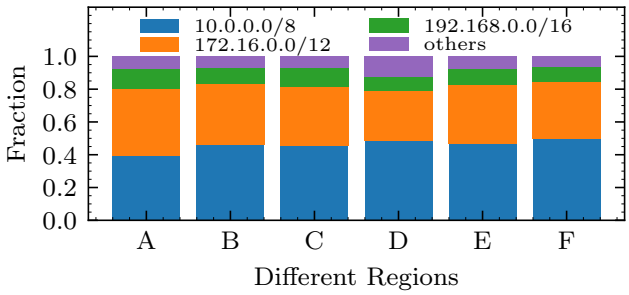

Figure 4: VPC CIDR-block distribution on public clouds.

3.3 Redundancy in Routing Entries

Considering VPC routing tables in a region as a whole, we notice the redundancy of routing entries across different VPCs.

IP Address Overlapping of VPCs. In public clouds, tenants are usually given only a few options of their VPCs' CIDR blocks (*e.g.*, 10.0.0.0/8, 172.16.0.0/12 and 192.168.0.0/16 [2, 7, 9], following RFC 1918 [4]). Most routing entries of a VPC should be within the CIDR block of the VPC itself or the peering VPC (*e.g.*, az-peer or cr-peer communications). Other CIDR blocks for VPCs are allowed but requiring special permissions.

Figure 4 shows the fractions of different CIDR blocks. Almost 80% of VPCs are in 10.0.0.0/8 and 172.16.0.0/12 blocks across all regions. A simple approach to reduce memory footprint would be splitting VRT based on which CIDR block the entry is in, and only storing the lower bits of the entries in that block. But this splitting is coarse-grained and requires maintenance of several tables.

Long-tail Distribution of Prefixes. Since there are a few popular CIDR blocks (*e.g.*, 10.0.0.0/8 and 172.16.0.0/12), we anticipate that there should be some popular prefixes that are favored by tenants. We first count how many entries are associated with each unique prefix. Then, we sort the prefixes so that the prefixes with more entries have smaller indexes. (That is, the prefix with index 1 has the most entries.) For each region, the number of entries for each prefix is normalized (i.e., divided) by the maximum value of that region so that we can see the trends for different regions in one figure.

Figure 5 shows the results.[4] The maximum value for the six regions are 19866, 12597, 5515, 10438, 10688, 2318. Some popular prefixes are 10.0.0.0/24, 192.168.0.0/24 and 172.16.0.0/24 for subnet routing, which coincide with the preference for the "/24" mask in Figure 3. Noting the log scale of x-axis, there is a clear long-tail distribution of prefixes for all six regions. The top 50 popular prefixes have a very large number of entries associated with them, whereas the vast majority of prefixes have only a few entries. For example, in region F, 90% of the prefixes have less than 10 associated entries.

[4]All VPCs have a cloud service entry with the same prefix, which we remove beforehand.

Sparsity and Entropy of k-blocks. Taking a step further, we consider the overlapping in upper bits of the prefixes. We call the upper k bits of a prefix the *k-block* if the prefix has a mask $> k$. We study how the distribution of k-blocks in the VRT varies with k. Figure 6 is a compact illustration of the entropy and the number of unique k-blocks in region A, described by the number of bits. The "index" line shows the number of bits required if we encode each unique k-block to a unique integer (i.e., index(k)= $\lceil \log_2(\#\text{unique } k\text{-block}) \rceil$) where $\lceil x \rceil$ is the minimum integer larger than or equal to x. The "entropy" line is calculated as entropy(k)= $-\sum_{P(i)} P(i) \log_2 P(i)$ where $P(i)$ is the probability that k-block takes value i.

The k-blocks become more and more sparse compared with the address space for $k > 8$. This make sense because only a few unique k-blocks are actually used compared with the huge address space. (i.e., k bits can represent as many as 2^k different k-blocks, whereas the number of unique k-blocks is much smaller.) Therefore, the naïve representation that uses k bits to store the k-block is wasteful. Encoding approaches may be applied. In addition, the entropy is even smaller, which shows the non-uniform distribution of the k-blocks. An aggressive compressing approach would be to compress the k-block to the entropy using variable-length encoding mechanisms. However, these approaches do not take the bits of VNI into consideration, and may violate the memory alignment of modern hardware architecture.

