# Panopticon: Reaping the Benefits of Incremental SDN Deployment in Enterprise Networks

Dan Levin$^{\dagger}$ Marco Canini$^{\star}$ Stefan Schmid$^{\dagger\ddagger}$ Fabian Schaffert$^{\dagger}$ Anja Feldmann$^{\dagger}$
$^{\dagger}$*TU Berlin* $^{\star}$*Université catholique de Louvain* $^{\ddagger}$*Telekom Innovation Labs*

## Abstract

The operational challenges posed in enterprise networks present an appealing opportunity for automated orchestration by way of Software-Defined Networking (SDN). The primary challenge to SDN adoption in the enterprise is the deployment problem: How to deploy and operate a network consisting of both legacy and SDN switches, while benefiting from simplified management and enhanced flexibility of SDN.

This paper presents the design and implementation of Panopticon, an architecture for operating networks that combine legacy and SDN switches. Panopticon exposes an abstraction of a logical SDN in a partially upgraded legacy network, where SDN benefits can extend over the entire network. We demonstrate the feasibility and evaluate the efficiency of our approach through both testbed experiments with hardware switches and through simulation on real enterprise campus network topologies entailing over 1500 switches and routers. Our results suggest that when as few as 10% of distribution switches support SDN, most of an enterprise network can be operated as a single SDN while meeting key resource constraints.

## 1 Introduction

Mid to large enterprise networks present complex operational requirements: The network must operate reliably and provide high-performance connectivity to a wide range of devices while enforcing organizational policy and traffic isolation across complex boundaries. The enterprise, thus represents a challenging environment for network management. Studies show, enterprise networks' operations largely rely on cumbersome and error-prone approaches that require labor-intensive configuration tasks by trained operators [5,6,28].

Software-Defined Networking (SDN) has the potential to provide a principled solution to both simplify management and enhance flexibility of the network. SDN is a paradigm that offers a programmatic, logically-centralized interface for specifying the intended network behavior. Through this interface, a software program acts as a network controller by configuring forwarding rules on switches and reacting to topology and traffic changes.

While commercial SDN deployment started within data-centers [2] and the WAN [11], the roots of today's SDN arguably go back to the policy management needs of enterprise networks [5,6]. In this paper, we focus on mid to large enterprise networks, *i.e.*, those serving hundreds to thousands of users, whose infrastructure is physically located at a locally-confined site. We choose this environment due to its complexity as well as the practical benefits that SDN network orchestration promises.

Enterprises stand to *benefit from SDN on many different levels*, including: (*i*) network policy can be declared over high-level names and enforced dynamically at fine levels of granularity [5, 13, 23], (*ii*) policy can dictate the paths over which traffic is directed, facilitating middlebox enforcement [28] and enabling greater network visibility, (*iii*) policy properties can be verified for correctness [19,20], and (*iv*) policy changes can be accomplished with strong consistency properties, eliminating the chances of transient policy violations [30].

Existing enterprises that wish to leverage SDN however, face the problem of how to deploy it. SDN is not a "drop-in" replacement for the existing network: SDN redefines the traditional, device-centric management interface and requires the presence of programmable switches in the data plane. Consequently, the migration to SDN creates new opportunities as well as notable challenges:

**Realizing the benefits.** In the enterprise, the benefits of SDN should be realized as of the first deployed switch. Consider the example of Google's software-defined WAN [11], which required years to fully deploy, only to achieve benefits after a complete overhaul of their switching hardware. For enterprises, it is undesirable, and we argue, unnecessary to completely overhaul the network infrastructure before realizing benefits from

SDN. An earlier return on investment makes SDN more appealing for adoption.

**Eliminating disruption while building confidence.** Network operators must be able to incrementally deploy SDN technology in order to build confidence in its reliability and familiarity with its operation. Without such confidence, it is risky and undesirable to replace all production control protocols with an SDN control plane as a single "flag-day" event, even if existing deployed switches already support SDN programmability. To increase its chances for successful adoption, any network control technology, including SDN, should allow for a small initial investment in a deployment that can be gradually widened to encompass more and more of the network infrastructure and traffic.

**Respecting budget and constraints.** Rather than a green field, network upgrade starts with the existing deployment and is typically a staged process—budgets are constrained, and only a part of the network can be upgraded at a time.

To address these challenges, we present **Panopticon**, a novel architecture for realizing an SDN control plane in a network that combines legacy switches and routers with incrementally deployable SDN switches . We call such networks *transitional networks*. Panopticon abstracts the transitional network into a logical SDN, extending SDN capabilities potentially over the entire network. As an abstraction layer, Panopticon is responsible for hiding the legacy devices and acting as a "network hypervisor" that maps the logical SDN abstraction to the underlying hardware. In doing so, Panopticon overcomes key limitations of current approaches for transitional networks, which we now briefly review.

## 1.1 Current Transitional Networks

We begin with the "dual-stack" approach to transitional or "hybrid" SDN, shown in Figure 1a, where the flow-space is partitioned into several disjoint slices and traffic is assigned to either SDN or legacy processing [22]. To guarantee that an SDN policy applies to any arbitrary traffic source or destination in the network, the source or destination must reside at an SDN switch. Traffic within a flow-space not handled by SDN forwarding and traffic that never traverses an SDN switch may evade policy enforcement, making a single SDN policy difficult to realize over the entire network.

In summary, this mode's prime limitation is that it does not rigorously address how to realize the SDN control plane in a partial SDN deployment scenario, nor how to operate the resulting mixture of legacy and SDN devices as an SDN. It thus requires a contiguous deployment of hybrid programmable switches to ensure SDN policy compliance when arbitrary sources and destinations must be policy-enforced.

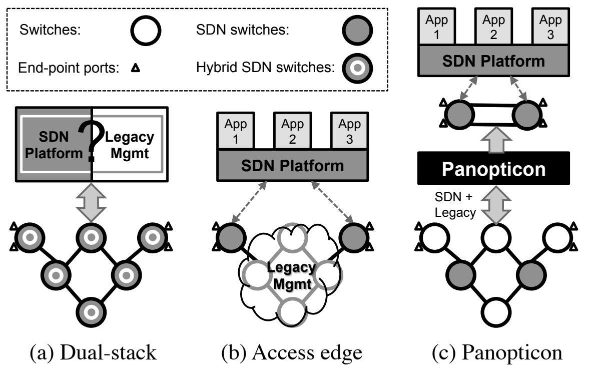


Figure 1: Current transitional network approaches vs. Panopticon: (a) Dual-stack ignores legacy and SDN integration. (b) Full edge SDN deployment enables end-to-end control. (c) Panopticon partially-deployed SDN yields an interface that acts like a full SDN deployment.

The second approach (Figure 1b) involves deploying SDN at the network access edge [7]. This mode has the benefit of enabling full control over the access policy and the introduction of new network functionality at the edge, *e.g.*, network virtualization [2]. Unlike a data-center environment where the network edge may terminate at the VM hypervisor, the enterprise network edge terminates at an access switch. At the edge of an enterprise network, to introduce new functionalities not accommodated by existing hardware involves replacing thousands of access switches. This mode of SDN deployment also limits the ability to apply policy to forwarding decisions within the network core (*e.g.*, load balancing, waypoint routing).

## 1.2 Panopticon

Panopticon realizes an SDN control plane for incrementally deployable software-defined networks. Our main insight is that *the benefits of SDN to enterprise networks can be realized for every source-destination path that includes at least one SDN switch.* Thus, we do not mandate a full SDN switch deployment—a small subset of all switches may suffice. Conceptually, a single SDN switch traversed by each path is sufficient to enforce end-to-end network policy (*e.g.*, access control). Moreover, traffic which traverses two or more SDN switches may be controlled at finer levels of granularity enabling further customized forwarding (*e.g.*, traffic load-balancing).

