Motivation

Software-defined networking is a recent advancement in network management that decouples the network control plane from the data plane. The logical behavior of the network is left to one central controller, while the devices and switches on the network behave essentially as simple packet forwarding devices. This allows the use of programmable behaviors to manage the network through a central controller, rather than implementing routing rules and updates for each device individually.

However, there are still issues with SDNs. First is the issue of head-of-queue blocking – that is, updates are applied one-by-one, in order, which leads to slow sequential execution. Secondly, there is no failsafe when communication between the devices and controller is disconnected, which may lead to incorrect routes or dependencies.

Proposed solution

In this project I will aim to work towards resolving these issues by implementing an abstract-algebra-driven update framework in conjunction with a probabilistic push-ahead algorithm to allow for resilience in the case of a network partition.

Update Algebra

Recent work has provided an algebraic framework to solve the head-of-queue blocking problem by treating each update as an element in a semigroup, allowing for asyn-
chronous composition of network updates. Combined with a dependency structure that
enforces order onto the updates, this allows for non-blocking updates to the network
structure.

I plan to continue work in the implementation of this framework as well as utilize
this in the design of a new algorithm for a more robust SDN paradigm.

**Partition-resistance**

In the case of a temporary loss of connection between the central controller and the
data plane, I aim to provide a buffer to ensure an uncompromised network state during
the outage. From a high-level overview, the central controller may have two channels
through which to communicate with devices. The first would be an execution channel,
where the central controller would send updates to be immediately executed in the
devices. Here, the devices would again essentially behave as simple rule tables and
ignore the computational components of the network. The second channel would be a
push-ahead channel, where the central controller would push updates to the devices to
be stored in a buffer. Here the devices would gain some autonomy, as the dependency
and

There are several goals to consider:

- **Update speed.** Here we will leverage previous work in developing datapath op-
eration containers (DOC) [1]. Moreover, I will conduct further research into
implementations of similar software, such as git, for the sake of implementation
details.

- **Minimizing duplicate updates.** A possible approach to this may involve a prob-
abilistic algorithm, taking into account the likelihood of future outages, and cal-
culating which updates to send to the execution and push-ahead channels.

**Deliverables and planning**

1. **Background research.** Here I will dive deeper into the designs and implementations
of systems such as DDP, Update Algebra, and other related works.
2. Design. I will design a system and algorithm to provide the robustness described in the previous section.

3. Implementation. I will implement the system I designed as a proof-of-concept and for evaluation purposes.

4. Evaluation. I will evaluate the system’s efficacy based on the goals mentioned above and other considerations as recommended.

By the end of the semester, I will provide a writeup detailing the work I’ve done, as well as any supporting code and results stemming from the implementation and evaluation portions of the project.

References
