Reflections on Data Integration for SDN

Anduo Wang
Temple University
adw@temple.edu

Jason Croft
University of Illinois at Urbana-Champaign
croft1@illinois.edu

Eduard Dragut
Temple University
edragut@temple.edu

ABSTRACT

Software-defined networking (SDN) provides an unprecedented opportunity to exercise computing principles in networking practice. This paper investigates *data integration*, a under-explored discipline from the database community. We propose to manage the various SDN control applications that collectively drive a shared dataplane using a data integration system. First, we develop a baseline design and study its feasibility on two networking challenges not adequately addressed in classic data integration systems: the extensibility requirement to add new controls on demand, and the performance requirement to cope with fast dataplane updates. Central to our baseline design is a relational model, where the entire SDN is represented as relational data (values in tables) with distinct roles. The control applications act as data sources generating network state, and the dataplane becomes the integrated whole. Based on this model, we explore extensions to data integration systems called behavioral dependency, a formal notion that captures the dynamic interactions between the control applications. While our design and extension are not intended to provide a comprehensive solution, we believe our study is a step toward reaping the benefits of data integration for SDN.

1. INTRODUCTION

In recent years, the networking community has looked to principles from the fields of programming languages, operating systems, and distributed systems to design software-defined networking's (SDN) salient features: network-wide control and higher-level abstractions. For example, programming language principles have enabled higher-level programming abstractions and highly optimized compilers [28, 30, 25]. Modular programming and functional constructs [11, 27, 23] have raised the level of abstraction in constructing forwarding applications. More advanced data structures such as automata and graphs [16, 26] have been proposed for network dynamics and service chains. Simultaneously, operating system and distributed system principles [13, 17, 3, 22, 5] have been used to create network-wide control. By offering network-wide structure and separating distributed state management from individual application control logic [6, 31], the complexity of adding new features is drastically reduced.

Indeed, these principles have resulted in improved management of a single, monolithic control application. However, their effectiveness in *integrating control applications*, i.e., combining a collection of potentially overlapping and conflicting networking controls into coherent forwarding behavior, is less evident. For example, with higher-level modular programming [15, 26, 16, 24], the operator must carefully construct a master program that foresees all possible control module interactions. This master program is restricted to applications that affect the same shared traffic. To handle applications over shared network resources, a complementary approach [31] is to detect and resolve conflicts via state management of the shared resources. The integration support then relies on hardwired state and dependency models. Therefore, integration support derived from programming language, operating system and distributed system principles is often too limiting and/or inflexible.

As a means to tackle this *network integration problem*, we propose an investigation into an under-explored discipline to influence SDN design: *data integration* [18, 10]. By building on principles from database and data integration research, we argue we can formulate many network management tasks as data integration problems of merging data sources into an integrated whole. In our formulation, we model the entire network as relational data, with values stored in tables. We assign distinct roles, where the control applications serve as the individual data sources that produce network data, and the dataplane becomes the integrated whole. We demonstrate the feasibility of data integration through a baseline design tailored for two basic yet challenging networking requirements: extensibility and performance. In addition, we examine the opportunities of this approach toward solving a long-standing networking problem: dependency discovery between complex control modules.

The baseline design is derived from two networking requirements: (1) the system extensibility requirement to accommodate an enlarging body of control applications, and (2) the performance requirement to cope with fast dataplane updates under complex control applications. The former entails a *local-as-view* (LAV) design of the source-global mapping where the source applications are described as *views* (virtual tables), implemented as queries over the dataplane tables. In LAV, adding a new control application simply requires extending the source-global mapping with a new descriptor and no additional changes. The latter requirement translates into the challenge of write-dominant data integration under complex integrity constraints. Our baseline addresses this challenge by altering the default management of constraints in a database system. Rather than checking for violations at each write — a strategy that incurs non-negligible overhead as constraint complexity increases — our

SDN-NFV Sec'17, March 22-24, 2017, Scottsdale, AZ, USA
© 2017 ACM. ISBN 978-1-4503-4908-6/17/03…$15.00
DOI: http://dx.doi.org/10.1145/3040992.3041006

baseline attempts to avoid network updates that might cause a violation.

