Scalable Flow-Based Networking with DIFANE

Minlan Yu* Jennifer Rexford* Michael J. Freedman* Jia Wang†

* Princeton University, Princeton, NJ, USA † AT&T Labs - Research, Florham Park, NJ, USA

ABSTRACT

Ideally, enterprise administrators could specify fine-grain policies that drive how the underlying switches forward, drop, and measure traffic. However, existing techniques for flow-based networking rely too heavily on centralized controller software that installs rules reactively, based on the first packet of each flow. In this paper, we propose DIFANE, a scalable and efficient solution that keeps all traffic in the data plane by selectively directing packets through intermediate switches that store the necessary rules. DIFANE relegates the controller to the simpler task of partitioning these rules over the switches. DIFANE can be readily implemented with commodity switch hardware, since all data-plane functions can be expressed in terms of wildcard rules that perform simple actions on matching packets. Experiments with our prototype on Click-based OpenFlow switches show that DIFANE scales to larger networks with richer policies.

Categories and Subject Descriptors: C.2.1[Computer Communication Networks]: Network Architecture and Design

General Terms: Algorithms, Design, Management

Keywords: Access Control, Network Architecture, OpenFlow, Scalability

1. INTRODUCTION

The emergence of flow-based switches [1, 2] has enabled enterprise networks that support flexible policies. These switches perform simple actions, such as dropping or forwarding packets, based on rules that match on bits in the packet header. Installing all of the rules in advance is not attractive, because the rules change over time (due to policy changes and host mobility) and the switches have relatively limited high-speed memory (such as TCAMs). Instead, current solutions rely on directing the first packet of each "microflow" to a centralized controller that reactively installs the appropriate rules in the switches [3, 4]. In this paper, we argue that the *switches themselves* should collectively perform this function, both to avoid a bottleneck at the controller and to keep all traffic in the data plane for better performance and scalability.

SIGCOMM'10, August 30–September 3, 2010, New Delhi, India.

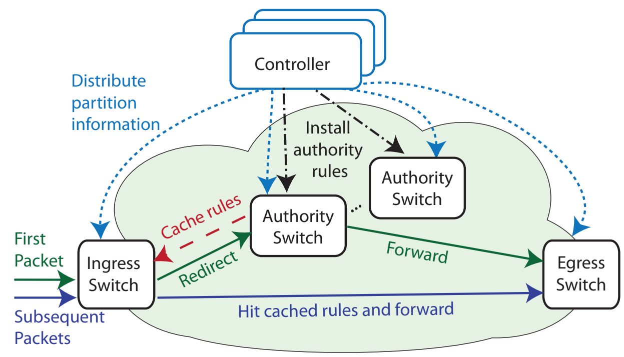


Figure 1: DIFANE flow management architecture. (Dashed lines are control messages. Straight lines are data traffic.)

1.1 DIFANE: Doing It Fast ANd Easy

Our key challenge, then, is to determine the appropriate "division of labor" between the controller and the underlying switches, to support high-level policies in a scalable way. Previous work has demonstrated that a logically-centralized controller can track changes in user locations/addresses and compute rules the switches can apply to enforce a high-level policy [4, 5, 6]. For example, an access-control policy may deny the engineering group access to the human-resources database, leading to low-level rules based on the MAC or IP addresses of the current members of the engineering team, the IP addresses of the HR servers, and the TCP port number of the database service. Similar policies could direct packets on customized paths, or collect detailed traffic statistics. The controller can generate the appropriate switch rules simply by substituting high-level names with network addresses. The policies are represented with 30K - 8M rules in the four different networks we studied. This separation of concerns between rules (in the switches) and policies (in the controller) is the basis of several promising new approaches to network management [3, 7, 8, 9, 10].

While we agree the controller should *generate* the rules, we do not think the controller should (or needs to) be involved in the real-time handling of data packets. Our DIFANE (DIstributed Flow Architecture for Networked Enterprises) architecture, illustrated in Figure 1, has the following two main ideas:

- The controller **distributes the rules** across (a subset of) the switches, called "authority switches," to scale to large topologies with many rules. The controller runs a partitioning algorithm that divides the rules evenly and minimizes fragmentation of the rules across multiple authority switches.
- The switches handle **all packets in the data plane** (*i.e.*, TCAM), diverting packets through authority switches as needed to access the appropriate rules. The "rules" for diverting packets are themselves naturally expressed as TCAM entries.

All data-plane functionality in DIFANE is expressible in terms of wildcard rules with simple actions, exactly the capabilities of commodity flow switches. As such, a DIFANE implementation requires only modifications to the control-plane software of the authority switches, and no data-plane changes in any of the switches. Experiments with our prototype, built on top of the Click-based OpenFlow switch [11], illustrate that distributed rule management in the data plane provides lower delay, higher throughput, and better scalability than directing packets through a separate controller.

Section 2 presents our main design decisions, followed by our DIFANE architecture in Section 3. Next, Section 4 describes how we handle network dynamics, and Section 5 presents our algorithms for caching and partitioning wildcard rules. Section 6 presents our switch implementation, followed by the performance evaluation in Section 7. Section 8 describes different deployment scenarios of DIFANE. The paper concludes in Section 9.

1.2 Comparison to Related Work

Recent work shows how to support policy-based management using flow switches [1, 2] and centralized controllers [4, 5, 6, 3]. The most closely related work is the Ethane controller that reactively installs flow-level rules based on the first packet of each TCP/UDP flow [3]. The Ethane controller can be duplicated [3] or distributed [12] to improve its performance. In contrast, DIFANE distributes wildcard rules amongst the switches, and handles all data packets in the data plane. Other recent work capitalizes on OpenFlow to rethink network management in enterprises and data centers [7, 8, 9, 10]; these systems could easily run as applications on top of DIFANE.

These research efforts, and ours, depart from traditional enterprise designs that use IP routers to interconnect smaller layer-two subnets, and rely heavily on inflexible mechanisms like VLANs. Today, network operators must configure Virtual LANs (VLANs) to scope broadcast traffic and direct traffic on longer paths through routers that perform access control on IP and TCP/UDP header fields. In addition, an individual MAC address or wall jack is typically associated with just *one* VLAN, making it difficult to support more fine-grained policies that treat different traffic from the same user or office differently.

Other research designs more scalable networks by selectively directing traffic through intermediate nodes to reduce routing-table size [13, 14, 15]. However, hash-based redirection techniques [13, 14], while useful for flat keys like IP or MAC addresses, are not appropriate for look-ups on rules with wildcards in arbitrary bit positions. ViAggre [15] subdivides the IP prefix space, and forces some traffic to always traverse an intermediate node, and does not consider on-demand cache or multi-dimensional, overlapping rules.

2. DIFANE DESIGN DECISIONS

On the surface, the simplest approach to flow-based management is to install all of the low-level rules in the switches in advance. However, preinstalling the rules does not scale well in networks with mobile hosts, since the same rules would need to be installed in multiple locations (*e.g.*, any place a user might plug in his laptop). In addition, the controller would need to update many switches whenever rules change. Even in the absence of mobile devices, a network with many rules might not have enough table space in the switches to store all the rules, particularly as the network grows or its policies become more complex. Instead, the system should install rules on demand [3].

To build a flow-processing system that has high performance and scales to large networks, DIFANE makes four high-level design decisions that reduce the overhead of handling cache misses and allow the system to scale to a large number of hosts, rules, and switches.

2.1 Reducing Overhead of Cache Misses

Reactively caching rules in the switches could easily cause problems such as packet delay, larger buffers, and switch complexity when cache misses happen. More importantly, misbehaving hosts could easily trigger excessive cache misses simply by scanning a wide range of addresses or port numbers — overloading TCAM and introducing extra packet-processing overhead. DIFANE handles "miss" packets *efficiently* by keeping them in the data plane and reduces the *number* of "miss" packets by caching wildcard rules.

