Applying NMAL OSiRIS to Network Slices on SLATE

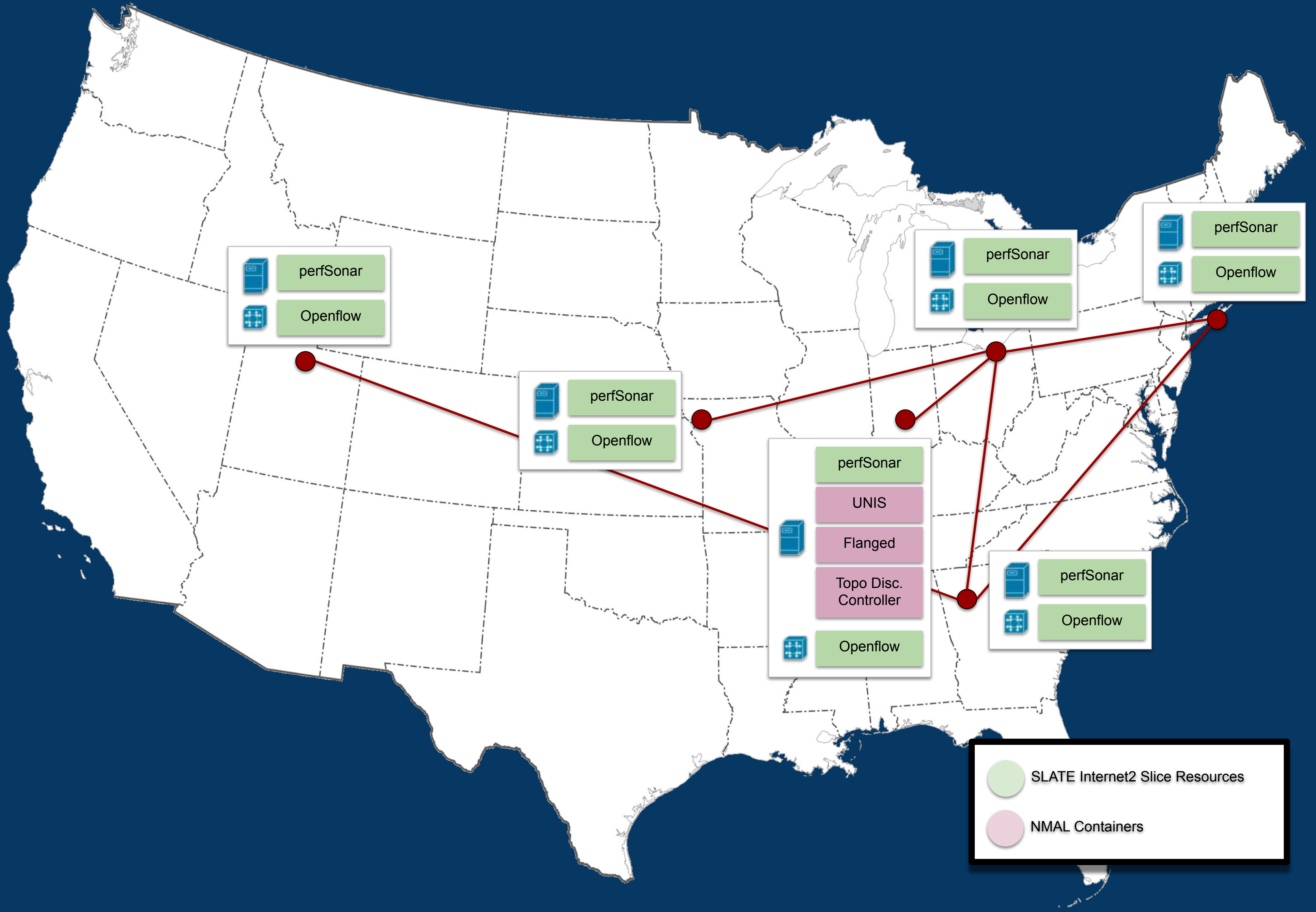
Jeremy Musser, Ezra Kissel, Douglas Swany (Indiana University)
Ben Meekhof, Shawn McKee (University of Michigan)
Joe Breen (University of Utah)

