Magneto: Unified Fine-grained Path Control in Legacy and OpenFlow Hybrid Networks

Cheng Jin*,Cristian Lumezanu†,Qiang Xu†,Hesham Mekky*,Zhi-Li Zhang*,Guofei Jiang†
* *University of Minnesota,* †*NEC Laboratories America*
{cheng,hesham,zhzhang}@cs.umn.edu, {lume,qiangxu,gfj}@nec-labs.com

ABSTRACT

Software-defined networking (SDN) provides fine-grained network control and monitoring that simplifies network management. Unfortunately, upgrading existing enterprise networks, comprised of numerous "legacy" switches, to SDN is often cost-prohibitive. We argue that it is possible to achieve most of the benefits of a fully deployed SDN *at a fraction of the cost* by strategically replacing only few legacy switches with – or introducing a few – new SDN-capable switches in a legacy network, thus creating a *hybrid* network.

We present Magneto, a *unified* network controller that exerts SDN-like, fine-grained path control over both OpenFlow and legacy switches in hybrid networks. Magneto i) introduces *magnet* MAC addresses and dynamically updates IP-to-magnet MAC mappings at hosts via gratuitous ARP messages for visibility and routing control; and ii) uses the ability of SDN switches to send "custom" packets into the data plane to manipulate legacy switches into updating forwarding entries with magnet MAC addresses for enhanced routing flexibility. Our evaluation on a lab testbed and through extensive simulations on large enterprise network topologies show that Magneto is able to achieve full control over routing when only 20% of network switches are programmable, with negligible computation and latency overhead.

CCS Concepts

•**Networks** → **Network manageability; Programmable networks;** *Network architectures; Network control algorithms;*

Keywords

Magneto; Hybrid SDN; MAC Learning and Forwarding

SOSR '17, April 03-04, 2017, Santa Clara, CA, USA

DOI: http://dx.doi.org/10.1145/3050220.3050229

1. INTRODUCTION

With a *(logically) centralized* control plane [12, 19] and a *programmatic match-action* data plane abstraction [26, 11, 9], software-defined networking (SDN) [25, 30, 28] enables flexible, fine-grained network control and monitoring, and offers the potential to transform network management: from today's largely *manual* process to an *automated* process governed by (high-level) *network policies*. Studies show that SDN can reduce the cost of operating a network by half [10]. Thanks to these benefits, earliest adoption of SDN occurs in data centers, where size renders manual network management difficult. SDN has also been applied to wide-area networks (WANs), *e.g.*, those connecting multiple data centers [16, 14], to more effectively manage expensive WAN bandwidth. Internet Service Providers (ISPs) or carrier networks have also started considering the adoption of SDN [2].

However, the majority of networks on the Internet are *enterprise* networks, where deployment of SDN faces major challenges. Unlike data center networks with well-structured topologies, enterprise networks often evolve in a not well-planned, "organic" fashion as the need for network connectivity and bandwidth grows. As a result, enterprise network topologies can be arbitrary—often with many quasi-tree like structures as access networks connected by a "semi-mesh" core network. Further, most enterprise networks [1, 18, 31] comprise layer 2 (L2) Ethernet switches supporting VLANs and use layer 3 (L3) IP routers as gateways to route between VLANs or for external Internet connectivity.

Converting enterprise networks to SDN is difficult. First, budget constraints make it cost-prohibitive [15] to perform a "wholesale" upgrade from a Ethernet-based "legacy" network to a programmable[1] SDN network. In addition, enterprises often run *mission-critical* applications that rely on existing legacy hardware devices and/or software components. Recent work has proposed partial SDN deployments where only a fraction of the switches are upgraded to SDN [6, 20, 23, 22, 15]. Operators control the SDN-enabled devices but cannot affect the paths through the legacy network. Much of the routing must be coarsely engineered using VLANs or tunnels [20, 22] or left to the latitude of L2 protocols such as

[1]We interchangeably use the terms *programmable*, *OpenFlow(-enabled)*, or *SDN(-capable)* to refer to devices whose forwarding tables can be configured remotely from a centralized controller.

Spanning Tree Protocol (STP) or ECMP. This limits network control as some policies cannot be installed. Most enterprise network operators have little experience in managing and operating SDN networks. They need to gradually gain experience and build confidence in running SDN networks.

In this paper, we present a novel framework for *incremental and graceful transition* of legacy networks comprised primarily of L2 Ethernet switches to SDN-capable networks. Rather than performing an expensive and disruptive *wholesale upgrade* or converting parts of the network into *"SDN islands"*, we argue and advocate that it is not only possible but in fact *advantageous* to migrate a network of legacy switches to a *hybrid* network of *mixed* legacy switches and SDN-capable switches while at the same time reaping as much benefit as a *fully deployed* SDN network. The key idea behind our proposed framework, which we call Magneto, is that by replacing one or a few *strategically placed* L2 legacy switches with SDN-capable switches, or by adding SDN switches, we can influence the forwarding behavior of legacy switches and end hosts (*i.e.*, "magnetize" them). This allows us to *gain visibility and exert control* over legacy devices *without the need to make any modifications to existing legacy hardware devices* or *software components* (*e.g.*, configuring VLANs or virtualization).

Magneto employs two key mechanisms to exert SDN-like control over legacy L2 switches: *telekinesis* where we leverage OpenFlow switches to inject *seed* packets to manipulate legacy switches' forwarding tables; and *magnet addresses* where we use gratuitous ARP messages to populate the ARP tables at end hosts with "fictitious" or "illusory" MAC addresses for the purpose of gaining network visibility and controlling routing and forwarding behaviors of end hosts and legacy switches. We describe the baseline *telekinesis* mechanism *without the use of magnet MAC addresses* in Section 3. This is the path control mechanism used in our prior work [17] for hybrid networks. This baseline mechanism injects seed packets with the *native* MAC address of a destination host of the path to install. This mechanism suffers from two shortcomings: i) it can only exert limited, coarse-grained (*i.e.*, per-destination) path control and ii) the path installed may be unstable. In Section 4, we introduce magnet addresses and outline how they can be used to exert *fine-grained* (*i.e.*, per source-destination pair) path control in hybrid networks and formulate the (path) *controllability* condition. We present the detailed Magneto fine-grained path control components in Section 5.

We evaluate Magneto using simulations on larger enterprise network topology and on a real-world testbed (Section 6). We demonstrate that Magneto is capable of enforcing fine-grained policies in hybrid networks, *e.g.*, routing along multiple disjoint paths to the same destination for congestion control or load balancing [33]. Magneto can install diverse paths with little control and data plane overhead, and exert full control over routing even when only 20% of the switches are SDN-capable.

In a nutshell, Magneto provides a *unified* network controller to exert SDN-like control over both programmable and legacy switches in hybrid networks. It enables network operators to transition legacy networks to SDN networks *in stages* by gradually replacing more and more legacy switches with SDN-capable switches *as needed and as budgets allow*. Further, it allows network operators to gracefully experiment with SDN networks to gain experience and build confidence while eliminating or minimizing service disruption. Our work demonstrates that it is possible to enjoy much of the benefits of a wholly deployed SDN network but *at a fraction of the cost* by strategically replacing only a few (e.g., 20%) legacy switches with SDN-capable switches.

2. BACKGROUND AND MOTIVATION

We discuss previous work on partial SDN deployment and identify their benefits and shortcomings. We then introduce our solution for unified network management for hybrid legacy and OpenFlow networks.

2.1 Hybrid Networks

There are several approaches to transition a legacy network to an SDN-capable network [23, 22, 13, 24, 6, 15]. First, vendors can install additional software modules on legacy switches to make them programmable. ClosedFlow [13] configures legacy switch features to mimic and support the OpenFlow API and make the switch appear OpenFlow-enabled to an SDN controller. This approach however requires modification and installation of additional software modules to process and support OpenFlow APIs; it is vendor-specific and highly depends on the features supported by legacy switches.

Another approach is through *access edge* control via virtualization. For example, VMWare's NSX [20] forgoes physical programmable switches altogether and implements SDN at the edge of the network as part of hypervisors. This approach requires upgrading and installing new networking software on all end devices in a network. This can be a challenging task in most enterprise networks, and may not be feasible in some enterprise networks where many devices are BYOD (bring your own device).