Process all packets in the data plane: Some flow management architectures direct the first packet (or first packet header) of each microflow to the controller and have the switch buffer the packet awaiting further instructions [3].[1] In a network with many short flows , a controller that handles "miss" packets can easily become a bottleneck. In addition, UDP flows introduce extra overhead, since multiple (potentially large) packets in the same flow may be in flight (and need to visit the controller) at the same time. The switches need a more complex and expensive buffering mechanism, because they must temporarily store the "miss" packets while continuing to serve other traffic, and then retrieve them upon receiving the rule. Instead, DIFANE makes it cheap and easy for switches to forward *all* data packets in the data plane (*i.e.*, hardware), by directing "miss" packets through an intermediate switch. Transferring packets in the data plane through a slightly longer path is much faster than handling packets in the control plane.

Efficient rule caching with wildcards: Caching a separate low-level rule for each TCP or UDP microflow [3], while conceptually simple, has several disadvantages compared to wildcard rules. For example, a wildcard rule that matches on all destinations in the 123.132.8.0/22 subnet would require up to 1024 microflow rules. In addition to consuming more data-plane memory on the switches, fine-grained rules require special handling for more packets (*i.e.*, the first packet of each microflow), leading to longer delays and higher overhead, and more vulnerability to misbehaving hosts. Instead, DIFANE supports wildcard rules, to

[1] Another solution to handle cache miss is for the switch to encapsulate and forward the entire packet to the controller. This is also problematic because it significantly increases controller load.

have fewer rules (and fewer cache "misses") and capitalize on TCAMs in the switches. Caching wildcard rules introduces several interesting technical challenges that we address in our design and implementation.

2.2 Scaling to Large Networks and Many Rules

To scale to large networks with richer policies, DIFANE divides the rules across the switches and handles them in a distributed fashion. We also keep consistent topology information among switches by leveraging link-state protocols.

Partition and distribute the flow rules: Replicating the controller seems like a natural way to scale the system and avoid a single point of failure. However, this requires each controller to maintain all the rules, and coordinate with the other replicas to maintain consistency when rules change. (Rules may change relatively often, not only because the policy changes, but also because host mobility triggers changes in the mapping of policies to rules.) Instead, we *partition* the space of rules to reduce the number of rules each component must handle and enable simpler techniques for maintaining consistency. As such, DIFANE has one primary controller (perhaps with backups) that manages policies, computes the corresponding rules, and divides these rules across the switches; each switch handles a portion of the rule space and receives updates only when those rules change. That is, while the switches *reactively* cache rules in response to the data traffic, the DIFANE controller *proactively* partitions the rules across different switches.

Consistent topology information distribution with the link-state protocol: Flow-based management relies on the switches having a way to communicate with the controller and adapt to topology changes. Relying on rules for this communication introduces circularity, where the controller cannot communicate with the switches until the appropriate rules have been installed. Rather than bootstrapping communication by having the switches construct a spanning tree [3], we advocate running a link-state protocol amongst the switches. Link-state routing enables the switches to compute paths and learn about topology changes and host location changes without involving the controller, reducing overhead and also removing the controller from the critical path of failure recovery. In addition, link-state routing scales to large networks, enables switches to direct packets through intermediate nodes, and reacts quickly to switch failure [13]. As such, DIFANE runs a link-state routing protocol amongst the switches, while also supporting flow rules that allow customized forwarding of traffic between end hosts. The controller also participates in link-state routing to reach the switches and learn of network topology changes.[2]

3. DIFANE ARCHITECTURE

The DIFANE architecture consists of a *controller* that generates the rules and allocates them to the *authority switches*, as shown in Figure 1. Authority switches can be a subset of existing switches in the network (including ingress/egress switches), or dedicated switches that have larger memory and processing capability.

[2]The links between the controller and switches are set with high link-weights so that traffic between switches do not go through the controller.

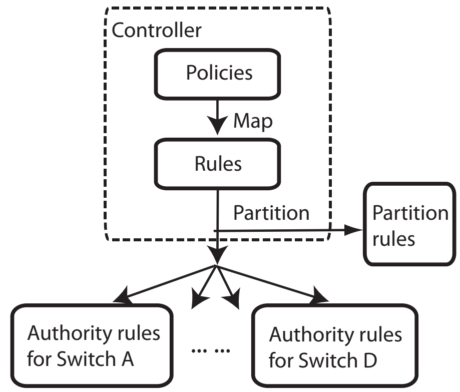


Figure 2: Rule operations in the controller.

Upon receiving traffic that does not match the cached rules, the ingress switch encapsulates and redirects the packet to the appropriate authority switch based on the partition information. The authority switch handles the packet in the data plane and sends feedback to the ingress switch to cache the relevant rule(s) locally. Subsequent packets matching the cached rules can be encapsulated and forwarded directly to the egress switch.

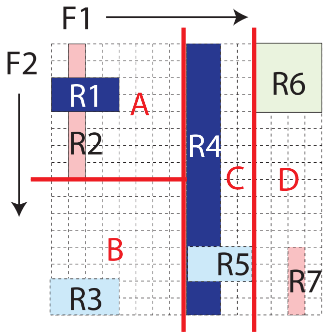
In this section, we first discuss how the controller partitions the rules and distributes the authority and partition rules to the switches. Next, we describe how a switch directs packets through the authority switch and caches the necessary rules, using link-state routing to compute the path to the authority switch. Finally, we show that the data-plane functionality of DIFANE can be easily implemented on today's flow-based switches using wildcard rules.

3.1 Rule Partition and Allocation

As shown in Figure 2, we use the controller to *pre-compute* the low-level rules, generate *partition rules* that describe which low-level rules are stored in which authority switches, and then distribute the partition rules to all the switches. The partition rules are represented by coarse-grained wildcard rules on the switches.

Precompute low-level rules: The controller pre-computes the low-level rules based on the high-level policies by simply substituting high-level names with network addresses. Since low-level rules are pre-computed and installed in the TCAM of switches, we can always process packets in the fast path. Most kinds of policies can be translated to the low-level rules in advance because the controller knows the addresses of the hosts when the hosts first connect to the ingress switch, and thus can substitute the high-level names with addresses. However, precomputation is not an effective solution for policies (like traffic engineering) that depend on dynamically changing network state.

Use partitioning to subdivide the space of all rules: Hashing is an appealing way to subdivide the rules and direct packets to the appropriate authority switch. While useful for flat keys like an IP or MAC address [13, 14], hashing is not effective when the keys can have wildcards in arbitrary bit positions. In particular, packets matching the same wildcard rule would have different hash values, leading them to different authority switches; as a result, multiple authority switches would need to store the same wildcard rule. Instead of relying on hashing, DIFANE *partitions* the rule space, and assigns each portion of rule space to one or more authority



(a) Low-level rules and the partition

Type	Priority	F_1	F_2	Action	Timeout	Note
Cache rules	210	00**	111*	Encap, forward to B	10 sec	R_3
	209	1110	11**	Drop	10 sec	R_7
	...	...	...	... ...	...	...
Authority rules	110	00**	001*	Encap, forward to D, trigger ctrl. plane func.	∞	R_1
	109	0001	0***	Drop, trigger ctrl. plane func.	∞	R_2
Partition rules	15	0***	000*	Encap, redirect to B	∞	Primary
	14	0***	1***	Encap, redirect to C	∞	
	13	11**	****	Encap, redirect to D	∞	
	5	0***	000*	Encap, redirect to B'	∞	Backup
	...	...	...	... ...	...	...

(b) Wildcard rules in the TCAM of Switch A

Figure 3: Wildcard rules in DIFANE (A-D are authority switches).

switches. Each authority switch stores the rules falling in its part of the partition.

