Poster: FDP: A teaching and demo platform for P4-based SDN

Heena Nagda[■], Rakesh Nagda[♦], Isaac Pedisich[♦], Nik Sultana[♦], Boon Thau Loo[♦]
[■]Georgia Institute of Technology [♦]University of Pennsylvania
heenan@gatech.edu {rakeshn, iped, nsultana, boonloo}@seas.upenn.edu

Abstract

There are a wealth of good-quality open-source tools for teaching, learning about, and experimenting with P4-based SDN. But this tooling and its mode of distribution is geared towards usage at the level of "nuts and bolts", requiring effort to setup even for casual or inexperienced users, and does not give "big picture" insight into the network as the system executes. Our poster describes FDP, a portable platform that builds on existing tooling to enable end-to-end experimentation and zero-effort in-browser interactive visualization, which we envision can benefit teaching of P4-based SDN and research demonstration.

1 Introduction

Teaching and demonstration that involves P4-based software-defined networks (SDN) usually relies on distributing a pre-setup VM for users to run. While this is sufficiently opaque from excessive detail for teaching or demonstration, it still requires effort and adequate resourcing for learners or observers to use these VMs. Further, while the VM approach is adequate for packaging dependencies, it does not provide the teacher nor learner with a network-centric and customizable "big picture" view of how the SDN is behaving.

In this poster we present the Flightplan Demo Platform (FDP). It is designed with the needs of both learners and teachers in mind, including researchers who want to produce an easy-to-access online demo of their SDN system.

Figure 1 shows an experiment in FDP running over a fat-tree topology. FDP's key features include: **i) easy to host:** the viewer-facing component of FDP runs in the browser at no setup cost; **ii) easy to use:** viewers are presented with an animated and explorable 3D topology enriched with visual cues, to see both the "big picture" of the SDN's behavior and the important details; **iii) runs offline:** the frontend and backend are decoupled, and demos or teaching examples can be replayed reliably by many users concurrently; **iv) generic:** the network topology, P4 programs, and the visual cues are all parameters to the system; **v) builds on widely-used systems:** such as Unity engine and Mininet, making FDP amenable to extension and customization; **vi) portable:** all the tooling we use and develop is portable across POSIX systems.

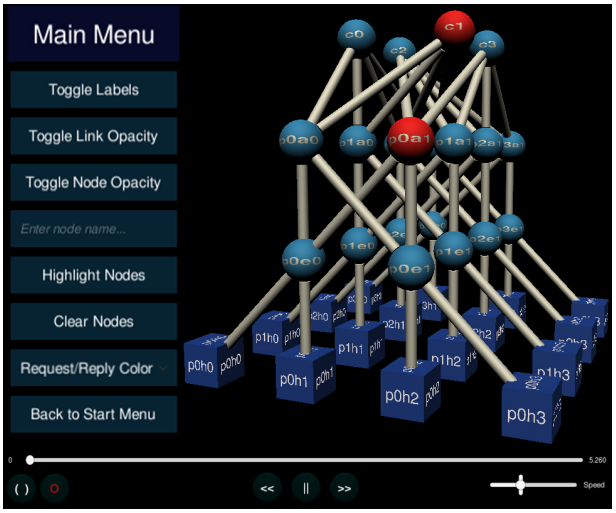

Figure 1: FDP facilitates the creation of 3D animations for viewers, showing the behavior of an SDN network including packet flows. Teachers or demonstrators using FDP can customize the interface or the presentation. This screenshot shows two switches in a fat-tree topology ($k = 4$) being highlighted. Users can change the speed of the visualisation, pause and rewind it, and navigate the 3D space.

2 FDP

FDP is structured into two parts: the **backend** takes experiment descriptions and executes them to produce outputs consisting of pcap files and other logs. The **frontend** takes this output and produces an in-browser 3D rendering of the topology showing the SDN's behavior. This rendering can be further customized to show graphs and visual cues to help the viewer better understand what they are seeing, or to understand a detail that is not evident from the visualization.

