DOCSDN: Dynamic and Optimal Configuration of Software-Defined Networks

♦ Timothy Curry, ♦ Devon Callahan, Benjamin Fuller, and Laurent Michel

University of Connecticut, Storrs CT 06268, USA
{timothy.curry,devon.callahan,benjamin.fuller,laurent.michel}@uconn.edu

Abstract. Networks are designed with functionality, security, performance, and cost in mind. Tools exist to check or optimize individual properties of a network. These properties may conflict, so it is not always possible to run these tools in series to find a configuration that meets all requirements. This leads to network administrators manually searching for a configuration.

This need not be the case. In this paper, we introduce a layered framework for optimizing network configuration for functional and security requirements. Our framework is able to output configurations that meet reachability, bandwidth, and risk requirements. Each layer of our framework optimizes over a single property. A lower layer can constrain the search problem of a higher layer allowing the framework to converge on a joint solution.

Our approach has the most promise for software-defined networks which can easily reconfigure their logical configuration. Our approach is validated with experiments over the fat tree topology, which is commonly used in data center networks. Search terminates in between 1-5 minutes in experiments. Thus, our solution can propose new configurations for short term events such as defending against a focused network attack.

Keywords: Network Configuration · Software Defined Networking · Reachability · Constraint Programming · Optimization

1 Introduction

Network configuration is a crucial task in any enterprise. Administrators balance functionality, performance, security, cost, and other industry specific requirements. The resulting configuration is subject to periodic analysis and redesign due to red team recommendations, emerging threats, and changing priorities. Tools assist administrators with this complex task: existing work assesses network reachability [27], wireless conflicts [40], network security risk [46, 52], and load balancing [48,51]. These tools assess the quality of a potential configuration. Unfortunately, current tools suffer from three limitations:

1. Tools assess whether a single property is satisfied, making no recommendation if the property is not satisfied. This leaves IT personnel with the task of deciding how to change the network.

2. Tools assess networks with respect to an individual goal at a time. This means a change to satisfy a single property may break another property. There is no guidance for personnel on how to design a network that meets the complex and often conflicting network requirements.
3. Tools do not react to changing external information such as the publication of a new security vulnerability.

Our Contribution This work introduces a new optimization framework that finds network configurations that satisfy multiple (conflicting) requirements. We focus on data center networks (DCN) that use software defined networking (SDN). Background on these settings is in Section 2. Our framework is called DocSDN (Dynamic and Optimal Configuration of Software-Defined Networks).

DocSDN searches for network configurations that simultaneously satisfy multiple properties. DocSDN is organized into layers that consider different properties. The core of DocSDN is a multistage optimization that decouples search on "orthogonal" concerns. The majority of the technical work is to effectively separate concerns so the optimization problems remain tractable. Our framework is designed to continually produce network configurations based on changing requirements and threats. It frees IT personnel from the complex question of how to satisfy multiple requirements and can quickly incorporate new threat information.

DocSDN focuses on achieving functional requirements (such as network reachability and flow satisfaction) and limiting security risk (such as isolating high risk nodes and nodes under denial of service attack). Naturally, other layers such as performance or cost can be incorporated. The search for a good configuration could be organized in many ways. State-of-the-art approaches assess different properties in isolation, frustrating search for a solution that satisfies all requirements. Ideally, a framework should search for a configuration that simultaneously satisfies all requirements. This extreme is unlikely to be tractable on all but the smallest networks. DocSDN mediates between these approaches separating the functional and security search problems but introducing a feedback loop between the two search problems based on *cuts*.

In the proposed organization the functional layer is "above" the security layer. Through the feedback loop, the security layer describes a problematic part of the network to the functional layer. The functional layer then refines its model and searches for a functional configuration that satisfies an additional *constraint*. This has the effect of blocking the problematic part of the configuration. Currently, the feedback signal is a pair of nodes that should not be proximate in the network. After multiple iterations the two layers jointly produce a solution that optimizes the SDN configuration both with respect to functionality and security risks.

DocSDN provides solutions of improving quality before the final solution. Thus, the network can be reconfigured once the objective improves on the current configuration by a large enough amount (to justify the cost/impact of reconfiguration).

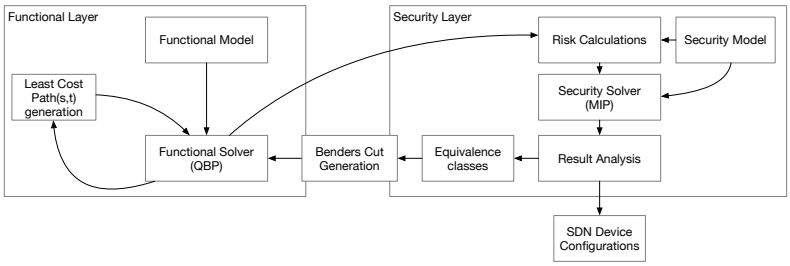

Fig. 1. DocSDN Framework. A layered decomposition that breaks down configuration synthesis into functional and security layers.

The underlying optimization problems are NP-hard but optimization technology has seen tremendous advances in performance during the past few decades. Since 1991, mathematical programming solvers have delivered speedups of 11 orders of magnitude [8,17]. Hybrid techniques such as Benders decomposition [6,12, 20,21] and column generation [3,19,31] (aka, Dantzig-Wolfe decomposition [14]) made it possible to solve huge problems thanks to on-demand generation of macroscopic variables and the dynamic addition of critical constraints. Large Neighborhood Search [47] further contributed to delivering high-quality solution within constrained time budgets.

These techniques are beginning to see adoption in network security. Yu et al. recently applied stochastic optimization with Bender's decomposition to assess network risk under uncertainty for IoT devices [52]. They used Bender's decomposition on a scenario-based stochastic optimization model to produce a parent problem that chooses a deployment plan while children are concerned with *choosing* the optimal nodes to serve the demands in individual scenarios. In comparison, our approach addresses *both functional and security requirements*. It relies on Bender's cuts from the security layer (child) to rule out vulnerable functional plans whose routing paths fail to adequately minimize risks and maximize served clients. We now briefly describe the framework (a formal description is in Section 3) and present an illustrative example.

