©2014 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works

# Wildcard Compression of Inter-Domain Routing Tables for OpenFlow-Based Software-Defined Networking

Wolfgang Braun and Michael Menth University of Tübingen, Department of Computer Science, Germany Email: {wolfgang.braun,menth}@uni-tuebingen.de

Abstract—In this paper we consider carrier networks using only OpenFlow switches instead of IP routers. Accommodating the full forwarding information base (FIB) of IP routers in the switches is difficult because the BGP routing tables in the defaultfree zone currently contain about 500,000 entries and switches have only little capacity in their fast and expensive TCAM memory.

The objective of this paper is the compression of the FIB in acceptable time to minimize the TCAM requirements of switches. The benchmark is simple prefix aggregation as it is common in IP networks where longest-prefix matching is applied. In contrast, OpenFlow-based switches can match general wildcard expressions with priorities. Starting from a minimum-size prefix-based FIB, we further compress that FIB by allowing general wildcard expressions utilizing the Espresso heuristic that is commonly used for logic minimization. As the computation time of Espresso is challenging for large inputs, we provide means to trade computation time against compression efficiency. Our results show that today's FIB sizes can be reduced by 17% saving up to 40,000 entries and the compression time can be limited to 1 - 2 s sacrificing only 1% - 2% compression ratio.

Index Terms—Software-Defined Networking, OpenFlow, Compression, Routing Tables, TCAM

#### I. INTRODUCTION

Software-defined networking (SDN) and OpenFlow [1] have gained a lot of attention in the last years due to their flexibility, programmability, and ease to introduce new features and services into the network. OpenFlow was mostly deployed in enterprise, campus, and data center networks. More recently, the interest in SDN for carrier networks has greatly increased. Several researchers have analyzed the applicability of that idea [2] and various proposals have been suggested. They are appealing for cost and operational reasons.

An important task in carrier networks is inter-domain routing which is performed by the Border Gateway Protocol (BGP). Current BGP routing tables hold more than 500,000 entries [3]. That information needs to be installed in OpenFlow switches if misses for flow tables entries should be avoided. However, forwarding tables in OpenFlow switches are usually smaller than those of core routers so that installing all necessary forwarding information is a challenge.

This work has been supported by the German Federal Ministry of Education and Research (BMBF) under support code 16BP12307 (EUREKA-Project SASER). The authors alone are responsible for the content of the paper. A reason for the small forwarding tables in OpenFlow switches is that they are implemented in Ternary Content Addressable Memory (TCAM). TCAM allows packet matching in constant time and outperforms software-based packet matching. However, TCAM has high power consumption and large footprint, suffers from heat generation [4], and TCAM is expensive compared to other memory such as SRAM [5]. Therefore, vendors tend to install small TCAMs into OpenFlow switches, e.g., many OpenFlow switches can handle in practice between 10K and 40K flow entries.

While conventional routers use only prefix-based match fields with longest prefix match (LPM) in their forwarding tables, OpenFlow switches support general wildcard-capable match fields with priorities. Since general wildcard expressions with priorities are more flexible than prefix-based expressions with LPM, compression techniques could reduce the number flow table entries for OpenFlow switches so that they can be better accommodated in the limited TCAM.

In this paper, we propose to use the Espresso heuristic [6] from logic minimization to compress a minimum-size set of prefix-based match fields generated by the ORTC algorithm [7] into a set of match fields with general wildcards. As the runtime of the Espresso scales exponentially with the input size, we suggest methods to keep the time for the compression of routing tables low. We apply this method to BGP routing tables of 2013 to quantify the compression potential for flow tables entries in carrier networks. We further analyze the structure of the compressed outcome and the tradeoff between runtime of the compression algorithm and its compression rate.

The remainder of this paper is structured as follows. In Section II we discuss a concept for OpenFlow-based SDN in carrier networks that sets the scene for our study. In Section III we review related work with regard to compression of table entries used in network devices. We explain our compression method including speedup options in Section IV. In Section V we apply the algorithms to BGP routing information, and analyze the compression potential and computation efforts. Section VI concludes this paper.