Sparsity of (VNI, k-block) pairs. In this section, we together consider the k-blocks with VNIs. Figure 7 shows the information of (VNI, k-block) pairs in region A. The definition of "index" and "entropy" is similar as in Figure 6.

On $k = 0$, the number of unique (VNI, k-block) pairs, which is essentially the number of unique VNIs, can be encoded by 15-bit integers. The entropy on $k = 0$ is about 14 bits, which coincide with the uneven distribution of number of entries each VPC has (§ 3.1). When k gets larger, the "index" and "entropy" changes slowly, staying around $12 - 18$ bits, with "index" being 1 to 2 bits larger than "entropy". This is different from the k-blocks, where "index" and "entropy" show apparent changes with k. This observation is significant in that simply encoding each unique (VNI, k-block) pair as an integer yields a compression rate that approximates the theoretical entropy without the need for a complex variable-length encoding mechanism, which is advantageous for implementation.In the next sections, we will show how we use this property for the design of QuarkTable.

4 DESIGN OF QUARKTABLE

QuarkTable is an organization of a series of match-action tables with guarantees of the LPM. The design of QuarkTable is supported by some key observations from § 3: *1)* Simply employing existing compression approaches for a single (or several) large FIB(s) would not work well because VRT consists of many small tables with only a few entries, where compressibility is small due to the less address overlapping in individual table (§ 3.1). *2)* The fraction of entries with mask <= k is not too large for a small k (e.g., < 5% for

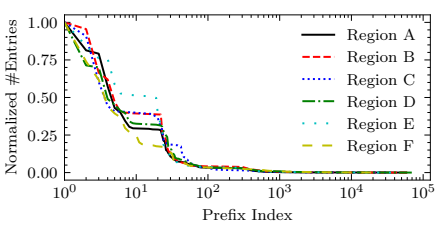

Figure 5: Normalized number of entries each prefix has. Note the log scale of x-axis.

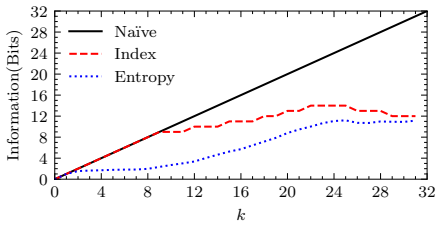

Figure 6: Information of k-blocks in region A by different representations.

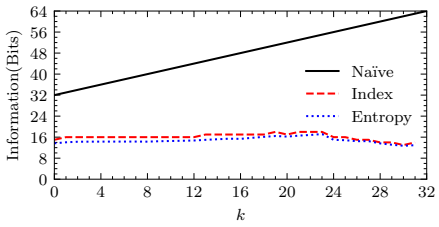

Figure 7: Information of (VNI, k-block) pairs in region A by different representations.

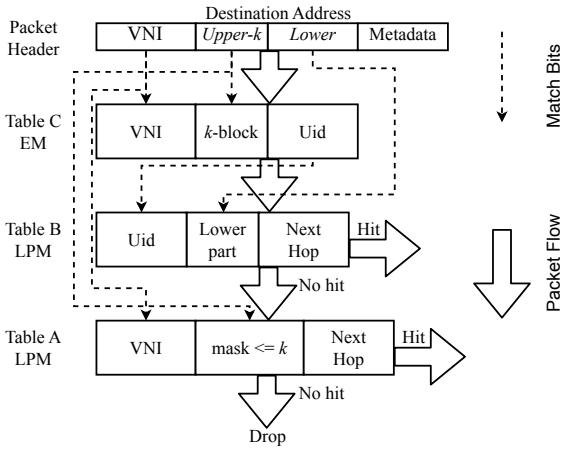

Figure 8: Packet processing flow. The left two blocks of a table is the key. The rightmost block of a table is the action.

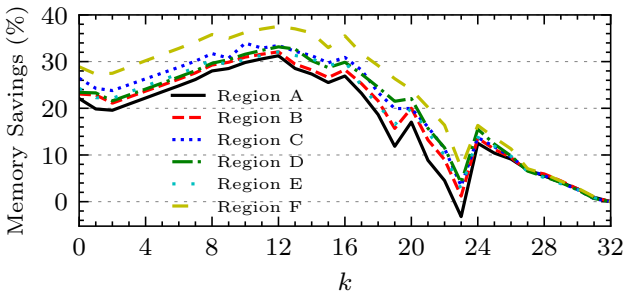

Figure 9: Memory savings in terms of k.

