ADD: Application and Data-Driven Controller Design

Yikai Lin*† University of Michigan
Yuru Shao† University of Michigan
Xiao Zhu† University of Michigan
Junpeng Guo University of Michigan
Kira Barton University of Michigan
Z. Morley Mao University of Michigan

ABSTRACT

Existing SDN controllers commonly adopt an event-driven model that minimizes southbound communication and control-plane overhead. This model satisfies most existing SDN applications' goals to maximize data plane performance while still being able to programmatically control with a decent level of visibility. However, as network composition becomes more heterogeneous with NFV and IoT, such model can be insufficient for future applications that rely more on data analysis and intelligent decision making.

In this paper, we present our findings in a case study on smart manufacturing systems, which have highly heterogeneous device compositions, and applications that are much less "throughput" hungry or "latency" sensitive than network applications but require a lot more data for (real-time) decision making. We share the insights we gain that help us design a new Application and Data-Driven (ADD) model for SDN controllers. We build a proof-of-concept ADD controller based on this model and develop two applications to showcase its new capabilities. Evaluation results show that ADD delivers satisfying scalability and performance. More importantly, applications enabled by ADD gain more insights of the data plane and can make better decisions faster.

CCS CONCEPTS

• **Networks** → **Network architectures**; **Programmable networks**; *Programming interfaces*;

KEYWORDS

SDN, data-driven, data analysis, network programming, application, interface, northbound, southbound

ACM Reference Format:
Yikai Lin, Yuru Shao, Xiao Zhu, Junpeng Guo, Kira Barton, and Z. Morley Mao. 2019. ADD: Application and Data-Driven Controller Design. In *Symposium on SDN Research (SOSR '19), April 3–4, 2019, San Jose, CA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3314148.3314351

*The contact author: yklin@umich.edu
†These authors contributed equally to this paper.

SOSR '19, April 3–4, 2019, San Jose, CA, USA

ACM ISBN 978-1-4503-6710-3/19/04…$15.00
https://doi.org/10.1145/3314148.3314351

1 INTRODUCTION

The rise of SDN not only marks the transition of a fully-distributed network architecture to a hybrid one with logically centralized control, it also puts a strong emphasis on the roles applications can play in network management and even design. One of the many outstanding features offered by SDN is a global view of the network, which empowers applications with greater freedom and visibility to programmatically manage the network. To some extent, the more information the controller is able to provide to applications, the more capable applications are in gaining insights of the underlying data plane and making (globally) optimal decisions.

Existing SDN controllers are mostly event-driven. In existing SDN controller implementations [1, 6, 9], applications mostly rely on network events such as *link down*, *link congested*, *new host up* and *no matching rules found for packet* to trigger reactive procedures like rerouting.

Systematic support for data access is limited. Various existing southbound protocols [5, 11, 21, 27] do allow SDN applications to proactively pull data and states such as *port/link stats*, *packet headers*, and even *full packets* from data plane components to gain more insights. However, lack of systematic support for such operations means application development is tightly coupled with specific southbound protocols. Moreover, each application acting independently leads to redundant data retrieval across different applications due to lack of request consolidation, and duplicate data copy residing in each application.

Future SDN applications will be more data-driven. As network composition continues to diversify in terms of

Table 1: Comparisons between SDN and Smart Manufacturing Systems

	Legacy SDN	**Smart Manufacturing Systems**	**Future Data-Driven SDN**
Data/Production Plane Composition	Programmable Switches	Conveyor belts, Robots, CNCs, Motors, Printers, *etc.*	Programmable Switches, Physical and Virtualized Network Functions
Applications/Tasks	1. Routing 2. Firewall 3. Traffic Engineering 4. Network diagnostics *etc.*	1. Routing 2. Predictive Maintenance Control 3. Scheduling 4. Anomaly Detection *etc.*	1. All legacy SDN apps but more complex 2. Device-specific service customization (Phones, Vehicles, wearables, sensors, *etc.*) 3. Emerging apps like VR, AR, *etc.* 4. Real-time network telemetry
Data/States	Mostly discrete, real-time	Both continuous and discrete, real-time and historical	

network function types (NFV) and end-host types (IoT, autonomous vehicles, VR/AR devices, *etc.*), the need for network service (*i.e.,* features offered by SDN applications) customization and flexibility will increase dramatically, so does the volume and diversity of data produced on the data plane. Different end-hosts might have drastically different requirements for network services in terms of QoS guarantees, policies, and even intensive data analysis. To satisfy these requirements, SDN applications cannot simply rely on existing network events and data digests that are usually device-agnostic and contain scarce information. Instead, they must have direct, flexible and efficient data access, as should also be provided by the controller.