Overview of DocSDN Figure 1 presents an overview of the framework. The functional layer takes as input a *Functional Model* that describes the network including the physical topology, capacity, the allowable communication patterns and the demand requirements. Network reachability begins with a priming procedure that generates the k-least cost paths to the optimizer for each source/destination pair in the demand requirements. The objective for the functional layer is to find a logical topology (a collection of routed paths) that meets all demand requirements while favoring shorter length routing paths and load balancing. The program is formulated as quadratic binary program (QBP). The solution as determined by the functional layer is passed to the security layer.

The output of the functional layer and a *security model* are the input for the security layer. The current configuration is fed to a module that uses risk assessments for the individual network devices (obtained for example using a

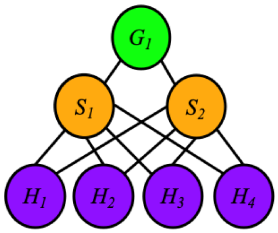

Fig. 2. Toy network example with a single gateway device G_1, two intermediate switches S_1 and S_2 and four hosts. We assume the switches are physically connected to all hosts.

vulnerability database) to assess the overall risk of the entire configuration. In our current implementation this risk calculation is based on a simple risk propagation model where a path's risk is based on the risk of nodes on the path and close to the source and sink. The security layer can deploy firewalls and deep packet inspection as network defenses. Since these mechanisms affect route capacity, the security layer has a dual objective function: 1) maximizing the functional objective and 2) minimizing security risk. The security objective is formulated as a mixed integer program (*MIP*). When the security search completes, it proposes nodes to the functional layer that should be separated. As an example, a high value node with low risk may be placed in a different (virtual) LAN than a high risk node. These *Benders cuts* are designed to entice a better logical topology from a subsequent iteration in the functional layer. This feedback loop between the two layers can iterate multiple times. When no further cuts are available, the overall output is a set of configuration rules.

An example configuration This section describes an application of our framework to automatically respond to a distributed denial of service (DDoS) attack. Current DDoS attacks demonstrate peak volume of 1 Tbps [29]. Many DDoS defense techniques require changes to the network behavior by rate limiting, filtering, or reconfiguring the network (see [23,24,37,43,53]). Recent techniques [15] leverage SDNs to react to DDoS attacks in a dynamic and flexible manner. We show how such a response would work in our framework using a toy network illustrated in Figure 2. A more realistic network and the framework's response are described in Section 5. We stress that DDoS attacks are often short in timescale making human diagnosis and reaction costly or impractical. Consider a focused DDoS attack against a number of services in an enterprise but not the entirety of its publicly accessible address space. (The Great Cannon's attack against GreatFire targeted two specific Github repositories [35].) We assume a service hosted by H_1 is targeted, while services on H_2, H_3 and H_4 are not.

Recall, the functional layer establishes a logical topology (forwarding rules) while the security layer adds network defenses (packet inspection modules and firewall rules). We elide how the attack is detected and assume it increases the risk score for H_1 in the security model.

The first iteration The functional layer proposes a candidate configuration where G_1 routes all traffic intended for H_1 and H_2 to S_1 which then forwards the traffic and G_1 routes traffic intended for H_3 and H_4 to S_2 which then forwards the traffic. This is the first candidate solution presented to the security layer.

Since H_1 is high risk the security layer proposes a firewall at S_1 to block all port 80 traffic. This reduces risk at the cost of blocking all traffic to H_2. Of course, in real firewalls more fine-grained rules are possible, this simplified example is meant to illustrate a case where collateral damage to the functional objective is necessary to achieve the security objective. Since traffic is being blocked to a node with low risk, the security layer asks the functional layer to separate H_1 and H_2 so H_2 does not suffer.

Repeated iterations The functional layer now has a constraint that H_1 and H_2 should not be collocated in the network. As such, it proposes a new configuration with H_1 and H_3 under S_1 and H_2 and H_4 under S_2. This is then sent to the security layer. The security layer makes a similar assessment and proposes a firewall rule at S_2, finds this recommendation hurts functionality and requests separation of H_1 and H_3.

This process repeats with the functional layer proposing to collocate H_1 and H_4. The security layer similarly asks to separate H_1 and H_4. Finally, H_1 is segregated from all other nodes. This produces a configuration where H_1 is the only child of S_1. Note that having H_2, H_3 and H_4 under a single switch may hurt performance but the effect is less than blocking traffic to one of the nodes entirely. DocSDN can then output the candidate solution as high level SDN fragments (using a high-level language like Frenetic [16]).

Recovery Importantly, when the DDoS abates, DocSDN automatically reruns with a changed risk for H_1, outputting a binary tree.

Organization The rest of the work is organized as follows: Section 2 provides background on our application and discusses related work, Section 3 describes our framework and accompanying optimization models, Section 4 describes our experimental setup, Section 5 evaluates the framework and finally Section 6 concludes.

2 Background and Related Work

Data Center Networks (DCN) host, process and analyze data in financial, entertainment, medical, government and military sectors. The services provided by DCNs must be reliable, accurate and timely. Services provided by DCNs (and the corresponding traffic) are heterogeneous. The network must adapt to changing priorities and requirements while protecting from emerging threats. They scale to thousands of servers linked through various interconnects. Protocols used for these services are split roughly 60 percent web (HTTP/HTPS) and 40 percent file storage (SMB/AFS) [7]. The interdependence of device configurations make modifying any single configuration difficult and possibly dangerous for network health. A seemingly simple update can cause significant collateral damage and unintended consequences.

Simultaneously, the network fabric is changing with the advent of Software Defined Networking (SDN) [30]. SDNs are flexible and programmable networks that can adapt to emergent functional or performance requirements. Openflow [36] is a common open source software stack. Researchers have proposed high-level languages and compilers [5, 16, 28, 44] that bridge the semantic gap between network administrators and the configuration languages used by SDN devices. These languages focus on *compositional* and *parametric* SDN software modules that execute specific micro-functions (e.g., packet forwarding, dropping, routing, etc.). The use of a high level language is prompted by a desire to be able to *select*, *instantiate* and *compose* SDN modules with guarantees.

