Compression of Interdomain SDN Policies at Exchange Points

Robert C. MacDavid

Master's Thesis

Presented to the Faculty
of Princeton University
in Candidacy for the Degree
of Master of Science in Engineering

Recommended for Acceptance
by the Department of Computer Science
Princeton University

Adviser: Jennifer Rexford

June 2016

© Copyright by Robert C. MacDavid, 2016.

All rights reserved.

Abstract

Internet Exchange Points (IXPs) with Software-Defined Networking (SDN) capabilities are a promising way of shaping the future of interdomain traffic delivery on the Internet—allowing IXP participants to express flexible SDN policies on interdomain traffic. Unfortunately, it has proven difficult to compile interdomain traffic policies into forward table sizes reasonable for available commodity switches, and indeed this shows in state-of-the-art Software-Defined IXP implementations. To compile interdomain traffic SDN policies, available prefix routes must be factored in to ensure correctness, resulting in an explosion in the size that a policy requires in the forwarding plane. To combat this scalabilty challenge, new compression techniques of forwarding table entries are required. In this work, we discuss the design and analysis of such a technique, which allows near-optimal compression of SDN policies for IXPs with large numbers of participants.

Acknowledgements

I would like to thank Arpit Gupta for his mentorship throughout the project, and acknowledge Rüdiger Birkner and Laurent Vanbever for their original proposal of the encoding scheme. I'd also like to thank my advisors Jen Rexford and Nick Feamster for their guidance throughout the project, and my iSDX coauthor Marco Canini for his excellent contributions.

Contents

List of Figures

Chapter 1

Introduction

Software-Defined Networking (SDN) is a promising advancement in networking, allowing users to continuously describe complex and flexible policies for forwarding traffic on their networks. Although it has mainly been considered in the Local Area Network (LAN) setting, there is a desire to see it applied to the Wide Area Network (WAN) setting as well. A natural way to introduce SDN to the wide area setting is by adding SDN support to Internet Exchange Points (IXPs). At an IXP, tens to hundreds of Autonomous Systems (ASes) come together at a single location to exchange routing information via BGP and interdomain traffic via layer 2 forwarding. Because of this relatively simple centralized structure, IXPs can provide a way to quickly add the power of SDN to significant portions of interdomain traffic.

Unfortunately, it has proven difficult to add SDN capabilities to an IXP in a scalable way. There are many challenges that must be addressed, including how to compile together the policies of multiple participants, how to ensure BGP routing is not violated, and how to incrementally deploy such a system. One of the most challenging obstacles has been preventing the violation of BGP routing by participant policies, and it is this obstacle that we will focus on.

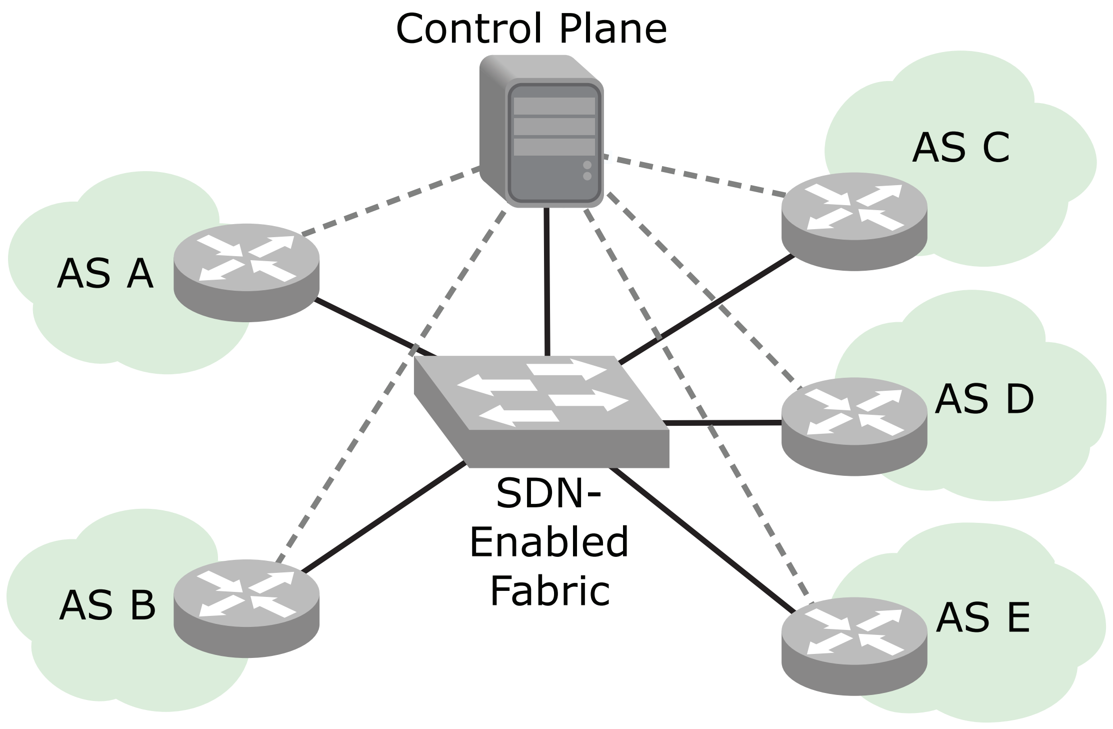

Figure 1.1: An example of a Software-Defined IXP. Pictured are 5 participant Autonomous Systems connected to the IXP's fabric and controller. Participants relay the policies they wish to install to the controller, which then installs it on their behalf on the fabric.

1.1 Software-Defined IXP

Before we dive into the problem, we will give a brief description of an exchange point. An IXP consists a route server, traffic fabric, and participant AS border routers. An example is shown in figure 1.1. Each border router connects to both the route server and the traffic fabric. Border routers send IP prefix route announcements and withdrawals to the route server via BGP messages, and the route server relays this information to the other participants' border routers. Each border router decides upon a *default next-hop* for each prefix based upon the advertised routes it receives, and then sends traffic to those next-hops via the IXP fabric.