Unfortunately, as we discussed earlier, existing SDN controller implementations commonly adopt an event-driven model that lack general support for applications to read data freely and efficiently. Therefore, in this paper, we present ADD, a new application and data-driven SDN controller design that aims to improve data access flexibility and efficiency for SDN applications. Several design choices are made in ADD to satisfy and resolve the aforementioned requirements and limitations: (1) Applications subscribe to data in addition to events; (2) Data subscriptions are consolidated to eliminate redundancies; (3) Southbound interface uses generic key-value based schema; (4) Northbound interface is divided into high-level intent-based APIs and low-level APIs.

ADD's design complements instead of contradicts existing event-driven designs. Although we focus on the data-driven aspect in this paper, we believe both event-driven and data-driven models are needed for different applications.

We make the following contributions in this paper:

- We identify the gaps between existing SDN controller design and the requirements of data-driven applications. We carry out a case study on smart manufacturing systems and applications to understand the implications of data-driven applications.
- We propose an Application and Data-Driven (ADD) controller design that provides general, flexible and efficient data access for applications, and generic interfaces for controller and applications to process data.
- We prototype ADD and present preliminary evaluation results that show the scalability, performance and usefulness of the new design. We develop two applications using the new model to gain more insights of the data plane and outperform existing solutions.

2 CASE STUDY

In order to gain a better understanding of data-driven applications, especially the volume and characteristics of data they deal with, and their requirements for the controller, we carry out a case study on smart manufacturing systems [20] and applications with an actual smart manufacturing testbed (Figure 2). Smart manufacturing systems are data-driven and heterogeneous in nature, which share many commonalities with a data-driven SDN. Table 1 shows the comparisons between both types of systems on their data/production plane composition, applications that they run, and the characteristics of data they produce and consume.

Networking v.s. Manufacturing. Despite being rather different domains, networking and manufacturing share quite some similarities. For instance, consider *Routing* in networking where a network path is selected to forward data packets: the manufacturing equivalence would be to select a physical path to transport materials or parts. Both applications need visibility of data/production plane's topology and diverse properties such as device capabilities, bandwidth, and queue length. As shown in Table 1, legacy SDN has a relatively homogeneous data plane with mostly programmable switches, while manufacturing production plane and future SDN data plane are much more heterogeneous. On a manufacturing production plane, each machine could process parts with different structures and generate a huge amount of data due to continuous physical properties like voltage, current, *etc.*. Manufacturing applications utilize these data to make real-time decisions, monitor machine status, and predict failures. Similarly, P4 [11] allows packet formats to be customized and different VNFs generate a wide spectrum of different data and states. Historical data also plays a significant role in manufacturing applications. If SDN applications can make use of all these data across different sources, they

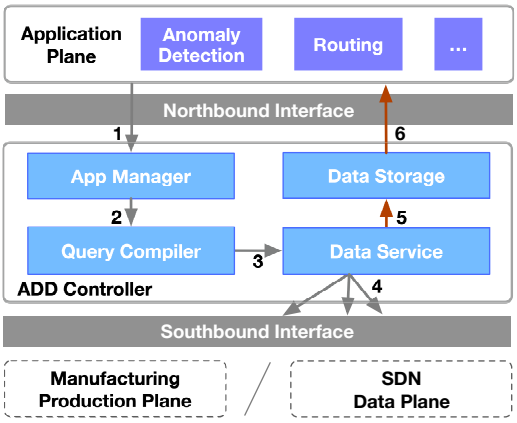

Figure 1: Data flow of the ADD controller design

can have a much better visibility of the network and deliver features that were not possible before.

Insights. Through this case study we intend to explore the data-driven aspect of future SDN. These comparisons give us a fresh perspective of how SDN composition might look like in near future and how data could become a dominant factor in future application development. We envision future SDN to benefit even more from such a data-driven controller design because of the diversifying end host types (phones, vehicles, sensors, wearables, *etc.*) in addition to the already heterogeneous network functions.