Third, operators can replace all legacy switches in a subnet with SDN switches to create *SDN islands* [4, 6, 23]. The SDN and legacy zones are independent and managed separately. The benefits of SDN are limited solely to SDN islands and cannot be extended to legacy networks. In addition, network operators must run multiple "control & management" planes, one for legacy networks, and one for each SDN island. This can add additional burden on network operators and further complicate their management tasks.

First proposed by Levin *et al.* [22], a fourth approach is to simply replace a few (strategically placed) legacy switches with, or introduce a few, new SDN switches *in piecemeal* fashion. We refer to such a network of mixed legacy and SDN-capable switches as a *hybrid* network. Hybrid networks offer the potential to benefit from the flexibility and visibility offered by SDN without the considerable initial investment of fully transitioning to SDN. By replacing legacy switches with SDN-capable switches (*e.g.*, OpenFlow switches), we add control entry points into the network to implement more complex policies and exploit path diversity in the un-

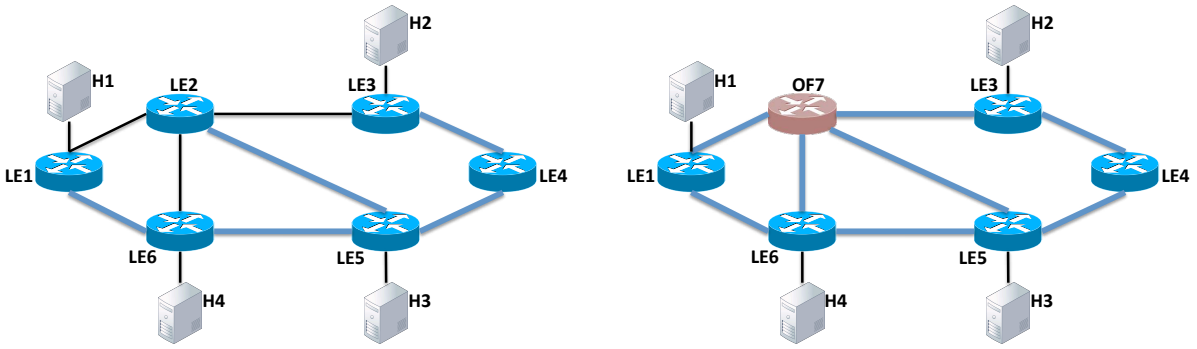


Figure 1: Path diversity in legacy (left) and hybrid (right) networks: In legacy networks, the spanning tree created by STP (solid blue lines) constrains the end-to-end paths. In hybrid networks, all links that are part of the spanning tree or adjacent to an OpenFlow switch can be used.

derlying physical network topology by going beyond the default spanning tree used by legacy switches.

Consider the example topology on the left in Figure 1. The paths between every pair of hosts in the legacy network are constrained by the L2 spanning tree constructed by STP (solid blue lines). This will create congestion on the spanning tree links, while the other links are not utilized. If the switch $LE2$ is upgraded to an OpenFlow switch $OF7$ that creates a hybrid network (Figure 1(right)), alternate paths are exposed through the OpenFlow switch. This allows us to install more diverse policies (*e.g.*, balance traffic across multiple links to eliminate congestion). Further, the addition of OpenFlow switches provides fine-grained flow-level visibility (*e.g.*, between two hosts).

Most existing approaches for managing hybrid networks incur significant management complexity, as they control legacy and SDN switches via different mechanisms. For example, Panopticon [22] resorts to VLANs (whereas NVP [20] employs tunnels) to set up paths through the legacy network, requiring additional (manual) configurations. Though effective at enforcing access control, using VLANs to force traffic through OpenFlow switch(es) restricts the flexibility of routing. Further, they do not provide sufficient agility (as VLANs cannot be reconfigured rapidly [23]) nor diversity (as tunnels cannot select the underlying physical path). In summary, while offering the potentials for increased flexibility and visibility at reduced cost, hybrid networks still face complex management issues. Ideally, we would like an unified framework to control both legacy and SDN switches that offers *flexible forwarding control* with *simple network management* and at *low operating cost*.

2.2 Our Solution

We propose Magneto, a network controller framework to *incrementally* and *gracefully* transition a legacy network to an *SDN-capable* network by *strategically* placing – or replacing a few legacy switches with – OpenFlow switches. We use SDN-capable switches to influence and exert control on the forwarding behavior of legacy switches and end hosts and to obtain similar network *visibility* and routing *control* as in a fully deployed SDN. This is achieved via two mechanisms: *telekinesis* where we leverage OpenFlow switches to inject *seed* packets to effect changes in legacy switches' forwarding tables; and *magnet addresses* where we employ gratuitous ARP messages to populate the ARP cache tables at end hosts with "fictitious" or "illusory" MAC addresses for the purpose of gaining network visibility and controlling forwarding behaviors of end hosts and legacy switches.

Magneto unifies hybrid network management using a single OpenFlow-based network controller. Unlike previous approaches, Magneto does not need switch-vendor support or additional modules on legacy switches. Although it does not obviate the use of VLANs or tunnels, Magneto provides path control and flexibility without the overhead of configuring VLANs or setting up tunnels.

Conceptually similar to Fibbing [32], Magneto indirectly affects network routing by injecting fake and harmless information into the network. However, due to the self-learning switch algorithm, STP and VLANs used by L2 switches, they pose unique and different challenges from L3 IP distributed routing, and therefore call for different mechanisms. Magneto operates at the data link layer by affecting the forwarding behavior of legacy L2 switches. In contrast, Fibbing [32] aims at introducing a centralized control over distributed L3 IP routing by injecting carefully crafted "fake" routing messages via OSPF. Fibbing's goal is to enhance the flexibility, diversity and reliability of L3 routing, *not to transition legacy enterprise networks to SDN-capable networks*, as enterprise networks comprise primarily legacy switches.

3. BASELINE MECHANISM

By replacing a few strategically placed legacy switches with SDN-capable switches, we are able to, not only directly control SDN switches, but also influence the forwarding behavior of legacy switches. This allows us to enhance routing flexibility and increase network utilization through path diversity. We first describe the baseline *telekinesis* mechanism to control paths through legacy devices, introduced in our prior work [17]. We then discuss the shortcomings of this baseline (coarse-grained) path control mechanism. In Section 4 and 5 we will describe an enhanced design for fine-grained path control which circumvents these shortcomings.

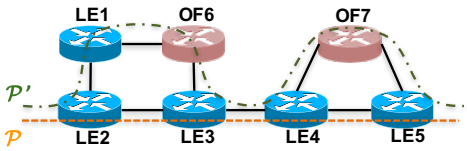


Figure 2: Example of path update: $\mathcal{P}$ is the current path, $\mathcal{P}'$ is the new path; $LE1$, $LE2$, $LE3$, $LE4$, $LE5$ are legacy switches, $OF6$ and $OF7$ are OpenFlow switches; $(LE2, LE1, OF6, LE3)$ and $(LE4, OF7, LE5)$ are the subpaths that need to be updated.

3.1 Basic Idea and Key Mechanisms

Assumptions. In a hybrid network with both *legacy* switches and *programmable* switches such as OpenFlow switches, we can only control the programmable switches via a central SDN controller, but cannot directly update the legacy switch forwarding entries. We assume that each legacy switch runs MAC learning and that the legacy network is configured, either manually or automatically, to avoid forwarding loops (*e.g.*, with STP). We call the collection of legacy links that results after this configuration the *network underlay*. The underlay is always a tree or a collection of trees. End hosts maintain ARP tables to map MAC addresses to IP addresses.

Goal. Given a path (*i.e.*, a sequence of switches) $\mathcal{P}$ between two hosts A and B in a hybrid network and a candidate new path $\mathcal{P}'$, reconfigure the network so that all traffic between A and B traverses $\mathcal{P}'$ [2] or decide that the new path is infeasible. This may require updating all switches along the new path. In Figure 2, $(LE2, LE3, LE4, LE5)$ is the old path $\mathcal{P}$ and $(LE2, LE1, OF6, LE3, LE4, OF7, LE5)$ is the new path $\mathcal{P}'$.