Run the partitioning algorithm on the controller: Running the partitioning algorithm on the switches themselves would introduce a large overhead, because they would need to learn the rules from the controller, run the partitioning algorithm, and distribute the results. In contrast, the controller is a more natural place to run the partitioning algorithm. The controller is already responsible for translating policies into rules and can easily run the partitioning algorithm periodically, as the distribution of low-level rules changes. We expect the controller would recompute the partition relatively infrequently, as most rule changes would not require rebalancing the division of rule space. Section 4 discusses how DIFANE handles changes to rules with minimal interruption to the data traffic.

Represent the partition as a small collection of partition rules: Low-level rules are defined as actions on a *flow space*. The flow space usually has seven dimensions (source/destination IP addresses, MAC addresses, ports, the protocol) or more. Figure 3(a) shows a two-dimensional flow space (F_1, F_2) and the rules on it. The bit range of each field is from 0 to 15 (*i.e.*, $F_1 = F_2 = [0..15]$). For example, F_1, F_2 can be viewed as the source/destination fields of a packet respectively. Rule R_2 denotes that all packets which are from source 1 ($F_1 = 1$) and forwarded to a destination in [0..7] ($F_2 = [0..7]$) should be dropped.

The controller partitions the flow space into M *ranges*, and assigns each range to an authority switch. The resulting partition can be expressed concisely as a small number of coarse-grain *partition rules*, where M is proportional to the number of authority switches rather than the number of low-level rules. For example, in Figure 3(a), the flow space is partitioned into four parts by the straight lines, which are represented by the partition rules in Figure 3(b). Section 5 discusses how the controller computes a partition of overlapping wildcard rules that reduces TCAM usage.

Duplicate authority rules to reduce stretch and failure-recovery time: The first packet covered by an authority rule traverses a longer path through an authority switch. To reduce the extra distance the traffic must travel (*i.e.*, "stretch"), the controller can assign each of the M ranges to *multiple* authority switches. For example, if each range is handled by two authority switches, the controller can generate two partition rules for each range, and assign each switch the rule that would minimize stretch. That way, on a cache miss,[3] a switch directs packets to the *closest* authority switch responsible for that range of rules. The placement of multiple authority switches is discussed in Section 5.

Assigning multiple authority switches to the same range can also reduce failure-recovery time. By pushing *backup* partition rules to every switch, a switch can quickly fail over to the backup authority switch when the primary one fails (see Section 4). This requires each switch to store more partition rules (*e.g.*, $2M$ instead of M), in exchange for faster failure recovery. For example, in Figure 3(b), switch A has a primary partition rule that directs packets to B and a backup one that directs packets to B'.

3.2 Packet Redirection and Rule Caching

The authority switch stores the authority rules. The ingress switch encapsulates the first packet covered by an authority switch and redirects it to the authority switch.[4] The authority switch processes the packet and also caches rules in the ingress switch so that the following packets can be processed at the ingress switch.

Packet redirection: In the ingress switch, the first packet of a wildcard flow matches a partition rule. The partition rule indicates which authority switch maintains the authority rules that are related to the packet. For example, in Figure 3 a packet with ($F_1 = 9, F_2 = 7$) hits the primary partition rule for authority switch B and should be redirected to B. The ingress switch encapsulates the packet and forwards it to the authority switch. The authority switch decapsulates the packet, processes it, re-encapsulates it, and forwards it to the egress switch.

Rule Caching: To avoid redirecting all the data traffic to the authority switch, the authority switch caches the rules in the ingress switch.[5] Packets that match the cache rules are encapsulated and forwarded directly to the egress switch (*e.g.*, packets matching R_3 in Figure 3(b) are encapsulated and forwarded to D). In DIFANE "miss" packets do not wait for rule caching, because they are sent through the authority switch rather than buffered at the ingress switch. Therefore,

[3]In DIFANE, every packet matches some rule in the switch. "Cache miss" in DIFANE means a packet does not match any cache rules, but matches a partition rule instead.

[4]With encapsulation, the authority switch knows the address of the ingress switch from the packet header and sends the cache rules to the ingress switch.

[5]Here we assume that we cache flow rules only at the ingress switch. We discuss the design choices of where to cache flow rules in our technical report [16].

we can run a simple caching function in the control plane of the authority switch to generate and install cache rules in the ingress switch. The caching function is triggered whenever a packet matches the authority rules in the authority switch. The cache rule has an idle timeout so that it can be removed by the switch automatically due to inactivity.

3.3 Implement DIFANE with Wildcard Rules

All the data plane functions required in DIFANE can be expressed with three sets of wildcard rules of various granularity with simple actions, as shown in Figure 3(b).

Cache rules: The ingress switches cache rules so that most of the data traffic hits in the cache and is processed by the ingress switch. The cache rules are installed by the authority switches in the network.

Authority rules: Authority rules are only stored in authority switches. The controller installs and updates the authority rules for all the authority switches. When a packet matches an authority rule, it triggers a control-plane function to install rules in the ingress switch.

Partition rules: The controller installs partition rules in each switch. The partition rules are a set of *coarse-grained* rules. With these partition rules, we ensure a packet will always match at least one rule in the switch and thus always stay in the data plane.

The three sets of rules can be easily expressed as a single list of wildcard rules with different priorities. Priorities are naturally supported by TCAM. If a packet matches multiple rules, the packet is processed based on the rule that has the highest priority. The cached rules have highest priority because packets matching cache rules do not need to be directed to authority switches. In authority switches, authority rules have higher priority than partition rules, because packets matching authority rules should be processed based on these rules. The primary partition rules have higher priority than backup partition rules.

Since all functionalities in DIFANE are expressed with wildcard rules, DIFANE does not require any data-plane modifications to the switches and only needs minor software extensions in the control plane of the authority switches.

4. HANDLING NETWORK DYNAMICS

In this section, we describe how DIFANE handles dynamics in different parts of the network: To handle rule changes at the controller, we need to update the authority rules in the authority switches and occasionally repartition the rules. To handle topology changes at the switches, we leverage link-state routing and focus on reducing the interruptions of authority switch failure and recovery. To handle host mobility, we dynamically update the rules for the host, and use redirection to handle changes in the routing rules.

4.1 Changes to the Rules

The rules change when administrators modify the policies, or network events (*e.g.*, topology changes) affect the mapping between policies and rules. The related authority rules, cache rules, and partition rules in the switches should be modified correspondingly.

Authority rules are modified by the controller directly: The controller changes the authority rules in the related authority switches. The controller can easily identify the related authority switches based on the partition it generates.

Cache rules expire automatically: Cached copies of the old rules may still exist in some ingress switches. These cache rules will expire after the timeout time. For critical changes (*e.g.*, preventing DoS attacks), the authority switches can get the list of all the ingress switches from the link-state routing and send them a message to evict the related TCAM entries.

Partition rules are recomputed occasionally: When the rules change, the number of authority rules in the authority switches may become unbalanced. If the difference in the number of rules among the authority switches exceeds a threshold, the controller recomputes the partition of the flow space. Once the new partition rules are generated, the controller notifies the switches of the new partition rules, and updates the authority rules in the authority switches.

The controller cannot update all the switches at exactly the same time, so the switches may not have a consistent view of the partition during the update, which may cause transient loops and packet loss in the network. To avoid packet loss, the controller simply updates the switches in a specific order. Assume the controller decides to move some authority rules from authority switch A to B. The controller first sends the authority rules to authority switch B, before sending the new partition rules for A and B to all the switches in the network. Meanwhile, switches can redirect the packets to either A or B for the authority rules. Finally, the controller deletes the authority rules in switch A. In this way, we can prevent packet loss during the change. The same staged update mechanism also applies to the partition change among multiple authority switches.

4.2 Topology Dynamics

Link-state routing enables the switches to learn about topology changes and adapt routing quickly. When authority switches fail or recover, DIFANE adapts the rules to reduce traffic interruption.