3 ADD CONTROLLER DESIGN

Figure 1 shows our proposed ADD controller design. In brief, ADD decouples applications from data retrieval and storage and provides general data access support with its internal pipeline and generic interfaces (described below). As a result, application developers can solely focus on "what data to request" and "what to do with it".

3.1 Key Controller Modules

The controller has four main modules that provide applications with efficient data access capabilities. As shown in Figure 1, (1) Apps register themselves to the App Manager with their interests (i.e. the data they want); (2) Query Compiler traverses all the interests, converts them into corresponding data fields, and consolidates all fields as a minimal set of queries and (3) passes them to the Data Service; (4) Data Service continuously queries data (a subset as defined by the interests) via the southbound interface and caches them in the Data Storage; (5) Applications fetch data from the Data Storage which contains both raw data (as explicitly requested by the applications) and network states (*e.g.,* global view). Both historical data and real-time data are stored in the Data Storage. Data storage supports both data-driven and event-driven models: applications can directly read data from it, or listen to changes of network states.

Applications' fetching data is asynchronous to controller modules' operations. This publish-subscribe-like messaging mechanism enables on-demand data extraction. Specifically, at the time data is produced or consumed, the producer and the consumer does not need to communicate with each other.

3.2 Southbound Interface

The southbound interface has to satisfy requirements of high scalability and low latency. As the number of applications grows, and with the increasing data volume they read from the production plane, the workload of the southbound interface rises significantly. Moreover, as discussed in §2, both manufacturing production plane and future SDN data plane will be heterogeneous. A generic abstraction is desired in order for the controller modules and applications to evolve/function independently from different devices.

To support abstractions and to cope with the potentially large data volume and the demands for structured, contextualized data from upper layers, we propose a customizable information model to describe the production/data plane. Specifically, data items (essentially key-value pairs) are grouped into objects which reflect device properties and functionalities. Each object can have multiple instances. For example, a CNC spindle has four axes, therefore the `Axis` object have four instances `Axis_X`, `Axis_Y`, `Axis_Z`, and `Axis_S`. The `Axis` object consists of spindle axis properties, such as `speed`, `acceleration`, and `voltage`. If certain data items in an instance are not of interest to any running applications, the southbound never queries them for better efficiency.

3.3 Northbound Interface

ADD provides flexible and efficient RESTful APIs for applications to communicate with the controller. As mentioned in §1, future applications will require frequent, sizable data exchanges with the controller, and these applications will need much more computing resources to perform data analytics and machine learning algorithms [16] which can easily overwhelm the controller. It is also challenging to manage application life-cycles, security and privacy [10] when they are co-located with the controller. Therefore, we envision applications to be moving out of the controller and be remotely communicating with it. The trend of geo-distributed computing [17, 26] and the aforementioned concerns make us believe that a bandwidth-efficient and low-latency northbound interface is desired. Existing RESTful northbound APIs [1, 6, 9] in SDN controllers have two major limitations: (1) Data is organized in a coarse-grained manner thus client could get more than they want while bandwidth is wasted;

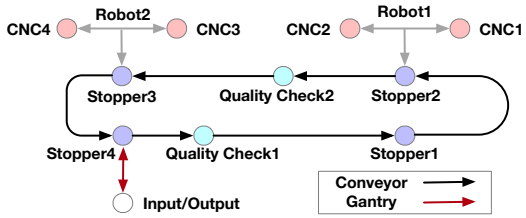

Figure 2: Physical layout of the testbed

(2) Client may have to submit multiple requests that correspond to different URLs to get all its desired data, which causes significant delay (although HTTP pipe-lining can help mitigate this problem, it can cause other problems such as Head-Of-Line blocking [23]).

ADD maintains a hierarchical data structure that houses fine-grained data fields and allows declarative query [19], in order to address the aforementioned issues. This data structure defines the capabilities (*e.g.,* available data items) of the northbound interface. Both the server-side (controller-side) northbound service and the client-side (application-side) northbound libraries have visibility of this definition. The client makes a single query to get all its desired data by strictly following the definition, *e.g.,* specifying a set of data items associated with a set of data plane components. Data service in the controller refers to it to execute queries from clients and only returns the exact data that clients need, nothing more, nothing less.