Based on this insight, we devise a mechanism called the Solitary Confinement Tree (SCT), which uses VLANs to ensure that traffic destined to operator-selected switchports on legacy devices passes through at least one SDN switch. Combining mechanisms readily available in legacy switches, SCTs correspond to a spanning tree connecting each of these switchports to SDN switches, overcoming VLAN scalability limitations.

Just as many enterprise networks regularly divert traffic to traverse a VLAN gateway or a middlebox, a natural consequence of redirecting traffic to SDN switches is an increase in certain path lengths and link utilizations. As we discuss later (§4), deployment planning requires careful consideration to mind forwarding state capacities and to avoid introducing performance bottlenecks. Consequently, Panopticon presents operators with various resource-performance trade-offs, *e.g.*, between the size and fashion of the partial SDN deployment, and the consequences for the traffic.

As opposed to the dual-stack approach, Panopticon (Figure 1c) abstracts away the partial and heterogeneous deployment to yield a logical SDN. As we reason later (§ 2.4), many SDN control paradigms can be achieved in a logical SDN. Panopticon enables the expression of any end-to-end policy, as though the network were one big, virtual switch. Routing and path-level policy, *e.g.*, traffic engineering can be expressed too [3], however the abstract network view is reduced to just the deployed SDN switches. As more of the switches are upgraded to support SDN, more fine-grained path-level policies can be expressed.

In summary, we make the following contributions:

1. We design a network architecture for realizing an SDN control plane in a transitional network (§ 2). This includes a scalable mechanism for extending SDN capabilities to legacy devices.
2. We demonstrate the system-level feasibility of our approach with a prototype (§ 3). Through an experimental evaluation with hardware switches we show that our proof-of-concept Panopticon implementation: (*i*) has no negative impact on application performance and (*ii*) recovers from link failures within a few seconds.
3. We conduct a simulation-driven feasibility study using real enterprise network topologies (with over 1500 switches) and traffic traces (§ 4). We find that with only a few SDN switches (∼30-40), it becomes possible to operate an enterprise as a single SDN while meeting key practical constraints.

## 2 Panopticon SDN Architecture

This section presents the Panopticon architecture, which abstracts a physical transitional network, where not every switch supports SDN, into a logical SDN. The goal is to enable an SDN programming interface, for defining network policy, which can be extended beyond the SDN switches to ports on legacy switches as well.

Our architecture relies on certain assumptions underlying the operational objectives within enterprise networks. To veryify these, we conducted five in-person in-

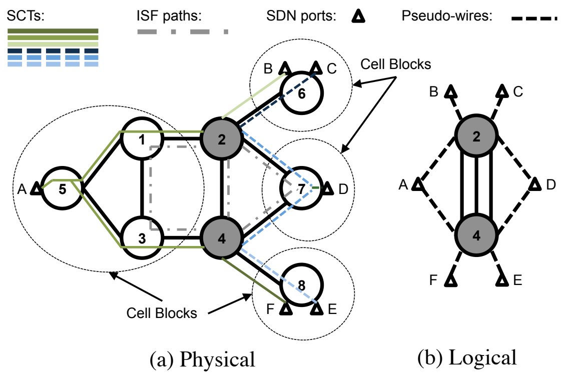


Figure 2: Transitional network of 8 switches (SDN switches are shaded). (a) The SCTs (Solitary Confinement Trees) of every SDNc port overlaid on the physical topology. (b) Corresponding logical view of all SDNc ports, connected to SDN switches via pseudo-wires.

terviews with operators from both large (≥10,000 users) and medium (≥500 users) enterprise networks and later, solicited 60 responses to open-answer survey questions from a wider audience of network operators [9].

Based on our discussions with network operators, and in conjunction with several design guidelines (*e.g.*, see [8, 17]), we make the following assumptions about mid to large enterprise networks and hardware capabilities. Enterprise network hardware consists primarily of Ethernet bridges, namely, switches that implement standard L2 mechanisms (*i.e.*, MAC-based learning and forwarding, and STP) and support VLAN (specifically, 802.1Q and per-VLAN STP). Routers or L3 switches are used as gateways to route between VLAN-isolated IP subnets. For our purposes, we assume a L3 switch is also capable of operating as a L2 switch. In addition, we assume that enterprises no longer intentionally operate "flood-only" hub devices for general packet forwarding.

Under these assumptions about legacy enterprise networks, Panopticon can realize a broad spectrum of logical SDNs: Panopticon can extend SDN capabilities to potentially every switchport in the network, however not every port need be included in the logical SDN. We envision an operator may conservatively choose to deploy Panopticon only in part of the network at first, to build confidence and reduce up-front capital expenditure, and then iteratively expand the deployment.

To accommodate iterative expansion of the logical SDN, we divide the set of switchports in the network into *SDN-controlled (SDNc) ports*, that is, those that need to be exposed to and controlled through the logical SDN and *legacy ports*, those that are not. Note that while an SDNc port is conceptually an access port to the logical SDN network, it is not necessarily physically located on an SDN switch (see port *A* in Figure 2): It may be connected to an end-host or a legacy access switch.

We extend SDN capabilities to legacy switches by ensuring that all traffic to or from an SDNc port is always restricted to a *safe end-to-end path*, that is, a path that traverses at least one SDN switch. We call this key property of our architecture *Waypoint Enforcement*. The challenge to guaranteeing Waypoint Enforcement is that we may rely *only on existing mechanisms and features readily available* on legacy switches.

## 2.1 Realizing Waypoint Enforcement

Panopticon uses VLANs to restrict forwarding and guarantee Waypoint Enforcement, as these are ubiquitously available on legacy enterprise switches. To conceptually illustrate how, we first consider a straightforward, yet impractical scheme: For every pair of ports which includes at least one SDNc port, choose one SDN switch as the waypoint, and compute the (shortest) end-to-end path that includes the waypoint. Next, assign a unique VLAN ID to every end-to-end path and configure the legacy switches accordingly. This ensures that all forwarding decisions made by every legacy switch only send packets along safe paths. However, such a solution is infeasible, as VLAN ID space is limited to 4096 values, and often fewer are supported in hardware for simultaneous use. Such a rigid solution furthermore limits path diversity to the destination according and cripples fault tolerance.

**Solitary Confinement Trees.** To realize guaranteed waypoint enforcement in Panopticon, we introduce the concept of a *Solitary Confinement Tree* (SCT): A scalable waypoint enforcement mechanism that provides end-to-end path diversity. We first introduce the concepts of cell block and frontier. Intuitively, the role of a cell block is to divide the network into isolated islands where VLAN IDs can be reused. The border of a cell block consists of SDN switches and is henceforth called the *frontier*.

**Definition 1** (Cell Blocks)**.** *Given a transitional network G,* Cell Blocks $CB(G)$ *is defined as the* set of connected components *of the network obtained after removing from G the SDN switches and their incident links.*

**Definition 2** (Frontier)**.** *Given a cell block $c \in CB(G)$, we define the* Frontier $\mathscr{F}(c)$ *as the subset of SDN switches that are adjacent in G to a switch in c.*

Intuitively, the solitary confinement tree is a spanning tree within a cell block, *plus* its frontier. Each SCT provides a safe path from an SDNc port $\pi$ to every SDN switch in its frontier—or if VLAN resources are scarce, a *subset* of its frontier, which we call the *active frontier*. A single VLAN ID can then be assigned to each SCT, which ensures traffic isolation, provides per-destination path diversity, and allows VLAN ID reuse across cell blocks. Formally, we define *SCT*s as:

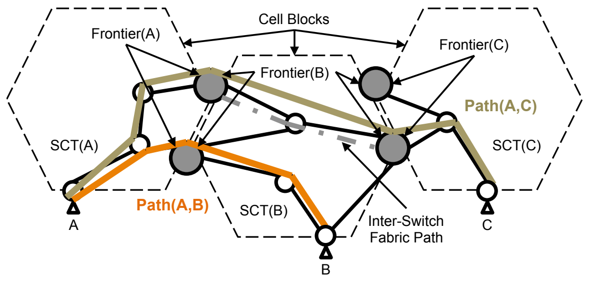


Figure 3: The forwarding path between A and B goes via the frontier shared by SCT (A) and SCT (B); the path between A and C goes via an Inter-Switch Fabric path connecting SCT (A) and SCT (C).