#### II. OPENFLOW-BASED SDN FOR CARRIER NETWORKS

Carrier networks require inter-domain routing information. Usually, one or several BGP routers serve as BGP speakers and collect such information via BGP from neighboring domains.

©Third European Workshop on Software Defined Networks (EWSDN), 2014, Budapest, Hungary, September 2014



Fig. 1: Inter-domain routing with OpenFlow-based SDN: a commodity server acts as a BGP speaker and collects the Border Gateway Protocol (BGP) runs in the control plane on commodity servers as SDN application or controller module.

To handle large routing tables and frequent BGP updates, such routers require additional CPU and memory resources as well as additional space in their forwarding table which increases their cost.

In addition, state distribution within a single carrier network can cause scalability and stability issues and, thus, the authors of [8] proposed a centralized BGP control plane called Routing Control Platforms (RCP) in 2005. This idea was recently reconsidered for OpenFlow-based SDN in [9] and [10]. Figure 1 illustrates that idea. Expensive routers are removed from the network and replaced by simpler and less expensive OpenFlow switches. The switches forward BGP control messages such as prefix announcements and withdrawals to the OpenFlow controller. The controller passes them to the BGP application. This application interacts with BGP speakers of neighboring domains and performs the BGP decision process. That means, it combines the inter-domain routing information from BGP and intra-domain routing information into a forwarding table for each switch in the network such that each prefix is associated with an appropriate next-hop. That information is generally compressed using prefix aggregation before being configured by the controller in the switches.

It is likely that the forwarding information cannot be completely included in a switch. Therefore, the authors of [11] take advantage of the fact that approximately 80% of the Internet traffic is caused by 20% of the IP prefixes so that the frequently used prefixes can always be installed in TCAM. Less frequently used prefixes are kept in the control plane but are not installed in the switch. If the switch receives a data packet for which it has no matching prefix, it forwards the packet to the controller that knows how to handle that packet. This approach generally degrades network performance.

The approach presented above may be improved through

compression of forwarding tables by combining appropriate forwarding rules through general wildcards in match fields that are supported by OpenFlow switches. In an SDN context this is easily feasibly as the control plane consists of cheap commodity hardware so that sufficient CPU power is available for computation-intensive wildcard compression. As a result, more entries fit into the limited TCAM and less traffic needs to be forwarded via the controller.

In [12], a software-defined Internet exchange point (SDX) is presented as well as a method to reduce the BGP state using virtual next-hops and addresses. Our method is orthogonal to this approach.

Routing table updates must be propagated fast so that there may not be enough time for the compression of the entire FIB after the routing change. The authors of [13] showed that the majority of the routing table entries are quite stable. Routing table updates are still frequent so that a compression method should be able to perform updates quickly which may happen incrementally on the existing compressed table. This also holds for the SDN context.

## III. RELATED WORK

One of the most important routing table compression method was developed in 1999. It is called Optimal Routing Table Constructor (ORTC) [7] and calculates minimum-size routing tables but does not support incremental routing table updates. The authors of [14] achieve fast incremental updates of compressed routing tables but only sub-optimal compression. Incremental updates are also considered in [15] and [16]; the latter is directly based on ORTC.

A different approach is the compression of the data structure that represents the routing table. In [17], entropy bound compression is investigated to significantly reduce the size of the routing table through shared data structures.

Compression of ACLs or firewall rules is similar to routing table compression. Entries of Access Control Lists (ACLs) may match more than a single field; they typically support source and destination address, port number, type, etc. In [18], a compression method based on decision trees and hypercuts is presented. The TCAM Razor [19] is able to minimize lists of ACLs into smaller lists. The authors also proposed a more efficient compressor, called firewall compressor [20], that outperforms the TCAM razor. These methods do not consider the use of general wildcard expressions.