Our framework is intended to be modular and allow integration of prior work on evaluating network configurations. As such there is a breadth of relevant work. Due to space constraints we focus on the most relevant works. In the conclusion we elaborate on the characteristics needed to integrate a prior assessment tool into our framework (see Section 6).

Measuring Network Risk Known threats against computer systems are maintained by governments and industry. Common Vulnerabilities and Exposures (CVE) is a publicly available dictionary including an identifier and description of known vulnerabilities [13], CVE does not provide a severity score or priority ranking for vulnerabilities. The US National Vulnerability Database (NVD) [41] is provided by the US National Institute of Standards and Technology (NIST). The NVD augments the CVE, adding severity scores and impact ratings for vulnerabilities in the CVE.

There are many mechanisms for measuring the security risk on a network [10, 25, 34, 49, 50]. Lippmann et al. present a network security model which computes risk based on a list of the most current threats [33]. This model implements a cycle of observe network state, compute risk, prioritize risk, and mitigate the risk.

This loop is often codified into an *attack graph* [22, 26, 46]. Attack graphs try to model the most likely paths that an attacker could use to penetrate a network. Attack Graphs often leverage one or more of the aforementioned vulnerability assessment tools as input, combined with a network topology and device software configurations to generate the graph. Current attack graph technologies provide recommendations to network administrators that effectively remove edges from the graph and trigger a re-evaluation of the utility for the attacker. To the best of our knowledge, current practice does not leverage network risk measurement into constraints used for the generation of new configurations.

Network Reachability The expansion of SDN has aided the applicability of formal verification to computer networks. Prior to SDN, the lack of clear separation between the data and control plain created an intractable problem when considering a network of any scale. Bounded model checking using SAT and SMT solvers [4, 54] can currently verify reachability properties in networks with several thousands of nodes.

Configuration Search Constraint Programming (CP) was introduced in the late 1980s [45] and is used for scheduling [2], routing, and configuration

problems. Large-scale optimization problems are often decomposed including Benders [12] and Dantzig-Wolfe [14]. *Soft constraints* or Lagrangian relaxation are used for over-constrained problems or when the problem is too computationally expensive. Stochastic optimization techniques have been used for many applications in resilience [9,39] and the underlying methodologies are a key part of this research. Prior work in configuration management with constraint programming [11, 32] focused on connectivity or security. We are not aware of any work that balances these two objectives in a meaningful way.

3 Implementation

Figure 1 outlines the overall structure of the DocSDN framework. layer interconnections as well as their internals. The functional layer uses a mathematical optimization model that is fed to a quadratic mixed boolean programming (QBP) solver alongside an initial set of least-cost paths to be considered to service the required flows. The security layer receives the topology chosen by the functional layer and a security model to solve, with a mixed-integer programming (MIP) solver, the risk minimization problem. The output can result in low-risk flows being blocked as a consequence of deploying firewalls to mitigate high-risk flows. A result analysis module then produces *equivalence classes* that are sent back to the functional layer to request the separation of specific flows that should not share paths, with the goal of minimizing the collateral damage to low-risk flows. These equivalence classes generate additional constraints, known as Bender's cuts, that are added to the functional solver for a new iteration. The remainder of this section describes the major modules in Figure 1.

3.1 Functional Layer

The mathematical optimization model in the functional layer is a quadratic mixed binary programming model. In constraint programming the four main components are Inputs, Variables, Constraints, and an Objective function. Inputs are below.

Inputs

$\mathcal{N}$ – the set of all network devices
$\mathcal{E}$ – the set of edges (pair of vertices) connecting network devices
$\mathcal{T}$ – the set of types of traffic to be routed
$\mathcal{F}$ – the set of $(s, t, T) \in \mathcal{N} \times \mathcal{N} \times \mathcal{T}$ tuples defining desired traffic flows of type T from source node s to sink node t.
$D(f) : \mathcal{F} \to \mathbb{R}$ – the actual demand for each flow $f \in \mathcal{F}$
$\mathcal{C} \subseteq 2^{\mathcal{N}}$ – a subset of sets of network devices
$\mathcal{R} \subseteq \mathcal{C} \times \mathcal{C}$ – pairs (c_1, c_2) of equivalence classes that segregate traffic from c_1 to c_2.
$\mathcal{P}$ – the set of all paths
$P(e) : \mathcal{E} \to \mathcal{P}$ – the set of all paths containing edge e
$P(n) : \mathcal{N} \to \mathcal{P}$ – the set of all paths containing node n

$P(c) : \mathcal{C} \to \mathcal{P}$ – the set of all paths containing a node in c
$N(p) : \mathcal{P} \to \mathcal{C}$ – the set of nodes appearing in path p
$P(s,t) : \mathcal{N} \times \mathcal{N} \to \mathcal{P}$ – the set of all paths $s \to t$
$cap(e) : \mathcal{E} \to \mathbb{R}$ – gives the capacity of an edge e.

Variables
$active_{p,T} \in \{0,1\}$, – for every path $p \in \mathcal{P}$ and traffic type $T \in \mathcal{T}$, indicates whether path p carries traffic of type T
$flow_{p,T} \in \mathbb{R}_{\geq 0}$ – for every path $p \in \mathcal{P}$ and traffic type $T \in \mathcal{T}$, amount of flow of type T that is sent along path p
$equiv_{c,n} \in \{0,1\}$ – does node $n \in \mathcal{N}$ appear in an active path together with a node in equivalence class c
$share_{c_1,c_2,n} \in \{0,1\}$ – indicates whether node $n \in \mathcal{N}$ appears on any active path with nodes in classes $c_1, c_2 \in \mathcal{C}$. Namely,

$$share_{c_1,c_2,n} \Leftrightarrow n \in \left(\left(\cup_{p \in P(c_1): active_{p,*}} N(p) \right) \cap \left(\cup_{p \in P(c_2): active_{p,*}} N(p) \right) \right)$$

$active_{p,*} = 1$ if there is a type $T \in \mathcal{T}$ where $active_{p,T} = 1$
$load_n \in \mathbb{R}$ – the amount of flow that goes through node n
$loadObj$ – the sum of squares of all $load_n$ variables.

