Tempus: Probabilistic Network Latency Verifier

Anonymous Author(s)

ABSTRACT

To combat problems and bugs that a given network might have, network verifiers have emerged as one of the promising solutions. State-of-the-art network verifiers mainly focused on evaluating qualitative properties under various scenarios of network failures, such as reachability under k-link failure. However, as modern networks evolved and performance need becomes more stringent – often expressed in terms of Service Level Agreement (SLA) – there is a need to evolve network verifiers to also reason about quantitative performance properties. Works in quantitative network verifiers that has arose in recent years mainly focuses on one side of the network performance metric: bandwidth and load violation properties. Questions about the other side of network performance metric, latency, were left unanswered.

In this work, we introduce a verifier framework, Tempus, that can answer the probability of a given temporal properties being true given latency distributions of individual links and nodes in the network. Early evaluation shows that Tempus can verify bounded reachability property – the probability that the average delay of packets traversing from a source to destination node is below T time unit – by only adding a fraction of the overhead introduced by the qualitative verifier its built upon.

1 INTRODUCTION

Modern networks consisted of various distributed protocols such as OSPF and BGP that exchange routing information so that a packet can reach their intended destination efficiently, even in the event of a failure. Due to their configuration intricacies however, these protocols are notoriously hard to get right. It is hard for a network engineer to reason about whether their configured network fulfilled some intended property in various possible states of the network. Thus, they are left between the choice of accepting the reality of Murphy's Law or getting a tool to help them in this task, namely, network control plane verifier.

Popular control plane verifiers, like ARC [1], are formulated to answer questions about a given property deterministically. In other words, given a set of states of the network (e.g. k-link failures) the verifier are expected to give a yes or no answer about the property (e.g. two nodes are reachable). While useful to some extent, this kind of models are sometimes too restrictive since network operators are usually able to tolerate small fraction of failure. For example, a given network might have an availability SLA of 99.999%. This kind of probabilistic properties have been the focus of a more recent works like NetDice [2].

The network property itself can also be divided into two kinds. Quantitative properties like reachability and loop existence, are the common properties that are studied by most of the existing verifiers. More recent work, like QARC [3], has also explored qualitative properties like link bandwidth violation for a given traffic.

In this work, we're exploring the other side of the network performance metric: latency. Certain network deployment often necessitates some latency requirement such as an ISP that has latency SLA [4] or deployments of Time-Sensitive Networking (TSN) [5]. We proposed a verification framework to probabilistically verify the latency property of packets traversing from a source to destination node under various failure scenarios, by using latency information of each components in the network.

2 PROPOSED APPROACH

We will start with the assertion that latency is a property that only make sense after connectivity between two nodes has been established. In other words, if two nodes in a network aren't connected (due to physical failure or ACL policy), then the latency between them will *always* be infinite. Because of this, we divide the problem of latency verification into two parts: verifying that two nodes are reachable (*functional* property) and only if the functional property is fulfilled, we would verify whether the latency between two nodes fulfilled some additional condition (*temporal* property)

2.1 Topology Graph

For functional property verification, NetDice [2] has laid the way for verifying reachability between two nodes under failure in a probabilistic and efficient manner. In this framework, the physical network is encoded in an edge-labeled directed graph $G_t = (V_t, E_t)$ where V_t represents the routers in the network and E_t represents the connectivity between a pair of source and destination router. The function $r : E_t \rightarrow \{x \in \mathbb{R} \mid 0 \leq x \leq 1\}$ is the edge-label that represents the failure rate of a given connectivity link.

On top of this topology, we could define additional information that would be used by a routing protocol to determine the valid path(s) between two nodes $src, dst \in V_t$ given a particular link failure scenario. In OSPF for example, we define a function $w_{ospf} : E_t \rightarrow \mathbb{N}$ as the edge-label that represents the positive weight of a link.

2.2 Latency Graph

There are a lot of components in a network that may introduce some latency into a transmitting packet. The principal challenge for us, then, is to determine the appropriate level of abstraction; detailed enough in order to accomplish our verification goal in a correct and accurate manner but abstract enough for the problem to remain tractable. We settle on modeling two sources of latency that we deem significant: propagation delay and queuing delay.

