Efficient Monte Carlo Evaluation of SDN Resiliency

David M. Nicol
Rakesh Kumar
Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign
{dmnicol,kumar19}@illinois.edu

Software defined networking (SDN) is an emerging technology for controlling flows through networks. Used in the context of industrial control systems, an objective is to design configurations that have built-in protection for hardware failures in the sense that the configuration has "baked-in" back-up routes. The objective is to leave the configuration static as long as possible, minimizing the need to have the controller push in new routing and filtering rules We have designed and implemented a tool that enables us to determine the complete connectivity map from an analysis of all switch configurations in the network. We can use this tool to explore the impact of a link failure, in particular to determine whether the failure induces loss of the ability to deliver a flow even after the built-in back-up routes are used. A measure of the original configuration's resilience to link failure is the mean number of link failures required to induce the first such loss of service. The computational cost of each link failure and subsequent analysis is large, so there is much to be gained by reducing the overall cost of obtaining a statistically valid estimate of resiliency. This paper shows that when analysis of a network state can identify all as-yet-unfailed links any one of whose failure would induce loss of a flow, then we can use the technique of importance sampling to estimate the mean number of links required to fail before some flow is lost, and analyze the potential for reducing the variance of the sample statistic. We provide both theoretical and empirical evidence for significant variance reduction.

Keywords

Software Defined Networking, Reliability, Fast Fail-over, Monte Carlo, Importance Sampling

1. INTRODUCTION

The computer/communication networks in industrial control systems (ICS) are unlike those in enterprise networks. An ICS often has real-time requirements, it always has safety requirements, and in the case of networks used in electrical utilities subject to NERC-CIP regulations, has requirements related to provable limited access to so-called "critical assets". Software defined networking [11] (SDN) is an emerging technology with great promise for aiding the design of ICS networks. A controller with a global view of the network is responsible for creating and installing routing and filtering rules into the network's collection of dumb switches. A standard called OpenFlow exists for expression of the rules and the interaction between controller and switches. Motivated by problems in networks for data centers, SDN's common application is for switches to dynamically communicate routing needs to the controller (e.g., first appearance of a flow) and in response the controller creates and installs rules in multiple switches to address that need, but also achieve system-wide properties such as load-balance.

SDN can be used differently in ICS, there is is value keeping switch configurations static. ICS network engineers like to know *exactly* what their networks are doing, and for the purposes of NERC-CIP audits, explaining a static set of switch rules to an auditor is possible, while trying to analyze an SDN controller program which dynamically *generates* switch rules seems a bridge too far for all involved. Instead, one can engineer all of the configuration rules for all of the network switches in such a way that the flows through the network have pre-defined properties (which we will sometimes call "policies"). The rules are installed when the network comes up, and to the greatest degree possible the switch configurations are left unaltered for as long as possible.

There is a challenge though, in that if the physical infrastructure fails (e.g., a link, or a switch), then some intervention by the controller might be needed to restore full functionality. Fortunately there is a mechanism in the OpenFlow specification that supports so-called "fast fail-over" in which a switch at the point of egress consults a prioritized table of links and pushes the frame through the live link with highest priority. Thus when a link drops, the switch can re-route according to this table without consulting the controller. Of course, careful construction of flow rules is needed to ensure that that local action still leads to ultimate delivery of the frame to its intended destination.

It is relatively straightforward to craft SDN rules which ensure that the failure of any *single* link can be tolerated, in the sense that every flow that formerly crossed that link is still delivered after following the fast fail-over link which covers for the failure. Beyond that, there are unresolved algorithmic issues in crafting configurations that ensure toleration of any 2 (or more) link failures. Still, we don't really

SIGSIM-PADS '16, May 15-18, 2016, Banff, AB, Canada
© 2016 ACM. ISBN 978-1-4503-3742-7/16/05…$15.00
DOI: http://dx.doi.org/10.1145/2901378.2901401

anticipate that the network tolerates only one link failure; while subsequent failure of the 1st link's fail-over back-up might indeed cause one or more flows to no longer be routed, there are many links that are logically and physically separated from the first link whose failure could be tolerated just as if they'd been the first failed link. We are led then to a question whose answer this paper supports: how resilient is the network configuration to link failures, where a measure of resilience is the number of link failures may occur before some flow that is supported by the original configuration is no longer supported, what we call a *loss of service*. A moment's reflection though shows that this question needs refinement. One measure might be the absolute minimum number of link failures that can be tolerated, another might be the average number of randomly chosen failures that can be tolerated. We opine that the second version gives a better overall sense of network resilience to normal "wear and tear" failures, while the first might be better to access resilience to attacks by a directed adversary. Note that inducing loss of service through a sequence of link failures in no way implies that the flow cannot be delivered by some other configuration. The point is that when loss of service occurs, new configurations are required of the controller, and in the ICS context we wish to interact with the controller as little as possible.

This paper considers a potential optimization that arises when using Monte Carlo sampling to evaluate a given configuration by estimating the mean number of link failures to loss of service. The standard technique would be to randomly sample sequences of link failures until loss of service occurs, and note the number of link failures involved. Statistical analysis of the sample mean and sample variance yields a confidence interval around the estimate. The technique we develop here does something different called *importance sampling*, which has the effect of tending to make the confidence interval around the mean smaller, for the same number of trials. From a statistical point of view this means that importance sampling is more efficient; for a stated statistical accuracy importance sampling tends to require fewer trials to achieve that accuracy than does ordinary sampling. The challenge when designing importance sampling strategies is to do so in a way that the sample variance is *provably* smaller than ordinary sampling. This is often a challenge, and our presentation of a importance sampling strategy for this problem with analysis of the variance reduction it provides is the key contribution of this paper. We also provide preliminary experimental evidence that importance sampling can reduce the number of samples needed to achieve a given statistical accuracy by an order of magnitude.

2. BACKGROUND

A network orchestrates packet delivery among all network devices by using packet headers. Its configuration drives the two functional components of the networking software: control and data planes. The control plane decides what packet forwarding and header transformations need to occur at network devices, while the data plane performs the actual actions on packets. In the traditional networking architecture, control and data planes coexist on individual network devices, thus requiring manual instrumentation of configuration at individual, heterogeneous devices to implement policy in a variety of configuration semantics. However, the SDN architecture simplifies access to the network configuration by logically centralizing the control-plane state in a device called the controller. This centralized state then drives the network devices that perform homogeneous forwarding plane functions [2]. The forwarding plane functions and the interface between controller and networking device are standardized. The OpenFlow specification [3] is one such standard. It provides an abstract model of a network switch. The OpenFlow switch model specifies the operations it performs and its interface with the controller. The packet-forwarding and modification behavior of a switch is driven entirely by the rules installed on it by the controller as part of the network configuration.

The OpenFlow rules support a fast fail-over feature. This is implemented by allowing the rules to choose an output port for a specified set of packets as a function of liveness of links. Hence, it is possible that the configuration may effectively utilize the topological redundancy by specifying rules that reroute traffic if links fail, without the intervention of the controller. However, in order to guarantee that the network configuration provides whatever degree of resilience to link failure is required, it needs to be validated. Such validation requires careful modeling of the network and the results of link failure events. Hence, for a network comprising of OpenFlow switches, a model that predicts the entire network's behavior needs to be constructed. This model needs to take into account the network state and construct data structures that allow for such validation tractably as a function of network's topology and configuration.

In order to model behavior of a network on per-packet basis, Jin et. al. [6] and Lantz et. al. [12] proposed approaches based on discrete-event simulation and container based emulation respectively. These approaches can predict the behavior of a network, given a configuration and offered traffic, but do not consider issues of the resiliency offered by a configuration.

