ECE-541 - Project - I:
Performance Analysis of a distributed, real-time network data-plane verification system

Rakesh Kumar

November 13, 2013

Abstract

We propose to use Stochastic Activity Networks to model a distributed system's performance. The purpose of distributed system is to verify that a large network's data-plane state always conforms to certain invariants. These invariants ensure the correct, secure operation of the system. Each update to the network state is verified before it is applied to the system. Hence this verification needs to be performed in *near* real-time. This verification is performed on system state that spans across multiple controllers and aggregation nodes in the system. The design described below presents a novel state aggregation and communication mechanism. Following the design, we describe key input and output variables of interest that we intend to study.

1 Problem

In a Software Defined Network [?], the network contains a set of *dumb* switches that are all connected to a central entity called the *controller*. The controller pushes *rules* to the switches. Each individual rule instructs the switch about the fate of traffic arriving at its ports. The functional output of a switch depends completely on the collection of rules that have been handed down to it. Hence the controller has a logically centralized view of network's data-plane state.

This centralization of state has recently motivated research problems concerning the verification of network state (and consequently its behavior), for presence of certain invariant properties, such as: loop-freeness, absence of black-holes. Moreover, what is particularly desirable is that, not only the network be verified to be *correct*, but it be done as the new rules are being applied to the network. Hence the speed of verification has to be *near* real-time. There have been two recent projects that tackle this very problem using two different approaches [?], [?].

However, as the network sizes grow, or as larger networks crossing boundaries of several administrative domains are considered, inevitably the forwarding state of network is expected to be distribured across multiple controllers [?]. There is a gap in the current literature on how to perform network invariant verification in this distributed scenario. It is this problem that this project proposes to solve. In the subsequent sections, a proposed design of a distributied invariant verification system is presented. Following that, key variables of interest are expressed. We then, propose to use a Stochastic Actvitiy Network (SAN) model to analyze the properties of design for certain guarantees.

2 Design

Consider a collection of N network *slices* each having its own controller containing state for the switches in the given slice. These N network slices are linked with each other using peering links and form the larger network. Each network corresponds to a topological slice and these N topological slices which form the larger network.

Each controller can be source of rule updates and the impact of those rules being applied in a given controller is not just to the network slice but to the overall network. Rules that may seem harmless in the local context of a slice may cause routing loops/black-holes in the larger network. But any real-time analysis that concerns the state of the larger network need to consider the state at all the controllers. Hence, the controllers need to somehow colloborate to answer invariant verification queries posed by the individual rules that are applied in a given controller.

2.1 Network Slices are Switches

The key idea of the proposed design here is to think of a network slice as an *equivalent switch* with peering links as ports. The entire state contained by the controller can be expressed as rules on a switch expressing just what enters and what portion of the entering traffic exits various other ports while abstracting details of state present on individual switch. Hence, each controller becomes just another switch in the larger network of switches. This switch representation is initially formed and subsequently updated as verified rules are applied to the local slice' network.

2.2 Aggregation Nodes

Each controller shares its switch representation with one aggregation node. Each aggregating node collects switch representations from multiple controllers. Hence the aggregation nodes are essentially looking at a collection of switches and they can further do what individual controllers did and compress the state again to form a single switch representing the state of a collection of controllers and further share it with other aggregation nodes in other parts of the network. This design choice follows the SDN philosophy of converging state - which comes at the cost of introducing central points of failures in the network while making computation model simple and in this case more amenable to analysis. Also, this enables one to model an aggregation node essentially as just another controller.

2.3 Rule Update Propagation

When a rule update arrives at a particular controller, it first checks whether the rule does not put the local slice's network in an incorrect state. If not, then the controller creates an *equivalent rule update* on its equivalent switch which represents the impact of applying the rule update on its network slice. It then forwards the equivalent rule update to the aggregation node which performs the same analysis *locally* on the state that now spans multiple controllers. Since the aggregation node is just another controller, it can further replicate the operation that happened on the individual controller, i.e. to check if the rule update is 'local' or whether it needs to be forwarded further up to a higher level aggregation node. Hence the process keeps repeating until the equivalent update has propagated to all the aggregation nodes that it can potentially affect.

2.4 Results Notification

When a rule update arrives at a controller, it has to wait for the update to propagate to all the aggregation nodes and for the result to propagate back. This choice allows for the network to always be in the invariant conformant state. However, we cannot allow for unbounded amount of time to be spent on each query and hence this time needs to be bounded by a constant.

3 Model Notation and Assumptions

Consider a model where we are analyzing a portion of the internet which has a hierarchy of controllers. The higher-level controllers in that hierarchy could be serving as aggregation nodes. We make following assumptions:

- There are L levels in the hierarchy of controllers, where L is finite. A typical value for L would be 10 but it is a system-wide parameter. Each controller on the ith level is called ci. Where i is a member of set [1, L]
- P is the punting probabiliy parameter. The probabiliy that a controller cannot completely answer the verification of a the rule update and has to punt to the higher level controller is pi. Furthermore, pi is a function of (P, i). A typical function for pi would be P * (L-i). (Come to think of it, it does not really need to be a probabiliy, it might as well be rate?)
- Di is the degree parameter. It indicates for a given level i, how many sub-controllers can be directly calling it an aggregator. Canonically, Di is 0 for i = 1, but it can be random or a deterministic function of i.
- There are arrivals on each level of controller. Arrival rates are given by lambda i. Lambda is a function of (Di).
- There are service rates for each level of controller. Service rates are given by mu i. Mu should be a function of Di to allow for scalable processing as one gets closer to the network core. So service is a function of (Di).

4 Modelling: Knobs and Rewards

For the design described in previous sections, we propose to build a Stochastic Activity Network model. We intend to leverage the techniques used by previous work [?] to model distributed computation problems using SANs.

There are several variables that affect how the system performs. The purpose of this study is to measure the impact of changes in the output of system caused by:

- **Rule Update Arrivals:** Rule updates arrive at a certain rate. They arrivals can be assumed to be poisson. The rate of arrivals of these rule updates may vary based on the nature of individual controller depending on the location of controller and the size of the underlying network slice it controls.
- **Rule Update Propogation Probability:** Rule updates are also propagated to the aggregation nodes if they are applied on the local slice successfully. The decision to propagate is random and depends on the nature of rule update.
- **Costs of communication:** Rule updates take a finite amount of time to be propagate over the network links among controllers and aggregation nodes. These costs need to be accounted for.
- **Costs of computation:** The algorithm that compresses the state of a controller and aggregation nodes and presents the equivalent switch output takes a finite amount of time. These times also need to be accounted for.

Since the purpose of the system is to deliver near real-time performance when the rule update arrives, it is crucial to analyze the time it takes for verification to take place.

- **Expected time until service:** The expected amount of time it takes for the results of verification of the rule update to propagate back towards the controller where the rule update originated.

References