Wildcard-based compression for ACLs is considered in [21]. It uses a heuristic that runs in polynomial time which is based on two functions: bit swapping and bit merging. However, this method is less efficient than other ACL compressors. Wildcard compression of ACLs using logic minimization is considered in [22]. It achieves a significant compression ratio for both artificial and real firewall rules but it is rather time-consuming. Compression of ACLs differs from compression of FIB entries. ACL rules are more complex than FIB entries, but then number of entries in a FIB in carrier networks exceeds the number of entries in ACLs by far: typical ACLs contain a few thousand entries while current BGP RIBs contain more than 500,000



Fig. 2: The prefix tree for the routing table in Table Ia.

entries. In addition, compression of FIBs has stricter runtime constraints because FIB updates must be processed faster than ACL updates.

Routing tables of virtual routers are commonly stored in the hypervisor. Running multiple virtual routers on a single physical router increases its TCAM requirement. To limit that requirement, the authors of [5] compress multiple virtual FIBs using merged prefix trees. Up to 14 virtual FIBs can be stored in one TCAM and incremental routing updates are possible.

### IV. METHODOLOGY

In this section we briefly describe the ORTC algorithm that provides the input and benchmark for our proposal. Then, we discuss longest-prefix matching (LPM) that is used in IP networks and wildcard matching using priorities. We present how prefixes can be compressed using the Espresso logic minimization heuristic. Finally, we discuss how Espresso can be incrementally applied and how routing table updates can be performed.

## A. Prefixes, Forwarding Tables, and Longest-Prefix Match

An IPv4 prefix p consists of a 32-bit IP address and a 32-bit network mask. The network mask starts with a series of ones in the most significant bits followed by a series of zeroes for the least significant bits. The number of leading ones is the length length(p) of the prefix p which is usually indicated by the notation /length(p). As an alternative, a prefix can also be represented by a match field containing the length(p) bits of the IP address followed by wildcard symbols '\*', also known as 'don't cares', for each remaining position in the address. Table Ia contains prefixes in that notation. A prefix matches an address if the leading length(p) bits of the prefix address equal the first length(p) bits of the address to be matched.

A routing or forwarding table associates a set of prefixes with a next-hop. It can be represented by a prefix tree. Figure 2 illustrates the prefix tree for the routing table in Table Ia. Each entry in the routing table is represented by a labeled node in the tree whereby the label indicates the next-hop. The prefix for such an entry is composed of the numbers on the edges from the root to the corresponding node. The root of the tree corresponds to the least specific prefix \*\*\*. If a node is empty, the corresponding routing tables has no entry associated with that prefix.

Forwarding in IP networks uses longest prefix matching (LPM). If multiple entries in a forwarding table match an

FIB entry	Next-hop	FIB entry	Next-hop	Priority
***	1	***	1	0
101		101	1	3
01*	3	1**	2	1
11*	3	*1*	3	2
(a) Prefix-base	ed forwarding	(b) Forwardin wildcard expre	ng tables us essions.	ing general

TABLE I: Two forwarding tables with identical forwarding behavior.

address, the most-specific one is used, i.e., the one with the longest prefix. To perform that procedure, the prefix tree is traversed according to the bits in the address as far as possible. Thereby, a '0' or '1' in the address denotes a move to the left or right child in the tree. The last visited labeled node corresponds to the most-specific forwarding entry.

The ORTC algorithm compresses a routing table with general prefixes into a minimum-size prefix-based table using prefix trees. Details are given in [7].

# B. Wildcards Expressions and Logic Minimization

A wildcard expression w is a match field with wildcards '\*' in arbitrary positions of the match field so that a prefix can be viewed as a very special wildcard expression. Wildcard expressions can be used in forwarding entries in OpenFlow. However, there are no analogue mechanisms for wildcardbased FIBs like prefix trees and LPM. Therefore, forwarding entries in OpenFlow require priorities to decide which of them is to be used when several of them match an address. We denote the priority of a wildcard expression as prio(w).

Logic minimization compresses a set of logical expressions into an equivalent set of logical expressions that covers the same on-set. The Espresso heuristic was developed for logic minimization in the context of very-large-scale integration (VLSI) synthesis. It does not guarantee minimum size results but is faster and requires less memory than exact minimizers [6]. In the remainder of the paper we denote  $E(\mathcal{P})$  as the minimized result of the Espresso on the prefix set  $\mathcal{P}$ .