Peyman et. al [9][8] and Khurshid et. al.[10] developed abstractions for representing sets of packets that are processed similarly by individual switches and thus making solution solvable in near real-time for small campus-sized networks. However, neither of these approaches support modeling of rules that support the fast fail-over feature. Thus, we have focused on exhaustive policy validation in an SDN using aggregated sets of packets when fast fail-over rules are used.

The tool we have developed, **Flow Validator** , is able to validate many properties of an SDN configuration. One of these is the ability of the configuration to tolerate failures of network links. This paper addresses how Monte Carlo simulation used to guide evolution of **Flow Validator** evaluations can estimate resiliency to link failure, measured as the mean number of links that must be failed before *some* flow that was supported by the pre-failure configuration is no longer supported. Modification of **Flow Validator** data structures is expensive with each link failure, and so it is worth-while to explore ways of reducing the number of link failure computations needed to derive a useable estimate.

Flow Validator data structures expose much information about all flows through the network. We can analyze those data structures to identify which links, if failed next, will induce loss of service. While the update of data structures to *implement* a link failure is expensive, identification of this set of vulnerable links is comparatively much much faster. The main result of this paper is that this information can be used in an importance sampling scheme for resiliency estimation and provably reduce the variance of the sample statistic.

3. SOFTWARE DEFINED NETWORKS

3.1 The Network Model

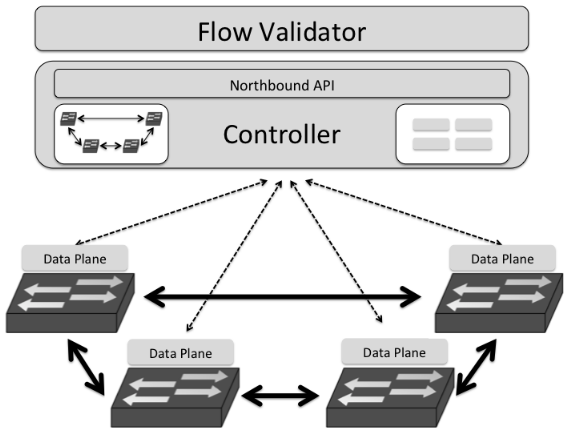

Figure 1: An SDN with a four switch topology connected in a ring.

In an SDN, as Figure 1 depicts, each switch is connected to the controller through a management port. The controller uses this port to gather information about link status and to place forwarding rules on each switch. The information from individual switches is used to construct the state of the entire SDN. The controller makes this state available through a "northbound" API to be used by the applications.

This logically-centralized state comprises a latest snapshot of SDN's topology as well as the data-plane configuration of each switch. Thus, our offline validation application (called Flow Validator) uses this API to access the snapshot of SDN state. The state is used to construct a model to predict the behavior of the SDN on all traffic. If the state of the SDN changes due to a modification of forwarding rules at a switch or a link failure/restoration event, a fresh snapshot of the the state is obtained and any changes are accommodated in the predictive model, incrementally.

3.2 The Switch

In order to predict the behavior of the entire SDN, we need to build a model for individual switches. We use the OpenFlow 1.3 [3] specification to construct one such model for switches, which provides a fast-failover feature.

An SDN switch contains a table processing pipeline and a collection of physical ports. Packets arrive at one of the ports, and are processed by the pipeline comprised of one or more flow tables. Each flow table contains multiple rules. Each flow rule is an atomic unit of decision-making regarding packets going through the pipeline. The decisions take the form of *actions*. During the processing of a single packet, these actions can modify the packet, forward it out of the switch via one or more of switch ports, or drop it. Below we provide more details regarding the OpenFlow specification that are relevant to our model of the switch.

3.2.1 Flow Rule

The set of actions that a switch applies to a packet is governed entirely by flow rules. Each flow rule has two parts:

- Match: A set of packet header field values that the given rule would apply to. Some packet header fields are characterized by single values (e.g. VLAN ID: 1, or TCP Destination Port: 80), while others can take a range of values (e.g. Destination IP Addresses: 10.0.0.0/8). If a packet header field is not specified then it is considered to be a wildcard for the purposes of match operation.
- Instructions: A description of the control operations performed by the flow rule to a matched packet. A switch can apply a variety of actions on the packet. These actions include:
  - Header Modification Actions: These include actions that modify existing protocol headers by adding or removing part of the headers.
  - Output Actions: These include actions that specify the ports on which the packet will be sent out. This can either be a set of ports, or a first live port in a sequence of ports.

3.2.2 Flow Table Pipeline

A switch processes the arriving packets through a pipeline of one or more flow tables. Each flow table is a collection of flow rules sorted by the priority in which they are matched against an arriving packet. A packet matches at most one flow rule in a flow table that has the highest priority.

When a packet arrives at the switch, it is associated with an empty action set and matched against rules in the first table. The action set can be manipulated by instructions in the matching flow rules in each table. Furthermore, the instructions associated with the matching flow rules can select the next table that will process the packet. Before the packet is sent to the next table, the matching flow rule can apply an action to it. If at any table, a matching flow rule for the packet is not found, then the switch stops processing the packet and actions in the associated action set are applied to it. If the action set is empty, then the packet is simply dropped.

The last table applied to a transiting packet chooses the egress port through which the packet is pushed. The table selected for the packet has a list of ports in priority order. The first port in the list whose associated link is live is chosen. Hence, it is possible that the configuration may effectively utilize the topological redundancy by specifying rules that reroute traffic if links fail, without the intervention of the controller. However, in order to guarantee that the network configuration conforms to a given policy in the face of link failures, it needs to be validated. Such validation requires careful modeling of the network and the results of link failure events. Hence, for a network comprising of OpenFlow switches, a model that predicts the entire network's behavior needs to be constructed. This model needs to take into account the network state and construct data structures that allow for such validation tractably as a function of network's topology and configuration.

3.3 Header Space Abstraction

As evident from the operations of a switch previously, in order model to its behavior and reason about properties of an SDN, it is crucial to model sets of packets. To that end, Peyman et. al. proposed a geometric abstraction for representing packet traffic called Header Space [9]. A typical packet contains more than one protocol header, each serving individual control-plane functions. However, the header space abstraction removes individual protocol semantics and represents packets as a point in the $\{0,1\}^L$ space, where L is the total bit length of the packet headers.

Similarly, sets of traffic are represented as hyper-rectangles in this space. These sets are expressed using wildcards for individual bit positions in an L bit header space expression. The utility of the header space abstraction is in its ability to compactly define sets of traffic by using wildcards for individual bit positions in the header. Such definition of traffic then allows set-theoretic operations such as intersection, union, complementation on it.

3.4 Switch Transfer Function

Peyman et. al. proposed representing a switch as a Transfer Function [9]. The transfer function is an abstract model for operations performed by a switch on the input traffic presented at on one of its ports. The switch *tranfers* a subset of traffic to one or more of its output ports. Beyond describing the subset of traffic that arrives at the output port for a corresponding input port traffic, the transfer function also captures any modifications that the subset of input traffic undergoes by the switch as specified by the OpenFlow specification [3].