Seed Packets. The key idea behind *telekinesis* is to use OpenFlow switches to send special ("custom-made") packets–referred to as *seed* packets–to the legacy switches on the new path. This relies on the ability of an SDN controller to send PacketOut control messages to OpenFlow switches and instruct them to send custom-made packets into the network. The seed packets take advantage of MAC learning to *manipulate* legacy switches into updating a single forwarding entry in their routing tables.

Under the baseline telekinesis mechanism, seed packets must satisfy two requirements. First, their source MAC address must be the same as the destination MAC of the path we want to install in the legacy switch. This ensures that only the forwarding entry corresponding to this MAC address is updated. Second, they must arrive at a legacy switch on a link that is part of the path we want to install. This ensures that the affected entry is correctly updated with the next-hop information. For example, if we want to modify the action of a forwarding entry for MAC m from *"send to port p1"* to *"send to port p2"*, we create a packet whose source address is m and make sure it arrives at the switch on port $p2$. The MAC learning algorithm sees the packet arriving on $p2$ and assumes its source address m is reachable on $p2$, therefore updating the corresponding forwarding entry.

[2]Throughout the paper, we refer to this process as installing, configuring, enforcing, or updating $\mathcal{P}'$.

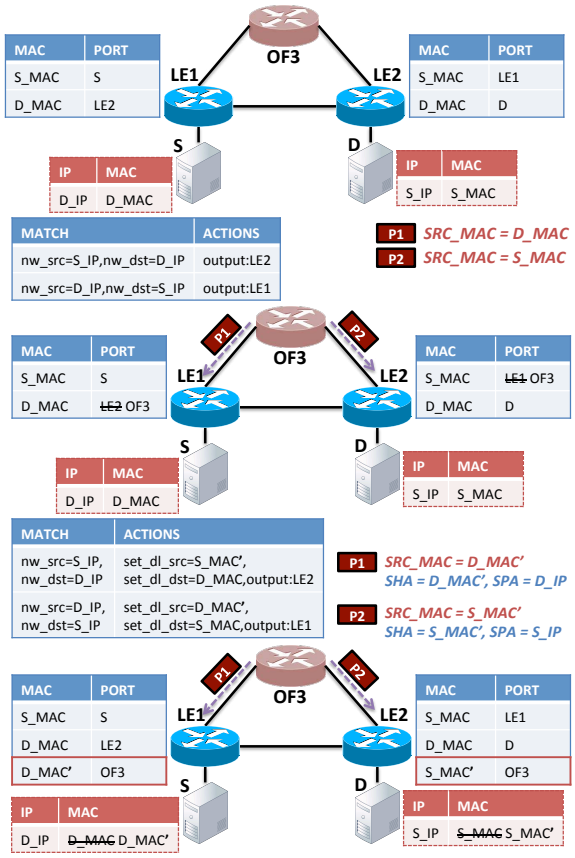


Figure 3: Path update between two hosts, S and D, in a hybrid network consisting of two legacy switches ($LE1$ and $LE2$) and one OpenFlow switch ($OF3$). Switch forwarding tables are in blue, host ARP caches are in red. (**top: original network state**) Traffic between S and D flows through path $(LE1, LE2)$; (**center: basic path update**) $OF3$ injects seed packets to $LE1$ and $LE2$, triggering updates in their forwarding tables and thereby changing the path between S and D to $(LE1, OF3, LE2)$; (**bottom: enhanced path update**) $OF3$ injects seed packets with magnet MACs to both legacy switches and end hosts changing the path to $(LE1, OF3, LE2)$.

Path Update. Updating the path $\mathcal{P}$ between two hosts to a new path $\mathcal{P}'$ requires updating all switches on $\mathcal{P}'$ if the two paths are disjoint. When old and new paths overlap, we need to update only the switches where the paths diverge. We define an *update subpath* as the sequence of adjacent switches that must be updated during a path change. For example, in Figure 2, we must update OpenFlow switches $OF6$ and $OF7$ and legacy switches $LE1$, $LE2$, $LE3$, $LE4$, and $LE5$. The update subpaths are $(LE2, LE1, OF6, LE3)$ and $(LE4, OF7, LE5)$.

The above example illustrates that by simply replacing one or a few legacy switches with OpenFlow switches, we can in fact gain more path control by leveraging these pro-

grammable switches to effect changes in legacy switches via telekinesis. The seed packets that used to remotely manipulate a legacy switch's forwarding table must arrive at the switch on a link that is part of the path telekinesis wants to install. This leads to the following *control* condition of the *baseline* telekinesis mechanism.

(Control condition of baseline telekinesis) *A path is feasible if and only if (a) every link on it is part of the L2 underlay or adjacent to an OpenFlow switch, and (b) every update subpath contains at least one OpenFlow switch.*

The first part of the condition ensures that a seed packet reaches the right interface on a legacy switch so it can trigger a forwarding entry update. The second part of the condition ensures that there is at least one OpenFlow switch to send a seed packet to every legacy switch on the update subpath. We will show in Section 4 how these conditions can be further relaxed via Magneto's enhanced *fine-grained* path control mechanisms.

3.2 Shortcomings of Baseline Telekinesis

This baseline telekinesis mechanism suffers from two shortcomings: i) it can only exert limited, *coarser-grained* (i.e., per-destination) path control and ii) the path installed may be *unstable*. We discuss them in more details below.

Coarse-grained paths. Legacy network L2 routing is destination-based: a destination MAC is associated with a single interface (and implicitly, path) on each switch. Legacy network operators create path diversity at increased management cost using VLANs or ECMP. OpenFlow networks can install more fine-grained paths as they can match traffic based on both source and destination MACs. Our basic scheme inherits the limitations of legacy networks: the update of a path triggers updates on all paths to the same destination. In the example on the right of Figure 1, both $H1$ and $H4$ send traffic to $H3$. The legacy switch $LE6$ will forward all the packets destined to $H3$ towards $OF7$, including the packets from $H4$ to $H3$, if we change the path between $H1$ and $H3$ to $(LE1, LE6, OF7, LE5)$.

Unstable paths. MAC learning reacts to all incoming packets, regardless of whether they are seed packets or not. A forwarding entry for a MAC address m may change every time the switch relays a packet from m. This can make even the simplest path update unstable. To better understand this limitation, we consider a common scenario that can lead to unstable paths: traffic between two hosts flows in both directions, such as when the hosts use TCP to communicate. Consider the example in Figure 2. If the update from $\mathcal{P}$ to $\mathcal{P}'$ on the direct path is not fast enough, packets on the reverse path (which is still $\mathcal{P}$) can invalidate the forwarding entry updates and revert them to the original states corresponding to $\mathcal{P}$. A simple solution to make paths stable when reverse traffic is present is to continually inject seed packets until forwarding entries reach a stable state. The frequency of seed packets depends on the rate of data packets. As long as seed packets arrive faster than data packets, they can override any change made by reverse path packets and the original direct path will eventually be updated.

We evaluate this scheme in a small real-world testbed, shown in Figure 3. Initially, the default path between servers traverses only the legacy switches. We continually send TCP traffic between the servers. At the same time, we update the path to traverse the OpenFlow switch as well. An update is successful if the path becomes stable in less than five seconds from the first seed packet. We compute the percentage of successful updates as we vary the data rate over one hundred runs. Table 1 shows the results. The basic update mechanism success rate decreases as the data plane rate increases and falls to 0 for rates of at least 100 Mbps. In summary, flooding legacy switches with seed packets does not guarantee a successful path update. In addition, it may generate significant network overhead. In the next section we present an enhanced path control mechanism that installs stable paths with almost zero network overhead.

Data Rate (Mbps)	Update Success
0.1	94%
1	80%
10	59%
100	0%

Table 1: Successful path updates using the basic *telekinesis* mechanism, when we vary the data plane rate. A path is successfully updated if it becomes stable in less than five seconds from the time when we send the first seed packet.

4. MAGNET MAC ADDRESSES AND FINE-GRAINED PATH CONTROL

We now enhance the baseline telekinesis by integrating it with magnet addresses to achieve *fine-grained* (*i.e.*, per source-destination pair) path control. In the following we first introduce magnet (MAC) addresses and briefly discuss how they can be used to gain visibility and enforce end-to-end access control in hybrid networks. We then outline the key ideas behind Magneto's fine-grained path control. The detailed path control processes is described in Section 5.