Looking forward, a key challenge of network integration not addressed by data integration is that networking data sources (i.e., control applications) are "active" entities interacting in complex ways. That is, their actions (i.e., network updates) depend on each other [1, 8, 14]. For example, a security application may rely on a separate routing application to remove an unsafe path. To formally capture such interactions, we developed *behavioral dependency*, an extension to classic data integration. Moreover, by building on *irrelevant updates* research [20, 4], we formulate behavioral dependency as a satisfiability problem that can be automatically solved using a SAT solver [33]. The automatically derived dependency can be used to combine the behavior of multiple applications into an orchestrated update. Each step of the update is executed by an individual application, followed by the applications it behaviorally depends on, and followed by those that depend on it. Intuitively, this orchestrated update simulates a master program that instructs the behavior of multiple applications that collectively drive a single network — a long-standing networking problem [24, 12].

In sum, our system design and extension do not intend to provide a comprehensive integration solution. Rather, this paper seeks to open a discussion on the long-standing network management problem in the SDN era from a data integration perspective. Our discussion is forth-looking and speculative, exploring the following questions for the benefit of the broader SDN community: Are data integration principles applicable to SDN? Can we customize and/or extend data integration to help long-standing networking problems? Can a data integration perspective lead to exciting new use cases?

2. FROM DATA INTEGRATION TO NETWORK INTEGRATION

2.1 Data integration

Data integration [18, 32, 19] examines the problem of combining data from a variety of sources to form a new, unified whole. More formally, a data integration system is defined as $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, with a global schema $\mathcal{G}$, semantic mappings $\mathcal{M}$, and data sources $\mathcal{S}$. The system exposes to users a single and coherent interface, through a mediated global schema $\mathcal{G}$. Users can query and access data from local sources while the system hides the details and inconsistencies of the constituting sources. Semantic mappings $\mathcal{M}$ specify the relationship between the schemas of the data sources $\mathcal{S}$ and the global schema $\mathcal{G}$. The semantic mappings allow answering queries over $\mathcal{G}$ using queries over $\mathcal{S}$. We consider data sources that are *relational*.

There are two approaches to (virtual) data integration, which differ in the way $\mathcal{M}$ is defined: *global-as-view* (GAV) and *local-as-view* (LAV). In GAV, the global tables are defined in terms of the local tables. Each global table is thus a *view* of the local tables. In LAV, $\mathcal{M}$ is defined in a somewhat dual manner: the local tables are defined in terms of the global tables. Hence, each local table is a *view* of the global tables.

A important problem in a data integration system is the reconciliation of inconsistencies between the data sources. Inconsistencies are captured by integrity constraints that specify what data instances are acceptable. Integrity constraints can be specified directly over the global schema, or expressed on the individual sources and eventually expanded and unfolded on the global schema. The constraints can accelerate data access (i.e., reads) in a read-dominant $\mathcal{I}$, but they deteriorate $\mathcal{I}$'s performance when faced with frequent updates (i.e., writes) due to the overhead of checking the integrity constraints.

2.2 Network integration

In SDN, *network integration* is the problem of enabling multiple independently created, potentially overlapping and/or conflicting control applications to collectively drive a network in a coherent manner. By modeling the entire network state (i.e., the control applications and dataplane) as relational data, we can cast this problem to a data integration problem and define a network integration system as $\mathcal{I}^{\mathcal{N}} = \langle \mathcal{G}^{\mathcal{N}}, \mathcal{S}^{\mathcal{N}}, \mathcal{M}^{\mathcal{N}} \rangle$. The global schema $\mathcal{G}^{\mathcal{N}}$, augmented with integrity constraints, characterizes a consistent dataplane. The source schemas $\mathcal{S}^{\mathcal{N}}$ describe network states contributed by independently operated control applications. The goal of network integration is to find a mapping $\mathcal{M}^{\mathcal{N}}$ that synchronizes between the control applications' states and the network dataplane under the integrity constraints.

While network integration shares a similar goal with data integration to merge sources into a coherent global, the two problems have a different focus. Data integration systems today are often optimized for read-only queries or read-dominated global instances [29] under simple integrity constraints. The need to check for constraint violations on every write, however, drastically hurts the performance of these systems for intensive writes. Moreover, the interactions between the sources is limited to those expressed as constraints on the global data. In contrast, a network integration system requires a much more difficult and complex problem with peculiar characteristics. *A network integration system must enable frequent writes on the global data under arbitrary constraints, and directly support complex interactions and dependencies between the sources.*

In this paper, we develop a baseline design to study two critical network requirements not adequately addressed in existing data integration systems:

Fast updates of global data under arbitrarily complex integrity constraints. The dataplane (the global database with schema $\mathcal{G}^{\mathcal{N}}$) has to support arbitrary integrity constraints to reflect the logic of the control applications and simultaneously accommodate real-time dataplane updates, i.e., writes to $\mathcal{G}^{\mathcal{N}}$. A network integration system must address the difficult and competing requirements of fast turn around time for frequent writes and checking complicated integrity constraints.