Constraints

$$\sum_{p \in P(e), T \in \mathcal{T}} flow_{p,T} \leq cap(e), \ \ \forall e \in \mathcal{E} \tag{1}$$

$$\sum_{p \in P(s,t)} flow_{p,T} \geq D(s,t,T), \ \ \forall (s,t,T) \in \mathcal{F} \tag{2}$$

$$active_{p,T} = 1 \to flow_{p,T} \geq 1, \forall (s,t,T) \in \mathcal{F}, p \in P(s,t) \tag{3}$$

$$\sum_{p \in P(s,t)} active_{p,T} = 1, \ \ \forall (s,t,T) \in \mathcal{F} \tag{4}$$

$$equiv_{c,n} = \bigvee_{p \in P(n) \cap P(c)} (active_{p,T}), \forall T \in \mathcal{T}, n \in \mathcal{N}, c \in \mathcal{C} \tag{5}$$

$$share_{c_1,c_2,n} = equiv_{c_1,n} \wedge equiv_{c_2,n}, \forall n \in \mathcal{N}, (c_1,c_2) \in \mathcal{R} \tag{6}$$

$$load_n = \sum_{p \in P(n), T \in \mathcal{T}} flow_{p,T}, \forall n \in \mathcal{N} \tag{7}$$

Equation 1 enforces the edge capacity constraint to service the demand of all paths flowing through it. Equation 2 ensures that enough capacity is available to meet the demand of an (s,t,T) flow. Equation 3 ensures that some non-zero capacity is used if a specific path is activated (conversely, an inactive path can only have a 0 flow). Equation 4 states that a single path should be chosen to service a given flow $f \in \mathcal{F}$. Equations 5 define the auxiliary variables $equiv_{c,n}$ as true if and only if node $n \in \mathcal{N}$ appears on an active path sharing a node with the equivalence class $c \in \mathcal{C}$. Equation 6 defines an active path that shares at

least one node with two classes. Finally, equation 7 defines the load of a node as the sum of the flows associated to active paths passing through node n.

Objective

$$\min \begin{pmatrix} \alpha_0 \sum_{p,T} len(p) * flow_{p,T} & + \\ \alpha_1 \sum_{(c_1,c_2)\in\mathcal{R},n\in\mathcal{N}} (share_{c_1,c_2,n} - 1) & + \\ \alpha_2 \sum_{n\in\mathcal{N}} (load_n)^2 & \end{pmatrix} \tag{8}$$

The objective function 8 in this model is a weighted sum of three terms. The first term captures the total flows which are penalized by the length of the path used to dispatch those flows (such policies are codified in OSFP [38] and BGP practice [18]). The second term gives a unit credit each time equivalence classes on the segregation list $\mathcal{R}$ do not share a node. (Due to this term, the objective value of the final solution may change between iterations of the functional layer.) The third and final term contribute to a bias towards solutions that achieve load balancing thanks to the quadratic component which heavily penalizes nodes with large loads.

Solving the Functional Model The functional model starts with empty sets $\mathcal{C}$ and $\mathcal{R}$ which are augmented with each iteration of the framework. New sets of nodes are added to $\mathcal{C}$ and new segregation rules are added to $\mathcal{R}$ (by the security layer). In the current implementation, least cost paths between pairs of nodes s, t are not generated "on demand". Instead, the generation is limited to the first best k such paths, for increasing values of k. This process will ultimately be improved to use column generation techniques [14].

3.2 Risk Calculation

After the functional layer finds an optimal solution, it passes this solution to the risk calculation procedure. This input is the set of active paths. This module calculates the effective risk to the network for each path and traffic type.

Inputs

$risk(n,T) : \mathcal{N} \times \mathcal{T} \to \mathbb{R}$ – the risk inherit to network device n for traffic of type T ($risk(n,T) \geq 1$)

$d_k(n) : \mathcal{N} \to 2^{\mathcal{N}}$ – the set of nodes at a distance at most k from n in the logical topology

Calculation

Given an *active* path $p \in \mathcal{P}$ with source s and sink t, the calculation proceeds by partitioning the set of nodes of the path into three segments: the nodes "close" to the source s, "close" to the sink t and the nodes "in between". Closeness is characterized by the function d_k and is meant to capture any connected node over the logical topology which sits no more that k hops away. Given this partition, $flowRisk(p,T)$ is:

$$\begin{aligned} flowRisk(p,T) = & \sum_{i\in d_2(s)\cup d_2(t)} risk(i,T)^2 + \\ & \sum_{i\in N(p)\setminus(d_2(s)\cup d_2(t))} risk(i,T)^2 \end{aligned}$$

We use $k = 2$ to model nodes on the same LAN. The rationale is to impart to source s and sink t risk resulting from *lateral movement* of attacks. All other nodes contribute to the overall path risk in proportion to the square of their own risks. We expect in most networks for $d_2(s)$ and $d_2(t)$ to include nodes not directly on the path (like nodes on the same LAN). The input path risk calculation $flowRisk(p, T)$ is modular and can be augmented using other risk calculation methods.

3.3 Security Layer

The mathematical optimization model in the security layer is a mixed integer programming model. We similarly present the inputs, variables, constraints, and objective for the security layer. Its inputs are given below. Also note that all the variables from the functional model are *constants*.