4.1 Magnet MAC Addresses & Visibility

Magneto introduces the key notion of *magnet (MAC) addresses* to influence and manipulate both end hosts' forwarding behaviors as well as those of legacy switches. A *magnet address* is a fictitious MAC address that does not correspond to any real host on the network, but is created by our Magneto controller for the purpose of gaining network visibility and controlling routing & forwarding behaviors of end hosts and legacy switches. We "magnetize" a hybrid network by controlling the (magnet) MAC address mappings at end hosts via unicast gratuitous ARP messages generated by the Magneto controller (via OpenFlow switches).

To gain visibility and enforce end-to-end access control, we can pre-populate hosts' ARP cache via gratuitous ARP messages to eliminate the broadcast ARP query process. For some "assets" servers that we want to monitor and control the access all the time, we can pre-populate the IP-MAC address mappings in all hosts on the same L2 LAN segment with the "assets" servers' magnet addresses. Since the ARP

packet size is small (though it may vary but is typically less than 80 Bytes), the overhead of doing this pre-population is negligible. Further, the controller can adjust the mappings dynamically via new gratuitous ARP messages to alter forwarding paths of host.

4.2 Telekinesis with Magnet Addresses

We now present the Magneto's fine-grained path control mechanism which seamlessly integrate telekinesis with magnet addresses to achieve fine-grained path control.

When sending seed packets, we set the source MAC address as a magnet MAC address associated with the path destination, rather than the real (native) MAC address of the destination host. The seed packet triggers the installation of a forwarding entry for the magnet MAC address. We also require that the seed packets are ARP packets and can reach the source host of the path. Thus, the source learns to associate the destination with its new magnet MAC address. Magneto enhances routing by enabling multiple paths between source-destination pairs, which enables re-routing a portion of the traffic on a congested path to a new path instead of the default spanning tree path. In the baseline mechanism, if one source changes its path to a destination, it will affect the paths from all other sources too. Magneto uses different magnet MAC addresses for other source hosts to update legacy switches, hence packets destined to the same destination from different sources can now traverse different paths. The last OpenFlow switch on the path rewrites the magnet MAC address to the native MAC address based on the destination IP address.

Figure 3 illustrates the enhanced path control at the granularity of per-source-destination pair. To install a new path between $(LE1, OF3, LE2)$ between S and D, we generate a new magnet MAC address D_MAC' associated with D and send a seed (*unicast*) ARP packet from $OF3$ to S with the new magnet MAC as the source MAC address in the Ethernet packet. The sender hardware address (SHA) field of the ARP message is also set to D's magnet address, *i.e.*, $SHA = D\_MAC'$. This packet triggers the addition of a new forwarding entry at switch $LE1$ and the update of the ARP table on S to add one entry for D's magnet MAC address and corresponding incoming port. The forwarding table of switch $LE2$ is updated in a similar manner.

By integrating telekinesis with magnet MAC addresses, we are able to exert fine-grained (per source-destination pair) path control, thereby significantly increasing path diversity that can be exploited for routing and traffic engineering. As a destination host can be associated with multiple magnet MAC addresses (for different source hosts), we can install multiple paths to the same destination host. Compared to the baseline telekinesis mechanism, this leads to the following *relaxed* path control conditions:

(Control condition of telekinesis with magnet MAC addresses) *A path is feasible if and only if every link on it is part of the L2 underlay or adjacent to an OpenFlow switch.*

This condition implies that there is at least one OpenFlow switch in the network. The use of magnet MAC addresses also isolates the old path (*e.g.*, the default spanning tree path)

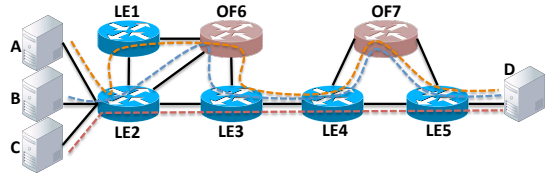


Figure 4: Three source hosts A, B, and C send traffic to the same destination host D via different paths.

and the new path between two hosts. This eliminates the unstable path problem associated with the baseline telekinesis mechanism. We note that packets traversing along the reverse direction of an old path (*e.g.*, the default spanning tree path $(LE1, LE2)$ in the bottom example in Figure 3) cannot rewrite the forwarding entries for a new path in the legacy switches, since these packets must contain either the native MAC address or a different magnet MAC address. In a sense, magnet MAC addresses achieve a form of network versioning, similar in spirit to the consistent network update mechanisms for SDNs proposed in [29]. As the native MAC addresses of hosts can always be learned by broadcasting on the default spanning tree, if we want to revert a new "off-spanning-tree" path back to the default spanning tree path, Magneto can generate a seed packet with the native MAC address in gratuitous ARP message (while the magnet MAC address is used as the source MAC address in Ethernet packet header) and send it via an OpenFlow switch on the off-spanning-tree path. Using the bottom example in Figure 3, to revert the path back from $(LE1, OF3, LE2)$ (the off-spanning-tree path) to the default spanning tree path $(LE1, LE2)$, Magneto crafts a seed packet and sends it towards S with $SRC\_MAC = D\_MAC'$ and $SHA = D\_MAC$ (the similar process is applied for D).

5. MAGNETO PATH CONTROL COMPONENTS

In this section we describe the detailed fine-grained path control components employed by Magneto: *path verification*, *path update*, and *magnet routing*. Given a network configuration (*i.e.*, forwarding tables on all switches and the network underlay) and a new path $\mathcal{P}'$ to install between two hosts attached to the network, Magneto first checks whether the path is feasible. It then installs the path by sending seed packets with magnet MACs to every legacy switch on the path. To route each packet to the destination along the new path, Magneto must rewrite packet headers and eventually replace the magnet MACs with the real MACs.

5.1 Path Verification and Path Update

Given a path $\mathcal{P}'$ and the current network configuration, path verification determines whether $\mathcal{P}'$ is feasible in the network. For each link in the new path that is not present in the old path, Magneto verifies whether it is part of the L2 spanning tree or adjacent to an OpenFlow switch. This ensures that seed packets can install the path. To maintain an updated view of the spanning tree, Magneto periodically

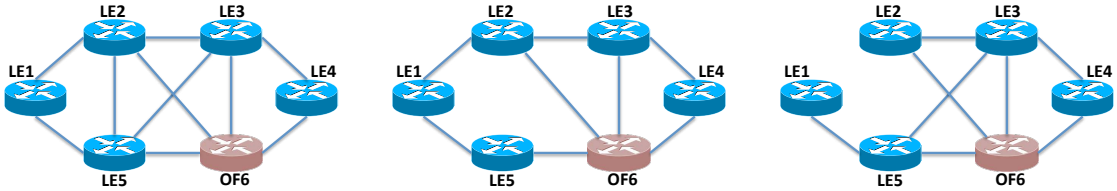


Figure 5: The network topology and underlay affect the diversity of paths enabled by Magneto. Given a topology with five legacy switches and one OpenFlow switch (a), the performance of Magneto varies across two possible sets of usable links (b,c) (spanning tree links plus OpenFlow-adjacent links).

queries port information from each legacy switch. In addition, Magneto checks that at least one switch on the new path is OpenFlow-enabled, unless the new path is only in the L2 spanning tree. This ensures that we can send seed packets.

To install a new path, Magneto generates seed packets and sends them to both legacy switches and hosts. We describe both actions next.

Generating seed packets. The role of seed packets is to trigger updates to legacy switch forwarding tables and host ARP caches. Each seed packet is an ARP packet whose source MAC address in the Ethernet header is a magnet MAC address associated with the destination of the path. In addition, we set the ARP header to map the magnet MAC to the destination's real IP address.

How do we generate magnet MAC addresses? The simplest way is to generate one magnet MAC address for each path through the network. However, this would create a large number of magnet MAC addresses and may inflate unnecessarily the size of switch forwarding tables. We observe that all feasible paths are constructed from the same set of usable links (*i.e.*, links that are part of the underlay or adjacent to OpenFlow switches). Further, adjacent legacy switches are controlled by the same seed packet.

We define a *magnet subpath* as a sequence of adjacent legacy switches on the path to install. A magnet subpath is part of the L2 underlay and lies between two OpenFlow switches or between a host and an OpenFlow switch. All legacy switches in the same magnet subpath can be updated by the same seed packet from the same OpenFlow switch. Magnet subpaths are different from update subpaths, defined in Section 3 as sequences of adjacent switches, not necessarily legacy, that must be updated when installing a new path.

