# Ravel: Orchestrating Software-Defined Networks

Anduo Wang Brighten Godfrey Matthew Caesar
*University of Illinois at Urbana-Champaign*

## 1. INTRODUCTION

Software-defined networking (SDN) use cases commonly involve multiple applications that collectively drive the network. The use cases are growing bigger and more complex, raising the need to break them into more manageable and reusable small pieces, as the design principle of separation of concerns suggests. Simultaneously, new business models [2] are driving the use cases beyond the control of one single administrator. For example, providers that lease cloud computing and network functions virtualization (NFV) are infrastructure services that host multiple user applications. The first SDN App store, recently announced by HP, is promoting SDN innovations that span over multiple applications contributed by different parties.

These applications can form complex behaviors due to interactions and interleavings that depend on the runtime dynamics. For example, the action of blacklisting a path by an access control application should invoke proper response by the routing application, such that the newly prohibited path is avoided. In addition to coordination, conflict occurs when applications disagree on what states are allowable: a lightly used path favored by the load balancer application could be rejected as unsafe by access control.

An operator could manually construct a master program — composition of the component applications — that statically specifies the inter-application interactions [3]. This method requires the control of a single administrator, and requires the components to commit to a fixed interface. However, the SDN ecosystem today has to accommodate use cases built from programs of disparate sources. Likewise, privacy and trust concerns hinder the naive composition approach. Besides, the master program is tightly coupled. Every time a new component is introduced, or an existing one is modified, the master program, as well as all relevant components need to be revised and tested. Programming interoperating software, in general, is expensive and challenging.

We propose a complementary solution — *data-centric orchestration* — that coordinates runtime interleaving by coordinating data access. The data-centric solution is based on a database-centered SDN design, inspired by a transition in online commercial data management in the 1980s. At that time operating system and programming language techniques were proven to fall short, and gave rise to database systems [1], which later developed into the solution for coordinating data access and mediating between multiple users. Likewise, we shift to a database-centered design of SDN that uses a standard SQL database. While the relational data representation leads to a clean orchestration semantics, the SQL language, together with the extensions of rule and trigger, enables powerful interoperability with existing management techniques and tools.

Specifically, we present Ravel, a database-centric solution to orchestrating SDN applications. Ravel takes the entire SDN network under the hood of a standard SQL database, including network configurations as stored (base) tables in the data-store, and network controls as virtual views that are derived from the store. In Ravel, applications are programs that interact with the network via its dedicated virtual view. A view is a specification that is queried on the fly: a SQL query produces the view from other views and/or the base. For example, the access control application is a program that reads and writes following view:

```
CREATE VIEW acl AS (
       SELECT src, dst
       FROM data_plane_base_tables ...);
```

Ravel views are extremely flexible and powerful. Applications can use such the view interface to introduce ad-hoc abstractions. The administrators can use views to restrict what an application sees. For example, the administrator can expose to an external application an aggregate view, a "summary" that hides the underlying sensitive data.

This relational representation leads to a clean orchestration semantics, which is implemented by two Ravel services: vertical and horizontal orchestration. Vertical orchestration coordinates applications by synchronizing their views. The enabling technique is data synchronization that keeps the views consistent as the top-level application writes its view and/or underlying network state changes. Horizontal orchestration, on the other hand, uses a priority-based data-sharing protocol that allows applications to act autonomously while live in harmony: Upon the write initiated by an application, the protocol engages all applications with higher priority for necessary responses, producing a collection of updates. These updates are then combined and populated to all applications with lower priority.

## 2. OVERVIEW

Figure 1 (left) shows the three main components of Ravel: the users, the database, and the network. The network component is the resources and services being controlled. The users are a collection of applications that are controlling the network component. An application can be a reactive program, or a human operator who manually responds to network notifications. Each application serves a particular purpose, such as routing and load balancing. The database component sitting in the middle is the engine that powers Ravel. First, The user-facing interface is ad-hoc extensible: the applications can contribute new network abstractions called views on the fly, and make them available to others. The views accept application updates , which are then translated by the database into proper network updates. Similarly, the