In a Software-Defined IXP, the setup is similar but enhanced. There are still border routers connected to a fabric and a route server, but the route server now doubles as an SDN controller, connecting directly to the fabric to install forwarding table entries. The fabric now supports these SDN forwarding table entries, and can communicate with the controller via some SDN protocol such as OpenFlow [1]. Participants that wish to express SDN policies relay their policies to the controller, and the controller installs the policies into the fabric on their behalf. The controller determines how best to combine all the participants' policies to prevent conflict.

Participants can express two kinds of policies at a Software-Defined IXP: outbound policies and inbound policies. Outbound policies are forwarding actions applied to traffic which a participant sends *to* the exchange point, and overrides the participants' default next-hop preferences when applied. Inbound policies are forwarding actions applied to traffic which a participant *receives* from the exchange point, and is useful for dropping unwanted traffic or for when a participant has multiple connections to the exchange and wishes to apply some form of load balancing.

Inbound policies are relatively straightforward to install at the exchange point, because all ports for a single participant lead to the same set of destinations and thus policies are not conditional upon available routes. Outbound policies, however, have proven difficult to correctly implement in the limited space of the forwarding plane. Throughout this work, we will be focusing on the compression of outbound policies and thus we will use the terms "outbound policy" and "policy" interchangeably.

Chapter 2

The Path Encoding Problem

We have mentioned that outbound has proven difficult to install without violating BGP routing. To understand the problem, let us consider a simple example. Say that some IXP participant AS S wishes to apply an outbound forwarding policy to all traffic it sends through the IXP fabric. Suppose that one rule in S's outbound policy is `dPort=80` $\rightarrow$ `fwd`(D)
In plain english, S wishes to reroute all HTTP traffic to IXP participant D, overriding the default BGP forwarding behavior for every matching flow. Left alone, this rule is incorrect, because participant D may not advertise routes to all destination prefixes. If D only advertises prefixes $[p_2, p_3, p_4, p_5]$, only the HTTP traffic from S destined for these four prefixes should be rerouted to D. If S generates HTTP traffic destined for p_1, it would be incorrect to send this traffic to D.

2.1 Naive Encoding

In order to correctly apply S's outbound forwarding policies to outbound packets, the flow rules placed in the IXP fabric which realize S's policy must take into account the paths available for each packet. We will refer to the list of IXP participants which have advertised routes to some prefix p as the valid next-hops for p. In this context, a correct implementation of a policy which forwards to participant D is a set of flow rules that always checks for whether D is in the valid next-hops before rerouting.

Consider the naive case for encoding correctness into a policy. Recall the rule
`dPort=80` $\rightarrow$ `fwd`(D)
for forwarding HTTP traffic to D. If D only advertises prefixes $[p_2, p_3, p_4, p_5]$, then, to ensure correct routing, this rule can be modified into the flow rules
`dPrefix=`$p_2 \wedge$ `dPort=40` $\rightarrow$ `fwd`(D)
`dPrefix=`$p_3 \wedge$ `dPort=40` $\rightarrow$ `fwd`(D)
`dPrefix=`$p_4 \wedge$ `dPort=40` $\rightarrow$ `fwd`(D)
`dPrefix=`$p_5 \wedge$ `dPort=40` $\rightarrow$ `fwd`(D)
Now this policy will only be applied to flows which can reach their destination via D. In this case, the simple HTTP policy was inflated by the number of prefixes which D advertised. If each participant advertises x prefixes, the forwarding policy is subject to an inflation factor of x in order to ensure correct routing. This factor can be quite large in practice, so this motivates our new approach.

2.2 Superset Encoding

For the sake of explanation, assume that it is possible to attach an arbitrary amount of metadata to each packet as it enters the IXP fabric, and that flow rules are able to read and write to this metadata arbitrarily. We can easily use this hypothetical metadata to correctly realize a participant's outbound policy. If there are N participants advertising routes at the exchange point, we can write N bits to the metadata: the ith bit corresponds to the ith participant. The moment the packet enters the exchange point, the ith bit is set to 1 if the ith participant has advertised a route to that packets destination, and 0 if they have not. As a result, if we wish to reroute a packet to participant D, we need only check that D's bit is 1 in the metadata before rerouting, rather than testing for every prefix which D advertises.

In our example, if N is 5 and bit 4 is assigned to D, then the flow rule for forwarding HTTP traffic to D becomes

`metadata=XXX1X` $\wedge$ `dPort=40` $\rightarrow$ `fwd`(D)

where X denotes a "don't care" match, so this checks only the 4th bit of the 5-bit metadata bitmask.

2.3 MAC as Metadata

Of course, we cannot actually attach an arbitrary amount of metadata to each packet as it enters the fabric, but we can use the same idea as the original SDX paper [2], where the destination MAC address field was repurposed for tagging. Before sending outbound traffic for some destination prefix p, a participant asks the exchange point, via ARP requests, for the next-hop MAC address. If we pad the metadata to 48 bits and send it as the reply, then every packet will arrive at the exchange point with metadata in the destination MAC field, effectively repurposing the field as metadata! We are able to do the required masked matching on this field using flow rules due to recent updates to the OpenFlow standard, which allowed arbitrary masked matching when reading the MAC address, permitting the reading of individual bits [3].

But, if we use the destination address as our metadata field, we lose knowledge of which direction the packet was originally headed if it does not match on any policy rule. This is simple enough to fix: if there are N nodes in the layer-2 network, we can reserve $\log_2 N$ bits in the metadata field to serve as a "mini MAC". If we assign each node in the network a $\log_2 N$-bit identifier and place it at the end of the metadata, we can simply do masked matching on that field to maintain layer-2 forwarding functionality. If we use x bits for the identifier, our approach will still work so long as our encoding needs less than $48 - x$ bits.