3.5 Flow Validator Data Structures

The data structures we use and the analysis we employ in **Flow Validator** for validating flows are well beyond the scope of the present paper. However our purposes here we point out that these data structures describe *every possible flow* through the SDN, and so expose the information we need to characterize each link as one whose next failure will induce loss of service, or not. The basic idea is as follows. Some switches connect to hosts, or to networks that are not part of the SDN fabric, others connect only to other switches. Thus there is a set $\mathcal{H}$ of switch ports connected to links that provide SDN ingress and egress. For every given port $p \in \mathcal{H}$ we present a header space that has wildcards in every dimension. **Flow Validator** pushes the abstraction through the rules and tables in the switch. This will fragment the initial abstraction: only some subspace will find rules that admit that subspace, different header modifications and output actions will be applied, different subspaces may be directed to other egress ports. Following application of the header space abstraction to one switch, we have generated a number of header space abstractions on egress links; each such abstraction is presented to the switch at the other end of the link and the process continues, until we discover the complete description of all flows that enter the SDN at the initial ingress port and are delivered to any egress port. Intuitively, we have computed all flows the SDN will route; we have identified for every link the set of flows that (for the given state of the network) cross that link. With supplementary analysis that is outside of the scope of this paper we can analyze these data structures and for every link determine whether if that link were to fail, every flow using that link reaches its destination somehow after using the back-up link. An ability to make this differentiation relatively efficiently is a crucial aspect of our approach.

Following selection of a link to fail, when the chosen link is in the set of links whose failures do not induce loss of service, **Flow Validator** needs to compute the impact that that failure has on all the routes impacted by the failure. This step is computationally expensive; not only are routes and their backups recomputed, other data structures that support compliance of the post-failure SDN with other user-defined policies are also recomputed. This expense helps motivate our work in reducing the amount of simulation workload needed to estimate resiliency as we have defined it.

4. MONTE CARLO EVALUATION

A body of sophisticated work exists on using Monte Carlo to estimate network reliability, e.g. see [1] and its references. The model we have for SDN's is in some ways simpler than such work (e.g., to the extent that we are concerned with link failure probabilities, ours are equal and other work considers various more complex relationships), and in some ways harder (e.g., the connectivity of interest to us is that provided by software configuration, and is dynamic, whereas the more typical model of network connectivity is purely topological.)

The idea behind fast fail-over paths is to provide a means by which link failures can be tolerated without calling the controller. A measure of the link failure resiliency then is the mean of the random variable N_F, defined as the number of link failures that can occur until the controller is called to find new routes.

We can express $E[N_F]$ precisely, but first need to identify some notation. Let $\mathcal{L}$ be the set of all switch links with cardinality $N = |\mathcal{L}|$, let $2^{\mathcal{L}}$ denote the power-set (set of all subsets) of $\mathcal{L}$, and for a given $S \in 2^{\mathcal{L}}$ let $Seq(S)$ be the set of all sequences constructed from permutations of S. For example, if $S = \{l_1, l_2, l_3\}$ then $(l_2, l_1, l_3) \in Seq(S)$ and $(l_3, l_2, l_1) \in Seq(S)$, along with four other unique sequences. For a given sequence $\mathbf{L} = (l_1, l_2, \ldots, l_k)$ we will refer to subsequences $\mathbf{L}_0 = ()$, $\mathbf{L}_1 = (l_1), \ldots, \mathbf{L}_j = (l_1, l_2, \ldots, l_j)$, and so on.

For any $S \in 2^{\mathcal{L}}$ and $\mathbf{L} \in Seq(S)$, we define an indicator function $\phi(\mathbf{L})$ to have value 1 if failing links in the order specified by $\mathbf{L}$ induces loss of service (else 0), and define indicator function $\gamma(\mathbf{L})$ to be 1 if $\phi(\mathbf{L}) = 1$ but $\phi(\mathbf{L}_j) = 0$ for all $j = 1, 2, \ldots, len(\mathbf{L}) - 1$ where $len(\mathbf{L})$ is the number of elements of $\mathbf{L}$. In other words, $\mathbf{L}$ is minimal in the sense that as links are failed in the sequence specified by $\mathbf{L}$, the first loss of service is induced by the failure of the last link. We define

$$\mathcal{S} = \{\mathbf{L} \mid \gamma(\mathbf{L}) = 1\}$$

to reference these sequences of interest.

Given a permutation $\mathbf{L}_N$ of all links in $\mathcal{L}$, there is exactly one j for which $\gamma(\mathbf{L}_j) = 1$, and we denote this index by $d(\mathbf{L}_N) = j$. We can express $E[N_F]$ as the expectation over all permutations $\mathbf{L}_N$, each permutation having equal probability $1/N!$:

$$E[N_F] = \sum_{\mathbf{L}_N} d(\mathbf{L}_N)/N!.$$

We partition the permutations into equivalence classes, where $\mathbf{L}_N^{(1)}$ and $\mathbf{L}_N^{(2)}$ are in the same class if and only if $d(\mathbf{L}_N^{(1)}) = d(\mathbf{L}_N^{(2)}) = k$ for some k, and the two permutations are identical in the first k links. That *stopping prefix* uniquely identifies the class. Given $d(\mathbf{L}_N^{(1)}) = k$ we know that the first

$k-1$ links did not cause loss of service, while the k^{th} selection did. Some care is needed in expressing the probability of choosing the common prefix. For every sequence $\mathbf{L}_i$ with i links and $\gamma(\mathbf{L}_i) = 0$, let $F(\mathbf{L}_i)$ denote the set of links not in $\mathbf{L}_i$ such that extending $\mathbf{L}_i$ with a member of $F(\mathbf{L}_i)$ induces loss of service. The notation captures the dependence of that set on the first i links chosen; different sequences expose different sets of vulnerable links. Also let $\bar{F}(\mathbf{L}_i)$ denote all links not in $\mathbf{L}_i$ and not in $F(\mathbf{L}_i)$. Now if we choose a sequence of k links with the property that the last link induces loss of service but none of the previous links do, we can express the probability that a uniformly sampled N-length sequence is a member of the equivalence class defined by that prefix. This is obtained by multiplying the probability p_{pre} of sampling the specific prefix times the probability p_{post} of sampling some specific sequence of $N-k$ links that are not in the prefix, times the number of such postfixes. To set up a later comparison with a different distribution, we express each term of the product p_{pre} as the probability of sampling from the set of links that do not induce loss of service times the (uniform) probability of sampling a specific link from that set. We likewise express p_{post} as the probability of sampling from the set of links that do induce loss of service times the probability of sampling a specific link from that set. Thus we express

$$\begin{aligned} p_{norm}(\mathbf{L}_k) &= \Pr\{\text{sample sequence } \mathbf{L}_k\} \\ &= p_{pre} \cdot p_{post} \cdot \prod_{i=0}^{N-k-1} \frac{1}{N-k-i} \cdot (N-k)! \\ &= \big(\prod_{i=0}^{k-2} \frac{|\bar{F}(\mathbf{L}_i)|}{N-i} \cdot p_l(\mathbf{L}_i) \big) \cdot \big(\frac{|F(\mathbf{L}_{k-1})|}{N-k+1} \cdot p_f(\mathbf{L}_{k-1}) \big) \\ &= \prod_{i=0}^{k-1} \frac{1}{(N-i)} \end{aligned} \quad (1)$$

where for notational convenience and future use we define

$$p_f(\mathbf{L}_i) = 1/|F(\mathbf{L}_i)|$$

and

$$p_l(\mathbf{L}_i) = 1/|\bar{F}(\mathbf{L}_i)|$$

e.g., the uniform probability of sampling a particular member of of $F(\mathbf{L}_i)$ (alt., $\bar{F}(\mathbf{L}_i)$) at the $(i+1)^{st}$ step.

