Norman Anderson March 21, 2025

CSC 579 Biweekly Update 3: An Intelligent Transportation System (ITS) Approach to Teletraffic Engineering

Summary: My project's direction is altered; I explain which decisions from my previous project update(s) are affected. I rectify gaps in my understanding of SDN. Most importantly, I detail how and why I chose my algorithm to implement. To aid my implementation, I explore resources for learning Mininet and Ryu. The summary concludes with challenges and next steps. A mix of online resources and academic sources are referenced in this update.

Changes in direction

This project has both a plan and a goal; it has also lacked disciplined pursuit of either. My journey is no beautiful, linear expedition. To the contrary—it has been "iterative" (I have wasted time exploring dead ends). In any case, some things have changed.

For example, my previous thoughts on prototyping a solution:

> *"Due to limited time and resources, I would prefer to use an already existing prototype, or something that is quick to set up on my own."*

Since then however, instead of working with an existing prototype, I have started from scratch. Why? As I will discuss later, Mininet and Ryu are ideal for prototyping this project, but existing research lacks open-source prototypes using those technologies. My hope is that although the presentation deadline is two weeks away, I will succeed due to the control I will have over my implementation. I have also intentionally kept the scope of my implementation small to ensure I can dedicate the time that is necessary.

My last biweekly update also brought this up:

“Unclear if I am searching for the correct terminology. Is it really a SDN ‘protocol’ that I want to implement?”

I agree with myself here. As I looked further into SDN in the past two weeks, implementing an algorithm rather than a protocol solidified as a clear choice. In fact, I could not justify implementing a protocol at all. It is a clear choice which is less complex: algorithms are considerably easier to implement. As for usefulness: in SDN, protocols such as OpenFlow already exist to communicate routing decisions. I could then design a protocol to make routing decisions, but if the controller is already centralized, what purpose would one serve?

Lastly, I also previously mused whether it would be feasible to use the Internet Topology Zoo [1] to create a realistic topology for experimentation. I haven’t thrown this idea out entirely, but considering my paper [2] would apply to a data center network (DCN) or corporate network, I am skeptical whether testing on a backbone makes sense.

Exploring further (from last time)

SDN

- Continued SDN research with the goal of clarifying the “picture in my head.”
- Started from our class’s reading list, and quickly found that the link to the advanced networks course at the University of Washington has readings on traffic engineering (TE) and SDN.
    - Took a second look at Feamster et al. [3] (which I previously consulted as a source), since it was recommended here.
        - Knowing I had previously explored “good” SDN resources gave me confidence that I was on the right path, especially with regards to technologies. Mininet is mentioned by name in the paper [3].
    - I also encountered a separate paper by Vahdat et al. [4] which discusses Google’s move internally to SDN. I ended up reading the entire thing; I’ve summarized it below.
        - SDN brings concepts from telecom network infrastructure, because we are getting these huge WANs like Google which carry approximately 10% as much traffic as the Internet, but they are all owned and operated by a single company. When all hosts are trusted,

        these networks do not have the same “push” towards decentralization that was fundamental to the original Internet design.
        - In the future, the authors predict SDN peering could overcome BGP's distrustful view of the network by dynamically sharing (some) additional info about your network.
            - For example, info about downstream traffic patterns would improve performance for end users *and* let ISPs use their "lightly loaded paths" better.
            - Also enables application-specific peering: "Hey, video traffic, go through this peer."
        - Currently SDN is applicable in cases where the controller platform and controller application are tightly integrated and developed by the same people; for SDN to apply more broadly we need reusable platforms to exist.
        - David Clark shares some interesting thoughts on fate-sharing, which differ from his original ones when he coined the term in his DARPA paper [5].

Algorithm selection

- Given that our reading summary assignments were due for our chosen topics, I had the opportunity to re-evaluate the relevance of the *GameTE* algorithm [6] to my project.
    - Pros:
        - Involves TE.
        - Consists of a single algorithm.
        - Algorithm seems reasonably straightforward.
    - Cons:
        - No reference implementation.
        - Real-world applicability is suspect (which decreased my interest in this topic, as I am looking for something that would be extensible into more advanced implementations).
        - Involves autonomous systems (ASes) which leads me to believe it might be harder to develop because ASes are *big*.
        - I worry constructing topologies from the Internet Topology Zoo [1] could be difficult to construct in Ryu.

