Loom Architecture

Sandhya Narayan
[work in progress]

Introduction

Loom is a distributed OpenFlow control plane designed to scale and control switch fabric that can comprise of thousands of OpenFlow switches or datapath elements (DPE). Loom is being written in Erlang with the goal of utilizing of Erlang and Erlang OTP's special features supporting distributed applications and concurrency. Loom sits between network applications, distributed applications, and toolkits that are specially designed to take advantage of Loom and the switch fabric.

A quick example

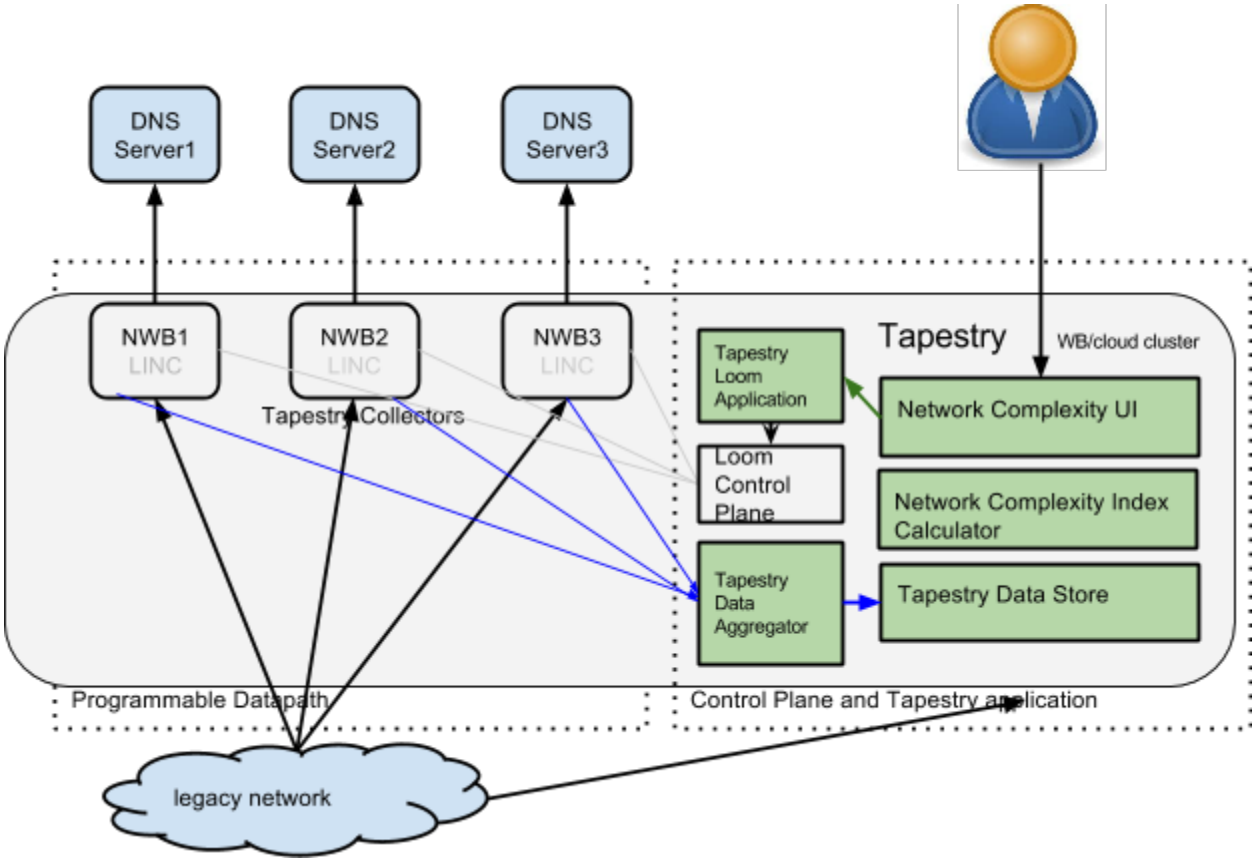


Figure 1

An example of how Loom is used is shown in Figure 1. Tapestry is a system that derives "network complexity" by analysing DNS requests made in a network. LINC switches running on Network White Boxes (NWB) are placed in the path to DNS servers and serve as network taps.