2.4 Optimizations

We must point out that by constraining ourselves to using only at most 48 bits of a MAC address field, we run into scalability issues. As our scheme currently stands, we are assigning a single bit of the metadata bitmask to every participant at the exchange point. If we have more than 48 participants, this approach immediately fails. To remedy this, we apply three optimizations:

1. We advertise different bitmasks to each participant.
2. We only reserve a bit in X's bitmasks for a next-hop participant Y if X has flow rules which forward Y.
3. The metadata field need not contain the full bitmask.

Disjoint Announcements For each list of next-hops which advertise a prefix, there is a corresponding bitmask. Following the model of the original SDX, it is up to the IXP controller to generate these bitmasks for every participant and convey them to participant's edge routers via gratuitous ARP replies. The participant routers will treat these bitmasks as normal MAC addresses and place them in every packet which is sent to the exchange point. If the IXP controller sends targeted gratuitous ARP replies rather than broadcasting, we can present different bitmasks for the same prefix to different participants. By itself, this has no effect on scalability, but it opens the door for further optimizations. Each participant may have a bitmask tailored to their policies, independent of all other participants. As we will soon see, this greatly improves the compression possible.

Active Set Masking Each participant S will have some set of policies which forward to some set of next-hops. If S has no policy which forwards to a participant T, then there is no need to reserve a bit in the bitmask for T, as S's flow rules will never check that T is a valid next-hop. Since S does not need to share their bitmask with any other participants due to our previous optimization, this is independent of all other participant policies.

We refer to the list of participants for which S has outbound flow rules as *S's active set*. As a result of this optimization, the amount of bits required for S's bitmask directly depends upon the size of S's active set, which can be thought of as the complexity of S's outbound policy. This allows the scheme to work on exchange points with an arbitrary number of participants, as long as no participant has an active set of size 48 or more. This still does not support larger policies, however, and so we must make one final optimization.

Multiple Masked Sets The purpose of the metadata bitmask attached to each packet is to concisely convey whether each participant is a valid next-hop for that packet or not. The bitmask can be thought of as recovering the desired list of next-hops by masking some *superset* which contains all the next-hops. Before our active set optimization, the superset which was being masked was the set of all participants. We noted that it was sufficient for the superset to be only the active set. This still does not scale enough, because the active set may be too large for a mask of it to fit in the metadata.

Consider the example in figure 2.1(a), where the left matrix contain the lists of next-hops that we wish to recover, and the right matrix shows the bitmasks we would currently generate. Figure 2.1(b) shows that the next-hops can be broken up into two categories: those which can be generated by masking over the superset $[A, B, C]$, and those which can be generated by masking over $[C, D, E]$. If we can attach to