As wildcard expressions can be interpreted as logical expressions, we can compress a set of wildcard expressions  $\mathcal{P}$  into a possibly smaller set of wildcard expressions  $E(\mathcal{P})$ . As a consequence, we can also minimize a set of prefixes  $\mathcal{W}$  into a smaller set of wildcard expressions  $E(\mathcal{W})$ .

# C. Compression of Forwarding Tables Using Logic Minimization

We apply logic minimization for compression of prefixbased forwarding tables. A naive approach is to combine all prefixes with the same next-hop. As a result, the prefixes '01\*' and '11\*' with next-hop 3, i.e., ('01\*', 3) and ('11\*', 3), can be combined to forwarding entry ('\*1\*', 3) and the forwarding entry ('1\*\*', 2) remains unchanged. It is important to assign higher priority to entry ('\*1\*',3) than for ('1\*\*',2) to preserve the forwarding behavior. Otherwise, packets addressed to addresses '11\*' would be forwarded to the wrong next-hop. A problem occurs if we combine ('\*\*\*', 1) and ('101', 1) to ('\*\*\*', 1) because then either packets destined to '1\*\*' or packets destined to '101' are forwarded to the wrong nexthop, depending on whether ('\*\*\*', 1) or ('1\*\*', 2) is assigned higher priority. This happens because prefix '\*\*\*' contains the other prefix '101' and the other node ('1\*\*', 2) with a different next-hop lies on the path between ('\*\*\*', 1) and ('101', 1). Thus, such situations need to be avoided in sets of prefixes  $\mathcal{P}$  with same next-hops that are compressed by Espresso.

To avoid the mentioned problems, we partition all entries in a forwarding table into sets with equal next-hops and equal prefix length. Thus we compress all prefixes  $\mathcal{P}_i^j$  with the same next-hop j on the same level i of a prefix tree with Espresso to a set of wildcard expressions  $E(\mathcal{P}_i^j)$ . The priority assigned them is the original prefix length, i.e., prio(w) = $i : w \in E(\mathcal{P}_i^j)$ . These priorities assure the same forwarding behavior as LPM on the prefix-based forwarding table. As the proposed compression approach cannot compress forwarding entries with different-length prefixes, we first run the ORTC on the original forwarding table and use a minimum-size prefix tree as input to the described procedure.

We have investigated two other methods to define sets of forwarding entries to be compressed and appropriate priorities. These sets contained different-length prefixes. Evaluation results showed that these approaches lead to less compression so that we do not consider them in this paper.

#### D. Compression Speedup

The runtime of the Espresso algorithm is exponential with regard to the number of input expression. Therefore, the compression is slow for a large set of prefixes to be compressed. To reduce the compression time, we propose a maximum set threshold T. We partition the sets  $\mathcal{P}_j^i$  into smaller sets with at most T forwarding entries, taken consecutively from the minimum-size prefix tree when going from left to right on the same level. These subsets of prefixes are individually compressed into sets of wildcard-expression by Espresso. They lead to less compression potential but to shorter overall compression time compared to the approach without threshold.

# E. Incremental FIB Updates

Routing table updates consist of prefix additions and removals. This also holds for changed next-hops in the forwarding table. ORTC is fast and can be used to quickly compute a new minimum-size prefix-tree. The changed nodes result in prefixes that need to be removed or added to the wildcardcompressed forwarding table. Adding prefixes is simple as they do not even need to be compressed; optimizations are possible.

We illustrate a procedure to remove a prefix p from a set of wildcard expressions  $\mathcal{W}$  by an example. Consider that the prefix p = `10011\*' of length 5 has to be removed from a set of wildcard expression  $\mathcal{W}$ . Let w = `1\*0\*1\*' be the only expression of priority 5 that matches p. We split w into a set of smaller wildcards  $\mathcal{W}' = \{110*1*, 10001*, 10011*\}$  and



Fig. 3: Number of FIB entries after pure prefix compression by ORTC and for further wildcard compression by Espresso with and without maximum set threshold T.

remove the prefix p from this set. The updated set of wildcard expressions is then  $\mathcal{W}^* = \mathcal{W} \setminus w \cup (\mathcal{W}' \setminus p)$ .