$k = 12$) (§ 3.2). *3)* The entries with mask $> k$ have the potential to be compressed on their k-blocks since the number of unique (VNI, k-block) pairs are small compared to the bits they occupied by the naïve implementation (§ 3.3).

We cut the VRT into two halves at position k. Entries with mask $<= k$ go to one half (say Table A) and the others go to Table B. For Table B, we encode the entries at position k and use an auxiliary encoding table (Table C) to map each unique (VNI, k-block) pair to a Universal id (Uid) of length m. Uid will be concatenated with the lower part of the prefix (i.e., the prefix without its k-block) as the key for Table B. Note that by design, one Uid may appear at several entries in Table B. As for Table A, since the prefixes all have masks $<= k$, we can throw away the bits that is larger than k to further save memory. All the entries are precomputed in the control plane and programmed to the dataplane. When k is fixed, incremental updates can be easily supported by update corresponding tables based on the prefix lengths. For extreme cases when the number of unique (VNI, k-block) pairs exceeds the maximum number Uid can encode (i.e., larger than 2^m), table reallocation and recomputation are required to increase the length of Uid.

As shown in Figure 8, the overall packet flow will be as follows. When a packet arrives, the three main components namely *VNI*, *Upper-k* and *Lower* are extracted from its header. VNI and Upper-k are first used to perform exact match (EM) in Table C to find its corresponding Uid. Then, Uid and Lower will be used to perform LPM in Table B. If there is a match, the packet is sent to the next hop. If there is no match, VNI and Upper-c are used to perform LPM in Table A. If there is a match, the packet is sent to the next hop. If there is no match, the packet is dropped. Note that a packet not matching any entry of Table C may be directly sent to Table A. But we remove this additional logic by adding a *default entry* in Table C that maps to Uid 0 and make sure Uid 0 does not exist in table B. One can check that the processing flow does not violate the LPM principle because longer prefixes are checked before possible shorter prefixes. Moreover, due to the match-action structure in programmable switches, latency and throughput penalty introduced by our design is negligible.

5 EVALUATIONS

5.1 Experimental Setup

We collect VRTs from six geographically distributed regions from a leading cloud vendor. VE is applied to these six VRTs and we calculate the theoretical memory savings. Furthermore, we also implement the match-action table and processing logic in Tofino [3] and report the SRAM and TCAM consumption.

5.2 Compression Ratios for Six VRTs

Different VRTs may have different distributions in terms of VNI and prefixes. Therefore, the best positions to cut (i.e., k) at each VRT may differ from one another. Intuitively, if k is too small, then encoding (VNI, k-block) pair is less appealing since the memory

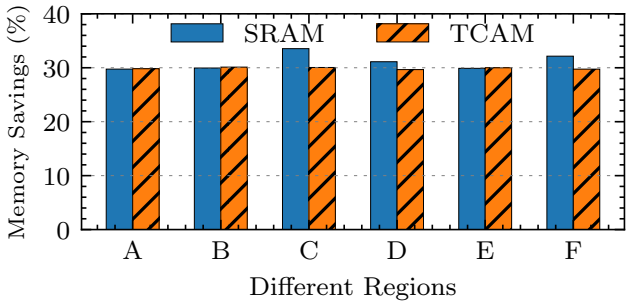

Figure 10: Memory comparison with and without VE.

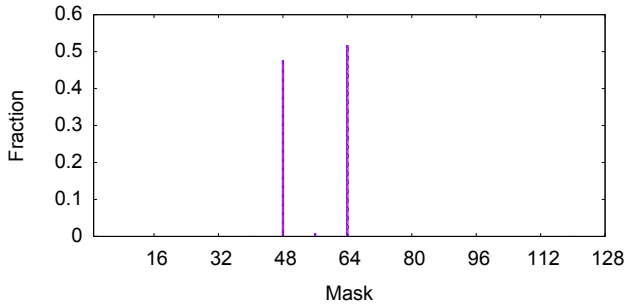

Figure 11: Distribution of prefix masks for IPv6 VRT (from Region A of Alibaba Cloud).

savings in bits is not high compared with naïve implementation (recall Figure 7). If k is too large, then the number of entries with masks $> k$ will be small (recall Figure 3 where only $\sim$ 20% of the entries have masks larger than 24), which makes the encoding irrelevant since many entries go to Table A.