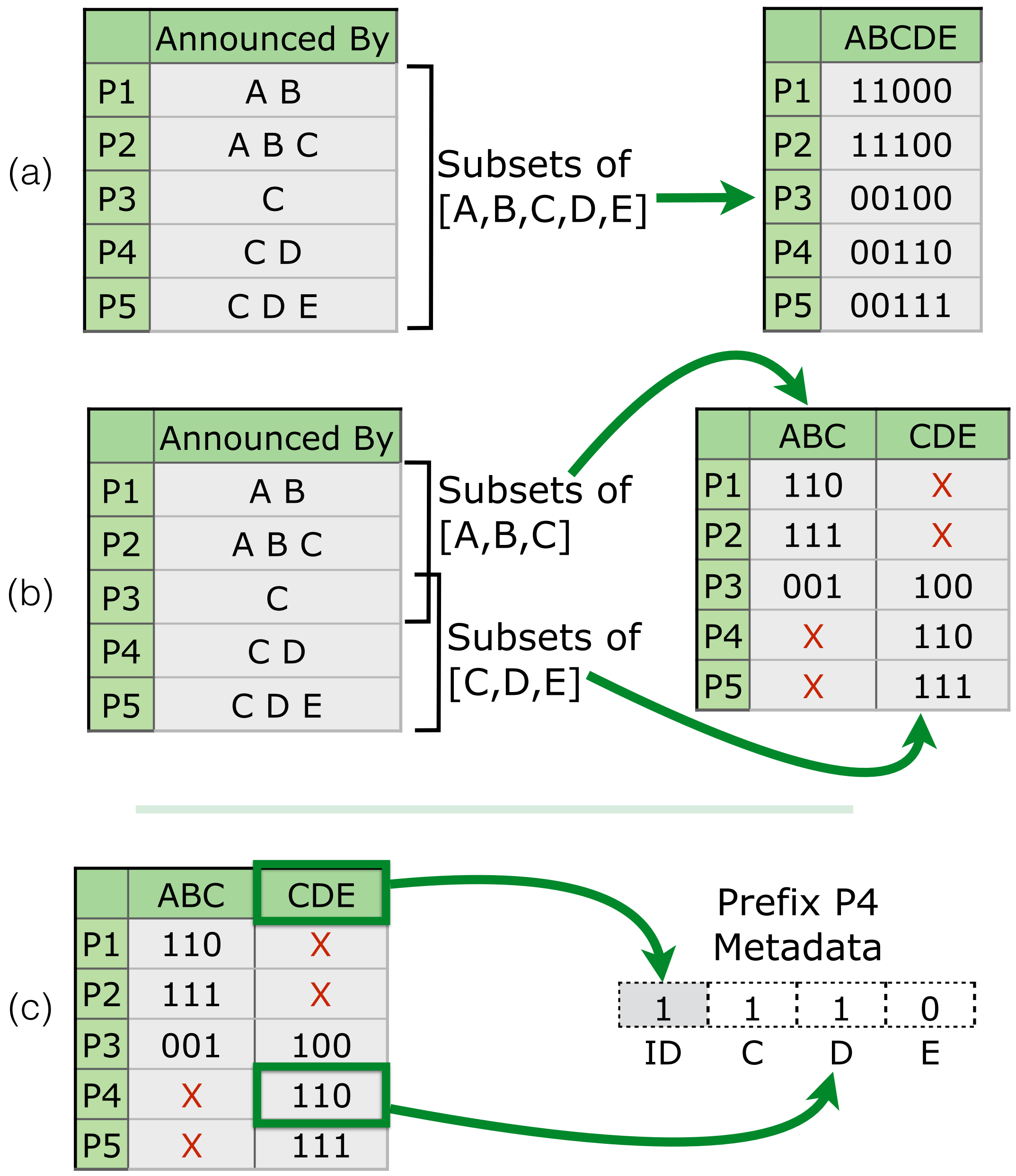

Figure 2.1: This figure demonstrates two different ways to recover a list of valid next-hops for each prefix. In (a), the next-hops are recovered by masking over $[A, B, C, D, E]$. In (b), the next-hops are recovered by masking over either set $[A, B, C]$ or set $[C, D, E]$. An X denotes that the list of next-hops cannot be fully recovered by masking over the given set. (c) shows how, if each set is identified by a binary integer, each entry can be converted to metadata consisting of an identifier and a mask.

our metadata an identifier of the superset over which we are masking, we can have a reduced mask size! As shown in figure 2.1(c), if superset $[A, B, C]$ is identified as superset 0, and $[C, D, E]$ is identified as superset 1, then the metadata for a prefix

p_4 which is advertised by $[C, D]$ becomes 1110, which is shorter than simply masking over the complete active set.

Now, if a policy wishes to forward port 80 traffic to D, it must be augmented to simultaneously check (1) the column identifier for the set that contains D and (2) the bit in the bitmask which corresponds to D. Thus, the rule that would be generated is

`metadata=1X1X` $\wedge$ `dPort=80` $\rightarrow$ `fwd`(D)

Where the first bit in the metadata match is for the superset identifier, and the remainder is for the mask.

However, looking again at figure 2.1(b), the case of policies which forward to C is not so simple. Since C appears in both supersets, we must have rules which check whether C's bit is 1 in either superset mask. For example, if we had the policy

`dPort=443` $\rightarrow$ `fwd`(C)

This would be augmented under our scheme to become

`metadata=0XX1` $\wedge$ `dPort=443` $\rightarrow$ `fwd`(C)

`metadata=11XX` $\wedge$ `dPort=443` $\rightarrow$ `fwd`(C)

Therefore, depending upon the matrix construction, there may still be some policy inflation. If no superset is too large to fit into the mask, it is feasible to merge columns of the matrix to create new, larger supersets which can decrease the inflation factor. However, this can only be performed until the supersets of each column become too large to fit into the available bit space. For example, in figure 2.1(b), the two columns could be merged into $[A, B, C, D, E]$ to eliminate the inflation of C rules. This would decrease the identifier size from 1 to 0 bits and increase the mask size from 3 to 5 bits. There is a balance to consider between the size of the superset identifier, the mask size, and the inflation factor, and when considered formally an optimization problem arises.

Chapter 3

Problem and Optimization

Let $P = \{p_1, p_2, \ldots, p_N\}$ bet a set of subsets of the ground set $U = \{1, 2, \ldots, M\}$ (i.e. $p_i \subseteq U \forall p_i \in P$) Associated with each element j of the ground set U is a weight term w_j

Goal: Partition P into Z groups and union each group to create supersets $\{s_1, s_2, \ldots, s_Z\} = S$ such that the following function is minimized:

$$\min \sum_{j \in U} w_j \cdot a_j$$

Subject to

$$\log_2 Z + \max_{s_i \in S} \{s_i\} \leq B$$

Where a_j is the number of supersets that element $j \in U$ belongs to.

In the context of a softward-defined IXP, the ground set U is the active set of participants for the current outbound policy. Each set $p_i \in P$ is the list of valid next-hops for the ith prefix. The weight term w_j on each element $j \in U$ is the number of flow rules which forward to next-hop j. The objective function reflects the number of rules required by a policy combined with a superset matrix. For every rule which forwards to a participant T, if T appears in a_j supersets, then our approach requires that the rule be replicated a_j times. If w_j is the number of such rules, then the objective function becomes clear. The constraint shows that a categorization is feasible if the identifier size plus the mask size is less than the bit constraint, B. The identifier size is $\log_2 Z$ if Z is the number of supersets, and the mask size required is the maximum superset size, or $\max_{s_i \in S}\{s_i\}$.

We suspect that this problem is NP-Complete for $M > B$, but a proof of the claim is left as future work.

3.1 Inflation Optimization

Now that we understand the problem, we will give a greedy heuristic for constructing a locally optimal solution. But, it is necessary to note that this is not a static problem. The key property which defines a feasible solution is that for every prefix and the set of participants that advertise that prefix, there is a superset which contains that set. After computing our solution, this set may expand or contract as participants announce and withdraw routes. Not only must we have an algorithm for the initial construction, we must also have a procedure for handling updates to sets of next-hops.

3.1.1 A Greedy Algorithm

We begin with N supersets, where superset i is the list of valid next-hops for prefix p_i. This solution is, by construction, able to generate every required next-hop list, but it may not be feasible. N may be so large that the superset identifier will dominate the metadata and exceed the bit limit. This solution will also certainly have large rule inflation, as no sets have yet been combined into supersets.

In the first step of the algorithm, we delete any supersets which are subsets of other supersets. If superset s_i is $[A, B, C]$, and superset s_j is $[B, C]$, then there is no use for s_j and it may be deleted. Deletion strictly improves the solution because any subset of s_j can still be recovered from s_i, and the superset identifier decreases in size as the number of supersets decreases. In practice, this reduces the number of sets down from hundreds of thousands to less than one hundred.

In step two, we attempt to greedily decrease the number of bits required by our feasible solution by merging pairs of small supersets together that do not increase the maximum mask size when unioned. The idea is that this will decrease the number of sets and thus the size of the superset identifier will decrease. This step repeats until every feasible merging action would increase the number of bits required.