**Definition 3** (Solitary Confinement Tree)**.** *Let $c(\pi)$ be the cell block to which an SDNc port $\pi$ belongs. And let* $\mathrm{ST}(c(\pi))$ *denote a spanning tree on $c(\pi)$. Then, the* Solitary Confinement Tree $\mathrm{SCT}(\pi)$ *is the network obtained by augmenting* $\mathrm{ST}(c(\pi))$ *with the (active) frontier $\mathscr{F}(c(\pi))$, together with all links in $c(\pi)$ connecting a switch $u \in \mathscr{F}(c(\pi))$ with a switch in* $\mathrm{SCT}(\pi)$.

**Example.** Let us consider the example transitional network of eight switches in Figure 2a. In this example, SCT ($A$) is the tree that consists of the paths $5 \rightarrow 1 \rightarrow 2$ and $5 \rightarrow 3 \rightarrow 4$. Instead note that SCT ($B$), which corresponds to the path $6 \rightarrow 2$, includes a single SDN switch because switch 2 is the only SDN switch adjacent to cell block $c(B)$. Figure 2b shows the corresponding logical view of the transitional network enabled by having SCTs. In this logical view, every SDNc port is connected to at least one frontier SDN switch via a pseudo-wire (realized by the SCT).

## 2.2 Packet Forwarding in Panopticon

We now illustrate Panopticon's basic forwarding behavior (Figure 3). As in any SDN, the control application is responsible for installing the necessary forwarding state at the SDN switches (*e.g.*, in accordance with the access policy) and for reacting to topology changes (fault tolerance is discussed in § 2.3).

Let us first consider traffic between a pair of SDNc ports $s$ and $t$. When a packet from $s$ enters $\mathrm{SCT}(s)$, the legacy switches forward the packet to the frontier based on MAC-learning, which establishes a symmetric path. Note that a packet from $s$ may use a different path within $\mathrm{SCT}(s)$ to the frontier for each distinct destination. Once traffic toward $t$ reaches its designated SDN switch $u \in \mathscr{F}(c(s))$, one of two cases arises:

**SDN switches act as VLAN gateways.** This is the case when the destination SDNc port $t$ belongs to a cell block whose frontier $\mathscr{F}(c(t))$ shares at least one switch $u$ with $\mathscr{F}(c(s))$. Switch $u$ acts as the designated gateway between $\mathrm{SCT}(s)$ and $\mathrm{SCT}(t)$: That is, $u$ rewrites the VLAN tag and places the traffic within $\mathrm{SCT}(t)$. For instance, in the example of Figure 2a, switch 2 acts as the gateway between ports *A*, *B* and *C*.

**Inter-Switch Fabric (ISF).** When no SDN switch is shared, we use an *Inter-Switch Fabric (ISF) path*: point-to-point tunnels between SDN switches which can be realized *e.g.*, with VLANs or GRE. In this case, the switch $u$ chooses one of the available paths to forward the packet to an SDN switch $w \in \mathscr{F}(c(t))$, where $w$ is the designated switch for the end-to-end path $p(s,t)$. In our example of Figure 2a, ISF paths are shown in gray and are used *e.g.*, for traffic from *B* or *C* to *E* or *F*, and vice versa.

We next turn to the forwarding behavior of legacy ports. Again, we distinguish two cases. First, when the path between two legacy ports only traverses the legacy network, forwarding is performed according to the traditional mechanisms and is unaffected by the partial SDN deployment. Policy enforcement and other operational objectives must be implemented through traditional means, *e.g.*, ACLs. In the second case, anytime a path between two legacy ports necessarily encounters an SDN switch, the programmatic forwarding rules at the switch can be leveraged to police the traffic. This is also the case for all traffic between any pair of an SDNc and a legacy port. In other words, Panopticon *always guarantees safe paths for packets from or to every SDNc port,* which we formally prove in the technical report [9].

## 2.3 Architecture Discussion

Having described all components of the architecture, we now discuss certain key properties.

**Key SCT properties.** Recall that one VLAN ID is used per SCT and that VLAN IDs can be reused across Cell Blocks. Limiting the size of the active frontier allows further VLAN ID reuse across fully-disjoint SCTs within the same cell block. A different path may be used within the SCT for each distinct destination. SCTs can be precomputed and automatically installed onto legacy switches (*e.g.*via SNMP) however, re-computation is required when the physical topology changes.

**ISF path diversity trade-offs.** Within the ISF, there may be multiple paths between any given pair of SDN switches. We expect that some applications may require a minimum number of paths. A minimum of two disjoint paths is necessary, to tolerate single link failures. If the ISF is realized using a VLAN-based approach, each path consumes a VLAN ID from every cell block it traverses. Alternative mechanisms, *e.g.*, IP encapsulation or network address translation can be used to implement the ISF depending on SDN and legacy hardware capabilities.

**Coping with broadcast traffic.** Broadcast traffic can be a scalability concern. We take advantage of the fact that each SCT limits the broadcast domain size, and we rely on SDN capabilities to enable in-network ARP and DHCP proxies as shown in [21]. We focus on these important bootstrapping protocols as it was empirically observed that broadcast traffic in enterprise networks is primarily contributed by ARP and DHCP [21, 26]. Last, we note that in the general case, if broadcast traffic must be supported, the overhead that Panopticon introduces is proportional to the number of SCTs in a cell block, which, at worst, grows linearly with the number of SDNc ports of a cell block.

**Tolerating failures.** We decompose fault tolerance into three orthogonal aspects. First, within an SCT, Panopticon relies on standard STP mechanisms to survive link failures, although to do so, there must exist sufficient physical link redundancy in the SCT. The greater the physical connectivity underlying the SCT, the higher the fault tolerance. Additionally, the coordination between SDN controller and legacy STP mechanisms allows for more flexible fail-over behavior than STP alone. When an SDN switch at the frontier $\mathscr{F}$ of an SCT notices an STP re-convergence, we can adapt the forwarding decisions at $\mathscr{F}$'s SDN switches to restore connectivity. A similar scheme can address link failures within the ISF.

Second, when SDN switches or their incident links fail, the SDN controller recomputes the forwarding state and installs the necessary flow table entries. Furthermore, precomputed failover behavior can be leveraged as of OpenFlow version 1.1 [29].

Third, the SDN control platform must be robust and available. In this respect, previous work [12] demonstrates that well-known distributed systems techniques can effectively achieve this goal.

## 2.4 Realizing SDN Benefits

By now, we have described how Panopticon shifts the active network management burden away from the legacy devices and onto the SDN control plane. This conceptually reduces the network to a logical SDN as presented in Figure 2b. Consequently, we want to be able to reason about what types of policy can be specified and which applications can be realized in such a transitional network.

Panopticon exposes an SDN abstraction of the underlying partial SDN deployment. In principle, any control application that runs on a full SDN can be supported in Panopticon since, from the perspective of the application, the network appears as though it is a full SDN deployment consisting of just the SDN switches. In practice, there are a small number of caveats.

**SDNc ports in the logical SDN.** An SDNc port in Panopticon is not necessarily physically located at an SDN switch, and it may be attached to multiple SDN switches. Accordingly, the SDN controller must take into account the multiple paths over which an SDNc port may be reached from its frontier. Furthermore, in general visibility into how resources are shared on legacy links can not be guaranteed.