Inputs

$mem(n) : \mathcal{N} \to \mathbb{R}$ – the memory resources of SDN device n
$fwCost(T) : \mathcal{T} \to \mathbb{R}$ – the memory footprint for a firewall blocking traffic type T
$piCost$ – the memory footprint for a packet inspection post
$fwComp$ – the complexity footprint for adding a firewall
$piComp$ – the complexity footprint for adding a packet inspection post to the network
$penalty(p, T) : \mathcal{P} \times \mathcal{T} \to \mathbb{R}$ – the penalty for blocking a unit of flow of type T along path p
$rank(n, p) : \mathcal{N} \times \mathcal{P} \to \mathbb{Z}$ – the position of node n in path p
$flowRisk(p, T) : \mathcal{P} \times \mathcal{T} \to \mathbb{R}_{\geq 0}$ – above risk calculation

Variables

$fw_{n,T} \in \{0, 1\}$ – does a firewall block traffic type T at n
$pi_n \in \{0, 1\}$ – is there packet inspection at network device n
$fwOR_{n,T} \in \{0, 1\}$ – is there a *block everything* or *block traffic of type* T firewall at network device n
$fwOP_{p,T} \in \{0, 1\}$ – is there a firewall on path p
$rf_{p,T} \in [0, 1]$ – risk factor for path $p \in P(s, t)$ servicing flow $(s, t, T) \in \mathcal{F}$
$RMfw_{p,n,T} \in [0, 1]$ – used in the riskFactor calculation
$RMpi_{p,n,T} \in [0, 1]$ – used in the riskFactor calculation

Constraints

$$fwOR_{n,T} = fw_{n,T} \vee fw_{n,*}, \forall n \in \mathcal{N}, T \in \mathcal{T} \tag{9}$$

$$\sum_{T \in \mathcal{T} \cup \{*\}} fwCost_T \cdot fw_{n,T} + piCost \cdot pi_n \leq mem_n, \forall n \in \mathcal{N} \tag{10}$$

$$fwOP_{p,T} = \bigvee_{n \in N(p)} (fwOR_{n,T}), \forall T \in \mathcal{T}, p \in \mathcal{P} : active_{p,T} \tag{11}$$

$$\begin{aligned} &RMfw_{p,n,T} = 1 - (.5)^{rank(n,p)} \cdot fwOR_{n,T}, \\ &\forall p \in P(s, t), n \in N(p), (s, t, T) \in \mathcal{F} \end{aligned} \tag{12}$$

$$RMpi_{p,n,T} = 1 - 0.1 \cdot (.5)^{rank(n,p)} \cdot pi_n, \quad \forall p \in P(s,t), n \in N(p), (s,t,T) \in \mathcal{F} \tag{13}$$

$$rf_{p,T} = \min \bigcup_{n \in N(p)} \{RMfw_{p,n,T}, RMpi_{p,n,T}\}, \quad \forall T \in \mathcal{T}, p \in \mathcal{P} : active_{p,T} \tag{14}$$

Equation 9 is used to define the presence of a firewall that will block traffic of type T at a node n. Equation 10 ensures that the memory footprint in SDN node n for the deployment of the firewall and the packet inspection logic does not exceed the device memory. Equation 11 links the presence of a firewall that will block traffic of type T on a path with the presence of a firewall that will block traffic of type T on any node along the active path. Equation 12 defines the minimum risk factor associated to a firewall. The earlier on the path the firewall is deployed, the lower the risk. Equation 13 similarly defines the minimal risk. Equation 14 defines the composite risk factor.

Objective

$$\min \begin{pmatrix} \beta_0 \left(\sum_{n,T} fwComp \cdot fw_{n,T} + \sum_n piComp \cdot pi_n \right) + \\ \beta_1 \sum_n load_n \cdot pi_n + \\ \beta_2 \sum_{p,T} penalty(p,T) \cdot flow_{p,T} \cdot fwOP_{p,T} + \\ \beta_3 \sum_{p,T} flowRisk_{p,T} \cdot rf_{p,T} \end{pmatrix} \tag{15}$$

The objective function defined in equation 15 is a weighted sum of four distinct terms that focus on minimizing the network complexity based on security resources deployed, the load induced by inspection posts, the penalties incurred from dropping desirable flows due to firewall placement and finally the residual risk. This model is a classic mixed integer programming formulation.

3.4 Result Analysis

The result analysis module tries to generate cuts for the functional layer with the goal of improving both functionality and security. To generate cuts, this module will form equivalence classes of network nodes and pass back certain pairs of these classes, one at a time, to the functional layer. Each pair of classes describes a segregation rule, or a cut, to which to functional layer will adhere to as much as possible.

After the functional and security layers are re-optimized using the most recent cut, the result analysis module determines whether the cut was beneficial or harmful based on the objectives of each layer. If the cut is deemed to have been beneficial, we permanently keep it as a constraint, repopulate the cut queue, and continue the process.

If the cut is deemed to have been harmful, it is removed from the functional layer's constraint pool. Then the next cut in the queue will be passed back to the functional layer. If the cut queue is empty, the feedback mechanism terminates and we output the best solution found.

We note that since this process only provides pairs of nodes it is a heuristic. It may be necessary for many nodes to simultaneously be separated to arrive at a global optimum. This mechanism performed well in our experiments.

3.5 Layer Coordination

It is valuable to review how the layers coordinate. The functional layer sends to the security layer a set of paths that implements the routing within the network to serve the specified flows while satisfying a set of segregation requirements. The security layer first computes *risks* for these paths based on its knowledge of the traffic. The paths, their risk and the security model are then tasked with deploying packet inspection apparatus as well as firewalls within that logical topology to monitor the traffic and block threats (risky traffic). Once the security model is solved to optimality, an analysis can determine whether the proposed logical topology is beneficial or not (w.r.t. its objective) and even suggest further equivalence classes for network nodes as well as segregation rules to be sent back to the functional layer for another iteration. Fundamentally, the coordination signal boils down to additional equivalence classes to group nodes together with segregation rules to separate paths that include network nodes in "antagonistic" equivalence classes.

3.6 Outputs

When the set of potential cuts is empty, the proposed configuration can be parsed and translated into SDN language fragments to be deployed on the network devices in order to obtain the desired logical network topology put forth by our framework.

4 Experimental Setup

A fundamental component of our work is the separation of the physical and logical networks. Our framework has potential in applications where many different logical topologies are possible from a single physical topology. Physical topology is an input to our framework and the empirical evaluation is based on a popular topology: Fat-Tree [1].