Authority switch failure: When an authority switch fails, packets directed through it are dropped. To minimize packet loss, we must react quickly to authority switch failures. We design a distributed authority switch takeover mechanism. As discussed in Section 3.1, the controller assigns the same group of authority rules to multiple authority switches to reduce stretch and failure-recovery time. Each ingress switch has primary partition rules directing traffic to their closest authority switch and backup partition rules with lower priority that directing traffic to another authority switch when the primary one fails.

The link-state routing protocol propagates a message about the switch failure throughout the network. Upon receiving this message, the switches invalidate their partition rules that direct traffic to the failed authority switch. As a result, the backup partition rule takes effect and automatically directs packets through the backup authority switch. For the switches that have not yet received the failure information, the packets may get sent towards the failed authority switch, but will finally get dropped by a switch who has updated its switch forwarding table.[6]

[6]If the switch can decapsulate the packet and encapsulate it with the backup authority switch as the destination, we can avoid such packet loss.

Authority switch addition/recovery: We use the controller to handle switches joining in the network, because it does not require fast reaction compared to authority switch failures. To minimize the change of the partition rules and authority rules, the controller randomly picks an authority switch, divides its flow range evenly into two parts. The controller then moves one part of the flow range to the new switch, and installs the authority rules in the new switch. Finally the controller updates the partition rules correspondingly in all the switches.

4.3 Host Mobility

In DIFANE, when a host moves, its MAC and perhaps IP address stays the same. The rules in the controller are defined based on these identifiers, and thus do not change as hosts move. As a result, the partition rules and the authority rules also stay the same.[7] Therefore we only need to consider the changes of cache rules.

Installing rules at the new ingress switch on demand: When a host connects to a new ingress switch, the switch may not have the cache rules for the packets sent by the hosts. So the packets are redirected to the responsible authority switch. The authority switch then caches rules at the new ingress switch.

Removing rules from old ingress switch by timeout: Today's flow-based switches usually have a timeout for removing the inactive rules. Since the host's old ingress switch no longer receives packets from the moved host, the cache rules at the switch are automatically removed once the timeout time expires.

Redirecting traffic from old ingress switch to the new one: The rules for routing packets to the host change when the host moves. When a host connects to a new switch, the controller gets notified through the link-state routing and constructs new routing rules that map the address of the host to the corresponding egress switch. The new routing rules are then installed in the authority switches.

The cached routing rules in some switches may be outdated. Suppose a host H moves from ingress switch S_{old} to S_{new}. The controller first gets notified about the host movement. It then installs new routing rules in the authority switch, and also installs a rule in S_{old} redirecting packets to S_{new}. If an ingress switch A receives a packet whose destination is H, A may still send the packet to the old egress point S_{old} if the cache rule has not expired. S_{old} then redirects packets to S_{new}. After the cache rule expires in switch A, A directs the packets to the authority switch for the correct egress point S_{new} and caches the new routing rule.

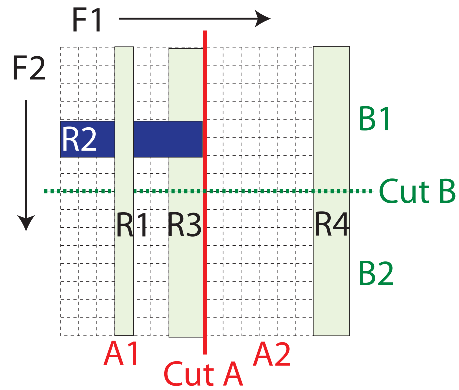
5. HANDLING WILDCARD RULES

Most flow-management systems simply use microflow rules [3] or transform overlapping wildcard rules into a set of non-overlapping wildcard rules. However these methods significantly increase the number of rules in switches, as shown in our evaluation in Section 7. To the best of our knowledge, there is no systematic and efficient solution for handling overlapping wildcard rules in *network-wide* flow-management systems. In this section, we first propose a simple and efficient solution for multiple authority switches to independently insert cache rules in ingress switches. We then discuss the key ideas of partitioning overlapping wildcard rules, deferring the description of our algorithm to the Appendix.

[7]In some enterprises, a host changes its identifier when it moves. The rules also change correspondingly. We can use the techniques in Section 4.1 to handle the rule changes.

Rule	F_1	F_2	Action
R_1	4	0-15	Accept
R_2	0-7	5-6	Drop
R_3	6-7	0-15	Accept
R_4	14-15	0-15	Accept

(a) Wildcard rules listed in the decreasing order of priority. ($R_1 > R_2 > R_3 > R_4$)



(b) Graphical view and two partition solutions.

Figure 4: An illustration of wildcard rules.

5.1 Caching Wildcard Rules

Wildcard rules complicate dynamic caching at ingress switches. In the context of access control, for example in Figure 4, the packet $(F_1 = 7, F_2 = \mathbf{0})$ matches an "accept" rule R_3 that overlaps with "deny" rule R_2 which has higher priority. Simply caching R_3 is not safe. If we just cache R_3 in the ingress switch, another packet $(F_1 = 7, F_2 = \mathbf{5})$ could incorrectly pass the cached rule R_3, because the ingress switch is not aware of the rule R_2. Thus, because rules can overlap with each other, the authority switch cannot *solely* cache the rule that a packet matches. This problem exists in all the flow management systems that cache wildcard rules and therefore it is not trivial to extend Ethane controllers [3] to support wildcards.

To address this problem, DIFANE constructs *one or more new wildcard rules that cover the largest flow range (*i.e., *a hypercube in a flow space) in which all packets take the same action.* We use Figure 4 to illustrate the solution. Although the rules overlap, which means a packet may match multiple rules, the packet only takes the action of the rule with the highest priority. That is, each point in the flow space has a unique action (which is denoted by the shading in each spot in Figure 4(b)). As long as we cache a rule that covers packets with the same action (*i.e.*, spots with the same shading), we ensure that the caching preserves semantic correctness. For example, we cannot cache rule R_3 because the spots it covers have different shading. In contrast, we can safely cache R_1 because all the spots it covers has the same shading. For the packet $(F_1 = 7, F_2 = \mathbf{0})$, we construct and cache a *new* rule: $F_1 = [6..7], F_2 = [0..3]$.

The problem of constructing new wildcard rules for caching at a *single* switch was studied in [17]. Our contribution lies in extending this approach to *multiple* authority switches, each of which can independently install cache rules at ingress switches. Given such a setting, we must prevent authority switches from installing *conflicting* cache rules. To guarantee this, DIFANE ensures that the caching rules installed by different authority switches do not overlap. This is achieved by *allocating non-overlapping flow ranges to the authority switches, and only allowing the authority switch to install caching rules in its own flow range*. We later evaluate our caching scheme in Section 7.

5.2 Partitioning Wildcard Rules

Overlapping wildcard rules also introduce challenges in partitioning. We first formulate the partition problem: The controller needs to partition rules into M parts to minimize the total number of TCAM entries across all M authority switches with the constraint that the rules should not take more TCAM entries than are available in the switches. There are three key ideas in the partition algorithm:

Allocating non-overlapping flow ranges to authority switches: As discussed in the caching solution, we must ensure that the flow ranges of authority switches do not overlap with each other. To achieve this goal, DIFANE first partitions the entire flow space into M flow ranges and then stores rules in each flow range in an authority switch. For example, the "Cut A" shown in Figure 4(b) partitions the flow space on field F_1 into two equal flow ranges A_1 and A_2. We then assign R_1, R_2 and R_3 in A_1 to one authority switch, and R_4 to another.

DIFANE splits the rules so that each rule only belongs to one authority switch. With the above partitioning approach, one rule may span multiple partitions. For example, "Cut B" partitions the flow space on field F_2, which results in the rules R_1, R_3, and R_4 spanning the two flow ranges B_1 and B_2. We split each rule into two independent rules by intersecting it with the two flow ranges. For example, the two new rules generated from rule R_4 are $F_1 = [14..15], F_2 = [0..7] \rightarrow$ *Accept* and $F_1 = [14..15], F_2 = [8..15] \rightarrow$ *Accept*. These two independent rules can then be stored in different authority switches. Splitting rules thus avoids the overlapping of rules among authority switches, but at the cost of increased TCAM usage.