In step three, we improve the remaining supersets in an iterative greedy fashion. We consider all feasible mergings of pairs of supersets, where a feasible merge is a union of the two supersets which does not result in the new mask size exceeding the bit limit. The *benefit* of a merging is the decrease in the number of flow rules which will result from the merge. The decrease is the sum of the weights of each participant which appears in the intersection of the two supersets, where the weight is the number of flow rules in which the participant appears. This is because after a merging of two sets, every participant which appeared in the intersection now appears one less time across the supersets, and thus every rule they appear in can be replicated one less time.

With these definitions in mind, the full algorithm is as follows:

The loops consider all pairs of supersets in each iteration in the worst case, so they have a quadratic running time. Fortunately, the removal of subset supersets in the first step, which runs linearly on lists of supersets with high redundancy, reduces the number of supersets in practice to a small constant, so the running time is reasonable.

```
**Data**: feasible supersets $S = \{s_1, s_2, \ldots, s_M\}$
**Result**: a list of supersets with maximally decreased rule inflation
remove subsets from $S$;
let $A =$ the set of merge pairs which don't increase bits required;
**while** $A$ *is nonempty* **do**
    choose pair $(s_i, s_j) = a \in A$ with smallest union;
    merge sets $s_i$ and $s_j$;
    update $A$;
**end**
let $A =$ the set of feasible merge pairs;
remove any subsets from **while** $A$ *is nonempty* **do**
    choose $(s_i, s_j) = a \in A$ with greatest benefit;
    merge sets $s_i$ and $s_j$;
    update $A$;
**end**
```

3.1.2 Handling Updates

We have given an algorithm for computing a static solution, but BGP routing is far from static. Routes are announced and withdrawn continuously, meaning the list of valid next-hops for many prefixes are constantly changing. Therefore, we must give a procedure for handling dynamic updates to the matrix.

We begin with a feasible solution as output by our algorithm, which is a collection of supersets and a mask for each prefix. Each time a prefix is announced or withdrawn by a participant, we consider that prefix's new list of valid next-hops. If it is still a subset of some superset, we need only update the mask.

If it is no longer a subset of any superset, we create a new superset solely for that new list, which is equal to the list. We then attempt to merge the new superset with an existing superset via the same greedy procedure we applied to the static context. If no merging is possible, we leave it as its own superset. If the introduction of the new superset causes the bit limit to be exceeded, we recompute the static solution entirely as a worst-case scenario. This worst-case may seem impractical, but in the iSDX [4] paper it is shown that this is *never* necessary. In fact, it was shown that the merging step almost never occurs as well, as every updated list of next-hops is almost always a subset of a superset in the existing solution. These procedures almost need only be defined for the purpose of correctness.

Chapter 4

Evaluation

We will now demonstrate the scalability of the supersets encoding scheme table size for a real exchange point.

4.1 Experimental Setup

We used a data set from the AMS-IX exchange point [5], which gave us access to 63 participants advertising over 600,000 prefixes. Although not on the scale of complete data sets of the largest exchange points, the data set is large enough that the partial masks optimization is required to fit masks into the MAC field. Our experiments were run on a laptop with a 4-core CPU running at 2.4GHz and 16GB of RAM.

The exchange point considered does not yet have support for SDN forwarding rules, so instead we simulate the number of forwarding entries required by the outbound policy of a hypothetical participant which is able to see all available route announcements. The participant was given 1000 uniformly random forwarding entries to a uniformly random subset of participants, with each subset size corresponding to a different experiment.

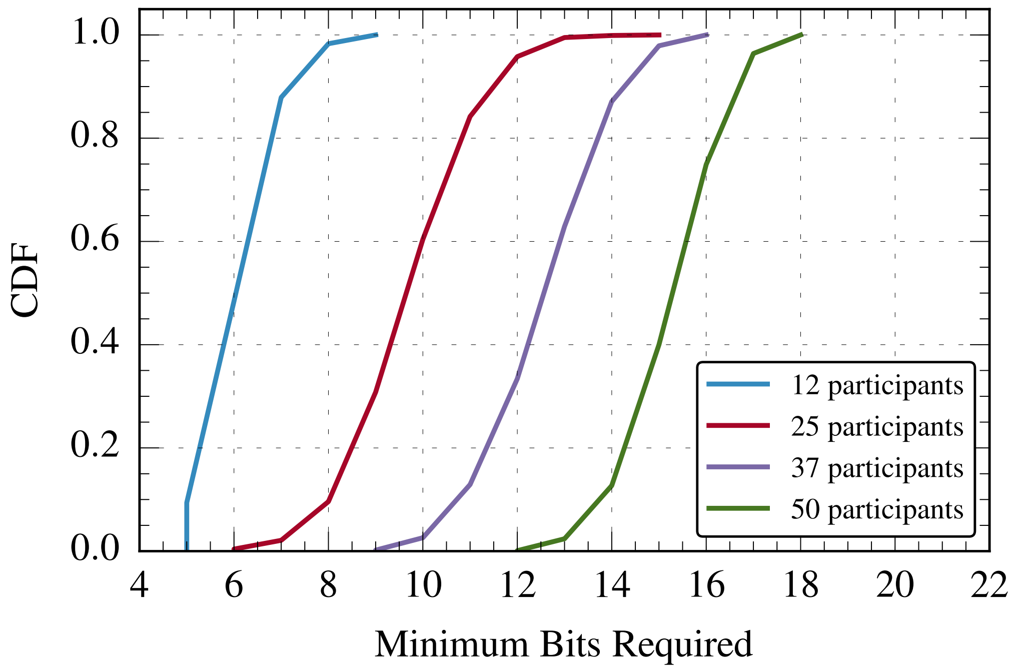

Figure 4.1: Minimum number of bits required by a feasible solution for a random policy which forwards to a random subset of participants.

4.2 Performance

We run our evaluations against a RIB dump of AMS-IX retrieved on May 1st, 2016 from the RIPE Routing Information Service raw data webpage [6].

