Optimizing Network Throughput for Software Defined Networks

William Scarbro

May 9, 2023

- Black - entirely written by me
- Orange - generated from notes
- Green - entirely written by GPT-4 (with paper as input)

Introduction

A Software Defined Network (SDN) provides centralized configuration of computer networks. In this field, a variety of tools exist for validating the configurations specified by this central controller against several important characteristics such as loop freedom, reachability, and waypoint enforcement [2]. To augment these tools we build a library which can estimate the throughput performance of a particular routing configuration. Our tool replaces the functionality of protocols such as Open Shortest Path First (OSPF) [1] in the context of SDNs. In particular our tool ingests the network topology, traffic patterns, and link and switch bandwidth to evaluate throughput in the context of a particular routing configuration.

This paper investigates the performance of network topologies in terms of throughput and presents a model for analyzing and improving routing configurations. In the Background and Related Work section, the paper discusses OSPF, a link-state routing protocol, and a previous study on measuring network topologies' throughput performance. The Motivating Example section demonstrates the usefulness of throughput analysis using two examples of routing configurations.

The Approach section describes the model's inputs and assumptions, intermediate heuristics, and the final throughput heuristic. It also discusses the implementation using Mininet for network virtualization, OpenFlow as the SDN protocol, and POX as the Python library for OpenFlow. The Results section presents the predicted throughput of example topologies, demonstrating that the model is capable of differentiating between some network topologies.

The Evaluation section acknowledges the limitations of using Mininet for testing the model's accuracy and discusses the modifications made to the POX

library to accommodate the model's requirements. The Future Work section outlines plans to extract routing configuration from POX, eliminate simplifying assumptions, add latency to the model, and collect timing results to validate the model's accuracy.

1 Background and Related Work

1.1 OSPF

OSPF (Open Shortest Path First) is a link-state routing protocol used for exchanging routing. It calculates the best path to a destination network using Dijkstra's shortest path algorithm and exchanges Link State Advertisements (LSAs) between routers. It supports features such as equal-cost multipath routing, route summarization, and authentication mechanisms. Equal-cost multipath routing allows multiple paths with the same cost metric to be used simultaneously, while route summarization reduces the size of the routing table by aggregating multiple contiguous network prefixes into a single route.

1.2 Modeling Throughput

The paper "Measuring and Understanding Throughput of Network Topologies" [3] analyzes network topologies in terms of their throughput performance. The authors assume an optimal routing configuration and evaluate network topologies based on their worst-case traffic scenarios, which helps identify potential bottlenecks and performance issues. The traffic in the network is represented by a matrix T(m,v), where m is the source and v is the destination. To establish a baseline, the authors use a naive traffic matrix where T(m,v) is set to 1 for all m and v. Through evaluating the worst-case traffic scenarios for different network topologies, the authors are able to measure and understand their throughput performance.

2 Motivating Example

To show how throughput analysis can be used to analyze and improve routing configurations consider the following two examples shown in figures 1 2. In these figures, there are four hosts (H1-H4) connected by three switches (S1-S3). Each link is labeled by the number of host-to-host routes which use this connection. In this model we assume that the traffic to and from a host takes the same path and that all traffic between two hosts takes the same path. These may not be reasonable assumptions in all routing configurations. In these examples we assume that the traffic between any two hosts is the same (and normalized to 1) and that the bandwidth of all links is the same (and normalized to 1).

Figure 1 shows a routing configuration produced by a greedy shortest path algorithm. This corresponds to the routes given in Table 1 and the throughput

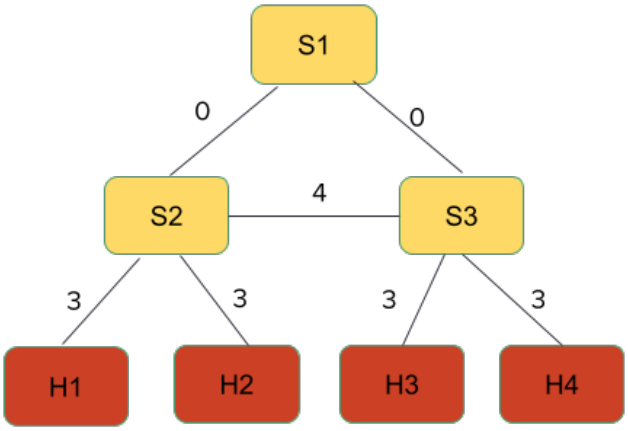

Figure 1: Greedy Shortest Path Routing

H1 $\Leftrightarrow$ H2	S2
H1 $\Leftrightarrow$ H3	S2, S3
H1 $\Leftrightarrow$ H4	S2, S3
H2 $\Leftrightarrow$ H3	S2, S3
H2 $\Leftrightarrow$ H4	S2, S3
H3 $\Leftrightarrow$ H4	S2