To reduce TCAM usage, we prefer the cuts to align with rule boundaries. For example, "Cut A" is better than "Cut B" because "Cut A" does not break any rules. We also observe that cut on field F_1 is better than F_2 since we have more rule boundaries to choose. Based on these observations, we design a decision-tree based partition algorithm which is described in [16].

In summary, DIFANE partitions the entire rule space into M independent portions, and thus each authority switch is assigned a non-overlapping portion. However, *within* the portion managed by a single authority switch, DIFANE allows overlapping or nested rules. This substantially reduces the TCAM usage of authority switches (see Section 7).

Duplicating authority rules to reduce stretch: We can duplicate the rules in each partition on multiple authority switches to reduce stretch and to react quickly to authority switch failures. Due to host mobility, we cannot pre-locate authority switches to minimize stretch. Instead, we assume traffic that is related to one rule may come from any ingress switches and place replicated authority switches to reduce the average stretch. One simple method is to randomly place the replicated switches to reduce stretch. Alternatively, by leveraging an approximation algorithm for the "k-median problem" [18], we can place the replicated authority switches so that they have minimal average stretch to any pair of switches. Both schemes are evaluated in Section 7.

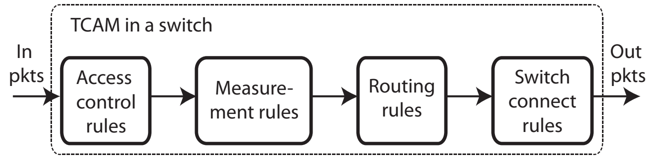


Figure 5: Rules for various management modules.

6. DESIGN AND IMPLEMENTATION

In this section, we present our design and implementation of DIFANE. First, we describe how our prototype handles multiple sets of rules from different kinds of high-level policies for different management functions. Second, we describe the prototype architecture, which just add a few control-plane functions for authority switches to today's flow-based switches.

6.1 Managing Multiple Sets of Rules

Different management functions such as access control, measurement and routing may have totally different kinds of policies. To make our prototype efficient and easy to implement, we generate different sets of rules for different policies, partition them using a single partition algorithm, and process them sequentially in the switch.

Generate multiple sets of low-level rules: Translating and combining different kinds of high-level policies into one set of rules is complicated and significantly increases TCAM usage. For example, if the policies are to monitor web traffic, and perform destination based routing, we have to provide rules for (dst, port 80) and (dst, other ports) for each destination dst. If the administrator changes the policy of monitoring port 21 rather than port 80, we must change the rules for every destination. In contrast, if we have different sets of rules, we only need one routing rule for each destination and a *single* measurement rule for port 80 traffic which is easy to change.

To distribute multiple sets of rules, the controller first partitions the flow space to minimize the total TCAM usage. It then assigns all the rules (of different management modules) in one flow range to one authority switch. We choose to use the same partition for different sets of low-level rules so that packets only need to be redirected to one authority switch to match all sets of authority rules.[8]

Processing packets through multiple sets of rules in switches: In switches we have one set of rules for each management module.[9] We process the flow rules sequentially through the rules for different modules as shown in Figure 5. We put access control rules first to block malicious traffic. Routing rules are placed later to identify the egress switch for the packets. Finally, the link-state rout-

[8]One can also choose to provide a different partition of the flow space for different sets of low-level rules, but packets may be redirected to multiple authority switches to match the authority rules of different management modules.

[9]To make the rule processing simple, we duplicate the same set of partition rules in the management modules.

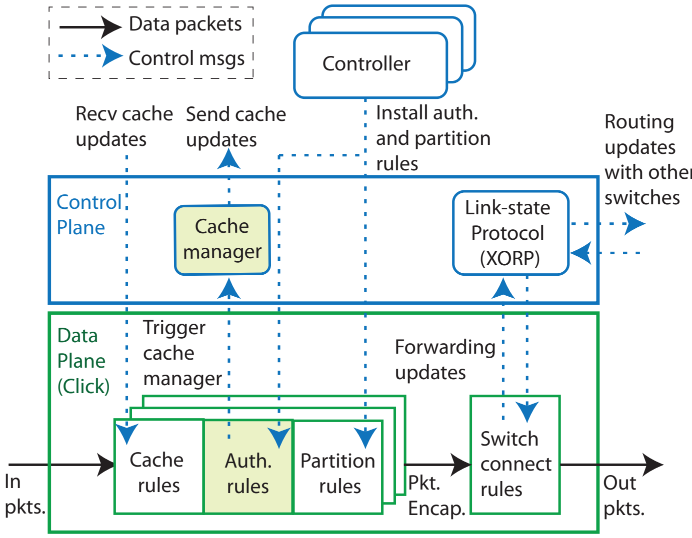


Figure 6: DIFANE prototype implementation. (Cache manager and authority rules (shaded boxes) only exist in authority switches.)

ing constructs a set of *switch connection rules* to direct the packets to their egress switches.

To implement sequential processing in the switch where all the rules share the same TCAM, the controller sets a "module identifier" in the rules to indicate the module they belong to. The switch first initializes a module identifier in the packet. It then matches the packet with the rules that have the same module identifier. Next, the switch increments the module identifier in the packet and matches to the next set of rules. By processing the packet several times through the memory, the switch matches the packet to the rules for different modules sequentially. The administrator specifies the order of the modules by giving different module identifiers for the high-level policies in the controller.

A packet may be redirected to the authority switch when it is in the middle of the sequential processing (*e.g.*, while being processed in the measurement module). After redirection, the packet will be processed through the following modules in the authority switch based the module identifier in the packet.

6.2 DIFANE Switch Prototype

Figure 6 shows both the control and data plane of our DIFANE switch prototype.

Control plane: We use XORP [19] to run the link-state routing protocol to maintain the switch-level connectivity, and keep track of topology changes. XORP also sends updates of the switch connection rules in the data plane.

The authority switch also runs a cache manager which installs cache rules in the ingress switch. If a packet misses the cache, and matches the authority rule in the authority switch, the cache manager is triggered to send a cache update to the ingress switch of the packet. The cache manager is implemented in software in the control plane because packets are not buffered and waiting for the cache rule in the ingress switch. The ingress switch continues to forward packets to the authority switch if the cache rules are not installed. The cache manager sends an update for *every* packet that matches the authority rule. This is because we can infer that the ingress switch does not have any related rules cached, otherwise it would forward the packets directly rather than sending them to the authority switch.[10]

Data plane: We run Click-based OpenFlow switch [11] in the kernel as the data plane of DIFANE. Click manages the rules for different management modules and encapsulates and forwards packets based on the switch connection rules. We implement the packet encapsulation function to enable tunneling in the Click OpenFlow element. We also modify the Click OpenFlow element to support the flow rule action "trigger the cache manager". If a packet matches the authority rules, Click generates a message to the cache manager through the kernel-level socket "netlink". Today's flow-based switches already support actions of sending messages to a *local controller* in order to communicate with the centralized controller [4]. We just add a new message type of "matching authority rules". In addition, today's flow-based switches already have interfaces for the centralized controller to install new rules. The cache manager then just leverages the same interfaces to install cache rules in the ingress switches.

7. EVALUATION

Ideally we would like to evaluate DIFANE based on policies, topology data, and user-mobility traces from real networks. Unfortunately, most networks today are still configured with rules that are tightly bound to their network configurations (*e.g.*, IP address assignment, routing, and VLANs). Therefore, we evaluated DIFANE's approach against the topology and access-control rules of a *variety* of different networks toexplore DIFANE's benefit across various settings. We also perform latency, throughput, and scalability micro-benchmarks of our DIFANE prototype and a trace-driven evaluation of our partition and caching algorithms.