To offer great usability, we design two types of APIs, *i.e.,* intent-level APIs and task-level APIs. Hierarchical northbound design [18] has been proposed to provide better application programming experience in event-driven SDN. We believe it is also helpful in our data-driven approach. Task-level APIs perform specific tasks such as querying a given set of data items, computing reconfigurations, *etc.*.

4 PROTOTYPE AND EVALUATIONS

To understand how ADD performs in practice, we build a prototype ADD controller (as described in §4.1) using open-source software. We also develop two applications (§4.3): *anomaly detection* and *routing*, which exist in both manufacturing and networking domains. In this section, we first introduce our prototype ADD controller and discuss some of the implementation choices, then present both quantitative and qualitative evaluation results. Specifically, we evaluate the ADD design in terms of scalability, usability and performance. We carry out most of our experiments on an actual manufacturing testbed in proximity with our controller. As shown in Figure 2, the testbed consists of 4 Computer Numeric Control milling machines (CNCs), 2 Robots, 2 quality check modules and a conveyor loop. As parts travel on the conveyor, they might encounter machines with different functions. For example, when a stopper pauses a part, a robot can pick it up and places it on either one of the two CNCs depending on the part model. We can clearly see the analogies between this testbed and an SDN data plane.

4.1 Prototype

Controller. Existing SDN controllers such as Floodlight [1] adopt a publish/subscribe model for applications to choose which events to listen to. ADD adopts a similar mechanism but with one major modification: instead of the controller dispatching data to applications, applications proactively pull data from the data storage, with the exception of events (as detailed below). Because of the heterogeneity of devices and the unstructured nature of their data, we implement the data storage with MongoDB [4]. The data storage stores not only the data read from southbound but also the global view, which reflects the dynamics of the physical layout of production plane. The global view provides the connectivity of manufacturing units to applications like *routing* which require machine capabilities and the available paths between them. The initial layout of the production plane is constructed from a pre-defined configuration that describes the interactions and paths between units. This global view updates automatically based on the streaming data obtained from southbound. Instead of waiting for applications to pull the data storage and potentially discover this event, the controller notifies interested applications. This shows how event-driven model and data-driven model can work together to better serve application needs.

Southbound. OPC [7] is the most widely deployed standard for data access in industrial automation. In OPC, all the data items are provided as *tags*. Hardware vendors define default tags that their devices produce. Meanwhile, operators can define specific input and output bits, internal temporary variables as tags. Instead of reading from or writing to I/O bits, machine controllers are able to use tag names in their code to easily read and write data. Our southbound interface implementation leverages OPC to get specific data tags from individual machines. Additionally, we define the information model using Protocol Buffers [8]. The benefits of the information model are twofold. First, it assembles scattered tags into meaningful data structures to describe machine status. Second, it provides an abstraction so that the controller can be completely agnostic to production plane details, which means the production plane (or data plane) can be swapped without changing the fundamental design of the controller. Both real-time data and historical data of the production plane are provided, and the controller can be deployed anywhere and retrieve data through remote procedure calls (RPCs) based on gRPC.

Table 2: *Intent-level* and *task-level* northbound API examples

API	Arguments	Description
`findRoute()`	`device_id, part_id`	Find an alternative route for a set of parts.
`getTopology()`	`status_type`	Return a graph representation of the topology with associated device status.
`isConnected()`	`initial_id, candidate_id`	Check whether the initial device and candidate device is connected.
`hasCapability()`	`device_id, part_id`	Check whether a specific machine is capable of processing a set of parts.
`isReady()`	`device_id, part_id`	Check whether a device is running the program that aims to process a part.
`detectAnomaly()`	`anomaly_type`	Detect anomalies within a specific type.
`getDevice()`	`device_id`	Get the statistics of a set of specific machines.
`isFailed()`	`device_id`	Check whether a specific machine is failed.

Northbound. We build our northbound interfaces on top of GraphQL [2] which provides a type system that is an ideal vantage point for us to define hierarchical data mappings. The GraphQL server directly interacts with MongoDB in the controller, while the GraphQL client resides in the app-side northbound libraries. We implement six task-level APIs and two intent-level APIs as shown in Table 2. `findRoute()` is an intent-level API that consists of four task-level APIs, which are `getTopology()`, `isConnected()`, `hasCapability()`, and `isReady()`. The other intent-level API, `detectAnomaly()`, is composed of two task-level APIs `getDevice()` and `isFailed()`.