The point of this rigor is to show that $p_{norm}(\mathbf{L}_k)$ accumulates the probability under uniform sampling of all full length sequences that match in the prefix up to the k^{th} selected link, which first induces loss of service. Therefore if we sample links until we select one that induces loss of service, it is a valid sample of N_F, with a probability given by $p_{norm}(\mathbf{L}_{d(\mathbf{L})})$. For a given k, the probability that $N_F = k$ is given by the sum over all k-length members of $\mathcal{S}$, call this subset $\mathcal{S}_k$, so that

$$\begin{aligned} E[N_F] &= \sum_{k=1}^{N} \sum_{\mathbf{L}_k \in \mathcal{S}_k} k \cdot p_{norm}(\mathbf{L}_k) \\ &= \sum_{\mathbf{L} \in S} len(\mathbf{L}) \cdot p_{norm}(\mathbf{L}). \end{aligned} \quad (2)$$

Equation 2 shows us that Monte Carlo simulation can be used to estimate $E[N_F]$ as follows. An experiment is a sequence of steps where with each step an un-chosen link is randomly selected, and the existing sequence is extended by this selection. On selecting the j^{th} such link, analysis determines whether the failure of that link induces loss of service, and if so the experiment stops. Effectively the simulation determines whether, given the sequence construction after $j-1$ steps, the chosen link is sampled from $F(\mathbf{L}_{j-1})$ (uniformly) or from $\hat{F}(\mathbf{L}_{j-1})$, also uniformly. In the former case value j is the sample of N_F that is recorded, and then another experiment may be run, independent of any other; in the latter case the experiment continues until the stopping condition is encountered. After m experiments the sample mean of the recorded values is our point estimate of N_F, and a confidence interval can be constructed around it in the usual way. For a given N-length sequence $\mathbf{L}_N$ the step where the experiment stops is deterministic, $d(\mathbf{L}_N)$, and so conditioned on this we know that for $i = 1, 2, \ldots, d(\mathbf{L}) - 1$, the sample must draw uniformly at random from $\bar{F}(\mathbf{L}_{i-1})$, and the last sample must draw uniformly from $F(\mathbf{L}_{d(\mathbf{L})-1})$. This is reflected in equation 2.

A class of techniques exist for reducing the overall cost of estimating resiliency called *variance reduction*. In these one conducts the experiments in a way that reduces the variance of the estimator.

4.1 Variance of Estimators

Basic statistical theory teaches that the sample mean we construct from repeated un-biased experiments is a random variable and has a mean that is the same as the mean of the random variable of interest (here, N_F). The confidence we have in a particular sample mean is expressed formally as a confidence interval. The smaller the width of the confidence interval, the greater the confidence we have; the smaller the variance of the probability distribution associated with the sample, the smaller the width of the confidence interval tends to be for the same number of samples.

Classically, when m independent repetitions are performed with m measurements $x_1, x_2, \ldots, x_m$, the sample mean $\hat{\mu} = (1/m)\sum_{i=1}^{m} x_i$ is computed, with confidence interval

$$\hat{\mu} \pm t^* \cdot \frac{s}{\sqrt{m-1}}$$

where $s^2 = \big(\sum_{i=1}^{m}(x_i - \hat{\mu})^2\big)/(m-1)$ is an unbiased estimator of the sampling distribution variance, and t^* is a critical value related to the certainty desired. The smaller $\sqrt{s^2/(m-1)}$, the tighter the interval around $\hat{\mu}$. One commonly seeks to achieve a confidence interval width that is 10% or less of the sample mean. The expression above reveals the limited impact of reducing the confidence interval width by increasing the number of samples: to make the confidence interval smaller by half requires that the number of samples be a factor of four larger.

The variance of N_F is given by

$$\begin{aligned} var(N_F) &= \sum_{\mathbf{L} \in \mathcal{S}} (len(\mathbf{L}) - E[N_F])^2 \cdot p_{norm}(\mathbf{L}) \\ &= \sum_{\mathbf{L} \in \mathcal{S}} \big(len(\mathbf{L})^2 \cdot p_{norm}(\mathbf{L})\big) - E[N_F]^2 \end{aligned} \quad (3)$$

where the expectation is taken with respect to the uniform sampling distribution.

The objective of *variance reduction* techniques is to craft a sampling distribution such that the mean value of the sample mean random variable is that of the random variable of interest (again, here N_F), and that the *variance* of the sample distribution (estimated by s^2) is smaller than that of ordinary random sampling, so that the confidence interval tends to be smaller for the same number of experiments. This paper develops ideas from importance sampling to accomplish this.

5. IMPORTANCE SAMPLING

Importance sampling is a different way to reduce the width of the confidence interval, by reducing the sampling variance s^2. One application of the technique is to increase the precision of the confidence interval for a given number of samples, another application is to achieve a desired precision using fewer samples. The problem motivating our work falls in the latter category. It is computationally expensive to execute an experiment, and so we seek means by which we can reduce the number of experiments needed for a desired accuracy.

Importance sampling has a long history, with reference as early as the 1950s by Kahn and Marshall [7]. Description of various techniques in general stochastic systems is reviewed by Glynn and Iglehart [4], its application to rare event simulation detailed by Heidelberger [5], and to communication systems by Smith, Shafi, and Gao [13]. Use of a variety of variance reduction techniques (including importance sampling) is covered by Cancela et. al [1]. Against the backdrop of extensive literature on importance sampling, our work is unique in its application to SDN, and in exploitation of separating links into the class of those that will induce loss of service when next failed, and the class of those that don't.

The basis for importance sampling is very simple. If $\mathbf{L}$ is a random sequence with discrete components having probability mass function $f(\mathbf{y}) = \Pr\{\mathbf{L} = \mathbf{y}\}$, and g is any other probability mass function on the same space with the property that $g(\mathbf{y}) > 0$ whenever $f(\mathbf{y}) > 0$, then if β is a scalar valued function of $\mathbf{L}$ we can write the expected value of $\beta(\mathbf{L})$ as

$$\begin{aligned} E_f[\beta(\mathbf{L})] &= \sum_{\mathbf{y}} \beta(\mathbf{y}) \cdot f(\mathbf{y}) \\ &= \sum_{\mathbf{y}} \beta(\mathbf{y}) \cdot \frac{f(\mathbf{y})}{g(\mathbf{y})} g(\mathbf{y}) \\ &= E_g\left[\beta(\mathbf{L}) \cdot R(\mathbf{L})\right] \qquad (4) \end{aligned}$$

where E_f denotes the expectation with respect to probability function f, E_g denotes the expectation with respect to probability function g, and the *likelihood ratio function* is $R(\mathbf{y}) = f(\mathbf{y})/g(\mathbf{y})$. This equivalence tells us that an unbiased estimator of $E_g[\beta(\mathbf{L}) \cdot R(\mathbf{L})]$ is also an unbiased estimator of $E_f[\beta(\mathbf{L})]$.

Applying these notions to our problem, the random vectors $\mathbf{L}$ are members of $\mathcal{S}$, i.e., sequences of link failures where the last link failed induces loss of service. $f(\mathbf{L}) = p_{norm}(\mathbf{L})$ is the probability of choosing that sequence of links. To apply importance sampling, at each step when we select a link, the probability distribution need not be uniform, and can depend on previously selected links. We denote the sequence of the first j links by $\mathbf{L}_j$ (with the boundary case of $\mathbf{L}_0 = ()$). Now if $\mathbf{L}_j$ does not cause loss of service and λ is any link not in $\mathbf{L}_j$, we must allow for the possibility that the biased sampling chooses λ, because the unbiased sampling can. We denote the probability under our sampling scheme of choosing λ to extend $\mathbf{L}_j$ by $p_{samp}(\lambda \mid \mathbf{L}_j)$. Therefore the probability of sampling sequence $\mathbf{L} = (l_1, l_2, \ldots, l_k) \in \mathcal{S}$ under importance sampling is

$$p_{skew}(\mathbf{L}) = p_{samp}(l_1) \prod_{i=2}^{k} p_{samp}(l_i \mid \mathbf{L}_{i-1}).$$

Once a link is chosen that induces loss of service the sampling stops, and we can think of $p_{skew}(\mathbf{L})$ as the probability of choosing some member of the equivalence class defined by prefix $\mathbf{L}$. With this interpretation we see that the skewed sampling is referring to the same underlying sample space as the uniform sampling does, and is just an alternative assignment of probabilities to all full permutations of links.