The switches are programmed by Loom with flow entries that forward the requests as before to the DNS servers, but also copy them to Tapestry Data Aggregator. Loom and Tapestry components all run on standard x86 Linux systems, also called White Boxes (WB).

The DNS data collected in the Tapestry Data Store is analyzed by a program called the Network Complexity Index Calculator that arrives at a “complexity” indicator. The data to be used for the analysis can be selected by using filters via a web-based GUI program called the NetWork Complexity UI.

Loom Block Diagram

Figure 2 provides a high level view of Loom.

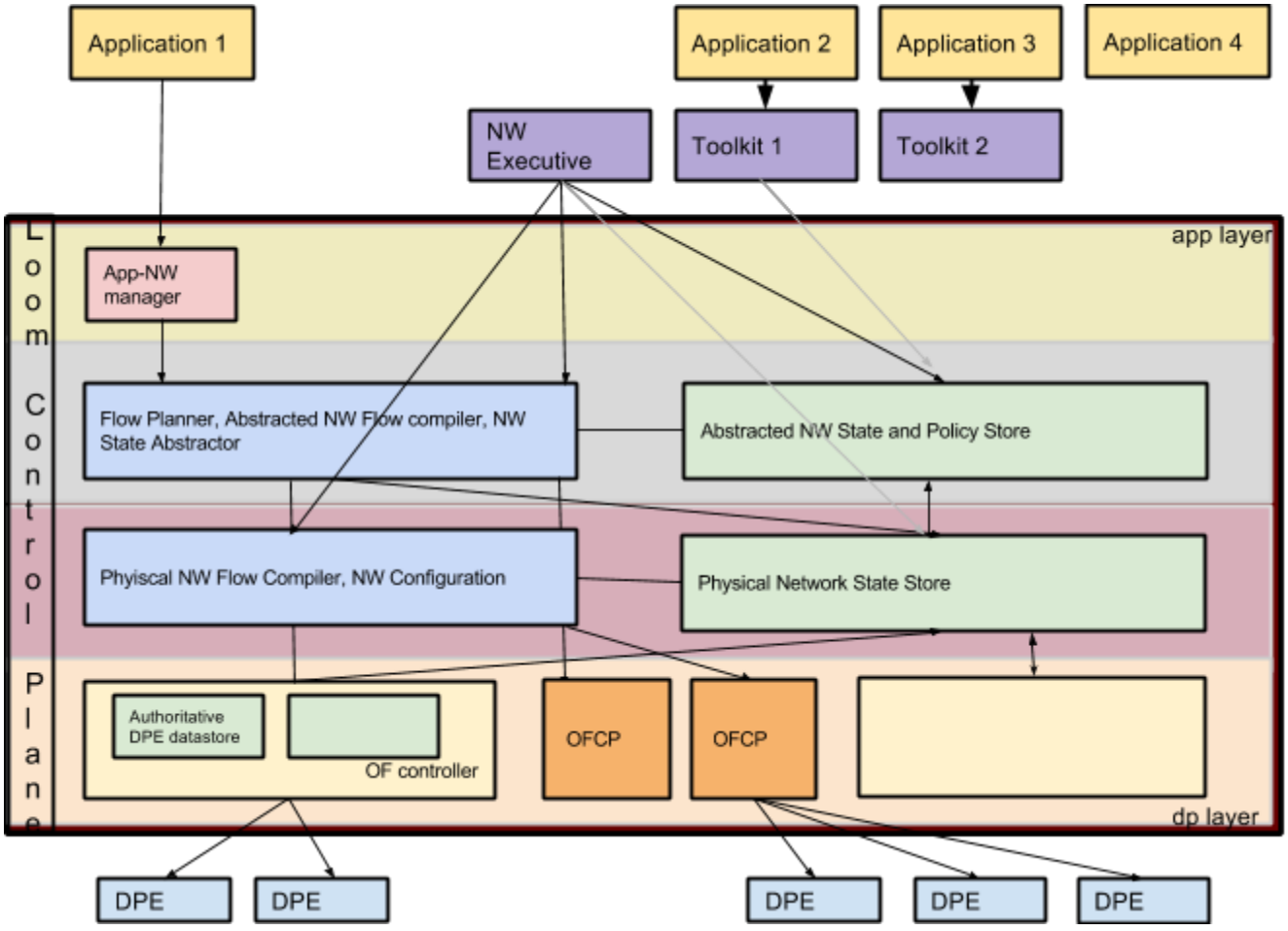


Figure 2

Loom Glossary

DPE: Datapath Element
OF Controller: Simple OpenFlow Controller that can construct and send OpenFlow messages to DPEs and receive OpenFlow messages from them.
OFCP: OpenFlow Configuration Point.
Authoritative DPE Datastore: This stores a DPE's configuration and flow table entries.
Network State Store/NIB: This stores the state of the physical network by gathering each datapath element, each link and network topology.
Abstracted NW State and Policy Store: This stores the network-wide policies and constraints and network state seen by all applications.
App-NW Manager: This gathers Application/Service Definition. This could be flow requirements: endpoints, type of traffic and priority and changes to traffic. This interacts with the Flow Planner to satisfy the application's requirements.
Flow Planner: Given an application's requirement for network resources or topology in terms of application flows and their type, and the configuration of the network, it arrives at the Network Definition. Then it uses the current state of the network to arrive at a route or network flow that can meet the requirement.
Abstracted NW Flow Compiler: Given a network flow and the state of the abstracted network, it generates flow entries to be sent to DPEs.
Physical NW Flow Compiler: This translates abstract flow-entries to the actual flow-entries to be sent to DPEs, based on the existing flow-entries.
Network Executive: Has knowledge of the network configuration and manages the selection of DPEs and ports to be controlled by an OF Controller. Gathers and implements network policies.

Layered Architecture

Loom consists of four main layers:

- At the bottom, the datapath layer of Loom interacts with the datapath elements. It consists of a number of OpenFlow Controllers each responsible for a set of datapath elements. Each OpenFlow Controller has an authoritative store for the flow entries and configuration of each datapath element. This layer also includes multiple OpenFlow Configuration Points (OFCP), each of which is responsible for a set of DPEs.
- The middle lower layer stores the global network state and configuration, and performs lower level compilation of flow entries.
- At the top, the application layer accepts and stores requests from the application for network resources.
- The middle upper layer provides an abstracted view of the network to the applications and stores network-wide policies and constraints. It is the brain of the control plane where network-flow plans are generated, network flows are compiled, network state is abstracted to what is required for each application and network configuration decisions are made.

Flow Design

Starting from an application's requirements, the flow design process involves several steps before there are flow entries that can be sent to the DPEs. Figure 2 provides an outline of the

steps and shows the different entities of taking part in flow generation.

The application specifies high level requirements called application-flows, which can be static or dynamic. An example application-flow is “Connect Host A to Host B with reliable connectivity”.

Flow Planner

The Flow Planner generates flows based on events such as:

- Change in application flows
- Network event such as port down, congestion detected etc.
- Packet-in from OpenFlow Controller

In SDN, the datapath elements are dumb and the route selection is done by the control plane. Since the routing algorithm is no longer embedded in black box end devices, there is no need to stick to one routing algorithm and metric for the selection of flow path. The metric itself does not need to be simple, because it is no longer computed by datapath elements for each packet. It can be as complex as needed to achieve the objective of path selection for a particular flow. It can be composed of different criteria. For example instead of selecting the shortest path, the requirement could be to select the shortest widest path. In addition, the routing algorithm used for path selection can be different for different flows.