        - No idea of the paper's long-term impact (published to recently).
- Without GameTE it was back to the drawing board. I quickly found a paper by Hadi et al. (*A simple security policy enforcement system for an institution using SDN controller*) [7].
- I didn't select this paper, and it was due to quality concerns. I don't recognize the journal, the authors' emails are Gmail addresses instead of academic ones, and overall I saw the paper as poorly structured.
- Instead I selected one of the competing algorithms the paper references: *Leveraging software-defined networking for security policy enforcement* by Liu et al. [2].
- I'm not entirely happy with this choice either, but I believe the Liu et al. paper is of high enough quality to warrant an implementation. A very brief summary:
    - The paper defines S-switches and F-switches.
        - An S-switch classifies packets (eg. ssh or https traffic).
        - An F-switch implements security policies (eg. dropping packets of certain traffic classes).
    - The main contribution is an algorithm which securely updates the security policies in the F-switches without incorrectly dropping approved traffic and/or letting denied traffic through at any point during execution.
- I will have more information on how I have implemented the various aspects of this paper in my next update.
- In the end I did not rely on *An overview of routing optimization for internet traffic engineering* to select my algorithm. I thought perhaps it might be easier to implement an algorithm which was already considered to be in the domain of SDN.

Mininet and Ryu

- After selecting an algorithm to implement, I needed to learn more about Mininet.
- Before long I found myself at Mininet's FAQ for "How do I implement a custom algorithm?" which directed me to the OpenFlow specification [8].
- Thinking back to common OpenFlow frameworks, I knew Ryu was suggested by Mininet as a "basic OpenFlow controller framework written in Python."
- So I set out to learn Ryu, starting at:
    - The Ryu book: as the introduction suggests, the first five chapters can familiarize a beginner with Ryu.
    - The Ryu tutorial: a simpler tutorial by Ryu. I haven't looked into this as much.
- From a cursory reading, I determined the following:

    - Because I want to build a simple system, extending the Ryu docs looks like a viable starting point.
  - For my project I require basic knowledge in:
    - Ryu
    - Open vSwitch
    - Mininet
- I saved these amazing inside-openflow resources:
  - Using Ryu with VirtualBox, Open vSwitch, and Mininet
  - Using Ryu with Postman
  - Building a datacenter topology (in Ryu)
  - OpenFlow with Ryu basics
  - More complicated OpenFlow concepts - based on Faucet (commercial-grade solution) - includes ACLs and packet filtering
  - Ryu for rapid prototyping
- I also read one of the resources in my last biweekly update (Implementing Different Network Topologies Using Mininet). It gives some practical tips for how to use common topologies and routing algorithms within Mininet.

Challenges

- I estimate another 50 hours are required to complete a “bare minimum” implementation (ChatGPT agrees, courtesy of Fig. 1).