Behavioral dependency between sources. The control applications (data sources $\mathcal{S}^{\mathcal{N}}$) collectively maintain a consistent network dataplane and are involved in complex interactions. To consistently update the dataplane, individual application updates must be orchestrated. A network integration system needs to resolve conflicts between overlapping applications and streamline updates for collaborative applications.

3. BASELINE DESIGN

In our baseline design of network integration system, all information is represented by values in tables, including those contributed by control applications and the global dataplane. The global dataplane $\mathcal{G}^{\mathcal{N}}$ has three relations:

```
topology(node_id, node_id, ...)
configuration(flow_id, switch_id, next_id, ...)
reachability_matrix(source, destination, ...)
```

The table `topology` stores link pairs `(node_id,node_id)`. `reachability_matrix` stores the reachability information from `source` to `destination` for each flow, as well as the live traffic requirement in the network. `configuration` stores the entries of a flow (the next hop `next_id` for a switch `switch_id`). Intuitively, `configuration` stores per-switch configurations.

With this design, the control applications serve as the data sources. Each has a partial view of the global schema of $\mathcal{G}^{\mathcal{N}}$ and controls a portion of $\mathcal{G}^{\mathcal{N}}$'s data. For example, consider the following schemas for three typical SDN applications:

```
routing(source, destination, path, ...)
load_balancer(server_id, load, ...)
firewall(source, destination, ...)
```

Integrity constraints capture the meaning of the network — the perceived acceptable states, and the intention or policy governing each control application. An integrity constraint is an arbitrary logic assertion over a schema. In our baseline design, these are defined over schemas in $\mathcal{S}^{\mathcal{N}}$. Ultimately, all constraints are mapped to $\mathcal{G}^{\mathcal{N}}$ via $\mathcal{M}^{\mathcal{N}}$.

3.1 System extensibility for on-demand source control

A key design component of a network integration system $\mathcal{I}^{\mathcal{N}}$ is the specification of the mapping $\mathcal{M}^{\mathcal{N}}$ between the source applications $\mathcal{S}^{\mathcal{N}}$ and the global dataplane $\mathcal{G}^{\mathcal{N}}$. There are two approaches to define $\mathcal{M}^{\mathcal{N}}$ (§2): GAV and LAV. GAV is simpler to implement, but less flexible in accommodating new sources. Adding a new data source may require revisions of the global schema and mappings between the global schema and source schemas. LAV favors system extensibility at the expense of harder query processing over the integrated whole. However, in many practical scenarios, such as the one presented in this paper, query processing can be accomplished efficiently [9].

We take the position that system extensibility is the more valuable property in network integration. The networking community today witnesses little support for extending the control plane without explicit global coordination [15, 5, 31, 21]. The problem of compiling individual control applications into the dataplane configuration, on the other hand, is better understood and seems to be more tractable [26, 16, 28, 27, 24, 11]. Thus, we focus on system extensibility and adopt the LAV approach to achieve it.

In the relational model, LAV specifications of data sources are simply SQL queries over the global schema. In network integration, the query extracts information relevant to the application's control. For example, the following query defines `load_balancer`:

```
CREATE VIEW load_balancer AS(
       SELECT node_id AS server_id,
              count(*) AS load
       FROM reachability_matrix)
```

The load balancer application monitors the network state by checking the contents of the `load_balancer` view. More importantly, as a data source, when overload occurs, it modifies and generates new network state by updating the view and pushing the changes to the integrated dataplane.

3.2 Fast dataplane updates under complex network control

The second important design decision is modeling the basic networking requirement: the network is under continuous updates from all control applications. Network updates correspond to a set of writes over the global dataplane schema. Each write must be validated against the integrity constraints that define the intention, or policy, of the control applications. As a result, the networking requirement translates into the following data integration challenge: *coping with frequent writes under arbitrarily complex integrity constraints*.

By default, an integration system checks each write against all the integrity constraints. If a constraint violation is detected, the integration system will roll back and abort the write. As the complexity of control applications increases, so does the computational complexity of their integrity constraints. Combined with an increasing number of writes, the overhead for constraint checking will quickly make the performance of the integration system impractical. In fact, most integration systems are read-dominant, supporting simple constraints that can be efficiently checked (e.g., inclusion dependency, or whether the set of values in attribute A are a subset of attribute B).