For small networks, the use of ad-hoc methods to compute paths may work. But as networks grow it will become essential to use some formal methods to specify the flow routing requirements and arrive at routing algorithms suitable for those requirements.

For a very general solution, one possibility is to use something like Metarouting [1], a theoretical framework which provides the means of describing and capturing routes and comparing route preference in algebraic structures that have rigorously defined semantics. The project has developed routing algebra that will allow a routing algorithm designer to compose complex routing metrics from simpler ones and make use of simpler algorithms. From their website: “Routing algebras provide an abstract formalism which captures the policy component of routing protocols --- how routes are described, how best routes are selected, and how routing policies are defined and applied.”
[1] http://www.cl.cam.ac.uk/~tgg22/metarouting/. This is work in progress, and a beta version of the framework has just become available in OCAML.

NOTE: Flow planning and compilation functions need not always be fully centralized, requiring packets being brought to a central location and then a decision made about how to route them. Some functions can be distributed close to the datapath, but it does not mean that we have to use fully decentralized distributed algorithms either. For example, what to do on a link failure may be handled at the edge, by an edge-component of Loom.

Flow Compiler

Using the information in network state store and the application level NW state and policy store,

the Flow Compiler compiles the network flows into flow entries that need to be sent to the DPEs [Figure 3]. These flow entries are sent to the the OpenFlow controller which speaks OpenFlow protocol and interacts with the DPEs to send flow entries and retrieve DPE state.

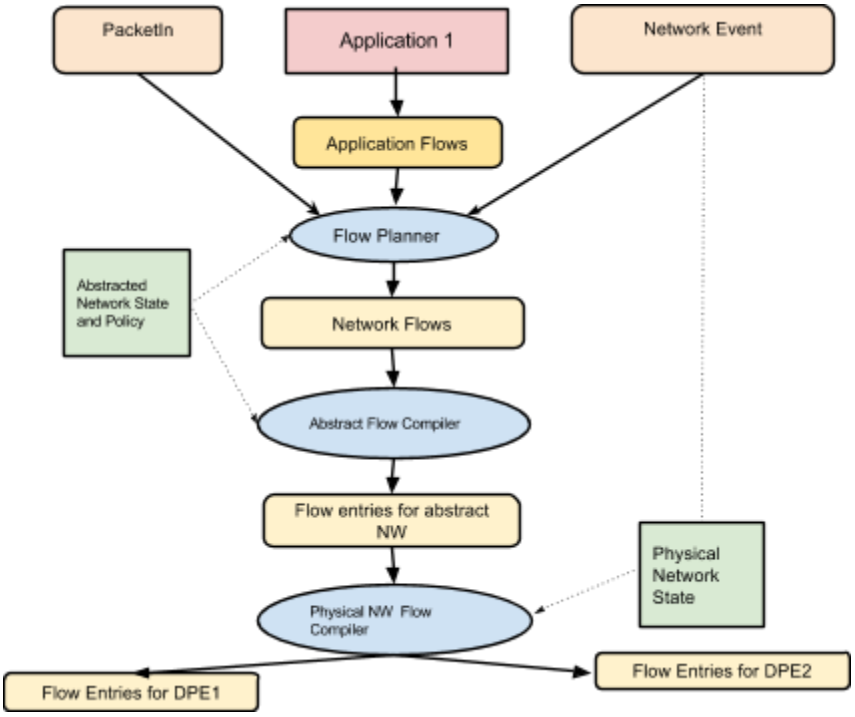


Figure 3

The Flow Compiler eliminates the need for the application developer to arrive at flow entries for each DPE separately and be concerned with how these entries interfere what already exists at the DPEs. In networks with large number of DPEs, without the Flow Compiler, the task of generating flow entries becomes too complex, resulting in using simplistic flow entries, and not really exploiting the power and freedom of being able to program each element separately.