The resulting sets of wildcard expressions are not minimal but did not require time-consuming Espresso minimization. Improved results can be obtained after another Espresso minimization  $E(W^*)$ .

#### V. RESULTS

In this section we apply the presented compression algorithms to realistic data sets. We evaluate and compare their compression ratios, analyze the structure of the compressed data, and study the algorithm runtime depending on the threshold parameter.

#### A. Investigated Data Set

As a base for our study, we use a RIB of 2013 from the Route Views project [3] with 500,495 IP prefixes. The RIB is converted into a pseudo-FIB by assigning a next-hop to each prefix. To that end, we assume up to  $n_{hops}^{next}$  different next-hops. Each prefix is assigned one of these hops with equal probability. We compress that FIB using the ORTC algorithm which yields a minimum size prefix-tree for further wildcard compression. For each number of next-hops  $n_{hops}^{next}$  we generate 10 random FIBs and report mean values for presented results. We also investigated data sets from prior years (2009 – 2012) but we do not show those results as they do not add further insights.

## B. Compression Ratio

We first quantify the compression potential of FIBs in carrier networks through wildcard compression by considering the compression ratios achieved by the heuristic approaches. Figure 3 compares the number of FIB entries generated by ORTC with the number of FIB entries after further wildcard compression for different numbers of next-hops  $n_{hops}^{next}$ . The number of FIB entries obtained after pure prefix compression by ORTC increases with increasing number of next-hops from 50,000 to 350,000. For a single next-hop, there are about 50,000 FIB entries instead of a single default route. The reason



Fig. 4: Fraction of FIB entries compressed by Espresso depending on the maximum set threshold T.



Fig. 5: Number of FIB entries per prefix length for ORTC and after wildcard compression by Espresso without threshold  $(T = \infty)$ . For Espresso, the number of FIB entries with wildcards is also given.

for that is the assumption of a full routing table so that packets need to be dropped in the absence of a matching entry. The figure also provides values for wildcard compression with Espresso with maximum set thresholds of T = 100 and  $T = \infty$ . We observe that this further compression reduces the number of FIB entries by 30,000 - 40,000 in the presence of at least  $n_{hops}^{next} = 2$  next hops. Furthermore, Espresso yields hardly more FIB entries when applying a low maximum set threshold of T = 100 than without such a threshold  $(T = \infty)$ .

As the loss of compression due to the maximum set threshold T is hardly visible in Figure 3, we present the compressed fraction  $\rho$  in Figure 4, i.e., the fraction by which further wild-card compression can reduce the size of the minimum prefixbased flow table. The compressed fraction ranges between 9% and 17% and depends both on the number of next-hops  $n_{hops}^{next}$  and the applied maximum set threshold T. The compressed fraction decreases with decreasing threshold, but the loss in compression is at most 2% even for a very low maximum set threshold of T = 100. A threshold of T = 500 leads only to 1% loss in compression.



Fig. 6: Compressed fraction per prefix length for wildcard compression with Espresso without threshold  $(T = \infty)$ .

# C. Analysis per Prefix Length

Figure 5 shows the number of FIB entries generated by ORTC sorted by prefix length. Most prefixes have length 24. Smaller prefixes down to prefix length 17 appear with decreasing frequency. Prefix length 16 is about as frequent as prefix length 18 and again, smaller prefixes down to prefix length 14 appear with decreasing frequency. All other prefix lengths hardly occur. The reason for this phenomenon is that standard prefix lengths are /16 and /24 which are already combined to shorter prefixes by CIDR in the BGP RIB or by the ORTC through prefix aggregation.

The figure also shows the number of FIB entries per prefix length after further wildcard compression and the number of FIB entries containing additional wildcards. The number of more general wildcard expressions is rather low and most of them contain only a single wildcard. So the compression potential is rather moderate.