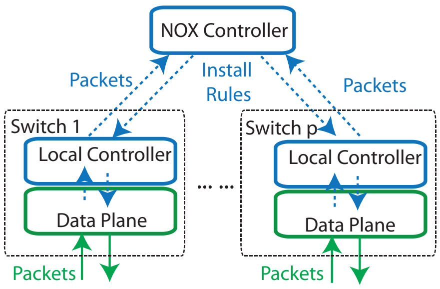
To verify the design decisions in Section 2, we evaluate two central questions in this section: (1) How efficient and scalable is DIFANE? (2) How well do our partition and caching algorithms work in handling large sets of wildcard rules?

7.1 Performance of the DIFANE Prototype

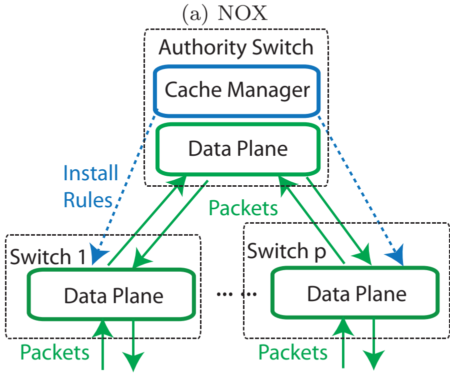
We implemented the DIFANE prototype using a kernel-level Click-based OpenFlow switch and compared the delay and throughput of DIFANE with NOX [4], which is a centralized solution for flow management. For a fair comparison, we first evaluated DIFANE using only one authority switch. Then we evaluated the throughput of DIFANE with multiple authority switches. Finally we investigated how fast DIFANE reacts to the authority switch failures.

In the experiments, each sender sends the packets to a receiver through a single ingress switch, which is connected directly to either NOX or a DIFANE authority switch as shown in Figure 7. (In this way, the network delay from the ingress switch to NOX and the DIFANE authority switch is minimized. We evaluate the extra delay caused by redirecting through authority switches in Section 7.2.) With NOX, when a packet does not match a cached rule, the packet

[10] For UDP flows, they may be a few packets sent to the authority switch. The authority switch sends one feedback for each UDP flow, because it takes very low overhead to send a cache update message (just one UDP packet). In addition, we do not need to store the existing cached flow rules in the authority switch or fetch them from the ingress switch.



(a) NOX



(b) DIFANE.

Figure 7: Experiment setup. (Dashed lines are control messages; straight lines are data traffic.)

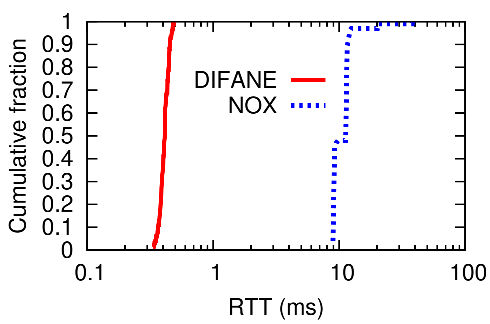


Figure 8: Delay comparison of DIFANE and NOX.

is buffered in the ingress switch before NOX controller installs a rule at the ingress switch. In contrast, in DIFANE the authority switch redirects the packet to the receiver in the data plane and installs a rule in the ingress switch at the same time. We generate flows with different port numbers, and use a separate rule for each flow. Since the difference between NOX and DIFANE lies in the processing of the first packet, we generate each flow as a single 64 Byte UDP packet. Based on the measurements, we also calculate the performance difference between NOX and DIFANE for flows of normal sizes. Switches, NOX, and traffic generators ("clients") run on separate 3.0 GHz 64-bit Intel Xeon machines to avoid interference between them.

(1) DIFANE achieves small delay for the first packet of a flow by always keeping packets in the fast path. In Figure 8, we send traffic at 100 single-packet flows/s and measure the round-trip time (RTT) of the each packet being sent through a switch to the receiver and an ACK packet being sent back. Although we put NOX near the switch, the packets still experience a RTT of 10 ms on average, which is not acceptable for those networks that have tight latency

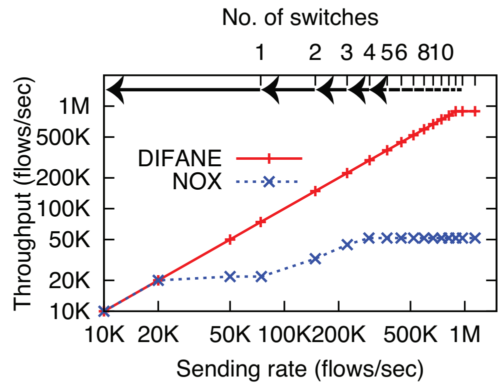


Figure 9: Throughput comparison of DIFANE and NOX.

requirement such as data centers. In DIFANE, since the packets stay in the fast path (forwarded through an authority switch), the packets only experience 0.4 ms RTT on average. Since all the following packets take 0.3 ms RTT for both DIFANE and NOX, we can easily calculate that to transfer a flow of a normal size 35 packets, which is based on the measurement in the paper [20]), the average packet transfer time is 0.3 ms (=(0.3*34+0.4)/35) transferring time for DIFANE but 0.58 ms (=(0.3*34+10)/35) for NOX. Others' tests of NOX with commercial OpenFlow switches observed similar delay [21].

(2) DIFANE achieves significantly higher throughput than NOX. We then increase the number of switches (p) to measure the throughput of NOX and DIFANE. In Figure 9, we show the maximum throughput of flow setup for one client using one switch. In DIFANE, the switch was able to achieve the client's maximum flow setup rate, 75K flows/s, while the NOX architecture was only able to achieve 20K flows/s. This is because, while all packets remain in the fast path (the software kernel) in DIFANE, the OpenFlow switch's local controller (implemented in user-space) becomes a bottleneck before NOX does. Today's commercial OpenFlow switches can only send 60-330 flows/s to the controller due to the limited CPU resources in the controller [22]. Section 8 discusses how to run DIFANE on today's commercial switches.

As we increase the number of ingress switches—each additional data-point represents an additional switch and client as shown in the upper x-axis—we see that the NOX controller soon becomes a bottleneck: With four switches, a single NOX controller achieves a peak throughput of 50K single-packet flows/s. Suppose a network has 1K switches, a single NOX controller can only support 50 new flows per second for each switch simultaneously. In comparison, the peak throughput of DIFANE with one authority switch is 800K single-packet flows/s.

Admittedly, this large difference exists because DIFANE handles packets in the kernel while NOX operates in user space. However, while it is possible to move NOX to the kernel, it would not be feasible to implement the entire NOX in today's switch hardware because the online rule generation too complex for hardware implementation. In contract, DIFANE is meant for precisely that — designed to install a set of rules in the data plane.[11]

[11]Today's TCAM with pipelined processing only takes 3–

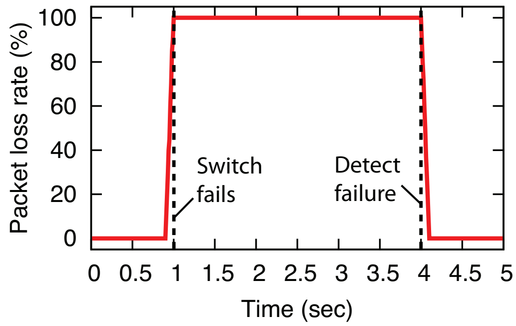


Figure 10: Authority switch failure.

Network	# switches/routers	# Rules
Campus	~1700	30K
VPN	~1500	59K
IPTV	~3000	5M
IP	~2000	8M

Table 1: Network characteristics.

We then evaluate the performance of the cache manager in authority switches. When there are 10 authority rules, the cache manager can handle 30K packets/s and generate one rule for each packet. When there are 9K authority rules, the cache manager can handle 12K packets/s. The CPU is the bottleneck of the cache manager.