4.2 Microbenchmarks

Southbound Performance. We read different numbers of data items from the production plane in order to measure southbound latency. Figure 3 shows the average reading time from 1 item to 200 items in 50 measurements. Overall, the latency is proportional to the number of data items being read, but there are some spikes of the average read latency. We find that those spikes come from outliers caused by unusually long OPC reading. Although there is no similar southbound implementation for comparison, the average latency of reading 200 tags is no worse than an industrial data collection tool named Rockwell Cloud Agent Elastic.

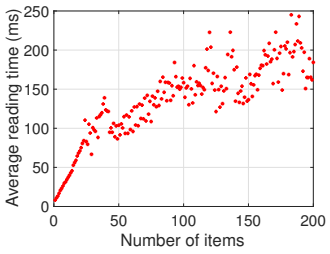

Figure 3: Southbound average reading time

Controller Scalability. Because of SDN controllers' centralized design, scalability is one of the most important criteria when evaluating their feasibility. For ADD's data-driven design, scalability is especially critical. For example, the overhead of controller writing data in the data storage should be contained despite large number of applications and interests. In order to evaluate this, we measure the time between a CNC going down on the production plane and the global view being updated in the data storage when varying the number of interests. If this time interval doesn't go up with the number of interests, it means (1) ADD's data storage has a stable overhead; (2) even with a large number of applications and interests, global view is updated fairly quickly.

The CNC we experiment with has a total of 800 tags, and we vary the number of tags queried by the controller from 15 to 800. The average time intervals measured are 9, 203ms, 9, 276ms, 11, 278ms respectively for 15, 100 and 800 such tags. This means with more than 50X the workload, the overhead increases for merely 22%. Note that since applications may query overlapping data items, ADD's data consolidation feature helps further improve the scalability. Besides, here we only use a single controller instance with a single southbound channel. In the future we plan to experiment with a larger scale of applications and data plane components, and better understand ADD's scalability by distributing the controller instances and parallelizing the streaming channels.

Northbound API Usability. To evaluate the ease of use of our northbound API, we develop two first-of-its-kind cyber manufacturing applications: anomaly detection app and routing app. The total LoC of the anomaly detection app and routing app is 78 and 210, respectively. More detailed evaluation of the application will be introduced in §4.3.

Northbound Performance. To evaluate bandwidth efficiency and latency performance of our northbound interface, we run a benchmark app on a laptop that is one hop (WiFi link) away from the controller. Our app monitors the TX bytes and the current bitrate of the first port of two switches. We compare with RESTful APIs in Ryu [9] which provides two related RESTful APIs for querying these two statistics. They are (1) `GET:/stats/port/<dpid>[/<port>]` and (2) `GET:/stats/portdesc/<dpid>/[<port>]`. For comparison, we implement both of them in our controller. We assume the controller already has the data stored in its database and study the performance of the northbound interface itself. We measure the amount of data exchanged and the

response latency for getting a pair of bitrate and TX bytes values. ADD's northbound interface consumes 255 bytes of data over the network while Ryu consumes 1,516 bytes. This disparity is attributed to the fact that ADD gets the exact data the app wants while Ryu delivers all the related data and lets the app do the data retrieval. The latency is 9.1ms and 43.4ms for ADD and Ryu, respectively. The improvement is brought by using just one query to fetch all the desired data, unlike Ryu where multiple requests have to be made to separately fetch data from different URLs.

4.3 System-level Test with Applications

We develop two example applications (co-located with the controller) to test if ADD can (1) provide data access with low latency and (2) offer great programmability to applications so that they can benefit from the extra data access.

Anomaly detection. To showcase the capabilities of ADD and the low latency of the southbound and the northbound interfaces, we develop an anomaly detection application that looks into network traffic and machine status simultaneously to detect anomalies and diagnose provenance. We are able to detect anomalies earlier than tools that rely purely on machine status. We compare our anomaly detection application with the Rockwell FactoryTalk program. While the testbed is running, we inject anomalies into CNC3 and measure the the detection delays of our application and FactoryTalk [3]. Our anomaly detection application reports an anomaly after 12.6 seconds, while FactoryTalk detects that more than 35 seconds later. The root cause is that the underlying OPC has a caching mechanism, but we are able to tell anomalies sooner with data collected from network communications. Once an anomaly is detected, we let the anomaly detection application notify the routing application for computing a new route and deploying the reconfigurations.