Propagation delay is the latency that is introduced by the links in the network, which is independent of the traffic load in the system. Queuing delay is the latency that the queuing process in the node introduced. Unlike propagation delay, queuing delay might be dependent on load in the system, since the more packets there are in the queue, the more delay the node will introduce to a subsequent packets.

To take both of these forms of delay into account, we've decided to model the latency of each components with a univariate continuous random variable. For propagation delay, the semantic of this random variable is relatively straightforward: it is the distribution of latency that a given link will introduce. For queuing delay however, this random variable represents the delay that a given queuing process will introduce, marginalized over various traffic pattern that a given network state might have resulted.

To convey this additional notion of latency distribution, we form a second edge-labeled directed graph (called a latency graph) $G_l = (V_l, E_l)$. For each network node $v_i \in V_t$, we want to "expand" this node to be able to represent the inner working of the queuing process within each node.

- Each v_i is expanded into $v_i^{in}, v_i^{out-1}, ..., v_i^{out-n} \in V_l$ where n is the amount of outward neighbors that v_i has in G_t.
- Every inward edge that v_i has in G_t will all get directed to v_i^{in}.
- Every outward edge that v_i has in G_t will be sourced from one of the v_i^{out-j} instead.
- Add additional edges e_i^j that connects from v_i^{in} to each v_i^{out-j}.
- Finally, we label each $e \in E_l$ with the random variable as the latency distribution.

Semantically, v_i^{out-j} represents a port, normal links that are copied from G_t represents a link with a propagation delay, and newly introduced links e_i^j represents an output queue in node v_i with a queuing delay.

2.3 Latency Verification

From these two graphs, we then do the verification in the following way:

First, we do the functional verification with G_t similar to what NetDice [2] have done. In addition to that, we also collect additional information from each state (e.g. convergent paths) to be used in the temporal verification step. After the functional verifier finished exploring, we then move to the next step.

Second, we compute the total path latency distribution from the convergent path(s) in each state with G_l. This is done by convolving over the latency distribution of each component in the path. Since not all convolutions can be calculated analytically, we implemented a numerical convolution method via mixture distribution, DIRECT [6], which is able to convolve two distribution with a KL divergence error bound.

Third, we calculate the probability of a given temporal property being true in that state. For example, we define **bounded reachability property** (probability of whether a packet can traverse from a source to destination below some time unit T) which can be computed by integrating the PDF from 0 to T. We then combine all of the *temporal* and *functional* in each state to get the probability of a given temporal property being true in this network.

3 PROPOSED EVALUATION

On the correctness side, we want to validate whether the result of our verifier is equivalent to another verifier that has lower fidelity (i.e. functional property only) if we set the temporal property to be virtually unconstrained (e.g. bounded reachability with a very high T).

On the performance side, we will evaluate the verifier by measuring the amount of additional overhead that the temporal verifier would introduce, measured by the amount of states that needs to be re-explored (in the second step) and / or the amount of convolution operation is performed.

REFERENCES

[1] A. Gember-Jacobson, R. Viswanathan, A. Akella, and R. Mahajan, "Fast control plane analysis using an abstract representation," in *Proceedings of the 2016 ACM SIGCOMM Conference*, pp. 300–313, 2016.

[2] S. Steffen, T. Gehr, P. Tsankov, L. Vanbever, and M. Vechev, "Probabilistic verification of network configurations," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pp. 750–764, 2020.

[3] K. Subramanian, A. Abhashkumar, L. D'Antoni, and A. Akella, "Detecting network load violations for distributed control planes," in *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 974–988, 2020.

[4] "Verizon global latency and packet delivery sla." https://www.verizon.com/business/terms/global_latency_sla/. Accessed: 2022-09-20.

[5] "Tsn." https://1.ieee802.org/tsn/. Accessed: 2022-09-20.

[6] C. Röver and T. Friede, "Discrete approximation of a mixture distribution via restricted divergence," *Journal of Computational and Graphical Statistics*, vol. 26, no. 1, pp. 217–222, 2017.