We generate one magnet MAC for each unique magnet subpath. We associate the first 42 bits of the MAC address with the OpenFlow switch used to update the magnet subpath (*e.g.*, we hash the OpenFlow switch DPID) and the last six bits with the interface[3] of the same switch used to send the seed packet that updates the path. This assignment ensures that *the maximum number of magnet MAC addresses is at most the sum of the number of interfaces across all OpenFlow switches in the network*. In our experiments, we generated at most 5,000 different magnet MAC addresses in a network with 100 OpenFlow switches.

Consider the example in Figure 4. The paths between A and D and between B and D have a common magnet subpath $(LE3, LE4)$. Magneto generates one, rather than two, magnet MAC address for this subpath. The OpenFlow switch $OF7$ sends a seed packet with the magnet MAC to both switches on the subpath.

Sending seed packets. To support forwarding entry updates on legacy switches via carefully crafted packets, we introduce a new primitive, called LegacyFlowMod. We use LegacyFlowMod to generate seed packets and send them to the switches we want to update. LegacyFlowMod relies on OpenFlow's PacketOut functionality, which allows us to use any OpenFlow switch we control to send a packet on the data plane. Given a path to update, LegacyFlowMod calls PacketOut for every legacy switch to update. We must be careful to call PacketOut with respect to an OpenFlow switch that can reach the intended legacy switch using a link that is on the new path we want to enforce.

Each seed packet must reach all legacy switches in the magnet subpath that precedes the OpenFlow switch sending the packet. In addition, the seed packet sent by the first OpenFlow switch on the path must reach the source host, to update its ARP table. In Figure 4, if C wants to reach D through the same path as B's, Magneto uses $OF6$ to send a seed packet to C to updates its ARP cache with the same magnet MAC address that B uses to reach D. In contrast, if A or B wants to use the default path in the spanning tree, Magneto uses $OF6$ to send a seed packet to A or B to update its ARP cache with the real MAC address of D.

5.2 Magnet Routing

Associating magnet MACs with subpaths rather than paths helps reduce the size of forwarding tables. However, because each magnet subpath of a path is installed using different magnet MACs, OpenFlow switches between subpaths must rewrite packet headers.

Given a path to be updated, the source hosts sends packets towards the magnet MAC associated to the first magnet subpath on the path (assuming a seed packet already updated the source's ARP cache). Legacy switches simply forward packets to the next hop according to their forwarding tables.

[3]We assume at most 48 interfaces on a switch; for more interfaces, we can change the bit distribution between the OpenFlow ID and the interface ID.

We insert rules in the OpenFlow switches that rewrite each packet's source and destination MAC fields according to the next magnet subpath along the path to be installed. The final OpenFlow switch rewrites the destination MAC field with the destination's real MAC address, as the last magnet subpath does not have its own magnet MAC.

In the example in Figure 4, to set up the both the direct and reverse paths between B and D, $OF6$ crafts a seed packet with source MAC address as OF6:2, source hardware address as OF6:2, source protocol address as D's IP, and sends it to B through $LE2$. Also, $OF6$ crafts another seed packet with source MAC address as OF6:3, source hardware address as OF6:3, source protocol address as B's IP, and send it to D through $LE3$. Similarly, $OF7$ crafts one seed packet with magnet MAC address OF7:1 to B and another seed packet with magnet MAC address OF7:2 to D respectively. A packet sent from B to D starts with source MAC address as B's real MAC address and destination MAC address as OF6:2. When it reaches $OF6$, $OF6$ rewrites its source MAC address to be OF6:3 and destination MAC address to be OF7:1. Later, $OF7$ rewrites the packet header again, whose source MAC address to be OF7:2 and destination MAC address to be D's real MAC address.

5.3 Interoperability, Reversibility & Incremental Deployment

We discuss various aspects of deploying Magneto in a real-world enterprise network environment.

Interaction with STP or one of its variants, such as Rapid STP, does not require additional configuration on legacy switches. Legacy switches exchange Bridge Protocol Data Units (BPDU) messages to create a loop-free L2 underlay. Magneto adds a rule in every OpenFlow switch to forward BPDU messages to the controller, so it can passively listen to all BPDU messages and *not forward* them further. This behavior guarantees any interface adjacent to an OpenFlow link is not blocked, while a loop-free underlay is still formed among legacy switches.

BUM traffic represents L2 broadcast, unknown unicast, and multicast traffic. The usage of magnet MAC addresses allows Magneto to coexist with broadcast/multicast traffic assuming that such traffic cannot update the hosts' ARP tables such as broadcast ARP messages (which are under the control of Magneto). Unknown unicast traffic with real destination MAC addresses can reach destinations through the default spanning tree path. On the other hand, unknown unicast traffic with magnet destination MAC addresses will be routed through OpenFlow switch(es) and their Ethernet packet headers will be rewritten.

ARP poisoning mitigation techniques protect the network against gratuitous or malicious ARP messages by building bindings of valid IP-MAC tuples. When such techniques are deployed, Magneto must be integrated with the network DHCP server and every magnet MAC address generated by Magneto must be bound to a magnet IP address assigned by the DHCP server. Thus, all ARP messages containing magnet MAC addresses correspond to valid IP-MAC bindings and are not blocked by the ARP poisoning mitigation.

Inter-VLAN routing and L3 routers are used in enterprise networks to isolate traffic and restrict broadcast domains. Magneto works with existing L3 routing by either: (1) utilizing OpenFlow switches on the path between source-destination pairs to rewrite VLAN tags, and therefore it can reduce the traffic latency and the load on the L3 router, or (2) it breaks the path into segments with one segment for each broadcast domain if the policy requires that the traffic go through the L3 router. Then, each segment can be assigned different magnet MAC addresses. Finally, Magneto enables diverse L2 paths, which can be combined with Fibbing [32] (which enables L3 diverse paths) to provide opportunities for joint L2/L3 routing optimization and traffic engineering.

Path diversity depends on the network underlay (*i.e.*, spanning tree), and the location of the OpenFlow switch. Consider the topology in Figure 5(a) where the OpenFlow switch is adjacent to four legacy switches. The controllable links change based on the network underlay. For instance, Figure 5(b) shows an examples of all controllable links (both spanning tree and OpenFlow links) when the spanning tree is rooted at $LE1$. Figure 5(c) shows another examples when the spanning tree is rooted at $LE4$ with less controllable paths. Consequently, the placement of OpenFlow switches during incremental deployment and configuring the STP is critical in enabling many paths in the network that can be controlled by Magneto, which we discuss later in Section 6.1.

Network failures may affect the functionality of Magneto. Magneto detects data plane failures by monitoring the Topology Change Notification (TCN) bit and root bridge ID fields in STP BPDU messages (for legacy links) or port status messages (for OpenFlow links). Once it identifies a failure, Magneto excludes the failed link from the known topology and recomputes and updates the flow paths affected by the failure. Because the STP failure recovery may change the original spanning tree containing the failed link, the newly updated paths may become unusable once the STP recovery finishes. To avoid frequent path recomputations, Magneto has the option to exclude the entire spanning tree containing the failed link, rather than the link itself, from the known topology before recomputing the affected paths.

If control links fail (*i.e.*, all OpenFlow switches are disconnected from the controller), the network data plane is still functional, although Magneto may not be able to update paths. Traffic using magnet MAC addresses still traverses the network until the OpenFlow forwarding entries expire. If the ARP entries associated with magnet MACs expire (either due to time-out or manual flush), new connections will start with the standard ARP process and use real MAC addresses to transmit traffic. The network eventually reverts to a standard L2 network. Making Magneto robust to control network failures [7, 21, 19] is subject of ongoing work.

6. EVALUATION

We evaluate Magneto from three perspectives. First, we show that Magneto provides high path diversity in various hybrid network topologies, with various OpenFlow placement strategies, even when the number of OpenFlow switches

Site	Source	# Switches	Max/Avg/Min Degree
Large	[31]	1577	65 / 2.15 / 1
Emulated	this paper	415	17 / 5.94 / 1
Small	[34]	16	15 / 4.5 / 3

Table 2: We evaluate Magneto on three diverse network topologies, two of them from large campus networks and one randomly generated. Figure 6 shows the node degree distribution of each topology.