Table 1: Greedy Shortest Path Routing: Routes

given by Table 2. The bandwidth of a particular route is calculated as the minimum bandwidth across all links on that route. The bandwidth of a particular link is calculated as the total bandwidth of a link divided by the number of routes using that link.

Figure 2 shows a routing configuration produced by a greedy shortest path algorithm. This corresponds to the routes given in Table 3 and the throughput given by Table 4. In this case the overall throughput has been improved by removing the link $S2 \rightarrow S3$ as a bottleneck. This causes the connection to each host to become the bottleneck in each route.

	H1	H2	H3	H4
H1		1/3	1/4	1/4
H2			1/4	1/4
H3				1/3

Table 2: Greedy Shortest Path Routing: Bandwidth

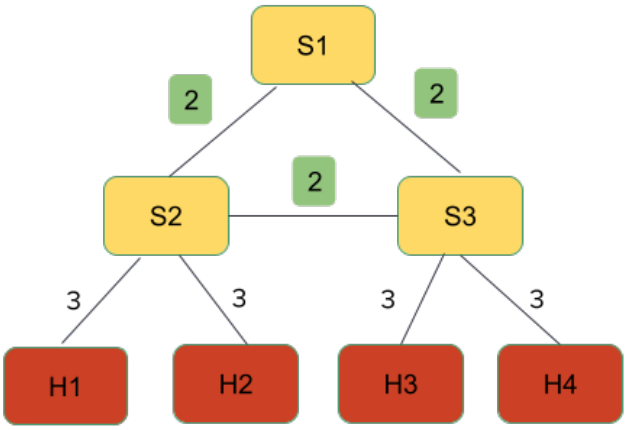

Figure 2: Improved Routing

H1 ⇔ H2	S2
H1 ⇔ H3	S2, S1, S3
H1 ⇔ H4	S2, S1, S3
H2 ⇔ H3	S2, S3
H2 ⇔ H4	S2, S3
H3 ⇔ H4	S2

Table 3: Improved Routing: Routes

	H1	H2	H3	H4
H1		1/3	1/3	1/3
H2			1/3	1/3
H3				1/3

Table 4: Improved Routing: Bandwidth

3 Approach

3.1 Model

The inputs for this model include host-to-host traffic and routes. Additionally, switch-to-switch bandwidth and switch bandwidth are considered. The modeling assumptions are as follows:

1. All traffic from host A to host B follows the same path.
2. The traffic from host A to host B follows the same path as the traffic from host B to host A.
3. Each host is only connected to one switch, with the connection not being a bottleneck. If this link is a bottleneck, then traffic from this host is reduced to the link bandwidth [3].

Intermediate heuristics used in this model include:

1. Link use: the sum of traffic flow through a link.
2. Switch use: the sum of traffic flow through a switch.
3. Host-to-host bandwidth: the minimum bandwidth at any connection (switch/link) in a host-to-host route.

The bandwidth at a connection is calculated as $cap/use * traffic$. The final heuristic in this model is throughput, defined as the average bandwidth across all host-to-host paths.

3.2 Implementation

3.2.1 Platform

The implementation of this model involves using several components. Mininet is used for network virtualization, OpenFlow serves as the SDN protocol, and POX functions as the Python library for OpenFlow with a built-in topology representation.

3.2.2 Integration

The integration process involved extracting the topology from POX, which required some modifications to the POX library. The model assumes Shortest Path First routing which is calculated using Dijkstra's algorithm. Finally, the throughput heuristic is calculated using the model outlined above.

Topology	Hosts	Switches	Throughput
Tree	4	3	0.10
Tree	8	7	0.027
Tree	16	15	0.0076
Tree	32	31	0.0020
Linear	4	4	0.10
Linear	8	8	0.027
Linear	16	16	0.0076
Linear	32	32	0.0020
Single	4	1	0.08
Single	8	1	0.017
Single	16	1	0.0042
Single	32	1	0.0010

Table 5: Predicted Throughput of Example Topologies

4 Results

In these results, the throughput model is fed a topology extracted from POX. A routing configuration is derived using SPF. Traffic between any two hosts is assumed to be uniform (normalized to one).