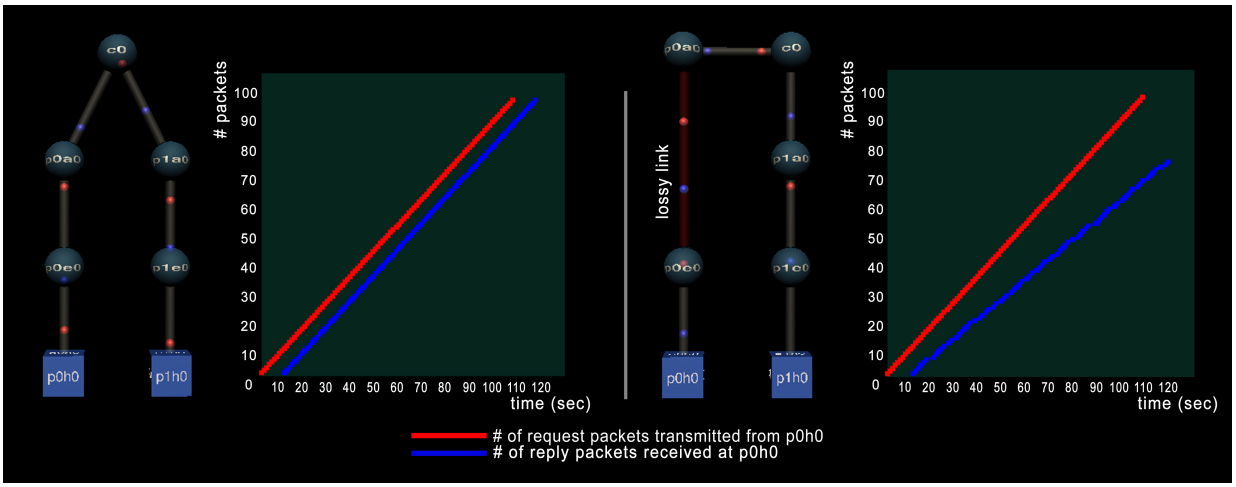

Figure 2: Screenshot of two instances of an experiment on a fat-tree topology ($k = 2$). Host p0h0 pings p1h0, and the latter replies. **Left pane:** the network is functioning correctly: we see the pings and replies flowing, and the graph shows that these are equinumerous. **Right pane:** same setup as in the left pane but with a lossy link between p0e0 and p0a0 resulting in fewer ICMP replies compared to requests over time.

2.1 FDP Backend

The backend's users consist of the designers of a demo or course. The backend relies on Mininet which emulates a virtual network of hosts, links, and switches. Switches can run teacher/demonstrator-selected P4 programs over the Behavioral Model version 2 (BMv2) software switch.

Inputs. The network topology is specified as a YAML file, and network traffic is generated using custom or third-party tools such as iperf3. The YAML file is used to configure a virtual Mininet network, and the BMv2 switch instances running in it.

Outputs. The traffic flow across all the links of the topology is captured. The switch behavior and important actions by software running in the network are logged.

2.2 FDP Frontend

The frontend's users do not necessarily overlap with the users of the backend. The frontend's users consist of the viewers of a demo or participants in a course. The frontend runs in the browser and is designed to provide a rich and customizable interactive visualization to users without burdening them with setup effort.

Analysis. The raw backend output is analyzed, collated and summarized into a timeline for the animation. The topology's YAML description is used to extract the network's links and names. Analysis is done offline, and a metadata file is produced containing all the attributes necessary to display the network traffic visualization.

Visualization. FDP uses two mechanisms to provide visual insight into the SDN. The first mechanism consists of rendering the topology and showing data packets travelling through the network. This view can be augmented using labels, colors and other visual cues added to the animation's timeline. The second mechanism consists of a 2D graph juxtaposed on the screen to show quantitative information—such as latency or throughput—that might be difficult for the viewer to discern from the 3D animation. Fig. 2 shows an example of both mechanisms being used.

User input. Users are completely free to move the camera around the network and zoom in or out. Since the network might be large, users can search for nodes and highlight them. Since the network's behavior might be complex, users are provided with buttons to change the speed of the animation, as well as pause or rewind it. This greatly helps in tracking the life-cycle of an individual packet.

Programmability. The frontend is mostly implemented in C# and builds on the Unity game engine. Teachers, demonstrators, or third-party developers who want to customize FDP with new features can do so within an expressive and portable programming environment.