(3) DIFANE scales with the number of authority switches. Our experiments show that DIFANE's throughput increases linearly with the number of authority switches. With four authority switches, the throughput of DIFANE reaches over 3M flows/s. Administrators can determine the number of authority switches according to the size and the throughput requirements of their networks.

(4) DIFANE recovers quickly from authority switch failure. Figure 10 shows the effect of an authority switch failure. We construct a diamond topology with two authority switches, both connecting to the ingress and the egress switches. We set the OSPF hello interval to 1 s, and the dead interval to 3 s. After the authority switch fails, OSPF notifies the ingress switch. It takes less than 10 ms for the ingress switch to change to another authority switch after the dead interval, at which time the ingress switch sets the backup partition rule as the primary one, and thus connectivity is restored.

7.2 Evaluation of Partitioning and Caching

We now evaluate DIFANE's partitioning algorithm using the topologies and access-control rules from several sizable networks (as of Sept. 10, 2009) including a large-scale campus network [24] and three large backbone networks that are operated by a tier-1 ISP for its enterprise VPN, IPTV, and traditional IP services. The basic characteristics of these networks are shown in Table 1. In these networks, each access control rule has six fields: ingress interface, source IP/port, destination IP/port, and protocol. Access control rules are configured on the ingress switches/routers. It is highly likely that different sets of rules are configured at different switches and routers, hence one packet may be permitted in one ingress switch but denied at another. In addition,

4 ns per lookup [23], which is more than three orders of magnitude faster than in software.

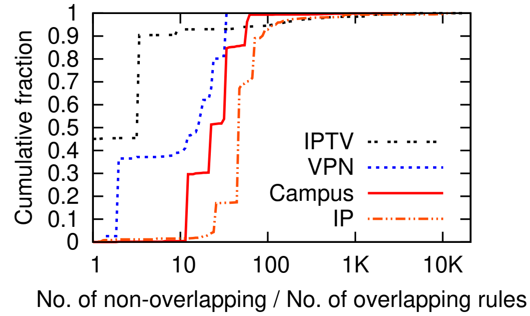


Figure 11: Comparison of overlapping and non-overlapping rules.

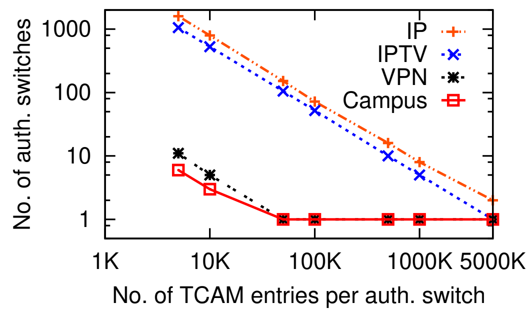


Figure 12: Evaluation on number of authority switches.

as Table 1 shows, there are a large number of access control rules configured in these networks. This is due to the large number of ingress routers that need to have access control rules configured and the potentially large number rules that need to be configured on even a single router. For example, ISPs often configure a set of rules on each ingress switch to protect their infrastructures and important servers from unauthorized customers. This would easily result in a large number of rules on an ingress switch if there are a large number of customers connected to the switch and/or these customers make use of a large amount of non-aggregatable address space.

In the rest of this section, we evaluate the effect of overlapping rules, the number of authority switches needed for different networks, the number of extra rules needed after the partitioning, the miss rate of caching wildcard rules, and the stretch experienced by packets that travel through an authority switch.

(1) Installing overlapping rules in an authority switch significantly reduces the memory requirement of switches: In our set of access control rules, the use of wildcard rules can result in overlapping rules. One straight forward way to handle these overlapping rules is to translate them into non-overlapping rules which represent the same semantics. However, this is not a good solution because Figure 11 shows that the resulting non-overlapping rules require one or two orders of magnitude more TCAM space than the original overlapping rules. This suggests that the use of overlapping wildcard rules can significantly reduce the number of TCAM entries.

(2) A small number of authority switches are needed for the large networks we evaluated. Figure 12 shows the number of authority switches needed under vary-

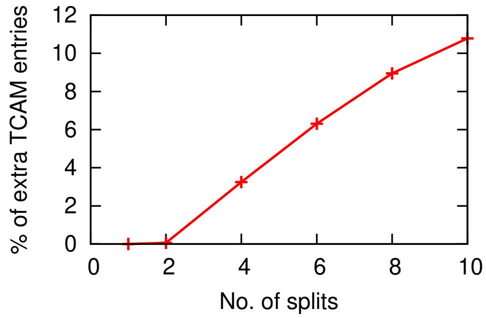


Figure 13: The overhead of rule partition.

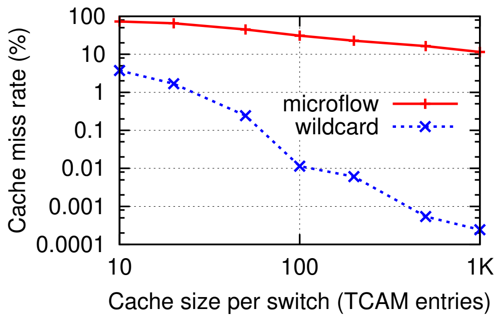


Figure 14: Cache miss rate.

ing TCAM capacities. The number of authority switches needed decreases almost linearly with the increase of the switch memory. For networks with relatively few rules, such as the campus and VPN networks, we would require 5–6 authority switches with 10K TCAM in each (assuming we need 16B to store the six fields and action for a TCAM entry, we need about 160KB of TCAM in each authority switch). To handle networks with many rules, such as the IP and IPTV networks, we would need approximately 100 authority switches with 100K TCAM entries (1.6MB TCAM) each.[12] The number of the authority switches is still relatively small compared to the network size (2K - 3K switches).

(3) Our partition algorithm is efficient in reducing the TCAM usage in switches. As shown in Figure 4, depending on the rules, partitioning wildcard rules can increase the total number of rules and the TCAM usage for representing the rules. With the 6-tuple access-control rules in the IP network, the total number of TCAM entries increases only by 0.01% if we distribute the rules over 100 authority switches. This is because most of the cuts are on the ingress dimension and, in the data set, most rules differ between ingresses. To evaluate how the partition algorithm handles highly overlapping rules, we use our algorithm to partition the 1.6K rules in one ingress router in the IP network. Figure 13 shows that we only increase the number of TCAM entries by 10% with 10 splits (100–200 TCAM entries per split).

(4) Our wildcard caching solution is efficient in reducing cache misses and cache memory size. We evaluate our cache algorithm with packet-level traces of 10M packets collected in December 2008 and the corresponding access-control lists (9K rules) in a router in the IP network. Figure 14 shows that if we only cache micro-flow rules, the miss rate is 10% with 1K cache entries. In contrast, with wildcard rule caching in DIFANE, the cache miss rate is only 0.1% with 100 cache entries: 99.9% of packets are forwarded directly to the destination, while only 0.1% of the packets take a slightly longer path through an authority switch. Those ingress switches which are not authority switches only need to have 1K TCAM entries for cache rules (Figure 14) and 10 - 1000 TCAM entries for partition rules (Figure 12).

[12] Today's commercial switches are commonly equipped with 2 MB TCAM.

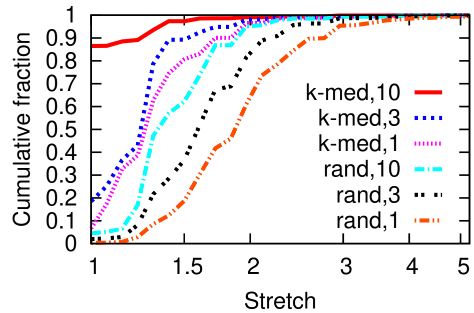


Figure 15: The stretch of campus network. (We place 1,3,10 authority switches for each set of the authority rules using k-median and random algorithms respectively.)