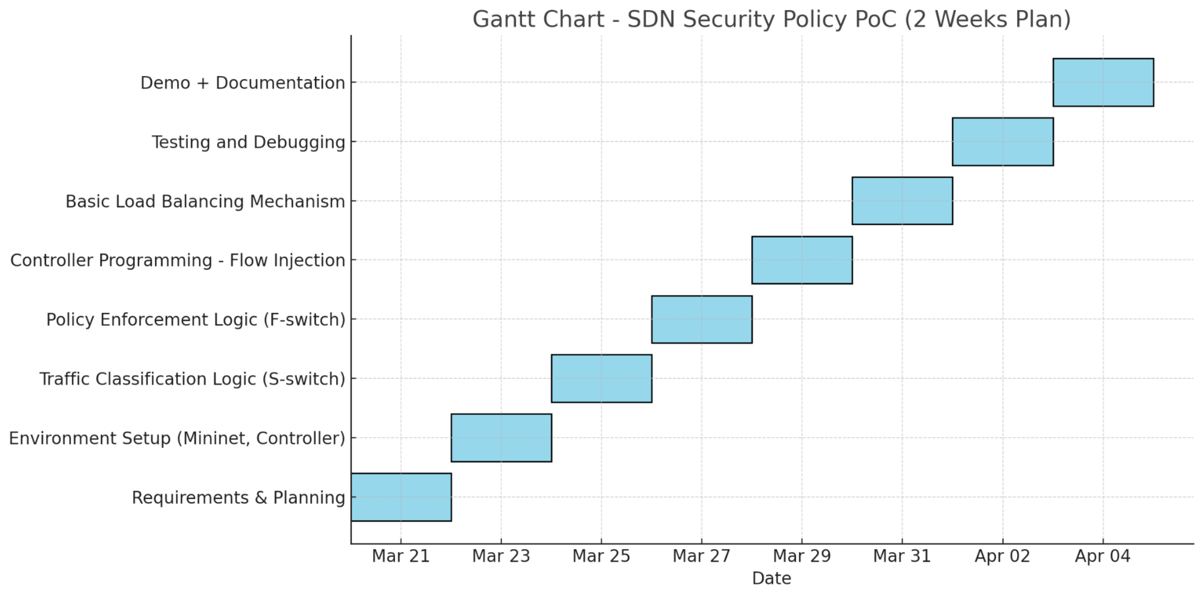

Fig 1. ChatGPT also estimated I require 40-60 hours and offered a “plan.”

- I still have to define which metrics I will use for my analysis.

Next steps

- Implementation.
- Rework project title in light of latest changes.
- Explore the Ryu implementation of SICC and this Java SDN example to familiarize myself with SDN controller code.
- Determine metrics (ASAP) probably by exploring related papers and their associated metrics, possibly by:
  - Using metrics from related academic papers.
  - Taking advantage of the Linux utilities which measure network statistics, since I can easily run them at each Mininet host.
  - Incorporating suggestions from the Mininet or Ryu documentation.
- Determine which kind of topology makes the most sense to use. If it is a data center topology:
  - Does the Internet Topology Zoo contain any DCN topologies?
  - Could I reuse something like the topology from Google's fat-tree paper [9] that we read in class?

References

[1] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, “The Internet Topology Zoo,” *IEEE J. Sel. Areas Commun.*, vol. 29, no. 9, pp. 1765–1775, Oct. 2011, doi: 10.1109/JSAC.2011.111002.

[2] J. Liu *et al.*, “Leveraging software-defined networking for security policy enforcement,” *Inf. Sci.*, vol. 327, pp. 288–299, Jan. 2016, doi: 10.1016/j.ins.2015.08.019.

[3] N. Feamster, J. Rexford, and E. Zegura, “The road to SDN: an intellectual history of programmable networks,” *SIGCOMM Comput Commun Rev*, vol. 44, no. 2, pp. 87–98, Apr. 2014, doi: 10.1145/2602204.2602219.

[4] A. Vahdat, D. Clark, and J. Rexford, “A Purpose-built Global Network: Google’s Move to SDN: A discussion with Amin Vahdat, David Clark, and Jennifer Rexford,” *Queue*, vol. 13, no. 8, pp. 100–125, Oct. 2015, doi: 10.1145/2838344.2856460.

[5] D. Clark, “The design philosophy of the DARPA internet protocols,” *SIGCOMM Comput Commun Rev*, vol. 18, no. 4, pp. 106–114, Aug. 1988, doi: 10.1145/52325.52336.

[6] Y. Liu, J. Hua, Y. Zhang, and S. Zhong, “GameTE: A Game-Theoretic Distributed Traffic Engineering in Trustless Multi-Domain SDN,” in *2024 IEEE 44th International Conference on Distributed Computing Systems (ICDCS)*, Jul. 2024, pp. 1248–1259. doi: 10.1109/ICDCS60910.2024.00118.

[7] F. Hadi, M. Imran, M. H. Durad, and M. Waris, “A simple security policy enforcement system for an institution using SDN controller,” in *2018 15th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*, Jan. 2018, pp. 489–494. doi: 10.1109/IBCAST.2018.8312269.

[8] N. McKeown *et al.*, “OpenFlow: enabling innovation in campus networks,” *SIGCOMM Comput Commun Rev*, vol. 38, no. 2, pp. 69–74, Mar. 2008, doi: 10.1145/1355734.1355746.

[9] M. Al-Fares, A. Loukissas, and A. Vahdat, “A scalable, commodity data center network architecture,” *SIGCOMM Comput Commun Rev*, vol. 38, no. 4, pp. 63–74, Aug. 2008, doi: 10.1145/1402946.1402967.