When a sampled link induces loss of service we need to compute the likelihood ratio $R(\mathbf{L}) = p_{norm}(\mathbf{L})/p_{skew}(\mathbf{L})$, and use $len(\mathbf{L}) \cdot R(\mathbf{L})$ as the value of the experiment.

While the intention of importance sampling is to reduce the variance in the sample mean, it is possible to actually *increase* the variance. Denoting the random sample under importance sampling by $\hat{\mu}$, the variance of $\hat{\mu}$ is given by

$$\begin{aligned} var(\hat{\mu}) &= \sum_{\mathbf{L}\in\mathcal{S}} (len(\mathbf{L}) \cdot R(\mathbf{L}) - E[N_F])^2 \cdot p_{skew}(\mathbf{L}) \\ &= \sum_{\mathbf{L}\in\mathcal{S}} (len(\mathbf{L}) \cdot \frac{p_{norm}(\mathbf{L})}{p_{skew}(\mathbf{L})})^2 \cdot p_{skew}(\mathbf{L}) - E[N_F]^2 \\ &= \sum_{\mathbf{L}\in\mathcal{S}} \big(len(\mathbf{L})^2 \cdot R(\mathbf{L}) \cdot p_{norm}(\mathbf{L})\big) - E[N_F]^2. \qquad (5) \end{aligned}$$

The objective is to define a skewed sampling strategy such that $var(\hat{\mu}) < var(\mu)$. Subtracting the expression in equation 5 from the variance of N_F (equation 3) we see that variance is reduced when the right-hand-side of the equation below is positive:

$$\begin{aligned} var(\mu) - var(\hat{\mu}) &= \sum_{\mathbf{L}\in\mathcal{S}} len(\mathbf{L})^2 (1 - R(\mathbf{L})) \cdot p_{norm}(\mathbf{L}) \qquad (6) \\ &= \sum_{\mathbf{L}\in\mathcal{S}} len(\mathbf{L})^2 \big(1 - \frac{p_{norm}(\mathbf{L})}{p_{skew}(\mathbf{L})}\big) \cdot p_{norm}(\mathbf{L}) \end{aligned}$$

This expression gives a clue to a skewed sampling approach that reduces variance. Suppose that $p_{skew}(\mathbf{L})$ could be constructed to be proportional to $len(\mathbf{L}) \cdot p_{norm}(\mathbf{L})$. Applying this to the expression above we obtain

$$\begin{aligned} &var(\mu) - var(\hat{\mu}) = \\ &\quad = \sum_{\mathbf{L}\in\mathcal{S}} len(\mathbf{L})^2 \big(1 - \frac{p_{norm}(\mathbf{L})}{len(\mathbf{L}) \cdot p_{norm}(\mathbf{L})/u}\big) \cdot p_{norm}(\mathbf{L}) \\ &\quad = \sum_{\mathbf{L}\in\mathcal{S}} len(\mathbf{L})^2 \big(1 - \frac{u}{len(\mathbf{L})}\big) \cdot p_{norm}(\mathbf{L}) \\ &\quad = \sum_{\mathbf{L}\in\mathcal{S}} \big(len(\mathbf{L})^2 - u \cdot len(\mathbf{L})\big) \cdot p_{norm}(\mathbf{L}) \\ &\quad = E[N_F^2] - u \cdot E[N_F] \qquad (7) \end{aligned}$$

where u is the normalization constant

$$u = \sum_{\mathbf{L}\in\mathcal{S}} len(\mathbf{L}) \cdot p_{norm}(\mathbf{L}) = E[N_F].$$

Substitution of u back into equation 7 makes the right-hand-side equal to $var(\mu)$, which implies that this particular (unrealizable) choice for $p_{skew}(\mathbf{L})$ yields an estimator with no variance! However, we can use this insight to construct $p_{skew}(\mathbf{L}) \approx len(\mathbf{L}) \cdot p_{norm}(\mathbf{L})/E[N_F]$.

From equation 7 we see that if constant of proportionality u is chosen with $u < E[N_F^2]/E[N_F]$ we should see a reduction in the sampling variance. The only constant of proportionality that *exactly* works is the unrealizable $u = E[N_F]$. In our approach we will construct p_{skew} dynamically, with each sampled link.

6. SKEWED SAMPLING STRATEGY

6.1 Method

We've seen that the strategy of trying to make $p_{skew}(\mathbf{L})$ proportional to $len(\mathbf{L}) \cdot p_{norm}(\mathbf{L})$ has the potential for variance reduction. We design a sampling strategy based on this observation. Earlier we saw that we can view an experiment under uniform sampling as a sequence of Bernoulli trials, where, at the j^{th} trial, while we sample uniformly from among all un-sampled links we are implicitly choosing from set $F(\mathbf{L}_{j-1})$ with probability $|F(\mathbf{L}_{j-1})|/(N-j+1)$, then choosing a specific link from that set with probability $p_f(\mathbf{L}_{j-1})$, thereby terminating the experiment. Under uniform sampling no specific knowledge of the membership in $F(\mathbf{L}_{j-1})$ or $\bar{F}(\mathbf{L}_{j-1})$ was required. Suppose though that the size of these sets *could* be determined explicitly. At the j^{th} step, we could change the probability of selecting $F(\mathbf{L}_{j-1})$ to be any $\alpha_j \in [0,1]$ we chose. We can then modify the expression for $p_{norm}(\mathbf{L})$ to create an expression for $p_{skew}(\mathbf{L})$:

$$p_{skew}(\mathbf{L}) = \alpha_{d(\mathbf{L})} \cdot p_f(\mathbf{L}_{d(\mathbf{L})-1}) \cdot \prod_{i=0}^{d(\mathbf{L})-2} (1-\alpha_{i+1}) \cdot p_l(\mathbf{L}_i) \tag{8}$$

recalling that $p_f(\mathbf{L}_{d(\mathbf{L})-1}) = 1/|F(\mathbf{L}_{d(\mathbf{L})-1})|$ and $p_l(\mathbf{L}_i) = 1/|\bar{F}(\mathbf{L}_i)|$.

By careful selection of the α_j we can *try* to make $p_{skew}(\mathbf{L})$ proportional to $len(\mathbf{L}) \cdot p_{norm}(\mathbf{L})$ when $\mathbf{L} \in \mathcal{S}$. The choice of α_j will depend on the state of sampling prior to step j; what is needed is a procedure that, given $\mathbf{L}_{j-1}$, we compute α_j and then if possible, choose the next link from $F(\mathbf{L}_{j-1})$ with probability α_j.

The qualifications above on selecting and using α_j follow from the observation that, while in principle given a constant proportionality u and $\mathbf{L} \in \mathcal{S}$ one can ascribe probability $len(\mathbf{L}) \cdot p_{norm}(\mathbf{L})/u$ to $\mathbf{L}$, that does not necessarily imply that the method we've described to create a skewed distribution is always able to accomplish that objective. The point is subtle but important. As a consequence, as we construct the skewed distribution we need to ensure that it has non-zero probability on every sequence for which the uniform distribution has non-zero probability. For example, we will see in the equations used tα_i the possibility for the equation to give a value greater than or equal to 1. For a value which is supposedly a probability values in excess are obviously problematic; the case where the equation for $\alpha_i = 1$ and $\bar{F}(\mathbf{L}_{i-1}) \neq \emptyset$ is equally problematic, for it cuts off the possibility of sampling from $\bar{F}(\mathbf{L}_{i-1})$, which the uniform distribution can, and so must then the skewed distribution.