The instance of Fat-Tree we use is shown in Figure 3. The network design avoids bottlenecks through multiple equal capacity links between layers. This design uses four layers of switches: gateway, core, aggregate and edge. The edge switches serve as top-of-rack switches and are where our hosts connect.

Within our sample network, we consider having two main types of devices: switches/routers and hosts. In order to model traffic between internal and external entities we utilize two gateway switches which represent the boundary of our network. For generality we consider two traffic types (A and B) which could represent any type of traffic such as web and storage. We also classify traffic as internal and external, with external traffic traversing one of the gateways. We

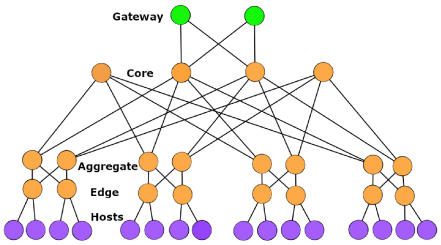

Fig. 3. Order 4 Fat-tree with 2 gateway switches at the top and 2 hosts per edge switch.

allow only half of our hosts to communicate with external sources by allowing them to connect to one of the two gateways. Further, all hosts are involved in internal communications. In this instance we have 16 hosts and we generated 60 flows, 44 of them being internal and 16 being external.

Additionally, our setup simulates an emergent vulnerability/active attack. We select two hosts that are highly vulnerable to, or being targeted on, a specific type of traffic, resulting in a significant increase in their risk for the corresponding type of traffic. In particular, this could represent at DDoS attack on these two hosts.

We run multiple experiments providing the framework increasing numbers of starting paths between source and destination (from the priming procedure) to determine the impact on the solution quality.

Our implementation was built in Python 3.6 using the Gurobi 8.1 optimization library [42]. The experiments were run on a machine running Ubuntu 18.04 and equipped with an Intel Core i9-8950HK processor operating at 2.90 GHz with a 12 MB Cache and 32 GB of physical memory.

5 Evaluation/Results

We begin with the model's resolution on the above scenario using the 10 shortest paths per source and destination pair to prime the optimization. We assume equal demand for each of the 60 flows and two high risk hosts (in red in Figure 4). In each iteration the functional layer of the framework generates a candidate topology and passes this solution to the security layer. The security layer then calculates the network risk and deploys firewalls (represented with rectangles).

In the initial configuration, the gateway on the right serves both low risk (shown in green) and high risk hosts (shown in red). Deploying a firewall on this gateway significantly reduces the network risk and selected as optimal by the security layer. Importantly, this results in collateral damage as flows to low risk hosts are blocked. In total, the first iteration through the framework deploys 3 firewalls which block 12 flows, 8 of which are high risk.

Iteratively, the security layer proposes separation of these collateral nodes from the high-risk nodes. The solution of the last framework iteration (493 candidates cuts are proposed, 40 prove beneficial) is shown on the right. This configuration routes all high risk flows through one core switch where a firewall is

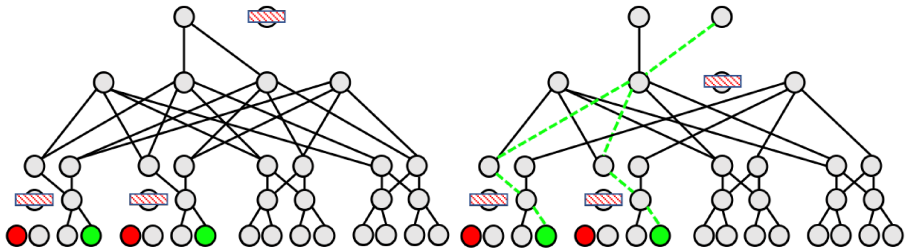

Fig. 4. Illustration of the Fat-Tree network after the first pass through both layers of the framework (left) and the final configuration (right). Note: Firewalls are depicted with rectangles, red nodes represent high risk nodes, green nodes represent nodes that are initially blocked and recovered in the final configuration, utilizing the updated routes shown in dashed lines. The modification of the logical topology allows for more intelligent firewall placement balancing both functionality and security.

now deployed. Meanwhile, all the low risk flows access the gateways through a separate core switch.

Overall we conducted six experiments with this configuration modifying only the number of paths ($\{10, 20, 30, 40, 50, 100\}$) being used to prime the functional layer for each source-destination pair. Ultimately, this process will be dynamic and use column-generation. Complete experimental results are shown in Table 1. A few observations are in order:

- The overall objective reflects both the functional and security layers. The other objective rows refer to each layer objective individually. The network risk rows quantify the risks and their change as the optimization proceeds. For instance, for the 10 paths benchmark, the risk degrades from 10425 to 11646 or 11.7% as a result of supporting an additional 4 good flows.
- The "nodes explored" rows indicate of the size of the branch and bound tree and remains quite modest throughout.
- Within each experiment we observe a meaningful search, as seen by numerous cuts sent back to the functional layer, to segregate high and low risk flows.
- All runs blocked all flows that contain a high risk host while preserving the low risk flows. All experiments delivered final configurations that preserved the same low-risk flows. We therefore hypothesize that a column-generation would quickly settle down and prove that no additional path can improve the quality of the solution. It is nonetheless interesting that adding more paths does not negatively impact the overall runtime.
- The objective functions of the functional and security layers use "scores" meant to ease the interplay between the two. Yet, it is wise to consult the *raw* properties of the solutions to appreciate the impact of the optimization. In particular, the number of flows blocked and the network risks. What is readily apparent is that improving functionality induces a slight degradation in the network risks, underlying the conflicting nature of the two objectives. The individual objective scores while moving in the correct direction are