We are exploring the Frenetic project [2] to understand and apply their work in the development of the Flow Compiler.
[2] http://www.frenetic-lang.org/overview.php

Loom as a Distributed System

Loom is designed as a distributed system for the following reasons:

- Fault Tolerance
    - High Availability

        - Synchronous replication of state and data
        - Only one slave hot standby (more will be slow and complex)
        - Replica located close by in the same “site”
        - Recovery is not an issue as the original master becomes the new slave
        - Addresses system and software failure
    - Disaster Recovery
        - Asynchronous replication of state and data with eventual consistency.There could be some loss of state and data due to the asynchronous nature of update.
        - N number of slaves (master candidates) that can be placed at different “sites”.
        - If the master fails then one of N slaves/ master candidates at a different “site” is chosen as the new master
        - Requires to be followed by remote recovery
        - Addresses site failure
    - Remote Recovery
        - Transfer of master ownership back to the recovered master site
        - In the most general case different components of a site can be placed with master candidates at different sites.

- Performance
    - If a component of loom can be placed at the edge, then some latency sensitive functions can be performed there.
    - Example: Rerouting- If there are multiple output ports on a DPE, and one outgoing link fails, then move all flows to the second link.
- Scale
    - Large number of DPEs may swamp the central controller. If some local function is clearly known and authority for that can be handed over, then having a distributed component of the logically centralized controller can address scaling issues.
- Use case complexity
    - SDN control plane may have many different kinds of applications and users which are potentially in different locations and within different organizational groups.

Scale of Loom

Loom distributed system is expected to grow to run on tens of thousands of “sites” and hundreds of thousands of instances of “applications”. A “site” can be an Erlang node, a VM running in a hypervisor, a cluster or a data center.

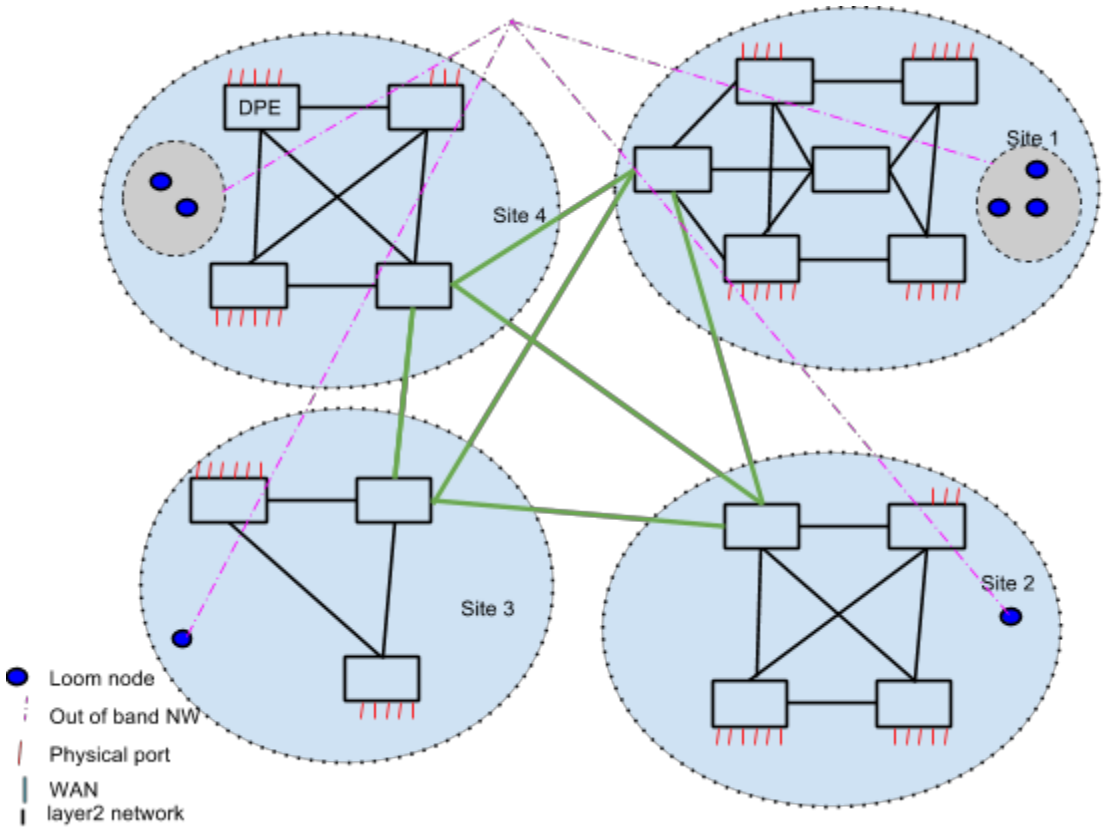
Note: The expected scale appears large, but current customer deployments have sites in the thousands.