**Logical SDN vs. full SDN.** As an abstraction layer, Panopticon is responsible for hiding the legacy devices and acts as a "network hypervisor" that maps the logical SDN abstraction to the underlying hardware (similar to the concept of network objects in Pyretic [23]). However, because the global network view is reduced to the set of SDN switches, applications are limited to control the forwarding behavior based on the logical SDN. This should not be viewed strictly as a limitation, as it may be desirable to further abstract the entire network as a single virtual switch over which to define high-level policies (*e.g.*, access policy) and have the controller platform manage the placement of rules on physical switches [18]. Nevertheless, the transitional network stands to benefit in terms of management simplification and enhanced flexibility as we next illustrate.

**More manageable networks.** Arguably, as control over isolation and connectivity is crucial in the enterprise context we consider, the primary application of SDN is *policy enforcement*. As in Ethane [5], Panopticon enables operators to define a single network-wide policy, and the controller enforces it dynamically by allowing or preventing communication upon seeing the first packet of a flow as it tries to cross an SDN switch.

The big switch [18] abstraction enables the network to support Ethernet's plug-and-play semantics of flat addressing and, as such, simplifies the handling of host mobility. This can be observed from the fact that our architecture is an instance of the fabric abstraction [7]. The ISF represents the network core and SCTs realize the edge. At the boundary between the SCTs and ISF, the SDN switches enable the decoupling of the respective network layers, while ensuring scalability through efficient routing in the ISF.

**More flexible networks.** The controller maintains the global network view and performs route computation for permitted flows. This provides the opportunity to efficiently enforce middlebox-specific traffic steering within the SDN-based policy enforcement layer, as in SIMPLE [28]. Integrating middleboxes in Panopticon requires that middleboxes are connected to SDNc ports.

A logical SDN also enables the realization of strong consistency semantics for policy updates [30]. Although legacy switches do not participate in the consistent network update, at the same time, they do not themselves express network policy—as that forwarding state resides exclusively on SDN switches.

Putting it all together, Panopticon is the first architecture to realize an approach for operating a transitional network as though it were a fully deployed SDN, yielding benefits for the entire network, not just the devices that support SDN programmability.

## 3 Panopticon Prototype

To cross-check certain assumptions on which Panopticon is built, this section describes our implementation and experimental evaluation of a Panopticon prototype. The primary goal of our prototype is to demonstrate feasibility for legacy switch interaction–namely, the ability to leverage path diversity within each SCT, and respond to failure events and other behaviors within the SCT.

Our prototype is implemented upon the POX OpenFlow controller platform [1] and comprises two modules: path computation and legacy switch interaction.

**Path computation.** At the level of the logical SDN, our path computation module is straightforward: It reacts to the first packet of every flow and, if the flow is permitted, it uses the global network view to determine the shortest path to the destination. Consequently, it installs the corresponding forwarding rules. Our implementation supports two flow definitions: (1) the aggregate of packets between a pair of MAC addresses, and (2) the microflow, *i.e.*, IP 5-tuple. As each SDNc port may be reached over multiple paths from the SDN switches on its frontier, our prototype takes into account the behavior of STP within the SCT (monitored by the component below) to select the least-cost path based on source-destination MAC pair.

**Legacy switch interaction.** The Spanning Tree Protocol (STP) or a variant such as Rapid STP, is commonly used to achieve loop freedom within L2 domains and we interact with STP in two ways. First, within each SCT, we configure a per-VLAN spanning tree protocol (*e.g.*, Multiple STP) rooted at the switch hosting the SCT's SDNc port. We install forwarding rules at each SDN switch to redirect STP traffic to the controller, which interprets STP messages to learn the path cost between any switch on the frontier and the SCT's SDNc port, but *does not reply* with any STP messages. Collectively, this behavior guarantees that each SCT is loop free. When this component notices an STP re-convergence, it notifies the path computation module, which in turn adapts the forwarding decisions at SDN switches to restore connectivity as necessary. Second, to ensure network-wide loop freedom for traffic from legacy ports, SDN switches behave as ordinary STP participants. When supported, this is achieved by configuring STP on the switches themselves. Otherwise, Panopticon can run a functionally equivalent implementation of STP.

### 3.1 Application: Consistent Updates

To showcase the "logical SDN" programming interface exposed by Panopticon, we have implemented per-packet consistency [30] for transitional networks. Our application allows an operator to specify updates to the link state of the network, while ensuring that the safety property of per-packet consistency applies over the entire network, even to legacy switches.

To implement this application, we modify the path computation to assign a unique configuration version number to every shortest path between SDNc ports. This version number is used to classify packets according to either the current or the new configuration.

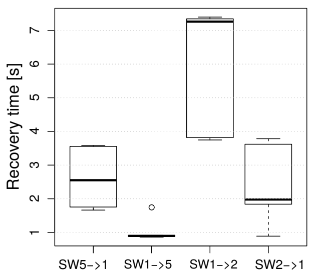
When the transition from current to new configuration begins, the controller starts updating all the SDN switches along the shortest path for both the forward and backward traffic. This update includes installing a new forwarding rule and using the IP *TOS* header field (i.e., in a monotonically increasing fashion) to encode or match the version number. The rules for the old configuration with the previous version number, if there are any, are left in place and intact. This procedure guarantees that any individual packet traversing the network sees only the "old" or "new" policy, but never both.

Once all the rules for the new configuration are in place at every switch, gratuitous ARP messages are sent over to the legacy switches along the new path so that the traffic is re-routed. After a operator-defined grace-period, when the last in-flight packet labeled with the "old" tag leaves the network, the controller deletes the old configuration rules from all the SDN switches, and the process completes.

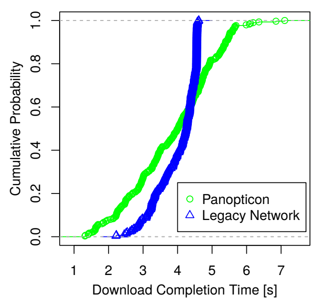
## 3.2 Evaluation

Our prototype is deployed on a network of hardware switches comprising two NEC IP8800 OpenFlow switches and one Cisco C3550XL, three Cisco C2960G, and two HP 5406zl MAC-learning Ethernet switches, interconnected as in Figure 2a. To emulate 6 hosts (*A* through *F*), we use an 8-core server with an 8-port 1Gbps Ethernet interface which connects to each SDNc port on the legacy switches depicted in the figure. Two remaining server ports connect to the OpenFlow switches for an out-of-band control channel.

We conduct a first experiment to demonstrate how Panopticon recovers from an STP re-convergence in an SCT, and adapts the network forwarding state accordingly. We systematically emulate 4 link failure scenarios between links (5,1) and (1,2) by disabling the respective source ports of each directed link. Host *A* initiates an iperf session over switch 2 to host *D*. After 10 seconds into the experiment, a link failure is induced, triggering an STP re-convergence. The resulting BDPU updates are observed by the controller and connectivity to host *D* is restored over switch 4. Figure 4a shows the elapsed time between the last received segment and first retransmitted packet over 10 repetitions and demonstrates how Panopticon quickly restores reachability after the failure



(a) SCT link failure recovery



(b) Bittorrent performance

Figure 4: Testbed experiments (a) Panopticon recovers from link failure within seconds. (b) Panopticon enables path diversity but also increases load on some links.

event. Interestingly, we observe that Panopticon reacts faster to link changes detected via STP reconvergence (*e.g.*, `sw5_to_sw1`) than to link changes at the OpenFlow switches themselves (`sw1_to_sw2`), which we observe to be breifly, internally delayed.