We applied the Espresso heuristic to equal-priority sets of prefixes with equal length. Figure 6 shows the compressed fraction per prefix length. For prefix length between 8 and 24 it is mostly in the order between 10% and 15%. Smaller and larger prefixes are extremely rare so that their overall fraction amounts to less than 0.1%. This suggests that Espresso can compress efficiently only if the set FIB entries to be compressed is sufficiently large so that there is a chance for similar entries. We have also studied other approaches to create equal-priority sets that do not reveal equal-length prefixes, but the achieved compression ratio was lower. Apparently, the compression potential is larger for sets of equal-length prefixes.

# D. Compression Time

We run Espresso on a computer with 8GB DDR3-RAM and an Intel CPU i5-2500K with 4 cores, 6 MB cache and 3.3 GHz. We implemented the compression in a single threaded C++ program that only uses one core. We measure the runtime for the Espresso-based FIB compression only, i.e., the partitioning work of the input FIB in smaller subsets is provided by a different program.

Figure 7 shows the measured runtime of Espresso for various numbers of next-hops  $n_{hops}^{next}$  and for various maximum



Fig. 7: Runtime of Espresso-based compression for complete FIBs depending on the maximum set threshold T.

set thresholds T. Without such a threshold  $(T = \infty)$ , the compression for a single next hop takes about 22 s, but for more next-hops the runtime increases from 341 s to 436 s. The runtime is greatly reduced by applying maximum set thresholds. For moderate thresholds of T = 500 the Espresso runtime decreases to a range between 7 s and 11 s. Low thresholds of T = 100 and T = 250 require even for  $n_{hops}^{next} = 5$  a maximum compression time of less than 1 s or 2 s. Thus, the application of maximum set thresholds reduces the compression time by orders of magnitude while degrading the compressed fraction by 1% - 2%.

#### VI. CONCLUSION

We presented a concept for the use of OpenFlow-based SDN in carrier networks. In some design variants, OpenFlow switches need to accommodate the full inter-domain routing information which is rather challenging as flow table size is limited. As OpenFlow supports match fields with general wildcards, we propose to compress prefix-based FIB entries into FIB entries using general wildcards using the Espresso heuristic. This is feasible with SDN as the computationintensive compression can be performed by a network application on a server instead by the switch.

We showed that our proposed method reduces FIB sizes based on a BGP RIB of 2013 with about 500,000 entries by up to 17%. As the runtime of Espresso is exponential with regard to input size, we propose a maximum set threshold to trade runtime against compression rate. Thereby, the compression time could be reduced by orders of magnitude down to less than 1 s or 2 s while sacrificing 1% - 2% compression ratio. Routing updates can be added incrementally.

This compression potential is certainly not large enough to fit the large amount of routing information into the TCAM of a today's custom OpenFlow switch to enable forwarding without misses of flow table entries. However, the method may be used to compress a rather stable part of the flow table to leave more room for other less frequently used flow table entries.