Network Management on SLATE

The **Network Management Abstraction Layer** (**NMAL**) on **SLATE** provides services for curating a near real-time model of the network as well as applying rules for the management and orchestration of the Open Storage Research Infrastructure (**OSiRIS**). SLATE represents a platform that allows for Services Layer At The Edge, specifically distributed science applications which deploy from a curated application catalog. To support these applications, NMAL includes topology discovery through active (SDN table inspection), and passive (LLDP sniffing) agents. Passive agents also extract resource state metadata. The network state is collected through plugins pulling from compatible measurement sources and managed by a network orchestration component called Flange. Recent efforts have allowed NMAL services to execute on the SLATE service deployment and orchestration platform.

NMAL gives us real-time feedback into the topology of our network from hardware switches and software **Open vSwitches**

Which we'll use for real-time network pathing decisions with **Flange** rules



As a demonstration of the system, NMAL has been applied on SLATE services within a slice of an Internet2 SDN testbed. NMAL agents were deployed to SLATE running on this testbed and given access to the testbed's SDN virtual switches.

NMAL Network Model Generation and Annotation

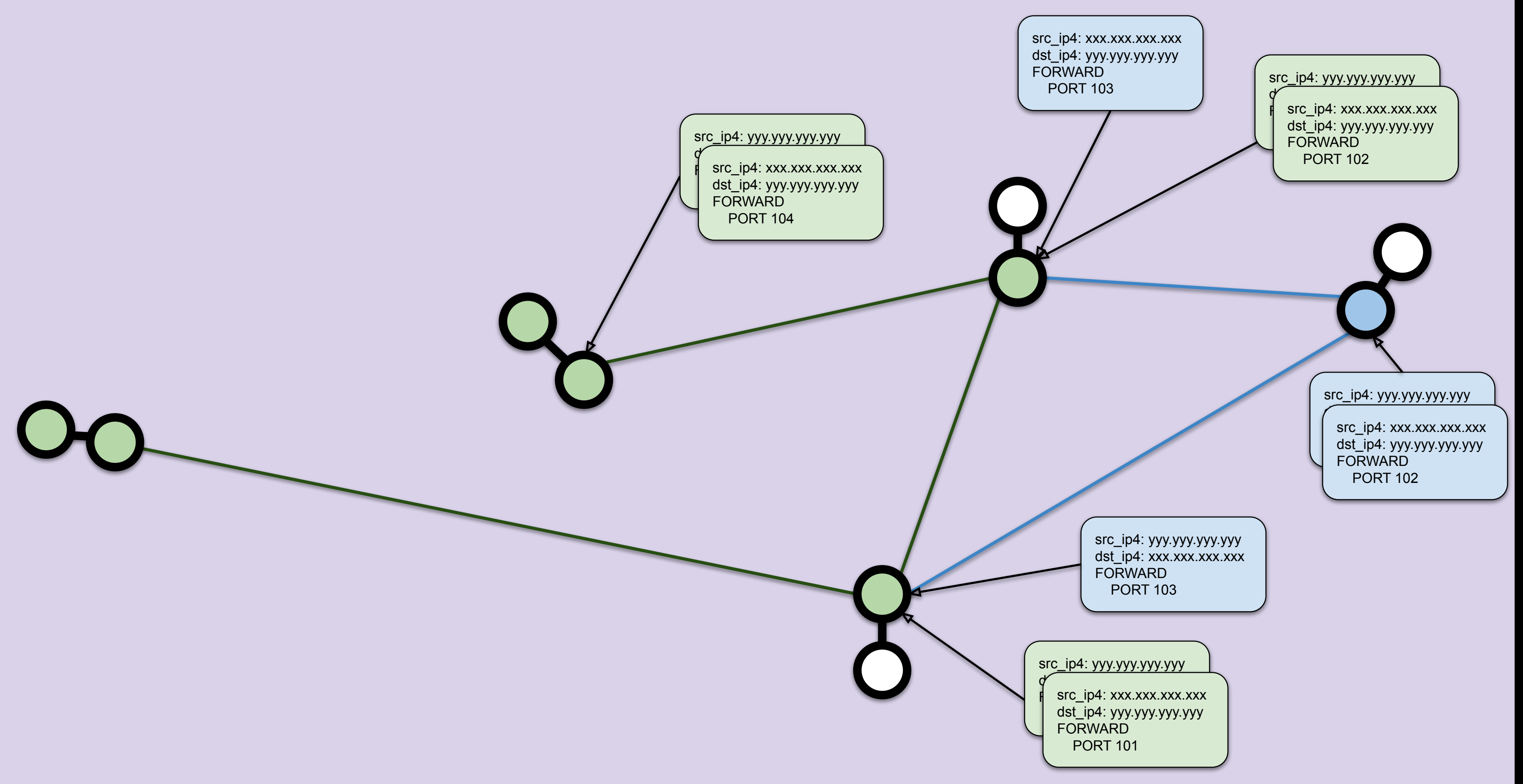
The NMAL topology discovery application uses the *zof* Openflow micro-framework to interrogate match-action rules within attached openflow switches. In addition, extra topological information is scraped from LLDP packets routed through the controller. This information is used to generate a graph representing the attached network's topology. Graph edges and vertices are annotated with descriptive metadata from passive information scraped from LLDP packets and active measurement agents such as perfSonar and BliPP, a measurement agnostic framework used on NMAL hosts to collect utilization statistics. Resource metadata is heterogeneous and polymorphic in nature, links may or may not contain bandwidth utilization or window average latency measurements depending on the availability of measurement agents.