We next conduct a second experiment to explore how the SCT impacts the performance of a BitTorrent file transfer conducted among the hosts attached to SDNc ports. In this experiment, we begin by seeding a 100MB file at one host (*A* through *F*), in an iterative fashion. All other hosts are then initialized to begin simultaneously downloading the file from the seeder and amongst one another. We repeat each transfer 10 times, and measure the time for each host to complete the transfer. We then compare each time with an identical transfer in a L2 spanning tree topology. Figure 4b, illustrates that some of the hosts (*i.e.*, *A* and *D*) are able to leverage the multipath forwarding of their SCTs to finish sooner. Others, *e.g.*, *B* and *C* experience longer transfer times, as their traffic shares the same link to their frontier switch.

## 4 Incremental SDN Deployment

Panopticon makes no assumptions about the number of SDN switches or their locations in a partial SDN deployment. However, under practical resource constraints, an arbitrary deployment may make the *feasibility* of the logical SDN abstraction untenable, as the flow table capacities at the SDN switches and the availability of VLAN IDs on legacy switches are limited.

Beyond feasibility, the SDN deployment also influences network *performance*. By ensuring Waypoint Enforcement, SDN switches may become choke points that increase path lengths and link loads, in some cases beyond admissible values. Deployment planning therefore becomes a necessity.

### 4.1 Deployment Planning

Deciding the number and location of SDN switches to deploy can be viewed as an optimization problem wherein the objective is to yield a good trade-off between performance and costs subject to feasibility. We envision that a tool with configurable parameters and optimized algorithms may assist operators in planning the deployment by answering questions such as "What is the minimal number of SDN switches needed to support all ports as SDNc ports?" or "Which switches should be first upgraded to SDN to reduce bottleneck link loads?"

In a companion technical report of this paper [9], we present a general integer programming algorithm to compute a partial SDN deployment optimized for different objective functions and resource constraints. This algorithm can assist operators in upgrading the network, starting from a legacy network or one that is already partially upgraded.

We observe however that specific objectives and constraints for planning an SDN deployment are likely to depend on practical contextual factors such as hardware life-cycle management, support contracts and SLAs, long-term demand evolution and more. Unfortunately, these factors are rather qualitative, vary across environments, and are hard to generalize.

Instead, we reason more generally about how deployment choices influence feasibility and performance of our approach. To navigate the deployment problem space without the need to account for all contextual factors, we focus on a few general properties of desirable solutions:

$(i)$ *Waypoint Enforcement:* Every path to or from an SDNc port must traverse at least one SDN switch.

$(ii)$ *Feasible:* SDN switches must have sufficient forwarding state to support all traffic policies they must enforce. VLAN requirements to realize SCTs must be within limits.

$(iii)$ *Efficient:* The resulting traffic flow allocations should be efficient. We reason about efficiency using two metrics: The first metric is the path *stretch*, which we define for a given path $(s,t)$ as the ratio between the length of the path under Waypoint Enforcement and the length of the shortest path in the underlying network. The second metric is the expected maximum load on any link.

## 4.2 Simulation-assisted Study

To explore Panopticon feasibility and efficiency, we simulate different partial SDN deployment scenarios using a large campus network topology under different resource constraints and traffic conditions. These simulations let us $(i)$ evaluate the feasibility space of our architecture $(ii)$ explore the extent to which SDN control extends to the entire network, and $(iii)$ understand the impact of partial SDN deployment on link utilization and path stretch.

| Site | Access/Dist/Core | max/avg/min degree |
|---|---|---|
| *LARGE* | 1296 / 412 / 3 | 53 / 2.58 / 1 |
| *EMULATED* | 489 / 77 / 1 | 30 / 6.3 / 1 |
| *MEDIUM* | – / 54 / 3 | 19 / 1.05 / 1 |
| *SMALL* | – / 14 / 2 | 15 / 3 / 2 |

Table 1: Topology Characteristics of our Evaluation

### 4.2.1 Methodology

To simulate Panopticon deployment, we first choose network topologies with associated traffic estimates and resource constraints.

**Topologies.** Detailed topological information, including device-level configurations, link capacities, and end-host placements is difficult to obtain for sizeable networks: operators are reluctant to share these details due to privacy concerns. Hence, we leverage several publicly available enterprise network topologies [34, 37] and the topology of a private, local large-scale campus network. The topologies range from SMALL, comprising just the enterprise network backbone, to a MEDIUM network with 54 distribution switches, to a comprehensive large-scale campus topology derived from anonymized device-level configurations of 1711 L2 and L3 switches. Summary information on the topologies is given in Table 1. Every link in each topology is annotated with its respective capacity. We treat port-channels (bundled links), as a single link of its aggregate capacity.

Simulation results on the SMALL and MEDIUM network gave us early confidence in our approach, however their limited size does not clearly demonstrate the most interesting design trade-offs. Thus, we only present simulation results for LARGE.

**Focus on distribution switches.** In our approach, we distinguish between *access switches*, *distribution switches*, and *core switches*. Access switches are identified both topologically, as well as from device-level configuration metadata. Core switches are identified as multi-chassis devices, running a L3 routing protocol. Due to their topological location, SCTconstruction to core switches becomes challenging, thus, we focus on distribution switches (in the following referred to as the *candidate set* for the upgrade). In case of the LARGE network, this candidate set has cardinality 412 of which, 95 devices are identified as L3 switches (running OSPF or EIGRP). Within this distribution network, we reason about legacy distribution-layer switchports as candidates to realize as SDNc ports, subject to Waypoint Enforcement. Each distribution-layer switchport leads to an individual access-layer switch to which end-hosts are attached. Thus, we identify 1296 candidate SDNc ports.

**Traffic estimates.** We use a methodology similar to that applied in SEATTLE [21] to generate a traffic matrix based on packet-level traces from an enterprise campus network, the *Lawrence Berkeley National Labora-*

*tory* (LBNL) [26]. The LBNL dataset contains more than 100 hours of anonymized packet level traces of activity of several thousand internal hosts. The traces were collected by sampling all internal switchports periodically. We aggregate the recorded traffic according to source-destination pairs and for each sample, we estimate the load imposed on the network. We note that the data contains sources from 22 subnets.

To project the load onto our topologies, we use the subnet information from the traces to partition each of our topologies into subnets as well. Each of these subnets contains at least one distribution switch. In addition, we pick one node as the Internet gateway. We associate traffic from each subnet of the LBNL network in random round-robin fashion to candidate SDNc ports. All traffic within the LBNL network is aggregated to produce the intra-network traffic matrix. All destinations outside of the LBNL network are assumed to be reachable via the Internet gateway and, thus mapped to the designated gateway node. By running 10 different random port assignments for every set of parameters, we generate different traffic matrices which we use in our simulations. Still, before using a traffic matrix we ensure that the topology is able to support it. For this purpose we project the load on the topology using shortest path routes, and scale it conservatively, such that the most utilized gigabit link is at 50% of its nominal link capacity.

**Resource constraints.** Most mid- to high-end enterprise network switches support 512-1024 VLAN IDs for simultaneous use. The maximum number of VLAN IDs expressible in 802.1Q is 4096. Accordingly, we focus on simulating scenarios where legacy switches support at most 256, 512, and 1024 simultaneous VLAN IDs. While first generation OpenFlow capable switches were limited to around 1k flow table entries many current switches readily support from 10k to 100k entries for exact and wild-card matching. Bleeding edge devices support up to 1M flow table entries [25]. To narrow our parameter space, we fix the flow table capacity of our SDN switches to 100k entries, and vary the average number of rules required to realize policy for a single SDNc port from 10 to 20. We furthermore ensure that every SDN switch maintains at all times both policy and basic forwarding state (one entry per SDNc port) to ensure all-to-all reachability in the absence of any policy. We note, this is a conservative setting; by comparison, if flow table entries were kept only in the temporal presence of their respective, active source-destination traffic in the LBNL dataset, the maximum number of entries would never exceed 1,200 flows/s [5].