In Figure 9, we illustrate how the memory savings change by the choice of k for the six VRTs. Note that we do not explicitly show the memory consumption of nexthops in this section because we show here the effectiveness of VE for the compression of match keys, and the compression of nexthops are considered out of scope. Region C has the largest memory saving of 34.0% on $k = 10$ which is slightly higher than 33.4% on $k = 12$. Other regions achieve the best results on $k = 12$. $k = 12$ is a sweat spot because the number of entries that have masks > 12 is large and the "index" encoding of (VNI, k-block) pairs can save a lot of bits from the naïve implementation, while keeping the encoding table small. The lengths of the Uid (i.e., m) for each region when the memory savings are maximized are 16,16,14,16,16,14. It shows the trend that smaller m may lead to large memory savings because the memory consumption of Table B and Table C can both be reduces.

5.3 Implementations

We implement a prototype on Tofino to validate our ideas. The Tofino chip has 12 stages in one pipeline. QuickTable are realized through a series of match-action units which span 6 stages, compared to 5 stages of original method. The additional one stage is to store the (VNI, k-block) mapping table.

We show the practical memory savings for six regions in Figure 10. As TCAM resources on Tofino cannot hold all routing entries of VRT, ALPM algorithm [34] is used to trade off SRAM for TCAM. We can notice that the overall memory savings are around 30% (up to 33.5% and 30.2% for SRAM and TCAM, respectively), slightly lower than the theoretical value in Section 5.2 due to the internal memory alignment. Because tables on Tofino have limitations that the underlying memory are aligned at a minimal element, word. Nonetheless, the practical memory savings are considered significant in our real-world deployment.

6 CONCLUSION AND FUTURE DIRECTIONS

This paper sheds light on the problem of forwarding table compression on public cloud. We extensively analyze the structure of VRTs including the entry sparsity for each VPC, mask distributions, and entropy of (VNI, k-block) pairs. We draw insights from the real-world data and develop a compact data structure to practically compressing the VRT by over 30%. Nonetheless, the proposed method is by no means complete or optimal. Further optimizations are encouraged in the following directions.

Entry reduction. Traditional algorithms like ORTC [16] are effective for reducing the number of entries on a large FIB. In the face of VRT, which consists of a large number of small FIBs, how to reduce the number of entries to near optimality is worth future study.

Variable-length coding. Though the fixed-length encoding mechanism proposed in this paper are proven effective, the encoded length is not optimal and the memory waste of the encoding table itself can be prohibitively large when the number of encoding entries increases. Therefore, more sophisticated variable-length coding algorithms (perhaps without encoding tables) can be employed to further reduce the overall memory consumption.

Smart realization of matches in programmable switches. EM and LPM are common operations in network packet processing. ALPM has demonstrated how SRAM can be used to trade off TCAM consumption for LPM operations. However, its effectiveness depends heavily on the distribution of the prefixes, which are not known beforehand. Elastic distribution-aware mechanisms are encouraged for higher memory savings and generality.

Support for IPv6 VRT. In the Tofino-based cloud gateways [27] of Alibaba Cloud, the IPv6 VRT consumes nearly equal on-chip memory consumption to the IPv4 VRT. Therefore, it is also important to build compact IPv6 VRTs to reduce memory consumption. We have a key observation to demonstrate that the proposed idea of QuarkTable can be extended to IPv6 VRTs. As shown in Figure 11, the prefix masks of IPv6 VRT concentrate on 48 and 64. Both the two masks are longer than IPv4 prefix masks, which means that the IPv6 VRT has more space to be compressed. Future work could be done to validate such ideas.

We believe that the statistical and analytical results obtained from real-world VRTs, along with QuarkTable, represent a significant advancement towards compressing forwarding tables in programmable switches, a less touched area that merits further researches.

REFERENCES

[1] [n. d.]. Cisco Silicon One. https://www.cisco.com/c/en/us/solutions/silicon-one.html.

[2] [n. d.]. Create a VPC with an IPv4 CIDR block - Virtual Private Cloud - Alibaba Cloud Documentation Center. https://www.alibabacloud.com/help/en/virtual-private-cloud/latest/create-an-ipv4-vpc.

[3] [n. d.]. Intel® Tofino™ Series Programmable Ethernet Switch ASIC. https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-series.html.