Mininet provides several different example typologies. The results of the throughput model for a few of these topologies is given in Table 4. The Tree topology constructs a binary tree of switches and attaches each host as a leaf of the tree. The Linear topology constructs a single line of switches, with each switch connected to its neighbors and a single host. The Single topology uses only a single switch connected to all hosts. In all three of these topologies, the SPF routing configuration is the only legal routing configuration (that allows all host pairs to communicate). Mininet also provides a Torus topology; we were unable to get the Torus topology to work with our modification of POX.

The Tree and Linear topologies produce the same throughput according to our model at all network sizes. This is because both topologies have similar bottlenecks. In both topologies the switch in the center (at the root in Tree and in the middle of the line for Linear) experiences the most traffic.

The Single topology produces lower throughput than the Tree and Linear topologies. This is because the single switch becomes a bottleneck for all routes. This is in contrast to the Linear and Tree topologies, which experience a bottleneck at the "center" of the network only for routes which traverse from the "left" to the "right" of the network.

5 Evaluation

Mininet cannot be used to test the accuracy of our model for predicting network characteristics because Mininet does not produce time accurate data.

Table 4 shows that our model is sufficiently complex to differentiate between some network topologies. In particular, Single is almost certainly a worse performing network than Linear and Tree, and our model was able to identify this. It is surprising that the throughput for Tree and Linear are exactly the same. Possible differences in these networks could be investigated and - if there are differences - they could be used to update the accuracy of our heuristic.

The implementation of this model and integration with POX was a useful test of the real world utility of this method. We found that POX already records much of the information necessary to build this model, however there were several cases where we needed to modify the POX library for our purposes. For example, POX does not - by default - add hosts to the topology. To overcome this we implemented a host discovery stage. This has the limitation of requiring that a host must communicate on the network before they are added to the topology (switches on the other hand are discovered automatically using LLDP packets). Further work is required to extract routing information from POX. In addition, there is not a clear way in POX to extract traffic data, we predict this would require the addition of local variables to the Openflow configuration of each switch.

6 Future Work

First, we plan to extract routing configuration from POX by leveraging the routing tables stored by POX in its topology structure. However, we note that POX routing tables are not populated by default, and specific update events are required to update the POX topology. To integrate throughput calculation into POX topology more seamlessly, we propose avoiding passing data around to multiple modules, which could add additional overhead and introduce unnecessary dependencies. Instead, we suggest incorporating throughput calculation into the POX topology. This will have the added benefit of allowing for dynamic updating of route bandwidth, rather than building a routing model statically at each route update.

Second, we intend to extend our model by eliminating simplifying assumptions, such as the assumption that there is only one routing path between two hosts. Additionally, we aim to add latency to our model to make it more comprehensive.

Finally, we plan to collect timing results to validate the accuracy of our model. This validation would likely require the use of a physical network. Collecting timing results would enable us to assess how well our model performs in real-world scenarios and identify any areas for improvement.

7 Conclusion

This paper presents a simple approach to analyze and improve the performance of network topologies in terms of throughput by developing a model that can

predict the throughput of various topologies. The integration of this model with Mininet, OpenFlow, and POX has demonstrated its practical applicability and ability to differentiate between different network topologies. However, the paper also acknowledges some limitations and simplifying assumptions made during the model's development and implementation.

Future work will focus on addressing these limitations, including extracting routing configuration from POX, eliminating simplifying assumptions, and incorporating latency into the model. Additionally, we plan to collect timing results and validate the model's accuracy in real-world scenarios. Despite these limitations, this research has successfully produced a tool which provides insights for network administrators and designers seeking to optimize their networks' throughput in the context of SDN.

References

[1] OSPFD Resource Page. Accessed May 8, 2023. http://ospf.org/.

[2] Arne Ludwig, Szymon Dudycz, Matthias Rost, and Stefan Schmid. Transiently policy-compliant network updates. *IEEE/ACM TRANSACTIONS ON NETWORKING*, 2017.

[3] John. West. *Measuring and Understanding Throughput of Network Topologies*. IEEE Press, 2016.