(5) The stretch caused by packet redirection is small. We evaluated the stretch of two authority switch placement schemes (random and k-median) discussed in Section 5. Figure 15 shows the distribution of stretch (delay normalized by that of the shortest path) among all source-destination pairs in the campus network. With only one authority switch for each set of authority rules, the average stretch is twice the delay of the shortest path length in the random scheme. Though some packets experience 10 times the delay of the shortest path, this usually happens to those pairs of nodes that are one or two hops away from each other; so the absolute delay of these paths is not large. If we store one set of rules at three random places, we can reduce stretch (the stretch is 1.8 on average). With 10 copies of rules, the stretch is reduced to 1.5. By placing the authority switches with the k-median scheme, we can further reduce the stretch (*e.g.*, with 10 copies of rules, the stretch is reduced to 1.07).

8. DIFANE DEPLOYMENT SCENARIOS

DIFANE proposes to use authority switches to always keep packets in the data plane. However, today's commercial OpenFlow switches have resource constraints in the control plane. In this section, we describe how DIFANE can be deployed in today's switches with resource constraints and in future switches as a clean-slate solution. We provide three types of design choices: implementing tunneling with packet encapsulation or VLAN tags; performing rule caching in the authority switches or in the controller; choosing normal switches or dedicated switches as authority switches.

Deployment with today's switches: Today's commercial OpenFlow switches have resource constraints in the control plane. For example, they do not have enough CPU resources to generate caching rules or have hardware to encapsulate packets quickly. Therefore we use VLAN tags to implement tunneling and move the wildcard rule caching to the DIFANE controller. The ingress switch tags the "miss" packet and sends it to the authority switch. The authority switch tags the packet with a different VLAN tag and sends the packet to the corresponding egress switch. The ingress switch also sends the packet header to the controller, and the controller installs the cache rules in the ingress switch.

Performing rule caching in the controller resolves the limitations of today's commercial OpenFlow switches for two reasons. (i) Authority switches do not need to run caching functions. (ii) The authority switches do not need to know the addresses of the ingress switch. We can use VLAN tagging instead of packet encapsulation, which can be implemented in hardware in today's switches.

This deployment scenario is similar to Ethane [3] in that it also has the overhead of the ingress switch sending packets to the controller and the controller installing caching rules. However, the difference is that DIFANE always keeps the packet in the fast path. Since we need one VLAN tag for each authority switch (10 - 100 authority switches) and each egress switch (at most a few thousand switches), we have enough VLAN tags to support tunneling in DIFANE in today's networks.

Clean slate deployment with future switches: Future switches can have more CPU resources and hardware-based packet encapsulation techniques. In this case, we can have a clean slate design. The ingress switch encapsulates the "miss" packet with its address as the source address. The authority switch decapsulates the packet, gets the address of the ingress switch and re-encapsulates the packet with the egress switch address as the packet's destination address. The authority switch also installs cache rules to the ingress switch based on the address it gets from the packet header. In this scenario, we can avoid the overhead and single point of failure of the controller. Future switches should also have high bandwidth channel between the data plane and the control plane to improve the caching performance.

Deployment of authority switches: There are two deployment scenarios for authority switches: (i) The authority switches are just normal switches in the network that can be taken over by other switches when they fail; (ii) The authority switches are dedicated switches that have larger TCAM to store more authority rules and serve essentially as a distributed data plane of the centralized controller. All the other switches just need a small TCAM to store a few partition rules and the cache rules. In the second scenario, even when DIFANE has only one authority switch that serves as the data plane of the controller, DIFANE and Ethane [3] are still fundamentally different in that DIFANE *pre-installs* authority rules in TCAM and thus always keeps the packet in the fast path.

9. CONCLUSION

We design and implement DIFANE, a distributed flow management architecture that distributes rules to authority switches and handles all data traffic in the fast path. DIFANE can handle wildcard rules efficiently and react quickly to network dynamics such as policy changes, topology changes and host mobility. DIFANE can be easily implemented with today's flow-based switches. Our evaluation of the DIFANE prototype, various networks, and large sets of wildcard rules show that DIFANE is scalable with networks with a large number of hosts, flows and rules.

10. ACKNOWLEDGMENT

We thank our shepherd Mark Handley, the anonymous reviewers, Nate Foster, Eric Keller, Wyatt Lloyd, Srini Seetharaman, and Rob Sherwood for their comments on earlier versions of this paper. We also thank Chad R. Meiners and Alex X. Liu for providing their code for the ACL compressor.

11. 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 Computer Communication Review*, Apr. 2008.

[2] "Anagran: Go with the flow." `http://anagran.com`.

[3] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. Gude, N. McKeown, and S. Shenker, "Rethinking enterprise network control," *IEEE/ACM Transactions on Networking*, 2009.

[4] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "NOX: Toward an operating system for networks," *ACM Computer Communication Review*, July 2008.

[5] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang, "A clean slate 4D approach to network control and management," *ACM Computer Communication Review*, 2005.

[6] H. Yan, D. A. Maltz, T. S. E. Ng, H. Gogineni, H. Zhang, and Z. Cai, "Tesseract: A 4D Network Control Plane," in *Proc. Networked Systems Design and Implementation*, Apr. 2007.

[7] A. Nayak, A. Reimers, N. Feamster, and R. Clark, "Resonance: Dynamic access control for enterprise networks," in *Proc. Workshop on Research in Enterprise Networks*, 2009.

[8] N. Handigol, S. Seetharaman, M. Flajslik, N. McKeown, and R. Johari, "Plug-n-Serve: Load-balancing Web traffic using OpenFlow," Aug. 2009. ACM SIGCOMM Demo.

[9] D. Erickson *et al.*, "A demonstration of virtual machine mobility in an OpenFlow network," Aug. 2008. ACM SIGCOMM Demo.

[10] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown, "ElasticTree: Saving energy in data center networks," in *Proc. Networked Systems Design and Implementation*, Apr. 2010.

[11] Y. Mundada, R. Sherwood, and N. Feamster, "An OpenFlow switch element for Click," in *Symposium on Click Modular Router*, 2009.

[12] A. Tootoocian and Y. Ganjali, "HyperFlow: A distributed control plane for OpenFlow," in *INM/WREN workshop*, 2010.

[13] C. Kim, M. Caesar, and J. Rexford, "Floodless in SEATTLE: A scalable Ethernet architecture for large enterprises," in *Proc. ACM SIGCOMM*, 2008.

[14] S. Ray, R. Guerin, and R. Sofia, "A distributed hash table based address resolution scheme for large-scale Ethernet networks," in *Proc. International Conference on Communications*, June 2007.

[15] H. Ballani, P. Francis, T. Cao, and J. Wang, "Making routers last longer with ViAggre," in *Proc. NSDI*, 2009.

[16] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with DIFANE," *Princeton University Computer Science Technical Report TR-877-10*, June 2010.

[17] Q. Dong, S. Banerjee, J. Wang, and D. Agrawal, "Wire speed packet classification without TCAMs: A few more registers (and a bit of logic) are enough," in *ACM SIGMETRICS*, 2007.

[18] J.-H. Lin and J. S. Vitter, "e-approximations with minimum packing constraint violation," in *ACM Symposium on Theory of Computing*, 1992.

[19] M. Handley, O. Hudson, and E. Kohler, "XORP: An open platform for network research," in *Proc. SIGCOMM Workshop on Hot Topics in Networking*, Oct. 2002.

[20] N. Brownlee, "Some observations of Internet stream lifetimes," in *Passive and Active Measurement*, 2005.

[21] "Stanford OpenFlow network real time monitor." `http://yuba.stanford.edu/~masayosi/ofgates/`.

[22] Personal communication with Stanford OpenFlow deployment group.

[23] "Netlogic microsystems." `www.netlogicmicro.com`.

[24] Y.-W. E. Sung, S. Rao, G. Xie, and D. Maltz, "Towards systematic design of enterprise networks," in *Proc. ACM CoNEXT*, 2008.