Network Executive

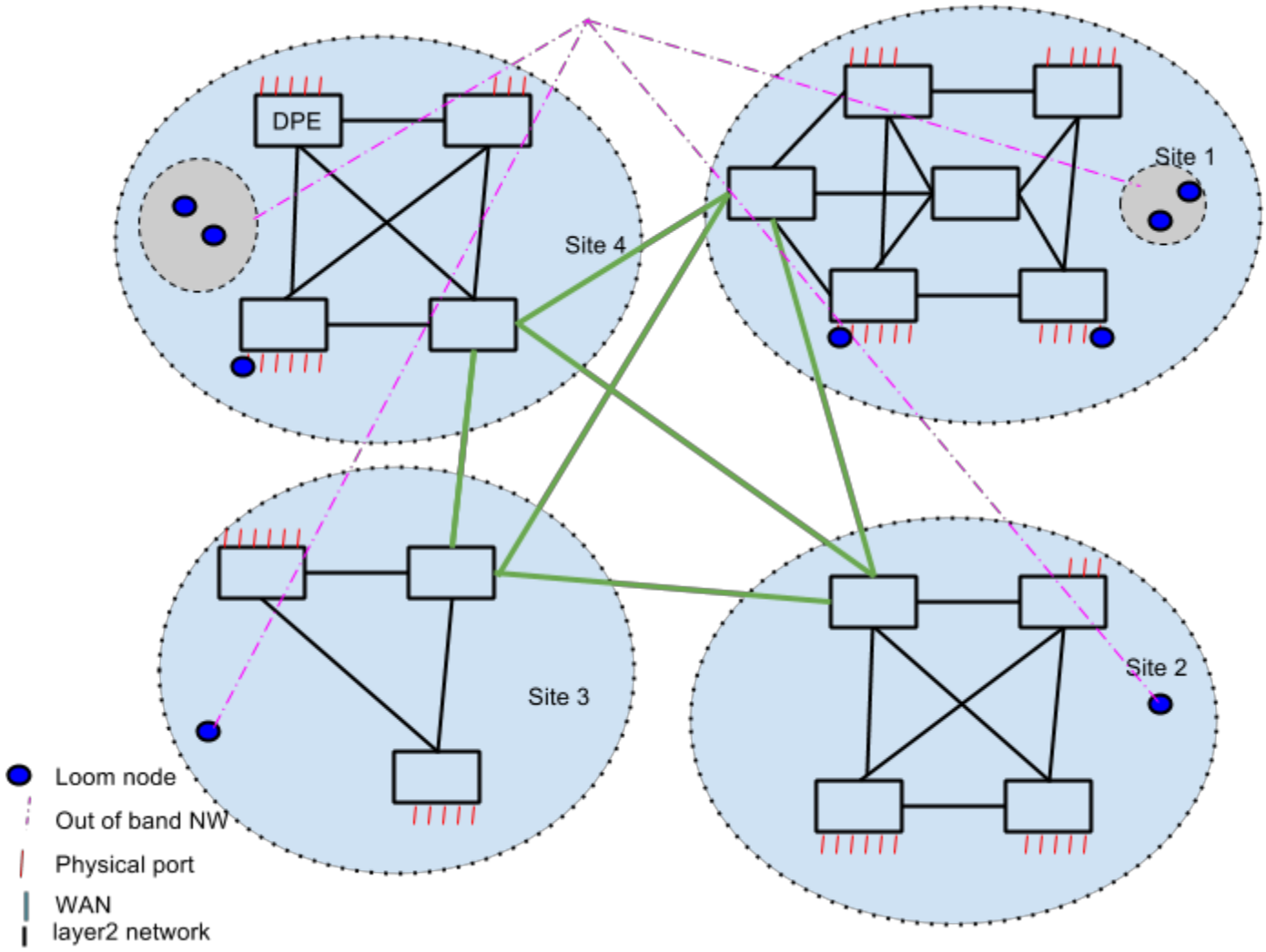
- has complete configuration information of all DPEs
- knows state of all DPEs (eventually consistent state)
- does not have or need flow level knowledge
- controls the placement of DPEs on controllers using OFCP, based on requirements of applications and organizational groups.
- starts/stops new controllers.
- controls placement of OpenFlow controller functions closer to DPEs and authorizes the transfer of authority for those functions to the local-OF controller process located at a DPE. The interaction between the local-OFC and the DPE still remains OpenFlow.
- If the DPE is LINC, then local-OFC can make use of more information than what can be collected by OpenFlow.
- controls the placement of some functions of the OFCP closer to the DPE and authorizes the transfer of authority for that function to the local-OFCP process running near the DPE.