### 4.2.2 Switch Deployment Strategies

Given our topology and traffic estimates, we next explore how SDN switch deployment influences feasibility and performance. As this problem is NP-Complete, we devise a simple, but effective deployment heuristic inspired by classical techniques such as Facility Location, called VOL. We have also developed an optimal Integer Programming formulation which we provide in our companion technical report [9].

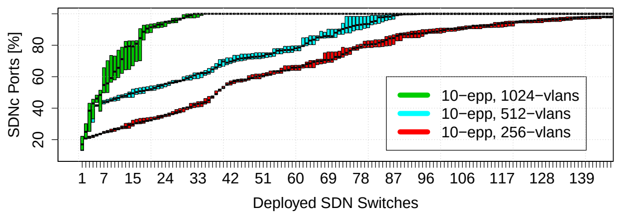
**VOL** iteratively selects one legacy switch to be replaced at a time, in decreasing order of switch egress traffic volume. SDNc candidate ports are then accommodated in the following greedy fashion: SDNc ports from the previous iteration are accommodated first. An SCT is constructed to the active frontier, whose size, chosen by the designer, defines a feasibility-efficiency trade-off we investigate later. If an SCT can be created, designated SDN switches from the active frontier are selected for each destination port respectively, and flow table entries are allocated. If flow table policy is accommodated, the traffic matrix is consulted and traffic is projected from the candidate port to every destination along each waypoint-enforced path. When no link exceeds its maximum utilization (or safety threshold value), the port is considered SDNc. The remaining SDNc candidates are then tried and thereafter, the next SDN switch candidate is deployed and the process repeats. As VOL is a greedy algorithm and does not backtrack, it may terminate prior to satisfying all SDNc candidates, despite the existence of a feasible solution.

For comparison, **RAND**, iteratively picks a legacy switch uniformly at random, subject to the same SDNc candidate VLAN, flow table, and link utilization constraint satisfaction. RAND allows us to evaluate the sensitivity of the solution to the parameters we consider and the potential for sophisticated optimizations to outperform naïve approaches. We repeat every RAND experiment with 10 different random seeds.

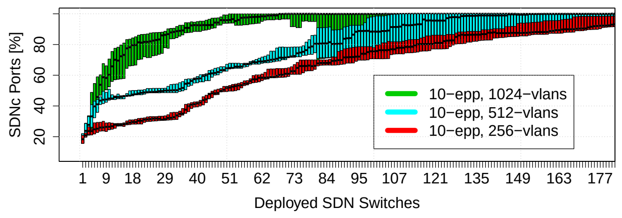
### 4.2.3 SDNc ports vs. deployment strategy

As Panopticon is designed to enable a broad spectrum of partial SDN deployment, we begin our evaluation by asking, "As a deployment grows, what fraction of candidate SDNc ports can be accommodated, under varying resource constraints?"

**Scenario 1:** To answer this question, we choose three values for the number of maximum simultaneous VLANs supported on any legacy switch (256, 512, 1024). We choose a policy requirement of 10 flow table entries on average for every (SDNc, destination port) pair as defined in the traffic matrix, so as to avoid a policy state bottleneck. We reason that policy state resource bottlenecks can be avoided by the operator by defining



(a) VOL Switch Selection Strategy



(b) Random Switch Selection Strategy

Figure 5: Percentage of SDNc ports as a function of deployed SDN switches, under different vlan availability. When more VLANs are supported by legacy devices, more SDNc ports can be realized with fewer SDN switches.

worst-case policy state needs in advance and then deploying SDN switches with suitable flow table capacity. We then compare our two deployment strategies VOL and RAND for different numbers of deployed SDN switches depicted by Figure 5 in which repeated experimental runs are aggregated into boxplots.

**Observations 1:** Figure 5 illustrates that the ability to accommodate more SDNc ports with a small number of SDN switches depends largely on the number of VLAN tags supported for use by the legacy hardware. Under favorable conditions with 1024 VLANs, 100% SDNc port coverage can be had for as few as 33 SDN switches. VLAN tag availability is necessary to construct SCTs and in Fig. 5a we see that when legacy switches support at most 256 VLANs, over 140 SDN switches must be deployed before achieving full SDNc port coverage. Fig. 5b shows the importance of choosing where to deploy SDN switches, as the earliest 100% SDNc feasible solution requires 20 additional SDN switch over VOL.

### 4.2.4 How will Panopticon affect my traffic?

We next ask: "As more SDNc ports are accommodated, what will waypoint-enforcement do to the traffic?"

**Scenario 2:** To answer this question, we evaluate the metrics path stretch and link utilization as we increase the SDN deployment, subject to two different VLAN resource constraints. As in Scenario 1, we assume average policy requirement of 10 flow table entries for every (SDNc, destination port) pair. Recall that our methodology scales up the baseline traffic matrix to ensure that the most utilized link in the original network is 50% utilized.

Figure 6 plots the relationship between the percentage of accommodated SDNc ports, the maximum link utilization, and the 90th percentile link utilization path stretch. Median values are shown for all metrics, across the repeated experiments. The feasible regions of each full "logical SDN" deployment with respect to all resource constraints are indicated by the vertical bar.

**Observations 2:** Figure 6a indicates that with 512 VLANs usable in the legacy network, a full logical SDN becomes feasible with 95 switches where the most utilized link reaching 55% of its capacity. 90th Percentile path stretch hovers around 2.1. As further switches are upgraded, the stretch and relative link utilization continue to improve. A more optimistic case is depicted in Figure 6c where full logical SDN is achieved with 33 switches. However, given fewer SDN waypoints, the maximum link utilization is higher at 60%. The key takeaway from this plot is that given conservative base link utilization, the additional burden imposed by SDN waypoint enforcement is small in many deployments.

### 4.2.5 Efficient 100% SDNc Port Feasibility

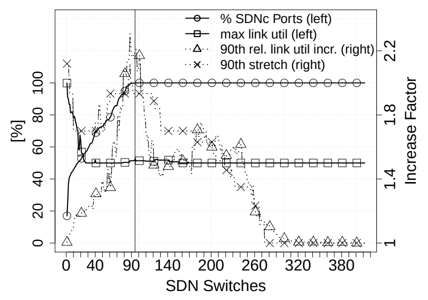
As we point out in our architecture section, Panopticon allows the designer to make efficiency trade-offs, where a full logical SDN can be realized with fewer SDN switches, at the expense of higher link utilization and path stretch. The parameter that governs this trade-off is the active frontier size. We next look to Figs. 6a and 6b which illustrate how this trade-off plays out.

Recall from Fig. 6a that for a legacy network supporting 512 VLANs, a full logical SDN becomes feasible with about 95 SDN switches when the number of available frontier switches is large (32). Each one of the paths to the 32 frontier switches consumes a VLAN however, which blocks other SDNc candidate ports later on. By limiting the active frontier to at most 2 switches, Fig. 6b illustrates that a feasible solution can be achieved with 39 fewer switches. The path stretch notably increases to a factor of 2.4, compared to less than 2 when a larger frontier is used. This trade-off underlines the flexibility of Panopticon: Operators can make design choices tailored to their individual network performance requirements.

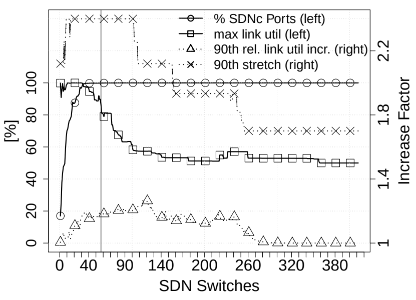
## 4.3 Traffic Emulation Study

To compliment our simulation-based approach and further investigate the consequences of Panopticon on traffic, we conduct a series of emulation-based experiments on portions of a real enterprise network topology. These experiments $(i)$ provide insights into the consequences of waypoint enforcement on TCP flow performance and $(ii)$ let us explore the extent to which the deployment size impacts TCP flow performance when every access point is operated as an SDNc port.