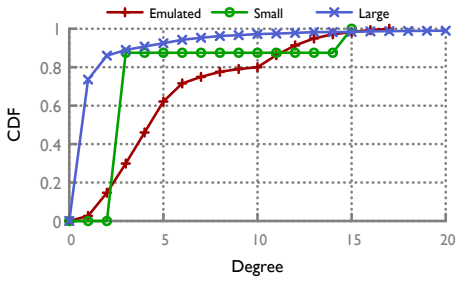


Figure 6: Switch degree distribution for the three evaluated network topologies.

is low. Second, we demonstrate that path updates are fast and introduce negligible delay to the data traffic. Finally, we show that Magneto introduces negligible network overhead.

We run Magneto both in simulation and on a small hybrid lab testbed. Our simulations use three topologies: two real-world and one synthetic, randomly generated. Table 2 describes the topologies and Figure 6 shows the degree distribution of switches in each topology. The "Large" topology represents a large-scale campus network [31] while the "Small" topology is the backbone network of another campus [34]. To generate the "Emulated" topology, we randomly choose the number of switches (between 400 and 600) and the number of links, ensuring the topology is connected. In our experiments, we vary the number and placement strategy of OpenFlow switches in each of these topologies, thus simulating various SDN transition scenarios.

6.1 Path Control

The main goal of Magneto is to provide control over the network without the cost of making the network fully programmable and at low management cost. We ask how effective Magneto is in installing paths across various hybrid network topologies. We run Magneto on each of the three topologies described in Table 2, and the degree distribution of switches is shown in Figure 6. For each run, we randomly select two hosts and compute the five paths with fewest hops between them. We select at random among them a new path to be installed. This makes the simulation realistic since we always install good paths.

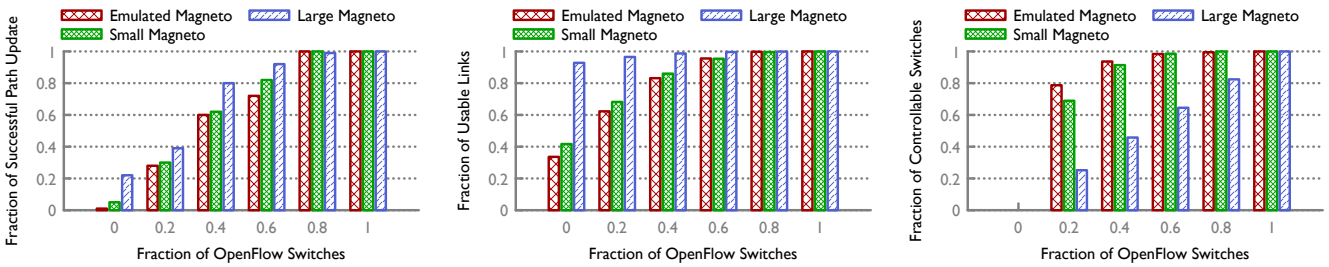
The number and location of OpenFlow switches play a key role in the performance of Magneto. We vary the percentage of switches that are OpenFlow and place them in the network using two strategies: random, where random switches are upgraded to OpenFlow, and greedy, where switches are upgraded in decreasing order of their degree.

Random OpenFlow switch placement. We upgrade random legacy switches to OpenFlow switches. We vary the percentage of OpenFlow switches and compute the fraction of successful path updates. Figure 7a shows averages over 100 runs. A spanning tree is built as the network underlay when there is no OpenFlow switch introduced (*i.e.*, the fraction of OpenFlow switches is 0). As expected, as we increase the number of OpenFlow switches the more paths we can install. This is because it is more likely that the feasibility condition in Section 4 is satisfied: links on the paths to install are more likely to be adjacent to an OpenFlow switch. Our results show that with as much as 40% of all switches transitioned to OpenFlow, we can install any path with a probability of 0.6. Recall that these paths are among the best five between the pair of end hosts. Other hybrid network controllers, such as Panopticon [22] may achieve a higher success rate but at the cost of increased management complexity due to the need to configure VLANs.

The results above are based on several realistic running scenarios and do not capture the number of total paths we can install. To understand this, we compute the number of links that Magneto can control. A link we cannot control cannot be part of a new path. These are the links that are adjacent to an OpenFlow switch or on the network underlay. Figure 7b shows that with less than half of OpenFlow coverage, at least 80% of the links are usable.

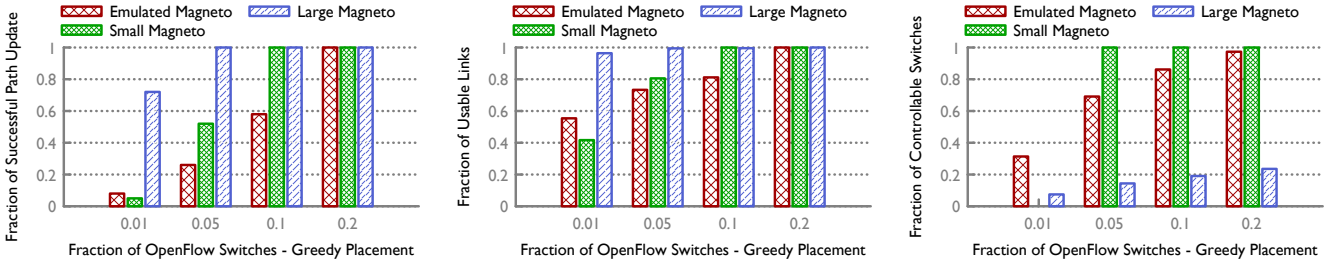
Finally, we define the controllable switches as the switches whose forwarding behaviors can be manipulated by Magneto. These are the OpenFlow switches and the legacy switches whose forwarding tables we can modify. Our results in Figure 7c show that even when only 20% of the switches are OpenFlow-enabled, Magneto can control as many as 75% total switches for their forwarding behaviors. The plots show a discrepancy among the different metrics used to evaluate the "Large" topology. While the path update success and fraction of usable links are high, the fraction of controllable switches is much lower than for the other topologies. This is because many switches (more than 70%) are with degree 1 in the "Large" topology, as shown in Figure 6. These switches provide usable links as part of the spanning tree but are not connected to OpenFlow switches therefore not controllable. While the results may depend on the switch placement strategy, we performed experiments with various random placements and consistently obtained results as the above.

Greedy OpenFlow switch placement. Strategic OpenFlow placement can improve the degree of control offered by Magneto. We propose to upgrade the most influential switches first. We rank the importance of switches according to their degree: the more adjacent links a switch has, the more important it is. Figure 8 shows the fractions of successful path updates, usable links, and controllable switches as we vary the percentage of OpenFlow switches. Greedy OpenFlow placement provides a significant boost in efficiency: we can install any path successfully when only 20% of the switches are programmable. Because the most connected switches are OpenFlow-enabled, we do not need to control many legacy switches. As Figure 8(c), controlling few legacy switches (less than 10%) is sufficient.



(a) (b) (c)

Figure 7: Magneto enables control over a hybrid network with a few OpenFlow switches. We show the path update success in (a), fraction of usable links in (b), and fraction of controllable switches in (c) achieved by Magneto as we upgrade more and more legacy switches to SDN. We assume which switch is updated is a random decision.



(a) (b) (c)

Figure 8: When we upgrade the high degree switches first, Magneto achieves control at a fraction of the cost incurred when the upgrade strategy is greedy. Only 20% of OpenFlow switches achieve full routing flexibility.

6.2 Control Delay

The *control delay* is the time it takes to install a new stable path, *i.e.*, the time between when the controller sends the first seed packet and when the first data packet traverses the new path without the path reverting to the original.

We perform experiments on a real-world testbed in our lab. The testbed consists of eight Dell servers, five Cisco Catalyst legacy switches [3], and two iwNetworks OpenFlow switches [5]. First, we consider a single update subpath and repeatedly vary the data rate on the path to update. Figure 9(a) shows that the control delay remains low when we increase the data rate. That the control delay decreases as we increase the data rate is an artifact of our measurement: when the data rate is low, the time between two consecutive packets is higher therefore our measurement error is higher.