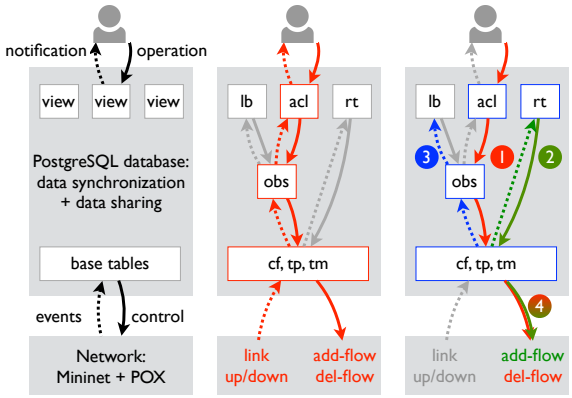

Figure 1: Ravel overview

network-facing of the database component features a uniform abstraction called the base tables, whose purpose is to hide implementation details. The base table interface collects network events, which are then transformed into updates on all the affected applications.

The ultimate goal of the Ravel database component is to allow multiple applications to control the network via their own user-defined views without worrying about interactions among them. To achieve this, Ravel offers two orchestration services shown in Figure 1 (middle and right). Figure 1 (middle) shows shows vertical orchestration. It synchronizes the high-level application views and the actual network state, thus allowing the individual application to control the network via high-level operations over the views. The two data synchronization mechanisms we utilize are view maintenance (dashed lines) and view update (solid line).

Figure 1 (right) shows horizontal orchestration. It coordinates operations of multiple applications with a priority-based (pre-determined) data sharing protocol. The protocol polices read and write on the shared table: Upon a write request (❶), the protocol engages those with higher priorities (❷), and lets them check the proposed updates and make modifications if necessary. The combined effects (❷, ❸) are populated back (❸) to lower ranked applications. After the orchestrated update transaction (❶ ❷ ❸) completes, the merged control is finally committed to the network (❹).

## 3. DETAILS OF DEMONSTRATION

We built a prototype of Ravel using PostgreSQL [6] database in Mininet [4] virtual machine. We choose PostgreSQL because, as one of the most advanced and popular database for both academic and commercial purpose, it is a highly customizable database with many features (e.g., pgRouting [5] for networking).

The Ravel prototype consists of 1000+ lines of SQL code and 100+ lines of Python code, and can be divided into four parts: the code for the shared network states, those for individual applications (routing, load balancer, access control, and tenants), orchestration (the vertical, horizontal orchestration and the combination of the two), and connection with

| Ravel component | Lines (#) | PostgreSql features |
|---|---|---|
| network base | 120 | SQL |
| routing (rt) | 230 | SQL, rule, trigger |
| load balancer (lb) | 30 | SQL, rule |
| access control (acl) | 20 | SQL, rule |
| tenant (rt, lb, acl) | 130 | SQL, rule, trigger |
| orchestration | 200 | rule, trigger |
| Mininet connection | 260(SQL)<br>102(Python) | SQL, PL/Python |

Table 1: Summary of Ravel components

Mininet. Table 1 summarizes for each Ravel components the number of lines of code, and the PostgreSQL features used.

In this demo, we show Ravel under three scenarios: First, we show the primitive operations of Ravel for individual applications, that is, the basic functionality of querying views and updating network states. We show how to use the Ravel database to collect the network statistics such as loads on a certain server. We also demonstrate how individual applications modify the Mininet configuration with Ravel database updates. In the next two scenarios, we show how Ravel orchestrates the operations in scenario 1. In vertical orchestration, we demonstrate how various applications control a virtual network. In horizontal orchestration, we show how the applications interact under the supervision of Ravel.

## 4. REFERENCES

[1] Abiteboul, S., Hull, R., and Vianu, V., Eds. *Foundations of Databases: The Logical Level*.
[2] HP SDN App Store - Transform Network Architecture. www.hp.com/sdnappstore.
[3] Jin, X., Gossels, J., Rexford, J., and Walker, D. Covisor: A compositional hypervisor for software-defined networks. In *NSDI* (2015).
[4] Mininet. http://mininet.org/.
[5] pgRouting Project. http://pgrouting.org/.
[6] PostgreSQL. http://www.postgresql.org.