These issues not withstanding, we will do what we can to cause each $\mathbf{L}$ to have $p_{skew}(\mathbf{L}) \propto len(\mathbf{L}) \cdot p_{norm}(\mathbf{L})$, while ensuring that the p_{skew} distribution has support everywhere that p_{norm} has support. Details will follow.

We must first start with a normalizing constant u, which we earlier saw might yield significant variance reduction if $u \approx E[N_F]$. We therefore estimate $E[N_F]$ analytically somehow, or do a few normal random trials to estimate $E[N_F]$.

Next we observe that if $F(\mathbf{L}_i)$ is empty, then $\alpha_{i+1} = 0$. As we sample then, the step where $F(\mathbf{L}_i)$ is first non-empty is important. Formally, given sequence $\mathbf{L}$, we define $b(\mathbf{L})$ to be the smallest index i for which $F(\mathbf{L}_i)$ is non-empty. We do not induce loss of service at any step $i \in [0, b(\mathbf{L})]$, so there obviously $p_{skew}(\mathbf{L}_i) = p_{norm}(\mathbf{L}_i)$. However, things change when $i = b(\mathbf{L})+1$. If it should happen that $F(\mathbf{L}_i)$ includes all links, then there is no skewing to be done; we'll sample a link, it will induce a loss of service, and the experiment will end. However if $\bar{F}(\mathbf{L}_i)$ is not empty we get to choose between the two lists, and $\alpha_{b(\mathbf{L})+1}$ will give the probability of sampling from $F(\mathbf{L}_{b(\mathbf{L})})$. We write

$$p_{skew}(\mathbf{L}_{b(\mathbf{L})+1}) = \beta_{b(\mathbf{L})+1} \cdot p_f(\mathbf{L}_{b(\mathbf{L})}) \cdot \prod_{i=0}^{b(\mathbf{L})-1} 1/(N-i)$$

which expresses the fact that when $F(\mathbf{L}_i)$ is empty we are sampling uniformly from all links. From this we express the property desired of $p_{skew}(\mathbf{L}_{b(\mathbf{L})+1})$:

$$\beta_{b(\mathbf{L})+1} \cdot p_f(\mathbf{L}_{b(\mathbf{L})}) \cdot \prod_{i=0}^{b(\mathbf{L})-1} 1/(N-i) = \frac{(b(\mathbf{L})+1) \cdot p_{norm}(\mathbf{L}_{b(\mathbf{L})+1})}{u}.$$

with solution

$$\begin{aligned}
\beta_{b(\mathbf{L})+1} &= \frac{(b(\mathbf{L})+1) \cdot p_{norm}(\mathbf{L}_{b(\mathbf{L})+1})}{u \cdot p_f(\mathbf{L}_{b(\mathbf{L})}) \cdot \prod_{i=0}^{b(\mathbf{L})-1} 1/(N-i)} \\
&= \frac{(b(\mathbf{L})+1) \cdot \prod_{i=0}^{b(\mathbf{L})} 1/(N-i)}{u \cdot p_f(\mathbf{L}_{b(\mathbf{L})}) \cdot \prod_{i=0}^{b(\mathbf{L})-1} 1/(N-i)} \\
&= \frac{(b(\mathbf{L})+1) \cdot (1/(N-b(\mathbf{L})))}{u \cdot p_f(\mathbf{L}_{b(\mathbf{L})})} \\
&= \left(\frac{b(L)+1}{u}\right) \cdot \left(\frac{|F(\mathbf{L}_{b(L)})|}{N-b(L)}\right).
\end{aligned} \tag{9}$$

There is no reason a priori why $\beta_{b(\mathbf{L})+1}$ so expressed necessarily satisfies $0 < \beta_{b(\mathbf{L})+1} < 1$. When this inequality is satisfied *and* $\bar{F}(\mathbf{L}_{b(L)}) \neq \emptyset$ we will assign $\alpha_{b(\mathbf{L})+1} = \beta_{b(\mathbf{L})+1}$. If it should happen that $\bar{F}(\mathbf{L}_{b(L)}) = \emptyset$ then obviously we must set $\alpha_{b(\mathbf{L})+1} = 1$. The final possibility is that $1 \leq \beta_{b(\mathbf{L})+1}$ and $\bar{F}(\mathbf{L}_{b(L)}) \neq \emptyset$. We have to allow for the possibility of sampling links from $\bar{F}(\mathbf{L}_{b(L)})$, and so choose the next link uniformly from among all links by assigning $\alpha_{b(\mathbf{L})+1} = |F(\mathbf{L}_{b(L)})|/(N-b(L))$. The choice of $\alpha_{b(\mathbf{L})+1}$ establishes the base for subsequent calculations of α_j for $j > b(\mathbf{L})+1$.

Continuing with the sampling, assuming no loss of service is induced up through step $j-1$, we can assume that α_i has been appropriately defined for all earlier steps, and now seek to compute α_j. As with the first step we'll define a variable β_j whose value will define α_j in exactly the same circumstances as it did with $\alpha_{b(\mathbf{L})+1}$: $0 < \beta_j < 1$ and $\bar{F}(\mathbf{L}_{j-1}) \neq \emptyset$. From the requirement $p_{skew}(\mathbf{L}_j) = j \cdot p_{norm}(\mathbf{L}_j)/u$ we set up

$$\beta_j \cdot p_f(\mathbf{L}_{j-1}) \prod_{i=0}^{j-2} (1-\alpha_{i+1}) \cdot p_l(\mathbf{L}_i) = \frac{j \cdot p_{norm}(\mathbf{L}_j)}{u}$$

which we rearrange to express β_j as a function of α_i with $i < j$:

$$\begin{aligned}\beta_j &= \frac{j \cdot p_{norm}(\mathbf{L}_j)}{u \cdot p_f(\mathbf{L}_{j-1}) \prod_{i=0}^{j-2}(1-\alpha_{i+1}) \cdot p_l(\mathbf{L}_i)} \\ &= \frac{j \cdot \prod_{i=0}^{j-1} 1/(N-i)}{u \cdot p_f(\mathbf{L}_{j-1}) \big(\prod_{i=0}^{j-2}(1-\alpha_{i+1}) \cdot p_l(\mathbf{L}_i)\big)} \\ &= \big(\frac{j}{u}\big) \cdot \big(\frac{|F(\mathbf{L}_{j-1})|}{N-j+1}\big) \cdot \big(\prod_{i=0}^{j-2} \frac{|\bar{F}(\mathbf{L}_i)|}{(1-\alpha_{i+1})(N-i)}\big)\end{aligned} \quad (10)$$

where the last step follows from simple algebra after using the definitions of $p_l(\mathbf{L}_i)$ and $p_f(\mathbf{L}_{j-1})$. As before, we cannot in the general case assert that β_j necessarily satisfies $0 < \beta_j < 1$. Therefore, just as with $\alpha_{b(\mathbf{L})+1}$, we assign α_j equal to β_j, 1, or $|F(\mathbf{L}_{j-1})|/(N-j+1)$, depending on whether β_j is a probability that allows for sampling from non-empty $\bar{F}(\mathbf{L}_{j-1})$, must sample from $F(\mathbf{L}_{j-1})$ because $\bar{F}(\mathbf{L}_{j-1}) = \emptyset$, or is not a proper probability, respectively.