Routing. The routing app awaits routing requests coming from the controller or other apps such as anomaly detection and machine maintenance. When the routing app receives notifications of a failure and that a part needs to be rerouted, it analyzes the topology and capabilities of each machine to make the rerouting decision. Our key metric is the application response time – the time between the reception of notification and when the rerouting decision is made. We repeat the experiment for 10 times and show the average values. It takes 10.9ms to query machine and topology data from the controller, 2.9ms to analyze the available machines, 1.7ms to check the capable machines, and 1.4ms to find the ready machines. The total application response time is 16.9ms.

Interactions between the two applications. With both apps installed, the administrator can leverage them to quickly detect failure and make rerouting decisions. In our evaluation, the routing app registers to listen for notifications from the anomaly detection app described above. Summing the response time of the two apps results in a 13s latency. Compared to traditional manufacturing system that usually takes hours, ADD significantly reduces the time it takes to (1) detect an anomaly and (2) deploy a new configuration.

5 RELATED WORK

Legacy SDN Controllers. Existing SDN controllers [1, 6, 9] have a common event-driven model without general data access support (usually offered by southbound protocols like OpenFlow or NetFlow). ADD provides systematic data access support for SDN applications and adopts a data-driven model that decouples applications from data retrieval and storage. ADD complements rather than contradicts existing controller designs in that one could still use protocols like OpenFlow or P4 for reconfigurations of certain devices.

Data-driven Approaches. The integration of big data or data analysis with SDN is becoming an interesting topic for both areas. Some architectures are proposed [12, 13, 15, 22] to facilitate data analysis in SDN. These controller designs either focus on data analysis on the application side without providing systematic support for efficient data query and storage, or do not address the limitations of existing proprietary southbound interfaces that support limited data access. Other works like Marple [25] and Sonata [14] improve the flexibility and scalability of network monitoring/telemetry by leveraging state-of-the-art programmable switches. ADD is a generic controller design that interfaces with different kinds of applications and data plane devices.

Hierarchical Controller Design. Previous works such as SoftMoW [24] propose hierarchical structure to improve SDN controller scalability. Such effort is orthogonal to our work and could potentially be utilized to further improve our controller design (e.g. improving southbound scalability by having multiple children controller instances).

6 CONCLUSION AND FUTURE WORK

In this paper, we propose an Application and Data-Driven controller design, or ADD, that aims to provide better data access capabilities for future SDN applications. ADD adopts a data-driven model that decouples applications from data retrieval and storage, and allows applications to gain more insights of underlying devices. We prototype ADD and evaluate it with two example applications that show the scalability and usability of ADD. Moreover, applications enabled by ADD can outperform existing solutions by utilizing the enriched insights. Although ADD's design is generic and the results in this paper are mostly not specific to manufacturing, we would like to expand our case studies in networking domain and experiment with more networking applications (*e.g.,* traffic engineering) and components (*e.g.,* P4 switches).

ACKNOWLEDGEMENTS

This work was supported by NSF under award CNS-1544678. We would like to thank our shepherd, Anduo Wang, and the anonymous reviewers for their valuable feedback.