In order for our encoding scheme to successfully work, it must compress the information to the point that it fits into the MAC address field, which is restricted to 48 bits if no other information is encoded alongside reachability. Figure 4.1 shows the number of bits required by our reachability encoding with uniformly randomly chosen active sets, repeated 500 times for each active set size. In the worst case, 18 bits were required when considering all 63 participants. The graphs appear to show that the number of bits required scales linearly with the number of participants present in the active set. However, we believe that the number of bits required actually scales sub-linearly with the size of the active set, and that the appearance of linear scaling is a

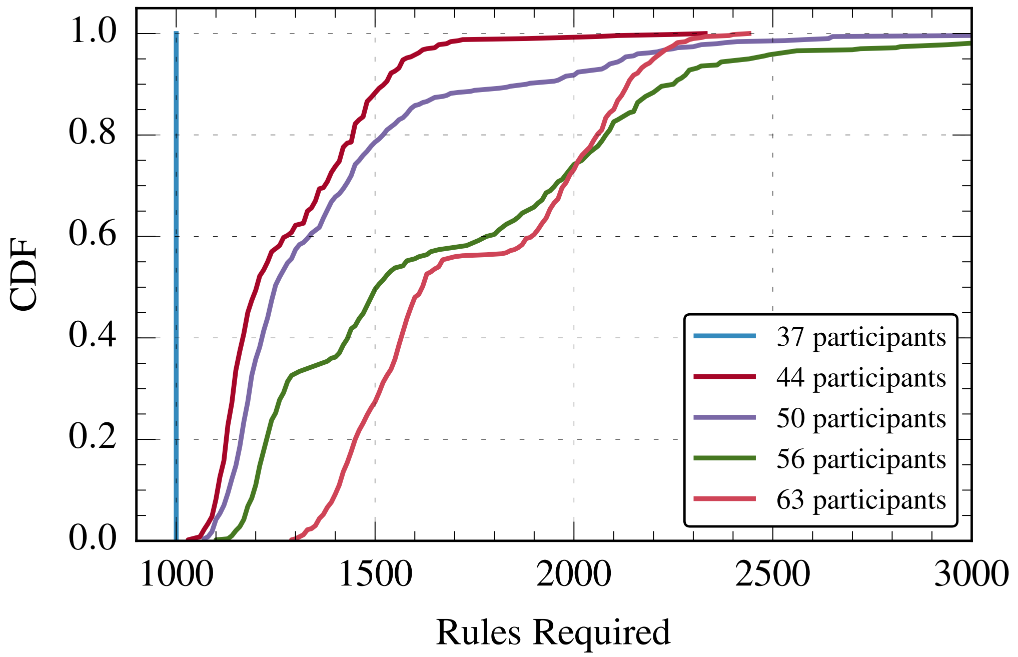

Figure 4.2: Number of rules required after encoding for a random policy of 1000 rules to random subsets of participants.

consequence of our simulation method. In order for the bits required to scale linearly, the number of participants that simultaneously advertise a single precept would have to also increase linearly in the size of the IXP, but we suspect this is not the case. Even if the number of bits required were indeed to scale linearly, extrapolating out the graph yields that, in the worst case, a participant's active set could contain over 100 participants, allowing very complex forwarding policies.

Figure 4.2 shows the number of rules required by our encoding scheme after running the greedy algorithm with a bit limit of 37, allowing 11 bits for the "mini-MAC". In this experiment, we began with a baseline policy of 1000 rules, with each rule forwarding to a participant chosen uniformly at random from a random active set. The experiment was repeated 500 times for each active set size, and the number of rules required after applying our encoding was plotted. When the active set is of size 37 or fewer, all participants can fit into a single mask, resulting in zero rule inflation. For

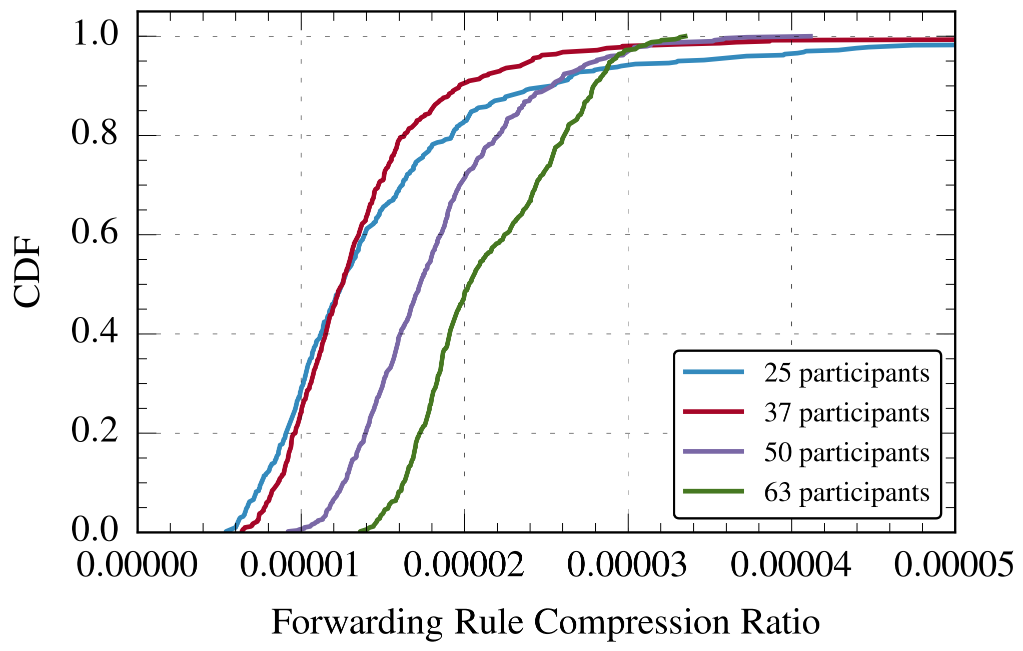

Figure 4.3: Ratio of flow rules required by our encoding algorithm versus the uncompressed case on random policies involving random subsets of participants.

active sets of size greater than 37, we can see that the inflation factor is at most 2 in the average case and 3 in the worst case for all active sets.