Network Topology

The network topologies of the Google's SDN and other similar SDN deployments are constrained to be simple and very tightly controlled. Google's internal SDN is fully controlled by Google, with DPEs, network topology, control plane and applications all developed by Google. Loom does not have such advantages of simplification and control. The deployment can be across a lot more sites than Google's data centers (e.g. Best Buy locations and data centers). The sites can have different topologies and organizational control. There is no regularity or pattern that can expected in customer deployments. The Figure below shows an example network controlled by Loom. The Loom control plane is composed of Loom nodes (blue) which are connected by an out of band network (pink). The OpenFlow datapath elements are shown with different number of ports (red) and are connected in different topologies. The sites are connected connected by WAN links shown in green. The Loom SDN control plane controls the entire network across many sites.



In the Figure below, some Loom nodes run close to the datapath, i.e. on the same system where the DPE is running. In the most general case, there is no out-of-band connectivity for Loom. Instead Loom nodes are just like other host machines connected to the DPE ports.



Data Stores

The green boxes in Figure 2 incorporate some form of a data store. Each of them has a different purpose and characteristics.

- The network state store keeps the state of all the switches in the fabric. The stored data is not expected to provide a consistent view, and it can be brought closer to the actual state based on how often it is refreshed. The network state can include other computed aggregates.
- Authoritative DPE Datastore: This stores a DPE's configuration and flow table entries.
- Network-wide policies and constraints as well as network-wide application stats are kept in the Abstracted NW State and Policy Store.

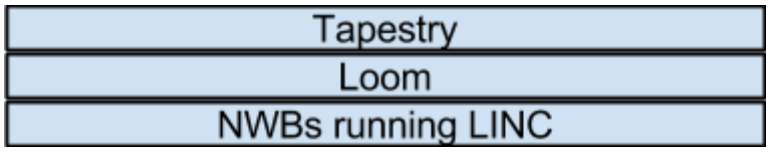
Authoritative DPE Datastore

This store is Loom's representation of configuration, flow entries, groups meters that exist at a switch and supports multiple versions of OpenFlow. For fault tolerance purposes this has a local hot standby and N remote replica stores.

Network State Store/NIB

This stores the state of the physical network by gathering each datapath element, each link and network topology. A graph database that maintains relationships and has pub-sub capabilities (e.g. IF-MAP server) could be used here.

Erlang Releases, applications, modules and libraries



This section will capture structure of the code in terms of releases, applications, modules and libraries.

Tool Kits

- Debugger
- Network Monitor
- Testing Framework
- GUI

Applications

- Tapestry
- OpenFlow Network for Erlang
- Riak
- Distributed MapReduce
- Network middlebox functions
- Traffic engineering