#### REFERENCES

- [1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," ACM SIGCOMM Computer Communications Review, vol. 38, no. 2, pp. 69–74, 2008.
- [2] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "OpenFlow: Meeting Carrier-Grade Recovery Requirements," *Computer Communications*, 2012.
- [3] "Route views project page," Mar. 2014. [Online]. Available: http://www.routeviews.org/
- [4] B. Agrawal and T. Sherwood, "Modeling TCAM POWER for Next Generation Network Devices," in *IEEE International Symposium on Performance Analysis of Systems and Software*, Mar. 2006, pp. 120– 129.
- [5] L. Luo, G. Xie, S. Uhlig, L. Mathy, K. Salamatian, and Y. Xie, "Towards TCAM-based Scalable Virtual Routers," in ACM Conference on emerging Networking EXperiments and Technologies (CoNEXT), 2012, pp. 73–84.
- [6] R. L. Rudell, "Multiple-Valued Logic Minimization for PLA Synthesis," EECS Department, University of California, Berkeley, Tech. Rep. UCB/ERL M86/65, 1986.
- [7] R. Draves, C. King, V. Srinivasan, and B. Zill, "Constructing Optimal IP Routing Tables," in *IEEE Infocom*, 1999, pp. 88–97.
- [8] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. van der Merwe, "Design and Implementation of a Routing Control Platform," in USENIX Syposium on Networked Systems Design & Implementation (NSDI), 2005, pp. 15–28.
- [9] C. E. Rothenberg, M. R. Nascimento, M. R. Salvador, C. N. A. Corrêa, S. C. de Lucena, and R. Raszuk, "Revisiting Routing Control Platforms with the Eyes and Muscles of Software-Defined Networking," in ACM Workshop on Hot Topics in Software Defined Networks (HotSDN), 2012, pp. 13–18.
- [10] R. Bennesby, P. Fonseca, E. Mota, and A. Passito, "An Inter-AS Routing Component for Software-Defined Networks," in *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2012, pp. 138–145.
- [11] N. Sarrar, S. Uhlig, A. Feldmann, R. Sherwood, and X. Huang, "Leveraging Zipf's Law for Traffic Offloading," ACM SIGCOMM Computer Communications Review, vol. 42, no. 1, pp. 16–22, Jan. 2012.
- [12] A. Gupta, M. Shahbaz, L. Vanbever, H. Kim, R. Clark, N. Feamster, J. Rexford, and S. Shenker, "SDX: A Software Defined Internet Exchange," Tech. Rep., 2013.
- [13] J. Rexford, J. Wang, Z. Xiao, and Y. Zhang, "BGP Routing Stability of Popular Destinations," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet Measurment*, 2002, pp. 197–202.
- [14] T. Yang, B. Yuan, S. Zhang, T. Zhang, R. Duan, Y. Wang, and B. Liu, "Approaching Optimal Compression with Fast Update for Large Scale Routing Tables," in *IEEE International Workshop on Quality of Service*, 2012, pp. 32:1–32:9.
- [15] Z. A. Uzmi, M. Nebel, A. Tariq, S. Jawad, R. Chen, A. Shaikh, J. Wang, and P. Francis, "SMALTA: Practical and Near-optimal FIB Aggregation," in ACM Conference on emerging Networking EXperiments and Technologies (CoNEXT), 2011, pp. 29:1–29:12.
- [16] Y. Liu, X. Zhao, K. Nam, L. Wang, and B. Zhang, "Incremental Forwarding Table Aggregation," in *IEEE Globecom*, Dec. 2010, pp. 1–6.
- [17] G. Rètvàri, J. Tapolcai, A. Kõrösi, A. Majdàn, and Z. Heszberger, "Compressing IP Forwarding Tables: Towards Entropy Bounds and Beyond," in ACM SIGCOMM. Hong Kong, China, Aug. 2013, pp. 111–122.
- [18] S. Singh, F. Baboescu, G. Varghese, and J. Wang, "Packet Classification Using Multidimensional Cutting," in ACM SIGCOMM, 2003, pp. 213– 224.
- [19] A. X. Liu, C. R. Meiners, and E. Torng, "TCAM Razor: A Systematic Approach Towards Minimizing Packet Classifiers in TCAMs," *IEEE/* ACM Transactions on Networking, vol. 18, no. 2, pp. 490–500, Apr. 2010.
- [20] A. X. Liu, E. Torng, and C. Meiners, "Firewall Compressor: An Algorithm for Minimizing Firewall Policies," in *IEEE Infocom*, Phoenix, Arizona, April 2008.
- [21] C. R. Meiners, A. X. Liu, and E. Torng, "Bit Weaving: A Non-prefix Approach to Compressing Packet Classifiers in TCAMs," *IEEE/ACM Transactions on Networking*, vol. 20, no. 2, pp. 488–500, Apr. 2012.
- [22] R. McGeer and P. Yalagandula, "Minimizing Rulesets for TCAM Implementation," in *IEEE Infocom*, Apr. 2009, pp. 1314–1322.