Figure 4.3 shows how the number of flow rules generated by our approach compares to the naive case of zero compression. The compression ratio of our approach versus the naive approach is 20,000 to 1 in the worst case for all active set sizes, and 50,000 to 1 in the median case.

Figure 4.4 compares our approach to the uncompressed case, as well as to the previous state-of-the-art, the MDS algorithm used in the original SDX system [2]. The comparisons were made using the same approach of generating 1000 random rules, except for the MDS simulation. The MDS algorithm requires each prefix's default next-hop as part of the input, so in each trial we chose next-hops uniformly at random from the list of available next-hops. The graph shows that our approach consistently compresses the number of flow rules by two orders of mangitude greater

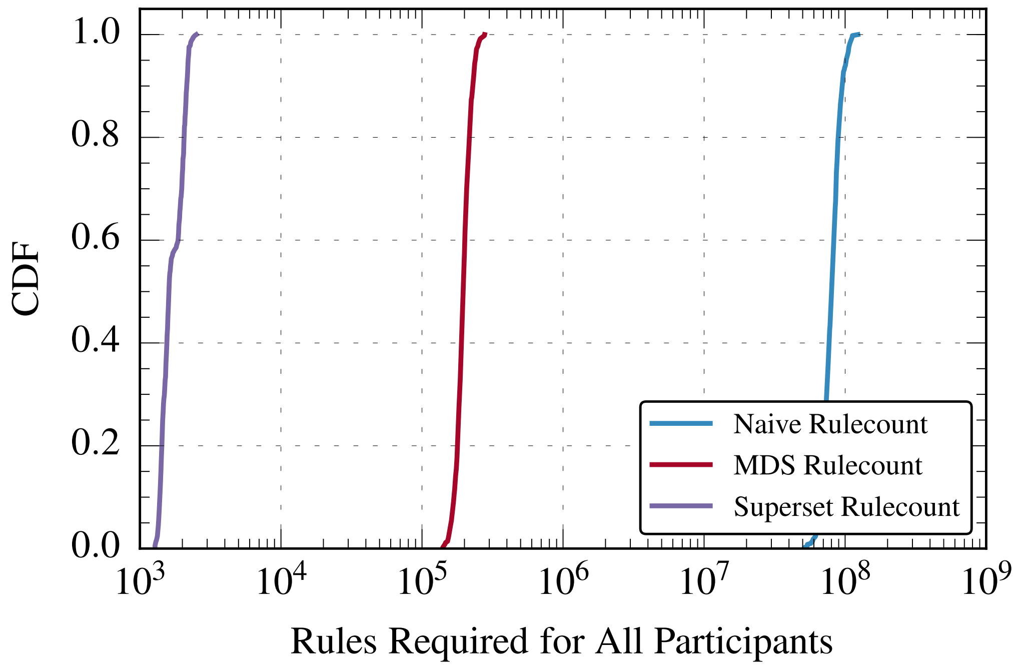

Figure 4.4: Comparison of the number of flow rules required by our encoding algorithm to the previous state-of-the-art MDS encoding algorithm and the uncompressed case for a random policy involving all participants.

than the MDS algorithm, which itself compressed the number of flow rules required by the naive case by three orders of magnitude.

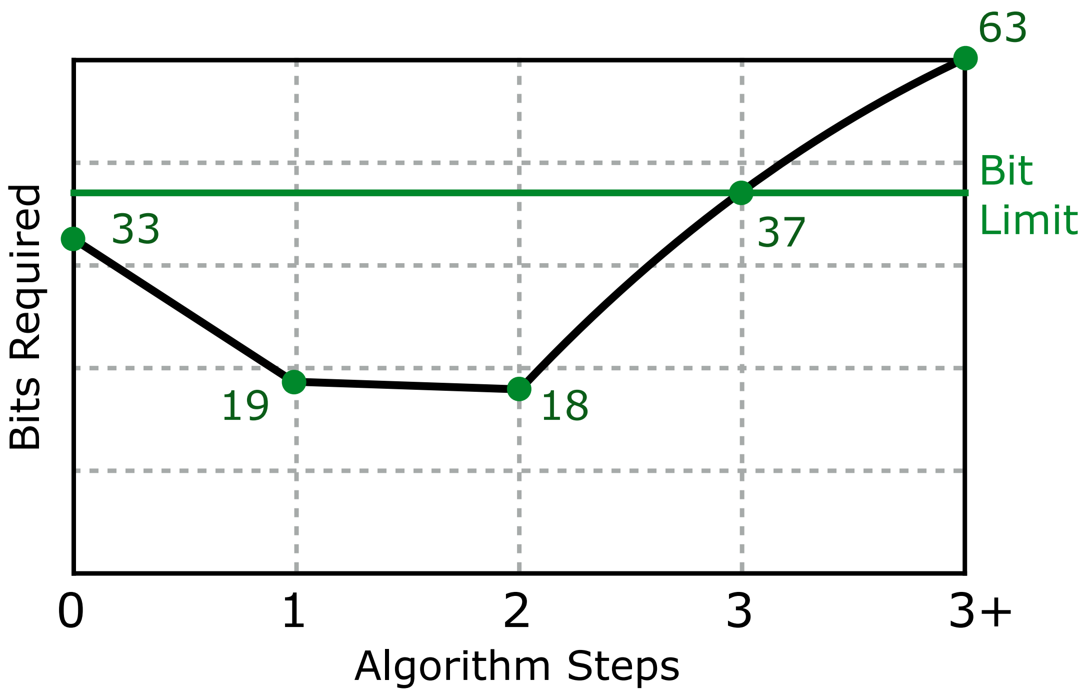

Figure 4.5: Graph of the number of bits required by the encoding scheme at each step of the algorithm, shown with real values computed over the AMS-IX RIPE data set. Step 0 is the input of all sets, step 1 is removal of subsets, step 2 is attempting greedy bit minimization, step 3 is after greedily decreasing inflation up to the bit limit, and step 3+ is with no bit limit.