[4] [n. d.]. RFC 1918 - Address Allocation for Private Internets (RFC1918). http://www.faqs.org/rfcs/rfc1918.html.

[5] [n. d.]. Trident4 / BCM56880 Series. https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs/bcm56880-series.

[6] [n. d.]. University of Oregon Route Views Project. http://www.routeviews.org/routeviews/.

[7] [n. d.]. VPC CIDR blocks - Amazon Virtual Private Cloud. https://docs.aws.amazon.com/vpc/latest/userguide/vpc-cidr-blocks.html.

[8] Hirochika Asai and Yasuhiro Ohara. 2015. Poptrie: A Compressed Trie with Population Count for Fast and Scalable Software IP Routing Table Lookup. In *ACM SIGCOMM*. https://doi.org/10.1145/2785956.2787474

[9] asudbring. 2023. Azure Virtual Network - Concepts and best practices. https://learn.microsoft.com/en-us/azure/virtual-network/concepts-and-best-practices

[10] Ran Ben Basat, Sivaramakrishnan Ramanathan, Yuliang Li, Gianni Antichi, Minian Yu, and Michael Mitzenmacher. 2020. PINT: Probabilistic in-band network telemetry. In *ACM SIGCOMM*. 662–680.

[11] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and others. 2014. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review* 44, 3 (2014), 87–95.

[12] Qizhe Cai, Shubham Chaudhary, Midhul Vuppalapati, Jaehyun Hwang, and Rachit Agarwal. 2021. Understanding host network stack overheads. In *ACM SIGCOMM*. 65–77.

[13] Mikael Degermark, Andrej Brodnik, Svante Carlsson, and Stephen Pink. 1997. Small forwarding tables for fast routing lookups. *ACM SIGCOMM Computer Communication Review* 27, 4 (1997), 3–14. https://doi.org/10.1145/263109.263133

[14] Sarang Dharmapurikar, Praveen Krishnamurthy, and David E Taylor. 2003. Longest prefix matching using bloom filters. In *ACM SIGCOMM*.

[15] Mihai Dobrescu, Katerina Argyraki, and Sylvia Ratnasamy. 2012. Toward predictable performance in software packet-processing platforms. In *USENIX NSDI*. 141–154.

[16] R.P. Draves, C. King, S. Venkatachary, and B.D. Zill. 1999. Constructing optimal IP routing tables. In *IEEE INFOCOM*. https://doi.org/10.1109/INFCOM.1999.749256

[17] Will Eatherton, George Varghese, and Zubin Dittia. 2004. Tree bitmap: hardware/software IP lookups with incremental updates. *ACM SIGCOMM Computer Communication Review* 34, 2 (2004), 97–122. https://doi.org/10.1145/997150.997160

[18] Jing Fu and Jennifer Rexford. 2008. Efficient IP-address lookup with a shared forwarding table for multiple virtual routers. In *ACM CoNEXT*. https://doi.org/10.1145/1544012.1544033

[19] Thomas Holterbach, Edgar Costa Molero, Maria Apostolaki, Alberto Dainotti, Stefano Vissicchio, and Laurent Vanbever. 2019. Blink: Fast connectivity recovery entirely in the data plane. In *USENIX NSDI*. 161–176.

[20] Hongjing Huang, Yingtao Li, Jie Sun, Xueying Zhu, Jie Zhang, Liang Luo, Jialin Li, and Zeke Wang. 2023. P4SGD: Programmable Switch Enhanced Model-Parallel Training on Generalized Linear Models on Distributed FPGAs. *IEEE Transactions on Parallel and Distributed Systems* (2023).

[21] Daehyeok Kim, Zaoxing Liu, Yibo Zhu, Changhoon Kim, Jeongkeun Lee, Vyas Sekar, and Srinivasan Seshan. 2020. TEA: Enabling State-Intensive Network Functions on Programmable Switches. In *ACM SIGCOMM*. https://doi.org/10.1145/3387514.3405855

[22] Attila Korosi, Janos Tapolcai, Bence Mihalka, Gabor Meszaros, and Gabor Retvari. 2014. Compressing IP Forwarding Tables: Realizing Information-Theoretical Space Bounds and Fast Lookups Simultaneously. In *IEEE ICNP*. https://doi.org/10.1109/ICNP.2014.55