	10 paths	20 paths	30 paths	40 paths	50 paths	100 paths
Initial Flows Blocked	12	12	12	12	12	12
Final Flows Blocked	8	8	8	8	8	8
Initial Functional Objective	2012	2012	2012	2012	2012	2012
Final Functional Objective	2014	2014	2014	2015	2014	2015
Cut Reward	-8450	-4260	-7210	-4900	-5540	-3840
Initial Security Objective	13735	13724	13729	13723	13696	13726
Final Security Objective	13356	13356	13356	13356	13356	13348
Initial Network Risk	10425	10414	10419	10413	10386	10416
Final Network Risk	11646	11646	11646	11646	11646	11638
Functional Nodes Explored	370	55	206	83	510	46
Security Nodes Explored	1922	1650	152	30	28	19
Beneficial Cuts	40	20	34	23	26	18
Harmful Cuts	453	70	237	81	68	74
Iterations Needed	494	91	272	105	95	93
Time in Model (s)	283	40	319	112	76	273

Table 1. Experimental results from applying the DocSDN framework to an order 4 Fat-tree. Each column refers to a separate experiment where the number of paths per source-destination pair given to the framework were varied. Note that the functional objective values in this table are calculated without the cut reward, the second term in Equation 8, in order to facilitate comparisons across columns.

not to be viewed as stand alone metrics to determine solution quality but rather inter layer communications indicating improvement or decline from a functional or security perspective.

- The objective scores vary across our experiments due to the stochasticity introduced by our heuristic-driven feedback module (see Section 3.4 for discussion). For instance, the functional objecive in the 30 path experiment is slightly worse than it is in other runs, but this difference does not impact the number of serviced flows in the final configuration.
- The variance in time, iterations and number of cuts produced by each experiment is due to symmetries in the formulation. Solutions that are symmetric in the functional layer may not be symmetric in the security layer and induce slightly different solutions there. This is especially true for a Fat-tree network due to its built in redundancy/symmetry.
- Beneficial cuts reflects the number of segregation proposals from the security layers that are adopted by the functional layer (these cuts remove the current best feasible solution). harmful cuts are segregation proposals that do not "cut" the current best feasible solution or worsen the functional solution.

6 Conclusion

Our framework is portable with respect to network risk assessment. Since the risk calculation/analysis is decoupled from the optimization model, the framework can be combined with any procedure that calculates risk on a per path

basis. Along with this procedure, the other requirements for implementing a different risk mechanism are 1) A way of evaluating how risk changes due to the deployment of network defenses and 2) The ability to propose candidate cuts that can be passed to the functional layer.

Our results show it is possible to effectively, automatically, and quickly find a network configuration that meets multiple conflicting properties. Our framework is modular, enabling integration of new desired properties. DocSDN will allow network administrators to effectively prioritize and choose their desired properties. The efficiency of DocSDN is enabled by the feedback/interplay between the functional and security optimization layers.

Acknowledgments

The authors thank the anonymous reviewers for their helpful insights. The authors would also like to thank Pascal Van Hentenryck, Bing Wang, Sridhar Duggirala and Heytem Zitoun for their helpful feedback and discussions. The work of T.C., B.F., and L.M. are supported by the Office of Naval Research, Comcast and Synchrony Financial. The work of D.C. is supported by the U.S. Army. The opinions in this paper are those of the authors and do not necessarily reflect the opinions of the supporting organizations.

References

1. Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. In *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, SIGCOMM '08, pages 63–74, New York, NY, USA, 2008. ACM.
2. P. Baptiste, C. Le Pape, and W. Nuijten. *Constraint-Based Scheduling*. Kluwer Academic Publishers, 2001.
3. Cynthia Barnhart, Ellis L Johnson, George L Nemhauser, Martin WP Savelsbergh, and Pamela H Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations research*, 46(3):316–329, 1998.
4. Ryan Beckett, Aarti Gupta, Ratul Mahajan, and David Walker. A general approach to network configuration verification. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 155–168. ACM, 2017.
5. Ryan Beckett, Ratul Mahajan, Todd Millstein, Jitendra Padhye, and David Walker. Network configuration synthesis with abstract topologies. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 437–451. ACM, 2017.
6. J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4(1):238–252, 1962.
7. Theophilus Benson, Aditya Akella, and David A. Maltz. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, IMC '10, pages 267–280, New York, NY, USA, 2010. ACM.

8. R.E. Bixby, M. Fenelon, Z. Gu, E. Rothberg, and R. Wunderling. *System Modelling and Optimization: Methods, Theory, and Applications*, chapter MIP: Theory and practice – closing the gap, pages 19–49. Kluwer Academic Publishers, 2000.
9. G. Byeon, P. Van Hentenryck, R. Bent, and H. Nagarajan. Communication-Constrained Expansion Planning for Resilient Distribution Systems. *ArXiv e-prints*, January 2018.
10. Yulia Cherdantseva, Pete Burnap, Andrew Blyth, Peter Eden, Kevin Jones, Hugh Soulsby, and Kristan Stoddart. A review of cyber security risk assessment methods for scada systems. *Computers & security*, 56:1–27, 2016.
11. Terry Coatta and Gerald W. Neufeld. Configuration management via constraint programming. In *CDS*, pages 90–101. IEEE, 1992.
12. Gianni Codato and Matteo Fischetti. Combinatorial Benders' cuts for mixed-integer linear programming. *Operations Research*, 54(4):756–766, 2006.
13. MITRE Corp. Common vulnerabilities and exposures, December 2018.
14. George B. Dantzig and Philip Wolfe. Decomposition principle for linear programs. *Oper. Res.*, 8(1):101–111, February 1960.
15. Seyed Kaveh Fayaz, Yoshiaki Tobioka, Vyas Sekar, and Michael Bailey. Bohatei: Flexible and elastic DDoS defense. In *USENIX Security Symposium*, pages 817–832, 2015.
16. Nate Foster, Rob Harrison, Michael J Freedman, Christopher Monsanto, Jennifer Rexford, Alec Story, and David Walker. Frenetic: A network programming language. *ACM Sigplan Notices*, 46(9):279–291, 2011.
17. B. Fourer. Amazing solver speedups. online, 2015. `http://bob4er.blogspot.com/2015/05/amazing-solver-speedups.html`.
18. Phillipa Gill, Michael Schapira, and Sharon Goldberg. A survey of interdomain routing policies. *ACM SIGCOMM Computer Communication Review*, 44(1):28–34, 2013.
19. Hassan Hijazi, Terrence W. K. Mak, and Pascal Van Hentenryck. Power system restoration with transient stability. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, pages 658–664. AAAI Press, 2015.
20. J.N. Hooker. Logic-based Benders decomposition. *Mathematical Programming*, 96:2003, 1995.
21. J.N. Hooker. *Logic-Based Methods for Optimization: Combining Optimization and Constraint Satisfaction*. John Wiley and Sons, 2000.
22. Kyle Ingols, Richard Lippmann, and Keith Piwowarski. Practical attack graph generation for network defense. In *Annual Computer Security Applications Conference, 2006.*, pages 121–130. IEEE, 2006.
23. John Ioannidis and Steven M. Bellovin. Pushback: Router-based defense against DDoS attacks, 2001.
24. John Ioannidis and Steven M Bellovin. Implementing pushback: Router-based defense against DDoS attacks. In *NDSS*, volume 2, 2002.
25. Wayne Jansen. *Directions in security metrics research*. Diane Publishing, 2010.
26. Kerem Kaynar. A taxonomy for attack graph generation and usage in network security. *Journal of Information Security and Applications*, 29:27–56, 2016.
27. Ahmed Khurshid, Wenxuan Zhou, Matthew Caesar, and P Godfrey. Veriflow: Verifying network-wide invariants in real time. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 49–54. ACM, 2012.
28. Hyojoon Kim, Joshua Reich, Arpit Gupta, Muhammad Shahbaz, Nick Feamster, and Russell J Clark. Kinetic: Verifiable dynamic network control. In *NSDI*, pages 59–72, 2015.