Next, we set the data rate at 1 Mbps and increase the number of subpaths that need to be updated. For this, we place one OpenFlow switch on every subpath. Recall that we need to generate and propagate a different magnet MAC for each update subpath. Figure 9(b) shows the results. The control delay is not significantly affected by the number of subpaths, as generating and propagating magnet MACs are independent operations and can be parallelized.

6.3 Overhead

We quantify the overhead introduced when running Magneto from two perspectives: impact on applications and impact on the network.

Data delay. The data delay is the additional delay introduced in the application traffic due to packet transformations along the path performed by OpenFlow switches, *i.e.*, rewriting MAC addresses. Recall that, because Magneto uses magnet MACs, OpenFlow switches must rewrite the source/destination MAC address of every packet traversing a newly installed path.

To measure the data delay, we connect five servers to an iwNetworks OpenFlow switch as shown in Figure 10 (left). Each server has four 1 Gbps Ethernet interfaces, and we use two interfaces as senders and the other two as receivers. Each server generates 2 Gbps traffic traversing the OpenFlow switch, together all servers generate traffic at 10Gbps (or 15 million packets per sec). Each server sends traffic that returns back to itself. To measure accurate one-way delay, we use PF_RING [8]. We modified the *pfsend* and *pfcount* codes to timestamp every packet before it is sent out and compute its one-way delay when it is received.

Figure 10 (right) shows the delay incurred when rewriting the Ethernet header of each packet and when simply forwarding the packet both under low and high (99%) CPU load. Rewriting packet headers introduces negligible data plane delay even at high CPU load. This matches the findings of an earlier work on application-aware data processing in SDN [27].

CPU and memory overhead. Injecting seed packets from OpenFlow switches could increase the CPU and memory overhead on both legacy switches and OpenFlow switches.

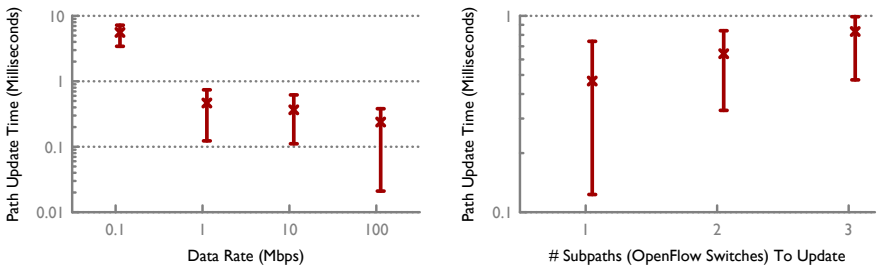


Figure 9: Control delay (the time to install a path) of Magneto remains low as we vary the data rate (left) and the number of update subpaths (right) on the path to install.

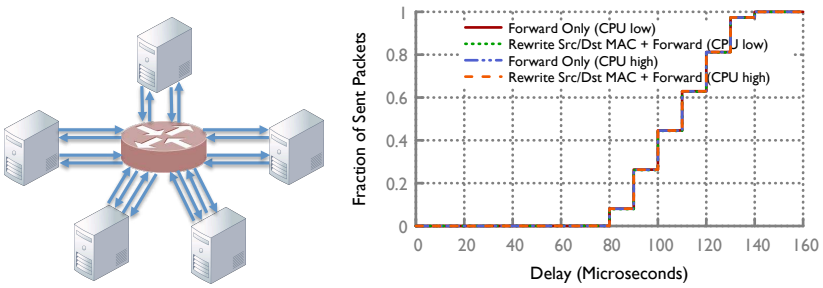


Figure 10: Packet header rewriting by OpenFlow switches does not affect the data plane delay. We use one OpenFlow switch and five servers, with each server sending 2 Gbps through the switch and back to itself (left); path installation introduces negligible delay even at high switch CPU loads (right).

We measure the CPU utilization and memory usage on our Cisco legacy switches and iwNetworks OpenFlow switches, when Magneto controller injects control packets with magnet MAC addresses.

Our results in Table 3 prove that Magneto introduces very little CPU and memory overhead on both legacy and OpenFlow switches. *Address Learning* in Cisco switches *ofswd* and *ofprotocol* in OpenFlow switches are the main processes affected by the sending of seed packets. Even with a large number of magnet MAC addresses (10,000), the total memory overhead increase was only 78 KB on Cisco switch and 146 KB for iwNetworks switch, a small fraction of the total memory available. We collected the CPU utilization on the switches every minute immediately after we started injecting seed packets. The utilization was systematically low, at most 7.36% for iwNetworks switch and 2.89% for Cisco switch.

Control traffic. Magneto introduces little control traffic into the network. In the worst case, the number of seed packets needed to update a path must be twice the number of subpaths. Because forwarding entries in legacy switches expire, Magneto must repeatedly re-inject the same seed packet. Given a standard timeout of five minutes, the additional network overhead is still negligible.

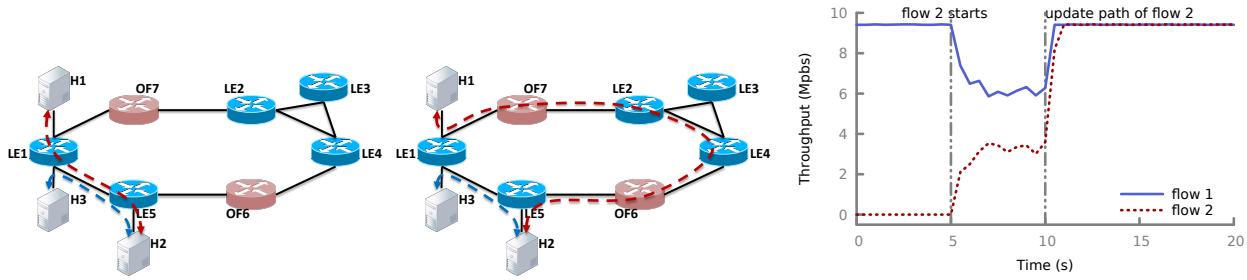
Number of magnet MACs	CPU (iwNetworks)	CPU (Cisco)	Mem (iwNetworks)	Mem (Cisco)
1,000	4.80%	1.75%	16 KB	8 KB
5,000	6.09%	2.46%	55 KB	35 KB
10,000	7.36%	2.89%	146 KB	78 KB

Table 3: CPU and memory load introduced by Magneto on OpenFlow and legacy switches when the number of magnet MACs varies.

7. CASE STUDY: BETTER ROUTING AND FAILURE RECOVERY WITH MAGNETO

We show how Magneto improves network performance by exploiting routing diversity and reacting quickly to network failures. We deploy a hybrid testbed consisting of three servers, five Cisco switches and two iwNetworks OpenFlow switches (Figure 11a). STP runs on the Cisco switches.

Flexible routing. To underline Magneto's ability to find alternate paths quickly, we start two flows, from $H3$ to $H2$ (*flow 1*) and from $H1$ to $H2$ (*flow 2*). Both flows share the link $(LE1, LE5)$ initially, whose capacity we artificially set to 10Mbps. *Flow 1* starts five seconds before *flow 2* (Figure 11a). As soon as *flow 2* starts, it will compete with *flow 1* for the entire capacity on the default path. Neither of the flows can benefit from the entire capacity. After 10 seconds, we use Magneto to update the default path of *flow 2* to $(LE1, OF7, LE2, LE4, OF6, LE5)$. As soon as the update finishes, both flows can run at full rate as they do not compete with each other. Figure 11c shows the rate of each flow during the experiment.

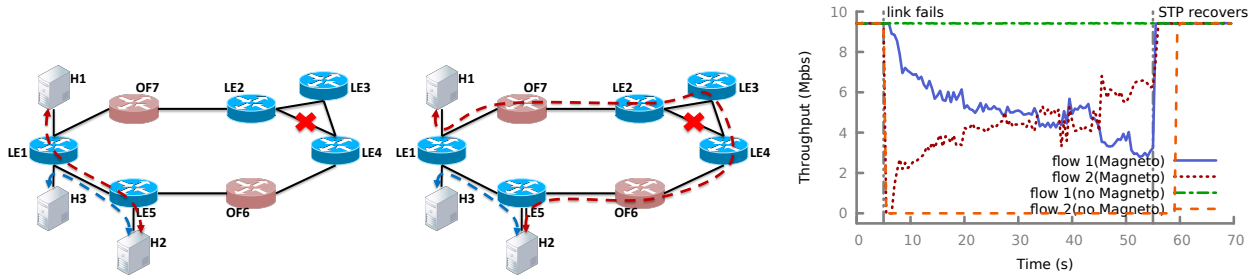


(a) *flow 1* and *flow 2* use the same path and compete for its bandwidth.