[23] Yaoqing Liu, Beichuan Zhang, and Lan Wang. 2013. FIFA: Fast incremental FIB aggregation. In *IEEE INFOCOM*. https://doi.org/10.1109/INFCOM.2013.6566913

[24] Layong Luo, Gaogang Xie, Kave Salamatian, Steve Uhlig, Laurent Mathy, and Yingke Xie. 2013. A trie merging approach with incremental updates for virtual routers. In *IEEE INFOCOM*. https://doi.org/10.1109/INFCOM.2013.6566914

[25] Sarah McClure, Zeke Medley, Deepak Bansal, Karthick Jayaraman, Ashok Narayanan, Jitendra Padhye, Sylvia Ratnasamy, Anees Shaikh, and Rishabh Tewari. 2023. Invisinets: Removing Networking from Cloud Networks. In *USENIX USDI*. 479–496.

[26] Rui Miao, Hongyi Zeng, Changhoon Kim, Jeongkeun Lee, and Minlan Yu. 2017. SilkRoad: Making Stateful Layer-4 Load Balancing Fast and Cheap Using Switching ASICs. In *ACM SIGCOMM*. https://doi.org/10.1145/3098822.3098824

[27] Tian Pan, Nianbing Yu, Chenhao Jia, Jianwen Pi, Liang Xu, Yisong Qiao, Zhiguo Li, Kun Liu, Jie Lu, Jianyuan Lu, Enge Song, Jiao Zhang, Tao Huang, and Shunmin Zhu. 2021. Sailfish: accelerating cloud-scale multi-tenant multi-service gateways with programmable switches. In *ACM SIGCOMM*. https://doi.org/10.1145/3452296.3472889

[28] Gábor Rétvári, János Tapolcai, Attila K\Horösi, András Majdán, and Zalán Heszberger. 2013. Compressing IP forwarding tables: Towards entropy bounds and beyond. In *ACM SIGCOMM*.

[29] Mariano Scazzariello, Tommaso Caiazzi, Hamid Ghasemirahni, Tom Barbette, Dejan Kostić, and Marco Chiesa. 2023. A High-Speed Stateful Packet Processing Approach for Tbps Programmable Switches. In *USENIX NSDI*. https://www.usenix.org/conference/nsdi23/presentation/scazzariello

[30] Naveen Kr Sharma, Antoine Kaufmann, Thomas E Anderson, Arvind Krishnamurthy, Jacob Nelson, and Simon Peter. 2017. Evaluating the Power of Flexible Packet Processing for Network Resource Allocation.. In *USENIX NSDI*. 67–82.

[31] Haoyu Song, Murali Kodialam, Fang Hao, and T. V. Lakshman. 2010. Building Scalable Virtual Routers with Trie Braiding. In *IEEE INFOCOM*. https://doi.org/10.1109/INFCOM.2010.5461960

[32] Hardik Soni, Myriana Rifai, Praveen Kumar, Ryan Doenges, and Nate Foster. 2020. Composing dataplane programs with μP4. In *ACM SIGCOMM*. 329–343.

[33] Zartash Afzal Uzmi, Markus Nebel, Ahsan Tariq, Sana Jawad, Ruichuan Chen, Aman Shaikh, Jia Wang, and Paul Francis. 2011. SMALTA: practical and near-optimal FIB aggregation. In *ACM CoNEXT*. https://doi.org/10.1145/2079296.2079325

[34] Henry Wang. 2019. Algorithmic longest prefix matching in programmable switch. US Patent 10,511,532.

[35] Tong Yang, Gaogang Xie, YanBiao Li, Qiaobin Fu, Alex X. Liu, Qi Li, and Laurent Mathy. 2014. Guarantee IP lookup performance with FIB explosion. In *ACM SIGCOMM*. https://doi.org/10.1145/2619239.2626297

[36] Marko Zec, Luigi Rizzo, and Miljenko Mikuc. 2012. DXR: towards a billion routing lookups per second in software. *ACM SIGCOMM Computer Communication Review* 42, 5 (2012), 29–36.

[37] Chaoliang Zeng, Layong Luo, Teng Zhang, Zilong Wang, Luyang Li, Wenchen Han, Nan Chen, Lebing Wan, Lichao Liu, Zhipeng Ding, et al. 2022. Tiara: A scalable and efficient hardware acceleration architecture for stateful layer-4 load balancing. In *USENIX NSDI*.