Rather, our baseline design disables the default integrity constraint checking in a database system and pursues a new approach. Our design avoids violations by allowing only constraint-compliant writes, employing *violation views* as a means to specify network constraints. A violation view is a SQL query that declares how to extract information that violates the desired network properties. For example, the violation view for `load_balancer` contains servers with load exceeding threshold t:

```
CREATE VIEW lb_violation AS (
  SELECT * FROM load_balancer WHERE load >= t);
```

With violation views, regardless of the complexity of the network policies, the network data remain as plain relational data. To maintain constraint compliance on this plain network data, rather than checking the emptiness of the violation after each write, we assume that the network updates are issued in an intelligent way such that they are free of violation.

4. FUTURE WORK: MANAGING BEHAVIORAL DEPENDENCY

Looking forward, we investigate the benefit of data integration in tackling a long-standing networking problem: managing interactions between multiple control application [2, 8, 1, 14]. As a first step, this section outlines possible extensions to our NIS baseline design.

With multiple applications jointly driving the behavior of a single network, the applications can interact in complex ways. Applications might conflict if they contribute network updates that contradict on network state. For example, some light-loaded paths might be marked "unsafe" by a firewall, but are considered "preferred" by a load balancer. At the same time, applications also collaborate. After a firewall application adds suspicious paths to its blacklist policy, it may depend on a separate routing application to handle the removal of any unsafe paths it flags.

In an NIS, this problem translates to the management of interactions between data sources. The data sources in a traditional integration systems are assumed to be autonomous and independent. When individual sources contradict each other, the integration system simply rolls back and aborts the write. However, moving beyond such simple conflict prevention, aside from the performance limitation discussed in §3, existing data integration does not offer adequate support needed for the networking scenario.

To address this limitation, we propose *behavioral dependency* to formally define the dependency between applications: an application depends on another if the former requires service from the latter. The notion of behavioral dependency leverages database research on independent queries (or irrelevant updates) [20, 4, 7]. Given a view v derived by a query q over base tables, v is said to be independent of update u if the updates u cannot affect v. Our goal is to characterize the dependency between the data sources (i.e., applications) by examining updates to the integrated dataplane contributed by those applications.

Assume an NIS consists of a set of control applications $c_1, c_2, \cdots$, and the integrated dataplane $\mathcal{G}^{\mathcal{N}}$. Each application is associated with a view d_i representing its local data. The constraint describing the application policy is expressed by a violation view v_i (§3).

The operation of each application is to restore its policy whenever a violation is detected in its violation view: whenever an update of $\mathcal{G}^{\mathcal{N}}$ results in an insertion to v_i, the application is triggered to push writes to $\mathcal{G}^{\mathcal{N}}$ to remove the violation in v_i. Following this operational model, behavioral dependency is formulated as follows. A source application c_i depends on another source c_j if there exists some emptiness repair of v_i that results in update u of $\mathcal{G}^{\mathcal{N}}$ that causes insertion to v_j that violates and thus requires further (repairing) action from c_j.

We now describe the application of behavioral dependency: automatically orchestrating network updates by multiple applications that collectively drive a single network. The idea is to first automatically derive a dependency graph for the applications via satisfiability reasoning. The resulting dependency is then used to coordinate updates contributed by the participating applications.

5. REFERENCES

[1] Bahl, P., Chandra, R., Greenberg, A., Kandula, S., Maltz, D. A., and Zhang, M. Towards highly reliable enterprise network services via inference of multi-level dependencies. SIGCOMM '07, ACM.

[2] Bahl, P. V., Barham, P., Black, R., Chandra, R., Goldszmidt, M., Isaacs, R., Kandula, S., Li, L., MacCormick, J., Maltz, D., Mortier, R., Wawrzoniak, M., and Zhang, M. Discovering Dependencies for Network Management. In *HotNets 2006* („ November 2006).

[3] Berde, P., Gerola, M., Hart, J., Higuchi, Y., Kobayashi, M., Koide, T., Lantz, B., O'Connor, B., Radoslavov, P., Snow, W., and Parulkar, G. Onos: Towards an open, distributed sdn os. HotSDN '14, ACM.

[4] Blakeley, J. A., Coburn, N., and Larson, P.-V. Updating derived relations: Detecting irrelevant and autonomously computable updates. *ACM Trans. Database Syst. 14*, 3 (Sept. 1989).

[5] Cai, Z., and et al. The preliminary design and implementation of the maestro network control platform, 2008.

[6] Canini, M., Kuznetsov, P., Levin, D., and Schmid, S. Software transactional networking: Concurrent and consistent policy composition. In *Proceedings of HotSDN* (New York, NY, USA, 2013).

[7] Chen, I. A., Hull, R., and McLeod, D. Local ambiguity and derived data update. In *Fourth International Workshop on Research Issues in Data Engineering: Active Database Systems, 1994*.