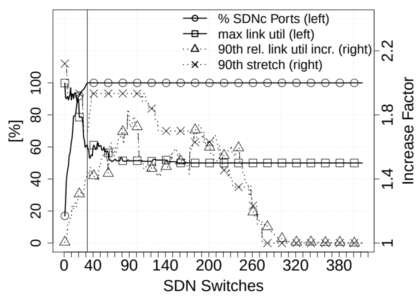
**Setup.** We use Mininet [15] to emulate a Panopticon



(a) 512 VLANs, 32 Active Frontier



(b) 512 VLANs, 2 Active Frontier



(c) 1024 VLANs, 32 Active Frontier

Figure 6: SDN deployment vs. max link utilization, 90th percentile path stretch and relative link util increase. Feasible 100% SDNc port coverage can be realized with 33 SDN switches, with acceptable link utilization and path stretch.

deployment. Due to the challenges of emulating a large network [15], we scale down key aspects of the network characteristics of the emulation environment. We (*i*) use a smaller topology, EMULATED, from Table 1 – a 567-node sub-graph of the LARGE topology – by pruning the graph along subnet boundaries, (*ii*) scale down the link capacities by 2 orders of magnitude, and (*iii*) correspondingly reduce the TCP MSS to 536 bytes to reduce packet sizes in concert with the reduced link capacities. This allows us to avoid system resource bottlenecks which otherwise interfere with traffic generation and packet forwarding, thus influencing measured TCP throughput.

We run our experiments on a 64-core at 2.6GHz AMD Opteron 6276 system with 512GB of RAM running the 3.5.0-45-generic #68 Ubuntu Linux kernel using OpenVSwitch version 2.1.90. Baseline throughput tests indicate that our system is capable of both generating and forwarding traffic of 489 simultaneous TCP connections in excess of 34Gbps, sufficiently saturating the aggregate emulated link capacity of every traffic sender in our experiments. We note that traffic in the subsequent experiments is generated on the system-under-test itself.

Thus, our emulation experiments involve 489 SDNc ports located at "access switches" at which traffic is sent into and received from the network. The distribution network consists of 77 switches and routers, comprising a L2/L3 network in which 28 devices are identified as IP router gateways, bridging Ethernet broadcast domains over the remainder of the switches. Within each Ethernet broadcast domain, we introduce a single spanning tree to break forwarding loops.

**Traffic.** We apply a traffic workload to our emulated network based on (*i*) a traffic matrix, defined over the 489 SDNc ports, and (*ii*) a synthetically generated flow size distribution where individual TCP flow sizes are obtained from a Weibull distribution with shape and scaling factor of 1, given in Table 2.

We re-use the traffic matrix used in the simulations to define the set of communicating source-destination pairs of SDNc ports in the network. For system scalability reasons, we limit the number of source-destination pairs to 1024, selected randomly from the traffic matrix. For each pair of SDNc ports, we define a sequence of TCP connections to be established in iterative fashion, whose transfer sizes are determined by the aforementioned Weibull distribution. The total traffic volume exchanged between each pair is limited to 100MB. When the experiment begins, every source-destination pair, in parallel begins to iterate through its respective connection sequence. Once every traffic source has reached its 100MB limit, the experiment stops.

| | min | median | avg | max |
|---|---|---|---|---|
| *Flow Sizes (in MB)* | 0.00005 | 6.91 | 9.94 | 101.70 |
| *Path Stretch A* | 1.0 | 1.33 | 1.25 | 3.0 |
| *Path Stretch B* | 1.0 | 1.0 | 1.16 | 3.0 |
| *Path Stretch C* | 1.0 | 1.0 | 1.002 | 1.67 |

Table 2: Traffic parameter and path stretch statistics.

**Scenarios.** We consider three deployment scenarios in which we evaluate the effects of Panopticon on TCP traffic: Scenario *A* in which 28 switches out of the 77 distribution switches are operated as SDN switches, and scenarios *B* and *C* which narrow down the number of SDN switches in *A* to 10 and 5 SDN switches respectively. SDN switch locations are selected at random based on location of IP routers, identified from the topology dataset.

**Results.** In each scenario, we compare TCP flow throughput in the Panopticon deployment versus the original network topology (which uses shortest-path IP routes with minimum cost spanning trees). Table 2 lists path stretch statistics for each scenario, namely, the ratio of SDN (waypoint-enforced) to legacy path length for every source-destination pair in the network.

Figure 7 illustrates the impact of waypoint enforcement on TCP performance in the three scenarios. The first observation we make is that in scenario *A*, when the 28 IP routers are replaced with SDN switches, the impact on median TCP throughput is negligible. This is perhaps expected, as all traffic across subnets must traverse some IP router in the legacy network, regardless. Some flows experience congestion due to waypoint enforcement. Other flows actually experience a performance in-

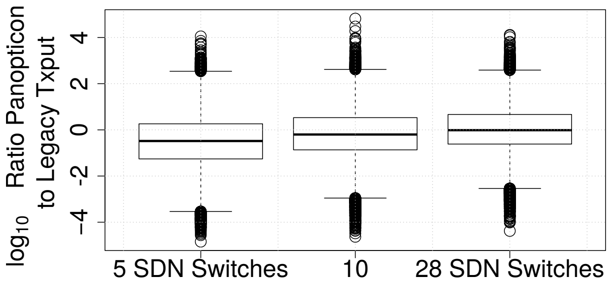


Figure 7: In both scenarios *A* and *B* (28 and 10 SDN switches), the median throughput over all experiments remains close to the performance of the legacy network.

crease due to the availability of multiple alternate paths in Panopticon. As the SDN deployment shrinks to more conservative sizes in scenarios *B* and *C*, the effects of waypoint enforcement becomes more prominent, supporting our observed simulation results.

## 4.4 Discussion

**Scalability.** As the number of SDNc candidates increases, the resource demands grow as well. We believe that one or two SDNc ports for every access switch however is a reasonable starting point for most partial SDN deployments. Even at one SDNc per access switch, a reasonable level of policy granularity, as end-hosts connected to the same physical access switch are often considered to be part of the same administrative unit as far as policy-specification is concerned. Should finer-grained SDNc port allocation be necessary, an approach such as "protected switchports" may be leveraged to extend waypoint enforcement to individual access-switch ports without the need for additional SCTs.

**Why fully-deploy SDN in enterprise?** Perhaps many enterprise networks do not need to fully deploy SDN. As our results indicate, it is a question of the trade-offs between performance requirements and resource constraint satisfaction. Our Panopticon evaluation suggests that partial deployment may in-fact be the right long-term approach for some enterprise networks.

## 5 Related Work

Our approach toward a scalable, incrementally deployable network architecture that integrates legacy and SDN switches to expose the abstraction of a logical SDN both builds upon and complements previous research.

**SDN.** In the enterprise, *SANE* [6] and *Ethane* [5] propose architectures to enforce centrally defined, fine-grained network policy. Ethane overcomes SANE [6]'s deployment challenges by enabling legacy device compatibility. Ethane's integration with the existing deployment is however, ad-hoc and the behavior of legacy devices falls out of Ethane's control. Panopticon by contrast, can guarantee SDN policy enforcement through principled interaction with legacy devices to forward traffic along "safe paths". Google's transition to a software-defined WAN involves an overhaul of their entire switching hardware to improve network performance [11]. In contrast to their goals, we take an explicit stance at transitioning to an SDN control plane without the need for a complete hardware upgrade. Considering a partial SDN deployment, Agarwal *et al.* [3] demonstrate effective traffic engineering of traffic that crosses at least one SDN switch. Panopticon is an architecture that enforces this condition for all SDNc ports.