Note that for any given $\mathbf{L} \in \mathcal{S}$, the question of whether $p_{skew}(\mathbf{L}) = len(\mathbf{L}) \cdot p_{norm}(\mathbf{L})/u$ depends entirely on whether $\alpha_{len(\mathbf{L})} = \beta_{len(\mathbf{L})}$. If so, the proportionality exists regardless of the details of how α_i with $i < len(\mathbf{L})$ was defined. Equation 11 describes the requirements of proportionality, so that when the solution of that equation is used for $\alpha_{len(\mathbf{L})}$, we necessarily have $p_{skew}(\mathbf{L}) = len(\mathbf{L}) \cdot p_{norm}(\mathbf{L})/u$.

Upon selecting the k^{th} link under the skewed sampling and discovering that its failure induces loss of service, we compute k times the likelihood ratio function and use that as the value representing the experiment. This is given by

$$\begin{aligned}&k \cdot \frac{p_{norm}(\mathbf{L}_k)}{p_{skew}(\mathbf{L}_k)} \\ &= k \cdot \frac{\big(\prod_{i=0}^{k-2} \frac{|\bar{F}(\mathbf{L}_i)|}{N-i} \cdot p_l(\mathbf{L}_i)\big) \cdot \big(\frac{|F(\mathbf{L}_{k-1})|}{N-k+1} \cdot p_f(\mathbf{L}_{k-1})\big)}{\big(\prod_{i=0}^{k-2}(1-\alpha_{i+1}) \cdot p_l(\mathbf{L}_i)\big) \cdot \big(\alpha_k \cdot p_f(\mathbf{L}_{k-1})\big)} \\ &= k \cdot \big(\prod_{i=0}^{k-2} \frac{|\bar{F}(\mathbf{L}_i)|}{(1-\alpha_{i+1}) \cdot (N-i)}\big) \cdot \big(\frac{|F(\mathbf{L}_{k-1})|}{\alpha_k \cdot (N-k+1)}\big)\end{aligned}$$

which is a form that suggests means of stable means of computation, provided that the α_i values used at each step are retained for use in this computation. That computation computes the ratio terms for each product index and multiplies the ratios. A naive computation would divide a very small number $p_{skew}(\mathbf{L}_k)$ into another very small number $p_{norm}(\mathbf{L}_k)$, which is a calculation rife with numerical issues.

The procedures described above will always create a skewed distribution that gives non-zero probability to any sequence that uniform sampling does. The degree to which that procedure yields reduction in variance depends on the structure of the network, configuration, and choice of u. We next consider those issues.

6.2 Variance Reduction

While the method we've described for constructing a skewed distribution strives to create values of $p_{skew}(\mathbf{L})$ that are proportional to $len(\mathbf{L}) \cdot p_{norm}(\mathbf{L})$, there will be $\mathbf{L} \in \mathcal{S}$ for which this is not true. Let $\mathcal{S}_F$ be the set of $\mathbf{L} \in \mathcal{S}$ for which $\alpha_{len(\mathbf{L})} \neq \beta_{len(\mathbf{L})}$ from equation 11 or equation 9. For each $\mathbf{L} \in \mathcal{S}_F$ define $p_{prop}(\mathbf{L}) = len(\mathbf{L}) \cdot p_{norm}(\mathbf{L})/u$, i.e., the probability we would have liked to ascribe to $\mathbf{L}$ but could not because the rules for defining $\alpha_{len(\mathbf{L})}$ selected $\beta_{len(\mathbf{L})} = 1$ or $\beta_{len(\mathbf{L})} = |F(\mathbf{L}_{len(\mathbf{L})-1}|/(N - len(\mathbf{L}) + 1)$. Then for each $\mathbf{L} \in \mathcal{S}_F$ define $R_{prop}(\mathbf{L}) = p_{norm}(\mathbf{L})/p_{prop}(\mathbf{L})$, i.e., the likelihood ratio function value we'd ascribe to $\mathbf{L}$ if it had probability $p_{prop}(\mathbf{L})$. $R(\mathbf{L}) = p_{norm}(\mathbf{L})/p_{skew}(\mathbf{L})$ as before.

The equations for β_i never have a negative solution, so the only time that $\alpha_{len(\mathbf{L})}$ describes uniform sampling rather than $\beta_{len(\mathbf{L})}$ is when $1 \leq \beta_{len(\mathbf{L})}$. We can think of $\beta_{len(\mathbf{L})}$ as a factor by which we need to multiply another term in order to make $p_{skew}(\mathbf{L})$ proportional to $len(\mathbf{L}) \cdot p_{norm}(\mathbf{L})/u$. But since $\alpha_{len(\mathbf{L})} < \beta_{len(\mathbf{L})}$ we can assert the following.

Lemma 1. *For every $\mathbf{L} \in \mathcal{S}_F$, $p_{skew}(\mathbf{L}) < p_{prop}(\mathbf{L})$ and so $R(\mathbf{L}) > R_{prop}(\mathbf{L})$.*

Note that for $\mathbf{L} \in \mathcal{S}/\mathcal{S}_F$, $R(\mathbf{L}) = R_{prop}(\mathbf{L})$. Define $\epsilon(\mathbf{L}) = R(\mathbf{L}) - R_{prop}(\mathbf{L})$, and observe that for all $\mathbf{L} \in \mathcal{S}_F$, $\epsilon(\mathbf{L}) > 0$. Recalling the derivation of the reduction of variance leading to equation 7, we express

$$\begin{aligned}&var(\mu) - var(\hat{\mu}) = \\ &\quad \sum_{\mathbf{L} \in \mathcal{S}/\mathcal{S}_F} len(\mathbf{L})^2 (1 - R(\mathbf{L})) \cdot p_{norm}(\mathbf{L}) \\ &\quad + \sum_{\mathbf{L} \in \mathcal{S}_\mathcal{F}} len(\mathbf{L})^2 (1 - R(\mathbf{L})) \cdot p_{norm}(\mathbf{L}) \\ &= \sum_{\mathbf{L} \in \mathcal{S}/\mathcal{S}_F} len(\mathbf{L})^2 (1 - R(\mathbf{L})) \cdot p_{norm}(\mathbf{L}) \\ &\quad + \sum_{\mathbf{L} \in \mathcal{S}_\mathcal{F}} len(\mathbf{L})^2 (1 - (R_{prop}(\mathbf{L}) + \epsilon(\mathbf{L}))) \cdot p_{norm}(\mathbf{L}) \\ &= \sum_{\mathbf{L} \in \mathcal{S}} len(\mathbf{L})^2 (1 - \frac{u}{len(\mathbf{L})}) \cdot p_{norm}(\mathbf{L}) \\ &\quad - \sum_{\mathbf{L} \in \mathcal{S}_\mathcal{F}} len(\mathbf{L})^2 \epsilon(\mathbf{L}) \cdot p_{norm}(\mathbf{L}) \\ &= \big(E[N_F^2] - u \cdot E[N_F]\big) \\ &\qquad - \sum_{\mathbf{L} \in \mathcal{S}_\mathcal{F}} len(\mathbf{L})^2 \cdot \epsilon(\mathbf{L}) \cdot p_{norm}(\mathbf{L}).\end{aligned}$$

This equation gives us insight into how variance reduction is impacted by choice of u. We recognize $E[N_F^2] - u \cdot E[N_F]$ from equation 7 and the insight that this term is $var(\mu)$ when $u = E[N_F]$. Unlike before though, the term summing weighted values of $\epsilon(\mathbf{L})$ reduces the amount of variance reduction. However if with u near $E[N_F]$ the nature of the network makes $|\mathcal{S}_\mathcal{F}|$ very small relative to $1/p_{norm}(\mathbf{L}) = \prod_{i=0}^{len(\mathbf{L})-1}(N-i)$, then this summation will be very small showing that we can expect excellent variance reduction. Note further that as u increases the size of $\mathcal{S}_F$ decreases. For if $\mathcal{S}_{F,i}$ is defined by $u = u_i$ for $i = 1,2$ and $u_1 > u_2$, then $\mathcal{S}_{F,1} \subseteq \mathcal{S}_{F,2}$. This says that if for u near $E[N_F]$ the number of terms in the sum is too large, the sum of *epsilon* terms can be made smaller by increasing u, but at the cost also of increasing the term $u \cdot E[N_F]$, which also reduces variance reduction.