Chapter 5

Related Work

Software-Defined IXPs have been gaining attention recently [7][8], with exchange point operators seeking testbeds and real deployments. Google's Cardigan project was deployed in a live internet exchange [9], but Cardigan does not appear to use any compression techniques and thus we assume that the inflation factor of their forwarding tables is comparable to the "naive" case we discuss, implying the number of forwarding entries would be far too large for available commodity switches when given reasonably-sized policies. The original SDX [2] paper introduced the concept of compression of routing information into the forwarding plane with the MDS algorithm for labeling packets which are subject to equivalent forwarding rules. It also introduced the idea of using the MAC address to store metadata, by advertising metadata as next-hop addresses. Despite these improvements, the project was unable to meet the scale or speed requirements of even moderately-sized IXPs. The recent iSDX [4] paper presents the full system which uses the compression technique detailed in this paper. In addition to a brief description of the technique, the work also discusses partitioning of outbound and inbound flow rules to avoid additional inflation, as well as tagging of packets to ensure participant flows are processed disjointly.

Chapter 6

Future Use

We will end with speculation about applications of the encoding scheme. In our problem setting at a high level, some actor attaches a list of boolean statements to each packet before it enters a layer-2 network. Once the packet has entered the network, forwarding table entries are able to quickly check for and react to the truth of any specific statement with only a small number of entries.

In our context, the actor which attaches the list of true statements is the SDN controller (indirectly), by relaying the metadata to be attached as a MAC to the participant border routers. The list of boolean statements is the list of valid routes, as advertised by BGP. If a bit is 0 or not present in the metadata, the statement is assumed to be false, or in other words the route is assumed to be invalid.

There are several properties that must be present for this scheme to work well. In order for the scheme to function at all, no list of statements must be too large, and there must be a large amount of redundancy in the possible lists of statements: many lists must be improper subsets of other lists. This is true in our setting because each list corresponds to participants which advertise some prefix, and many prefixes are announced by identical small sets of participants.

An additional property that is worth note is that OpenFlow does not yet support masked writing, only masked reading. Because of this, the list of statements cannot be partially modified as it traverses the network. Instead, it must be either be fully overwritten or left unmodified while in the core. However, end hosts are of course able to modify the set one bit at a time, so long as they are able to modify individual, arbitrary bits of the ethernet fields.

We also cannot forget that the metadata is composed of a column identifier, mask, and "mini MAC". The number of bits which must be allocated for the mini MAC is logarithmic in the number of nodes in order for each node to have a unique address. Formally, for the scheme to work, if you have X non-subset statement lists where the largest list is length Y, and you have Z nodes in your network, then it must be the case that $\log_2 X + Y + \log_2 Z \leq 48$

One example to which our scheme could apply would be selective anycast in a local network. If a flow could be accepted by any single host in a specified set of hosts, the set could be encoded with our scheme. Switches in the network could check this set and choose any output port which leads to one of the hosts in the set.

Another example is encoding an unordered list of network functions which a flow should traverse before it reaches its destination. If a switch sees that a boolean in the metadata is true for a nearby network function, it could forward the flow to that function. Once the function has processed the flow, it could then flip the boolean to false. This is only possible for network functions implemented as hosts which can modify arbitrary bits in the header; the network functions could not be on current switches or hardware routers.

Although maybe without real motivation, these are just examples showing that any problem which fits the mold of encoding a set can be solved by our encoding scheme, if the required properties hold.

Chapter 7

Conclusion

Our reachability encoding scheme has shown the ability to compress a large amount of routing information into a small number of bits. Although our experiments were only for the RIPE-available information on 63 participants, in the full iSDX paper the encoding scheme was able to represent subsets of over 500 participants in only 33 bits [4]. We have yet to find an example of an exchange point where this technique breaks down, and so it seems that this approach will allow real software-defined exchange points to be deployed in the near future.

Since this technique is completed, any future work would be in finding new uses for the technique. However, it would be interesting to explore how this technique could be expanded or modified when additional capabilities are added to SDN switches.

Bibliography

[1] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. OpenFlow: Enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 2008.

[2] Arpit Gupta, Laurent Vanbever, Muhammad Shahbaz, Sean Patrick Donovan, Brandon Schlinker, Nick Feamster, Jennifer Rexford, Scott Shenker, Russ Clark, and Ethan Katz-Bassett. SDX: A Software Defined Internet Exchange. In *ACM SIGCOMM*, pages 579–580, Chicago, IL, 2014. ACM.

[3] Openflow 1.3 specifications. http://bit.ly/1eyrkxY.

[4] Arpit Gupta, Robert MacDavid, Rüdiger Birkner, Marco Canini, Nick Feamster, Jen Rexford, and Laurent Vanbever. An industrial-scale software defined internet exchange point. In *NSDI*, 2016.

[5] AMS Internet Exchange. https://www.ams-ix.net/ams-ix-route-servers/, 2013.

[6] RIPE Routing Information Service (RIS). http://www.ripe.net/ris.

[7] Workshop on Prototyping and Deploying Experimental Software Defined Exchanges. https://www.nitrd.gov/nitrdgroups/images/4/4d/SDX_Workshop_Proceedings.pdf.

[8] Could IXPs Use OpenFlow To Scale? http://www.menog.org/presentations/menog-12/116-Could_IXPs_Use_OpenFlow_To_Scale.pdf, 2013.

[9] J. Stringer, D. Pemberton, Qiang Fu, C. Lorier, R. Nelson, J. Bailey, C.N.A. Correa, and C. Esteve Rothenberg. Cardigan: Sdn distributed routing fabric going live at an internet exchange. In *Computers and Communication (ISCC), 2014 IEEE Symposium on*, pages 1–7. IEEE, June 2014.