**Enterprise network design and architecture.** Scalability issues in large enterprise networks are typically addressed by building a network out of several (V)LANs interconnected via L3 routers [8, 17]. *TRILL* [27] is an IETF Standard for so-called *RBridges* that combine bridges and routers. Although TRILL can be deployed incrementally, we are not aware of any work regarding its use for policy enforcement in enterprise networks.

Sun *et al.* [33] and Sung *et al.* [34] propose a systematic redesign of *enterprise networks* using parsimonious VLAN allocation to ensure reachability and provide isolation. These works focus on legacy networks only. The *SEATTLE* [21] network architecture uses a one-hop DHT host location lookup service to scale large enterprise Ethernet networks. However, such clean-slate approach is not applicable for the transitional networks we consider.

**Scalable data-center network architectures.** There is a wealth of recent work towards improving data-center network scalability. To name a few, Al-Fares *et al.* [4], *VL2* [14], *PortLand* [10], *NetLord* [24], *PAST* [32] and *Jellyfish* [31], offer scalable alternatives to classic data-center architectures at lower costs. As clean-slate architectures, these approaches are less applicable to transitional enterprise networks, which exhibit less homogeneous structure and grow "organically" over time.

**Evolvable inter-networking.** The question of how to *evolve* or run a *transitional* network, predates SDN and has been discussed in many contexts, including Active Networks [36]. Generally, changes in the network layer typically pose a strain to network evolution, which lead to overlay approaches being pursued (*e.g.*, [16, 35]). In this sense, the concept of Waypoint Enforcement is grounded on previous experience.

## 6 Summary

SDN promises to ease network management through principled network orchestration. However, it is nearly impossible to fully upgrade an existing legacy network to an SDN in a single operation.

Accordingly, we have developed **Panopticon**, an enterprise network architecture realizing the benefits of a logical SDN control plane from a transitional network which combines legacy switches and routers and SDN switches. Our evaluation highlights that our approach can deeply extend SDN capabilities into existing legacy networks. By upgrading between 30 to 40 of the hundreds of distribution switches in a large campus network, it is possible to realize the network as an SDN, without violating reasonable resource constraints. Our results motivate the argument, that partial SDN deployment may indeed be an appropriate long-term operational strategy for enterprise networks. Going forward, we plan to expand our prototype implementation to serve end-users.

## 7 Acknowledgements

We wish to thank the network administrators and Rainer May for providing us insights into their operations. Thanks to Srini Seetharaman for providing inputs to our cost model. Thanks to Srinivas Narayana for giving us an early implementation of STP on NOX. We thank Ho-Cheung Ng, Julius Bachnick and Arno Töll who contributed to an early Panopticon prototype. Thanks to Odej Kao and Bjoern Lohrmann for providing cluster time and support. We wish to thank Jennifer Rexford, Fred Baker, Don Towsley, James Kempf, and Dave Meyer for early feedback, as well as our anonymous reviewers for their helpful feedback.

## References

[1] POX Controller. `http://noxrepo.org`.

[2] VMWare NSX. `http://bit.ly/1iQZzDj`.

[3] S. Agarwal, M. Kodialam, and T. Lakshman. Traffic engineering in software defined networks. In *INFOCOM*, 2013.

[4] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *SIGCOMM*, 2008.

[5] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. Ethane: Taking control of the enterprise. In *SIGCOMM*, 2007.

[6] M. Casado, T. Garfinkel, A. Akella, M. J. Freedman, D. Boneh, N. McKeown, and S. Shenker. SANE: a protection architecture for enterprise networks. In *USENIX Security Symposium*, 2006.

[7] M. Casado, T. Koponen, S. Shenker, and A. Tootoonchian. Fabric: a retrospective on evolving SDN. In *HotSDN*, 2012.

[8] Cisco. Campus Network for High Availability Design Guide, 2008. `http://bit.ly/1ffWkzT`.

[9] D. L. et al. Panopticon: Reaping the benefits of partial sdn deployment in enterprise networks. Technical report, TU Berlin `http://bit.ly/1n1U3LD`, 2013.

[10] N. M. et. al. PortLand: a scalable fault-tolerant layer 2 data center network fabric. In *SIGCOMM*, 2009.

[11] S. J. et al. B4: Experience with a Globally-Deployed Software Defined WAN. In *SIGCOMM*, 2013.

[12] T. K. et al. Onix: A distributed control platform for large-scale production networks. In *OSDI*, 2010.

[13] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker. Frenetic: A network programming language. In *ICFP*, 2011.

[14] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. VL2: A scalable and flexible data center network. In *SIGCOMM*, 2009.

[15] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown. Reproducible network experiments using container-based emulation. CoNEXT, 2012.

[16] D. Joseph, J. Kannan, A. Kubota, K. Lakshminarayanan, I. Stoica, and K. Wehrle. OCALA: An architecture for supporting legacy applications over overlays. In *NSDI*, 2006.

[17] Juniper. Campus Networks Reference Architecture, 2010. `http://juni.pr/1iR0vaZ`.

[18] N. Kang, Z. Liu, J. Rexford, and D. Walker. Optimizing the "one big switch" abstraction in software-defined networks. In *CoNEXT*, 2013.

[19] P. Kazemian, G. Varghese, and N. McKeown. Header space analysis: static checking for networks. In *NSDI*, 2012.

[20] A. Khurshid, X. Zou, W. Zhou, M. Caesar, and P. B. Godfrey. Veriflow: Verifying network-wide invariants in real time. In *NSDI*, 2013.

[21] C. Kim, M. Caesar, and J. Rexford. Floodless in Seattle: A scalable ethernet architecture for large enterprises. In *SIGCOMM*, 2008.

[22] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: enabling innovation in campus networks. *SIGCOMM CCR*, 38(2), 2008.

[23] C. Monsanto, J. Reich, N. Foster, J. Rexford, and D. Walker. Composing Software Defined Networks. In *NSDI*, 2013.

[24] J. Mudigonda, P. Yalagandula, J. Mogul, B. Stiekes, and Y. Pouffary. NetLord: a scalable multi-tenant network architecture for virtualized datacenters. In *SIGCOMM*, 2011.

[25] NoviFlow. 1248 Datasheet. `http://bit.ly/1baQd0A`.

[26] R. Pang, M. Allman, M. Bennett, J. Lee, V. Paxson, and B. Tierney. A first look at modern enterprise traffic. In *ACM IMC*, 2005.

[27] R. Perlman, D. Eastlake, D. G. Dutt, S. Gai, and A. Ghanwani. Rbridges: Base protocol specification. In *IETF*, 2009.

[28] Z. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu. SIMPLE-fying Middlebox Policy Enforcement Using SDN. In *SIGCOMM*, 2013.

[29] M. Reitblatt, M. Canini, A. Guha, and N. Foster. FatTire: Declarative Fault Tolerance for Software-Defined Networks. In *HotSDN*, 2013.

[30] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker. Abstractions for network update. In *SIGCOMM*, 2012.

[31] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey. Jellyfish: networking data centers randomly. In *NSDI*, 2012.

[32] B. Stephens, A. Cox, W. Felter, C. Dixon, and J. Carter. PAST: Scalable Ethernet for data centers. In *CoNEXT*, 2012.

[33] X. Sun, Y. E. Sung, S. D. Krothapalli, and S. G. Rao. A systematic approach for evolving vlan designs. In *INFOCOM*, 2010.

[34] Y.-W. E. Sung, S. G. Rao, G. G. Xie, and D. A. Maltz. Towards systematic design of enterprise networks. In *CoNEXT*, 2008.

[35] N. Takahashi and J. M. Smith. Hybrid hierarchical overlay routing (hyho): Towards minimal overlay dilation. *IEICE*, 2004.

[36] D. J. Wetherall. Service introduction in an active network. Technical report, M.I.T. PhD Thesis, 1999.

[37] H. Zeng, P. Kazemian, G. Varghese, and N. McKeown. Automatic test packet generation. In *CoNEXT*, 2012.