7. EXPERIMENTS

We use a simple example to illustrate the potential for using importance sampling to reduce the number of samples required to achieve high statistical accuracy. We consider a ring architecture, as illustrated in Figure 2, not only because of its simplicity but also because networks in the ICS domain of interest are in fact often rings. In this example all but two switches have just two ports; two switches have three

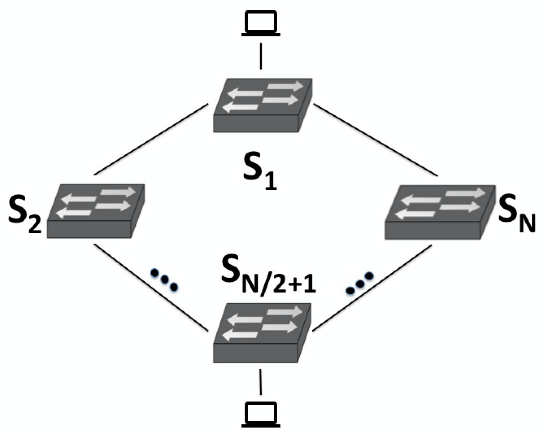

Figure 2: Ring architecture, with two hosts

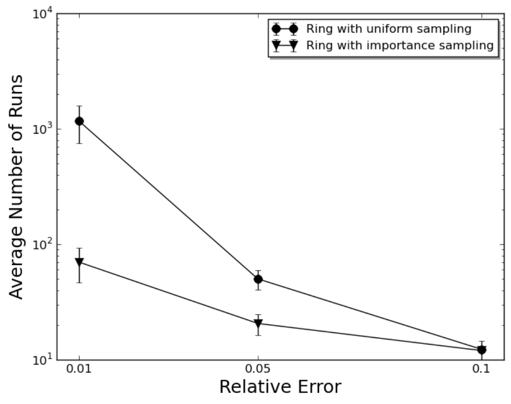

Figure 3: Replications required for given statistical accuracy

ports, with an attached host. A flow is established between the hosts; rules are synthesized so that the backup link is the "other" inter-switch link at that switch.

The statistical accuracy asked of a Monte Carlo simulation is frequently given as "relative error", the ratio of the width of the confidence interval to the mean. So a relative error of 10% means that the confidence interval is one tenth the size in magnitude of the mean it surrounds. As we have seen earlier, to shrink the confidence interval by a factor of two normally requires a factor of four increase in the number of replications. This means that ordinary Monte Carlo simulation will tend to require significantly more replications to achieve a relative error of, say, 1% than a relative error of 10%.

Figure 3 gives the results of experiments we've done with **Flow Validator** on the ring topology with four switches. To understand the meaning of this data, think of each estimation task as a sample of the number of replications required to achieve a given statistical accuracy (in this graph, 1%, 5%, and 10%). Each such estimation task will have a variable number of replications; we plot the mean and standard deviation of 10 estimation tasks. We observe that there is very little difference between standard Monte Carlo and importance sampling for the relative error of 10%, but significant differences for more stringent accuracies. Note that the y-axis is logarithmic, and at a relative error of 1% importance sampling uses over an order of magnitude fewer replications to achieve that accuracy.

8. CONCLUSION

The application of software defined networking in an industrial control system context motivates the development of configurations that can tolerate link failures and minimize interaction with the controller. Assessment of these fail-over paths, along with many other properties is the objective of a tool under development called **Flow Validator** . This paper considers how we might use **Flow Validator** to assess overall resilience of an SDN to link failures, in the sense of estimating the mean number of links that may randomly fail before any flow that formerly was routed can no longer be routed (necessitating involvement of the controller to attempt to repair the network.) The characteristics of **Flow Validator** make it possible for us to approach the problem via Monte Carlo sampling, and exploit deep knowledge about the network state to identify all as-yet un-failed links such that the immediate failure of any one of them will cause some loss of service. We show how to exploit this knowledge to design an importance sampling scheme for the Monte Carlo estimation of the mean of interest. We show in this paper conditions under which we can expect significantly less less variance in the sample statistic, with the potential for significant reduction in the computational effort needed to estimate resiliency with good accuracy. Preliminary experiments confirm this expectation.

Acknowledgements

This work was supported in part by Department of Energy contracts DE-OE0000679 and DE-OE0000780[1], and DHS contract 2015-ST-061-CIRC01[2] .

9. REFERENCES

[1] H. Cancela, M. E. Khadiri, G. Rubino, and B. Tuffin. Balanced and approximate zero-variance recursive estimators for the network reliability problem. *ACM Trans. Model. Comput. Simul.*, 25(1):5:1–5:19, Nov. 2014.

[2] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. Gude, N. McKeown, and S. Shenker. Rethinking enterprise network control. *IEEE/ACM Transactions on Networking (TON)*, 17(4):1270–1283, 2009.

[3] O. N. Foundation. Openflow switch specification 1.3. "https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf".

[4] P. W. Glynn and D. L. Iglehart. Importance sampling for stochastic simulations. *Manage. Sci.*, 35(11):1367–1392, Nov. 1989.

[1]The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

[2]The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the U.S. Department of Homeland Security.

[5] P. Heidelberger. Fast simulation of rare events in queueing and reliability models. *ACM Trans. Model. Comput. Simul.*, 5(1):43–85, Jan. 1995.

[6] D. Jin and D. M. Nicol. Parallel simulation of software defined networks. In *Proceedings of the 2013 ACM SIGSIM conference on Principles of advanced discrete simulation*, pages 91–102. ACM, 2013.

[7] H. Kahn and A. W. Marshall. Methods of reducing sample size in monte carlo computations. *Journal Operations Research*, pages 263–278, 1953.

[8] P. Kazemian, M. Chan, H. Zeng, G. Varghese, N. McKeown, and S. Whyte. Real time network policy checking using header space analysis. In *NSDI*, pages 99–111, 2013.

[9] P. Kazemian, G. Varghese, and N. McKeown. Header space analysis: Static checking for networks. In *NSDI*, pages 113–126, 2012.

[10] A. Khurshid, W. Zhou, M. Caesar, and P. Godfrey. Veriflow: verifying network-wide invariants in real time. *ACM SIGCOMM Computer Communication Review*, 42(4):467–472, 2012.

[11] K. Kirkpatrick. Software-defined networking. *Commun. ACM*, 56(9):16–19, Sept. 2013.

[12] B. Lantz, B. Heller, and N. McKeown. A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, page 19. ACM, 2010.

[13] P. J. Smith, M. Shafi, and H. Gao. Quick simulation: a review of importance sampling techniques in communications systems. *IEEE Journal on Selected Areas in Communications*, 15(4):597–613, May 1997.