(b) Magneto changes *flow 2* to take another route to alleviate congestion.

(c) Flow throughput over time.

Figure 11: Magneto alleviates congestion by reconfiguring flows traversing both legacy and OpenFlow switches. *flow 1* and *flow 2* start on the same path and compete for its bandwidth. As soon as Magneto updates the path of *flow 2*, both flows can use all available bandwidth.



(a) Magneto updates *flow 2* to take the route used before to restore end-to-end connectivity.

(b) Magneto changes the route of *flow 2* after STP recovers.

(c) Flow throughput over time.

Figure 12: In face of the link failure on $(LE2, LE4)$, Magneto switches *flow 2* to the original path $(LE1, LE5)$ to rapidly restore connectivity instead of waiting for STP to recover. After STP converges, Magneto updates the path of *flow 2* again to achieve maximum throughput.

Quick failure recovery. We now demonstrate how Magneto can recover from network failures. Consider the end of the previous experiment where *flow 1* and *flow 2* take non-overlapping paths to their destination. After five seconds, the link $(LE2, LE4)$ fails. As STP forms a loop-free underlay among connected legacy switches, no alternative path is available for *flow 2* until STP recovery finishes. On the other hand, Magneto can adapt immediately by detecting the propagated STP BPDU frames and re-routing *flow 2* on its original path $(LE1, LE5)$. Although *flow 2* competes once again with *flow 1* for the capacity on $(LE1, LE5)$, the end-to-end connectivity is restored. Magneto detects when STP finishes recovery by sending probes between $OF6$ and $OF7$ every second. Once the probe is received on the other end, Magneto knows STP finishes recovery and redirects *flow 2* to the path $(LE1, OF7, LE2, LE3, LE4, OF6, LE5)$. Relying solely on on STP to recover from the failure disconnects *flow 2* during the STP recovery process, whereas with Magneto, end-to-end connectivity is preserved.

8. CONCLUSIONS

We present Magneto, a network controller that enables unified, fine-grained routing control in hybrid networks. Magneto uses OpenFlow's ability to send custom-made packets into the data plane to manipulate legacy switches into updating forwarding entries for specific MAC addresses. Via magnet addresses, Magneto gains visibility to the network and achieves fine-grained path control. Our evaluation on a lab testbed and simulations on large enterprise network topologies show that Magneto is able to achieve full control over routing when only 20% of network switches are programmable and with negligible computation and latency overhead. Magneto also poses a number of new research questions such as the strategic placement and number of SDN switches as well as magnet addresses needed to exert SDN-like control over legacy networks and to what extent such control can be exercised.

9. ACKNOWLEDGMENTS

We are grateful to SOSR anonymous reviews for their insightful comments. We thank our college network administrators, especially Irene Jacobson, for providing switches and helping us build the testbed. This research was supported in part by NSF grants CNS-1411636, CNS 1618339 and CNS 1617729, DTRA grant HDTRA1-14-1-0040 and DoD ARO MURI Award W911NF-12-1-0385.

10. REFERENCES

[1] Cisco. Campus Network for High Availability Design Guide. http://bit.ly/1ffWkzT.

[2] AT&T Vision Alignment Challenge Technology Survey - AT&T Domain 2.0 Vision White Paper. https://www.att.com/Common/about_us/pdf/AT&T%20Domain%202.0%20Vision%20White%20Paper.pdf.

[3] Cisco switches. http://www.cisco.com/c/en/us/products/switches/index.html.

[4] Global ONOS and SDN-IP deployment. http://onosproject.org/wp-content/uploads/2015/06/PoC_global-deploy.pdf.

[5] iwNetworks switches. http://www.iwnetworks.com/main/products.

[6] New Generation Network testbed. http://www.jgn.nict.go.jp/jgn2plus_archive/english/index.html.

[7] ONOS. http://onosproject.org/.

[8] PF_RING. http://www.ntop.org/products/packet-capture/pf_ring/.

[9] P. Bosshart, D. Daly, M. Izzard, N. McKeown, J. Rexford, D. Talayco, A. Vahdat, G. Varghese, and D. Walker. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communications Review*, July 2014.

[10] Software Defined Networks Study. http://www.currentanalysis.com/news/2014/pr-SDN-NFV-Deployment.asp.

[11] O. N. Foundation. Openflow switch specification, version v1.3.4, 2014.

[12] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. Nox: towards an operating system for networks. *ACM SIGCOMM Computer Communication Review*, 38(3):105–110, 2008.

[13] R. Hand and E. Keller. Closedflow: Openflow-like control over proprietary devices. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 7–12. ACM, 2014.

[14] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer. Achieving high utilization with software-driven wan. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 15–26. ACM, 2013.

[15] D. K. Hong, Y. Ma, S. Banerjee, and Z. M. Mao. Incremental deployment of SDN in hybrid enterprise and ISP networks. In *SOSR*, 2016.

[16] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, et al. B4: Experience with a globally-deployed software defined wan. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 3–14. ACM, 2013.

[17] C. Jin, C. Lumezanu, Q. Xu, Z.-L. Zhang, and G. Jiang. Telekinesis: controlling legacy switch routing with openflow in hybrid networks. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, page 20. ACM, 2015.

[18] Juniper. Juniper Campus Networks Reference Architecture, 2010. http://juni.pr/1rR0vaZ.

[19] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, et al. Onix: A distributed control platform for large-scale production networks. In *OSDI*, volume 10, pages 1–6, 2010.

[20] T. Koponen and et al. Network virtualization in multi-tenant datacenters. In *USENIX NSDI*, 2014.

[21] M. Kuźniar, P. Perešíni, N. Vasić, M. Canini, and D. Kostić. Automatic failure recovery for software-defined networks. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 159–160. ACM, 2013.

[22] D. Levin, M. Canini, S. Schmid, F. Schaffert, and A. Feldmann. Panopticon: Reaping the Benefits of Incremental SDN Deployment in Enterprise Networks. In *USENIX Annual Technical Conference*, 2014.

[23] H. Lu, N. Arora, H. Zhang, C. Lumezanu, J. Rhee, and G. Jiang. HybNET: Network Manager for a Hybrid Network Infrastructure. In *Middleware*, 2013.

[24] M. Markovitch and S. Schmid. Shear: A highly available and flexible network architecture. In *ICNP*, 2015.

[25] N. McKeown. How sdn will shape networking. Open Networking Summit 2011, 2011. Available from https://www.youtube.com/watch?v=c9-K5O_qYgA.

[26] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: enabling innovation in campus networks. *ACM Sigcomm CCR*, 38:69–74, 2008.

[27] H. Mekky, F. Hao, S. Mukherjee, Z.-L. Zhang, and T. Lakshman. Application-aware data plane processing in sdn. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 13–18. ACM, 2014.

[28] S. A. Overview. v1.0, 2013.

[29] M. Reitblatt, N. Foster, J. Rexford, C. Schelsinger, and D. Walker. Abstractions for network update. In *ACM Sigcomm*, 2012.

[30] S. Shenker. The future of networking, and the past of protocols. Open Networking Summit 2011, 2011. Available from https://www.youtube.com/watch?v=YHeyuD89n1Y.

[31] Y.-W. E. Sung, S. G. Rao, G. G. Xie, and D. A. Maltz. Towards systematic design of enterprise networks. In *Proceedings of the 2008 ACM CoNEXT Conference*, page 22. ACM, 2008.

[32] S. Vissicchio, O. Tilmans, L. Vanbever, and J. Rexford. Central control over distributed routing. In *ACM SIGCOMM*, 2015.

[33] R. Wang, D. Butnariu, and J. Rexford. OpenFlow-based Server Load Balancing Gone Wild. In *Hot-ICE*, 2011.

[34] H. Zeng, P. Kazemian, G. Varghese, and N. McKeown. Automatic test packet generation. In *Proceedings of the 2012 ACM CoNEXT Conference*, pages 241–252. ACM, 2012.