29. Sam Kottler. February 28th DDoS incident report, March 2018.
30. Diego Kreutz, Fernando MV Ramos, Paulo Esteves Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, 2015.
31. Edward Lam and Pascal Van Hentenryck. A branch-and-price-and-check model for the vehicle routing problem with location congestion. *Constraints*, 21(3):394–412, July 2016.
32. Siamak Layeghy, Farzaneh Pakzad, and Marius Portmann. SCOR: software-defined constrained optimal routing platform for SDN. *CoRR*, abs/1607.03243, 2016.
33. Richard P Lippmann and James F Riordan. Threat-based risk assessment for enterprise networks. *Lincoln Laboratory Journal*, 22(1):33–45, 2016.
34. RP Lippmann, JF Riordan, TH Yu, and KK Watson. Continuous security metrics for prevalent network threats: introduction and first four metrics. Technical report, Massachusetts Inst of Tech Lexington Lincoln Lab, 2012.
35. Bill Marczak, Nicholas Weaver, Jakub Dalek, Roya Ensafi, David Fifield, Sarah McKune, Arn Rey, John Scott-Railton, Ronald Deibert, and Vern Paxson. China's great cannon. *Citizen Lab*, 10, 2015.
36. Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.
37. Jelena Mirkovic and Peter Reiher. A taxonomy of DDoS attack and DDoS defense mechanisms. *ACM SIGCOMM Computer Communication Review*, 34(2):39–53, 2004.
38. John T Moy. *OSPF: anatomy of an Internet routing protocol*. Addison-Wesley Professional, 1998.
39. Harsha Nagarajan, Emre Yamangil, Russell Bent, Pascal Van Hentenryck, and Scott Backhaus. Optimal resilient transmission grid design. In *PSCC*, pages 1–7. IEEE, 2016.
40. Pedro Neves, Rui Calé, Mário Rui Costa, Carlos Parada, Bruno Parreira, Jose Alcaraz-Calero, Qi Wang, James Nightingale, Enrique Chirivella-Perez, Wei Jiang, et al. The SELFNET approach for autonomic management in an NFV/SDN networking paradigm. *International Journal of Distributed Sensor Networks*, 12(2):2897479, 2016.
41. NIST. National vulnerability database, December 2018.
42. Gurobi Optimization. Inc.,"gurobi optimizer reference manual," 2015. *URL: http://www. gurobi. com*, 2014.
43. Tao Peng, Christopher Leckie, and Kotagiri Ramamohanarao. Survey of network-based defense mechanisms countering the DoS and DDoS problems. *ACM Computing Surveys (CSUR)*, 39(1):3, 2007.
44. Joshua Reich, Christopher Monsanto, Nate Foster, Jennifer Rexford, and David Walker. Modular SDN programming with Pyretic. *Technical Reprot of USENIX*, 2013.
45. Francesca Rossi, Peter van Beek, and Toby Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., New York, NY, USA, 2006.
46. Bruce Schneier. Attack trees. Blog, 1999.
47. P. Shaw. Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems. In *Proceedings of Fourth International Conference on the Principles and Practice of Constraint Programming (CP'98)*, pages 417–431. Springer Verlag, October 1998.

48. Richard Skowyra, Andrei Lapets, Azer Bestavros, and Assaf Kfoury. A verification platform for SDN-enabled applications. In *IEEE International Conference on Cloud Engineering (IC2E)*, pages 337–342. IEEE, 2014.
49. Sal Stolfo, Steven M Bellovin, and David Evans. Measuring security. *IEEE Security & Privacy*, 9(3):60–65, 2011.
50. Gary Stoneburner, Alice Y Goguen, and Alexis Feringa. SP 800-30. Risk management guide for information technology systems. 2002.
51. Richard Wang, Dana Butnariu, Jennifer Rexford, et al. Openflow-based server load balancing gone wild. *Hot-ICE*, 11:12–12, 2011.
52. Ruozhou Yu, Guoliang Xue, Vishnu Teja Kilari, and Xiang Zhang. Deploying robust security in internet of things. In *IEEE Conference on Computer and Network Security*, 2018.
53. Saman Taghavi Zargar, James Joshi, and David Tipper. A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks. *IEEE communications surveys & tutorials*, 15(4):2046–2069, 2013.
54. Shuyuan Zhang and Sharad Malik. SAT based verification of network data planes. In *International Symposium on Automated Technology for Verification and Analysis*, pages 496–505. Springer, 2013.