On the Internet2 SLATE testbed, each host runs a perfSonar instance that provides periodic bandwidth and latency measurements. Network orchestration policies can draw from this topological metadata to make more nuanced decisions when generating circuits. In our tests on the SLATE testbed, we predicated circuit creation and routing on bandwidth measurements between Atlanta and Cleveland.

```
let client = { x | x.name == 'saltlake' }
let server = { x | x.name == 'kansas' }
let rules = { f | f.throughput_bps >  80000000 }

exists client ~rules> server
exists server ~rules> client
```

🟢	Optimal Uncongested Routing
🔵	Possible Alternate Routing Under Load



Configuration programs written in the network orchestration language, **Flange**, generate rules for a provided topology. When run against the SLATE testbed, the program above generates a single bi-directional circuit between Salt Lake City and Atlanta. This circuit is predicated on the measured performance of the underlying network. When NMAL detects a spike in bandwidth utilization, the program is partially re-evaluated and a new set of rules is generated to satisfy the program.

Solutions are generated as needed, and the blue circuit above is resolved only in the case where the green circuit does not satisfy the program as a result of congestion. Routing rules are also generated as needed; new rules hide previously established circuits. In the case that the invalidating congestion abates, the new rules may be removed, revealing existing routes. This approach minimizes network state mutations and presents NMAL with a unified model of network behavior.

NMAL Policy Definition

Policies are defined in NMAL with the **Flange** network orchestration language. Flange uses declarative syntax to define a goal state on the network. The compilation stage generates a difference graph and registers it as a requested network state. A Flange daemon - *flanged* - tracks changes to the network, including newly registered difference graphs. When a new state is detected, the program is run against the new network state.

The behavior of the circuit is defined by the program, but enacted and enforced by the flange daemon. The resulting forwarding rules are generated entirely by *flanged* and are subject to the active network model. Evaluation may be triggered by a new policy, but also results from disruptions in the network, such as chronic congestion or link failure.

With NMAL, circuits can be defined by source and destination, or by property, or some composition of the two. Additionally, the modifications to the network are made through back-end agents which interpret a generic *netpath* intermediate representation. As such, multiple back-end agents may operate on the policy at once on different devices. An Openflow switch can be programmed with the same *netpath* as a P4 switch as long as a compatible interpreter exists.

Containerizing NMAL on SLATE^{CI}

NMAL components execute within docker containers on the SLATE deployable services platform. Each service registers with a centralized registry for service discovery. When the NMAL containers are run on SLATE, each agent queries the centralized community of services and begins collecting topological and measurement data and exposes public API endpoints. One goal is to allow NMAL to be distributed in a number of dynamic environments using SLATE with minimal manual reconfiguration.

Supported by NSF Awards 1541335 and 1724821.