[8] Chen, X., Zhang, M., Mao, Z. M., and Bahl, P. Automating network application dependency discovery: Experiences, limitations, and new solutions. OSDI'08, USENIX Association.

[9] Doan, A., Halevy, A., and Ives, Z. *Principles of Data Integration*, 1st ed. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2012.

[10] Dong, X. L., and Srivastava, D. Big data integration. In *ICDE 2013* (2013).

[11] Foster, N., Guha, A., Reitblatt, M., Story, A., Freedman, M. J., Katta, N. P., Monsanto, C., Reich, J., Rexford, J., Schlesinger, C., Walker, D., and Harrison, R. Languages for software-defined networks. *IEEE Communications Magazine 51* (2013).

[12] Goldszmidt, G., Yemini, Y., and Yemini, S. Network management by delegation: The mad approach. CASCON '10, IBM Corp.

[13] Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N., and Shenker, S. Nox: Towards an operating system for networks. *SIGCOMM Comput. Commun. Rev. 38*, 3 (July 2008).

[14] III, B. P., Ning, P., and Jajodia, S. On the accurate identification of network service dependencies in distributed systems. In *LISA 12* (San Diego, CA, 2012), USENIX.

[15] Jin, X., Gossels, J., Rexford, J., and Walker, D. Covisor: A compositional hypervisor for software-defined networks. In *NSDI* (2015).

[16] Kim, H., Reich, J., Gupta, A., Shahbaz, M., Feamster, N., and Clark, R. Kinetic: Verifiable dynamic network control. NSDI'15, USENIX Association.

[17] Koponen, T., Casado, M., Gude, N., Stribling, J., Poutievski, L., Zhu, M., Ramanathan, R., Iwata, Y., Inoue, H., Hama, T., and Shenker, S. Onix: a distributed control platform for large-scale production networks. OSDI'10.

[18] Lenzerini, M. Data integration: A theoretical perspective. PODS '02, ACM.

[19] Levy, A. Y. Logic-based artificial intelligence. Kluwer Academic Publishers, Norwell, MA, USA, 2001, ch. Logic-based Techniques in Data Integration.

[20] Levy, A. Y., and Sagiv, Y. Queries independent of updates. VLDB '93, Morgan Kaufmann Publishers Inc.

[21] Mogul, J. C., AuYoung, A., Banerjee, S., Popa, L., Lee, J., Mudigonda, J., Sharma, P., and Turner, Y. Corybantic: Towards the modular composition of sdn control programs. HotNets-XII, ACM.

[22] Monaco, M., Michel, O., and Keller, E. Applying operating system principles to sdn controller design. HotNets-XII, ACM.

[23] Monsanto, C., Foster, N., Harrison, R., and Walker, D. A compiler and run-time system for network programming languages. *SIGPLAN Not. 47*, 1 (Jan. 2012).

[24] Monsanto, C., Reich, J., Foster, N., Rexford, J., and Walker, D. Composing Software-Defined Networks. Proc. Networked Systems Design and Implementation, April 2013.

[25] Nelson, T., Ferguson, A. D., Scheer, M. J. G., and Krishnamurthi, S. Tierless programming and reasoning for software-defined networks. In *NSDI 2014* (2014), pp. 519–531.

[26] Prakash, C., Lee, J., Turner, Y., Kang, J.-M., Akella, A., Banerjee, S., Clark, C., Ma, Y., Sharma, P., and Zhang, Y. Pga: Using graphs to express and automatically reconcile network policies. SIGCOMM '15, ACM.

[27] Reich, J., Monsanto, C., Foster, N., Rexford, J., and Walker, D. Modular sdn programming with pyretic.

[28] Reitblatt, M., Canini, M., Guha, A., and Foster, N. Fattire: Declarative fault tolerance for software-defined networks. HotSDN '13, ACM.

[29] Silberschatz, A., Korth, H., and Sudarshan, S. *Database Systems Concepts*, 5 ed. McGraw-Hill, Inc., New York, NY, USA, 2006.

[30] Soulé, R., Basu, S., Kleinberg, R., Sirer, E. G., and Foster, N. Managing the network with merlin. HotNets-XII, ACM, pp. 24:1–24:7.

[31] Sun, P., Mahajan, R., Rexford, J., Yuan, L., Zhang, M., and Arefin, A. A network-state management service. SIGCOMM '14, ACM, pp. 563–574.

[32] Ullman, J. D. Information integration using logical views. ICDT '97, Springer-Verlag.

[33] Yices. http://yices.csl.sri.com/.