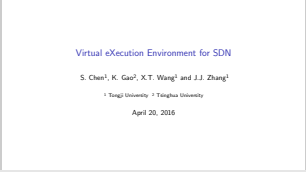
Virtual eXecution Environment for SDN

S. Chen[1], K. Gao[2], X.T. Wang[1] and J.J. Zhang[1]

[1] Tongji University [2] Tsinghua University

April 20, 2016

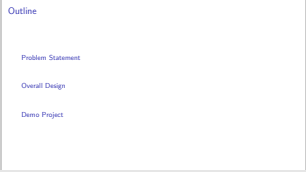
Virtual eXecution Environment for SDN



Hello, everyone! Nice to meet you all!

I'm Kai, a Ph.D student at Tsinghua University and my teammates are Shenshen, Tony and Jensen, who are all from Tongji University. Today we are going to present the *Virtual eXecution Environment* for SDN programming, which is aimed to solve the data consistency problem, and to simplify programming in SDN.

Outline



Here is the outline of our presentation.
First we discuss a little bit about what kind of problems we are trying to solve, and why current solutions are not good enough.
Second we provide an overview of our proposal and show what we have implemented during the ONUG competition.
Finally we walk through the code and present a demo of the system.

Problem Statement

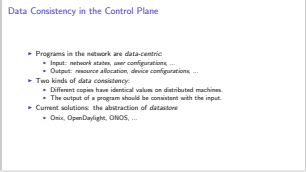
- Data consistency in the control plane
- Complexity in SDN programming

Our group has been working on the problem of SDN programming models, in the sense that we want to understand how SDN programming differs from generic programming by identifying domain-specific features, and to design a programming framework that satisfies the demand of correctness, efficiency and simplicity.

In this presentation we discuss mainly about two issues that are identified: the data consistency in the control plane and the programming complexity in modern SDN systems.

Data Consistency in the Control Plane

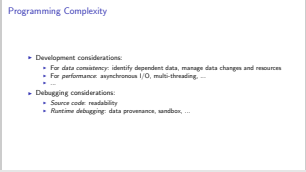
- Programs in the network are *data-centric*:
  - Input: *network states*, *user configurations*, ...
  - Output: *resource allocation*, *device configurations*, ...
- Two kinds of *data consistency*:
  - Different copies have identical values on distributed machines.
  - The output of a program should be consistent with the input.
- Current solutions: the abstraction of *datastore*
  - Onix, OpenDaylight, ONOS, ...



Most SDN applications depend on certain network states or user configurations and their final output includes resource allocations and forwarding rules on devices. All these input and output are modelled as *data* and the problem of data consistency has two meanings:
First, the data should be synchronized on different machines. Most modern controllers have provided mechanism such as distributed data base to solve this.
Second, the output of an application should be consistent with the input, which means if the input of the application changes, the output may need to be updated.
Modern SDN controllers have provided the abstraction of datastore to help solve these two consistency problems. However, the mechanisms they provide are low-level operations and make SDN programming complex.

Programming Complexity

- Development considerations:
  - For *data consistency*: identify dependent data, manage data changes and resources
  - For *performance*: asynchronous I/O, multi-threading, ...
  - ...
- Debugging considerations:
  - *Source code*: readability
  - *Runtime debugging*: data provenance, sandbox, ...

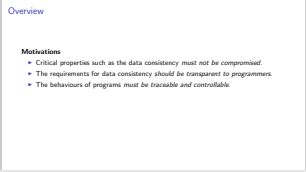


The complexity of SDN programming can affect the following aspects:
First is the development.
For example, to guarantee the data consistency we talked about in the last slide, programmers must identify dependent data in the program, register listeners to manage data changes, and be very careful with allocated resources. Common techniques such as asynchronous I/O and multi-threading may also lead to programming complexities.
Second is debugging.
Programming complexities often result in difficulty in reading the source code, which makes it extremely hard to debug. Also when a new program is being tested in a real network, we want to make sure its consequence is traceable and controllable.

Overview

Motivations

- Critical properties such as the data consistency *must not be compromised*.
- The requirements for data consistency *should be transparent to programmers*.
- The behaviours of programs *must be traceable and controllable*.


Overview

Motivations

- Critical properties such as the data consistency *must not be compromised.*
- The requirements for data consistency *should be transparent to programmers.*
- The behaviours of programs *must be tracable and controllable.*

Our project is motivated by the following conclusions:
First, data consistency is an important property in the control plane and should not be compromised.
Second, identifying data dependencies, as well as dealing with data changes or managing allocated resources, can and should be handled automatically, beyond the programmers' concern.
Finally, for debugging and security reasons, it is important to trace and to control the behaviours and possible consequences of SDN programs.
Thus we propose the VXE system, hoping to satisfy all the three demands. What we want to provide, is a simple API for the SDN programs to get the access to the datastore, and a powerful runtime where system requirements, such as the data consistency, can be enforced.

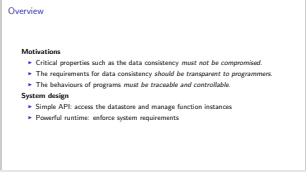
Overview

Motivations

- Critical properties such as the data consistency *must not be compromised*.
- The requirements for data consistency *should be transparent to programmers*.
- The behaviours of programs *must be traceable and controllable*.

System design

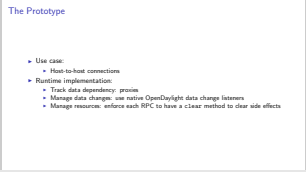
- Simple API: access the datastore and manage function instances
- Powerful runtime: enforce system requirements



Our project is motivated by the following conclusions:
First, data consistency is an important property in the control plane and should not be compromised.
Second, identifying data dependencies, as well as dealing with data changes or managing allocated resources, can and should be handled automatically, beyond the programmers' concern.
Finally, for debugging and security reasons, it is important to trace and to control the behaviours and possible consequences of SDN programs.
Thus we propose the VXE system, hoping to satisfy all the three demands. What we want to provide, is a simple API for the SDN programs to get the access to the datastore, and a powerful runtime where system requirements, such as the data consistency, can be enforced.

The Prototype

- Use case:
  - Host-to-host connections
- Runtime implementation:
  - Track data dependency: proxies
  - Manage data changes: use native OpenDaylight data change listeners
  - Manage resources: enforce each RPC to have a `clear` method to clear side effects



In the ONUG competition, we have followed the design principles and have implemented a simple version of the VXE system to demonstrate the basic workflow. This prototype system is built on top of the OpenDaylight controller and we have implemented an application, which calculates the shortest path between two specific end hosts and sets up the path using OpenFlow.

We have also implemented a graphical user interface to help observe the results.

Demo

THANK YOU

Q & A

That's all for our demonstration, thank you all for your time.
We'd like to get feedbacks and to answer any questions.