REFERENCES

[1] 2018. Floodlight SDN Framework. http://www.projectfloodlight.org/floodlight/.
[2] 2018. GraphQL. https://graphql.org/.
[3] 2018. Manufacturing Software Solutions, Rockwell FactoryTalk. https://www.rockwellautomation.com/rockwellsoftware/products/overview.page.
[4] 2018. MongoDB. https://www.mongodb.com/.
[5] 2018. NetFlow. https://www.solarwinds.com/what-is-netflow.
[6] 2018. ONOS SDN Framework. https://onosproject.org/.
[7] 2018. OPC Foundation. https://opcfoundation.org/.
[8] 2018. Protocol Buffers. https://developers.google.com/protocol-buffers/.
[9] 2018. Ryu SDN Framework. https://osrg.github.io/ryu/.
[10] Yousra Alkabani and Farinaz Koushanfar. 2007. Active Hardware Metering for Intellectual Property Protection and Security.. In *USENIX security symposium*. 291–306.
[11] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al. 2014. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review* 44, 3 (2014), 87–95.
[12] Alexander Clemm, Mouli Chandramouli, Nitish Gupta, Robert Lerche, Ashwin Pankaj, Manjunath Patil, Ganesan Rajam, V Anbalagan, Joe Zhang, and Yifan Zhang. 2015. DNA: An SDN framework for Distributed Network Analytics (Demo Paper). In *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*. IEEE, 1143–1144.
[13] Laizhong Cui, F Richard Yu, and Qiao Yan. 2016. When big data meets software-defined networking: SDN for big data and big data for SDN. *IEEE network* 30, 1 (2016), 58–65.
[14] Arpit Gupta, Rob Harrison, Marco Canini, Nick Feamster, Jennifer Rexford, and Walter Willinger. 2018. Sonata: Query-driven streaming network telemetry. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. ACM, 357–371.
[15] Haojun Huang, Hao Yin, Geyong Min, Hongbo Jiang, Junbao Zhang, and Yulei Wu. 2017. Data-driven information plane in software-defined networking. *IEEE Communications Magazine* 55, 6 (2017), 218–224.
[16] Chien-Chun Hung, Ganesh Ananthanarayanan, Peter Bodik, Leana Golubchik, Minlan Yu, Paramvir Bahl, and Matthai Philipose. 2018. VideoEdge: Processing Camera Streams using Hierarchical Clusters. In *Proceedings of the Third ACM/IEEE Symposium on Edge Computing*. IEEE, 115–131.
[17] Chien-Chun Hung, Ganesh Ananthanarayanan, Leana Golubchik, Minlan Yu, and Mingyang Zhang. 2018. Wide-area analytics with multiple resources. In *Proceedings of the Thirteenth EuroSys Conference*. ACM, 12.
[18] Yikai Lin, Ulaş C Kozat, John Kaippallimalil, Mehrdad Moradi, Anthony CK Soong, and Z Morley Mao. 2018. Pausing and resuming network flows using programmable buffers. In *Proceedings of the Symposium on SDN Research*. ACM, 7.
[19] Boon Thau Loo, Joseph M Hellerstein, Ion Stoica, and Raghu Ramakrishnan. 2005. Declarative routing: extensible routing with declarative queries. In *ACM SIGCOMM Computer Communication Review*, Vol. 35. ACM, 289–300.
[20] Felipe Lopez, Yuru Shao, Z Morley Mao, James Moyne, Kira Barton, and Dawn Tilbury. 2018. A software-defined framework for the integrated management of smart manufacturing systems. *Manufacturing Letters* 15 (2018), 18–21.
[21] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. 2008. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review* 38, 2 (2008), 69–74.
[22] Albert Mestres, Alberto Rodriguez-Natal, Josep Carner, Pere Barlet-Ros, Eduard Alarcón, Marc Solé, Victor Muntés-Mulero, David Meyer, Sharon Barkai, Mike J Hibbett, et al. 2017. Knowledge-defined networking. *ACM SIGCOMM Computer Communication Review* 47, 3 (2017), 2–10.
[23] Xianghang Mi, Feng Qian, and Xiaofeng Wang. 2016. Smig: Stream migration extension for http/2. In *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*. ACM, 121–128.
[24] Mehrdad Moradi, Wenfei Wu, Li Erran Li, and Zhuoqing Morley Mao. 2014. SoftMoW: Recursive and reconfigurable cellular WAN architecture. In *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*. ACM, 377–390.
[25] Srinivas Narayana, Anirudh Sivaraman, Vikram Nathan, Prateesh Goyal, Venkat Arun, Mohammad Alizadeh, Vimalkumar Jeyakumar, and Changhoon Kim. 2017. Language-directed hardware design for network performance monitoring. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 85–98.
[26] Qifan Pu, Ganesh Ananthanarayanan, Peter Bodik, Srikanth Kandula, Aditya Akella, Paramvir Bahl, and Ion Stoica. 2015. Low latency geo-distributed data analytics. In *ACM SIGCOMM Computer Communication Review*, Vol. 45. ACM, 421–434.
[27] Minlan Yu, Lavanya Jose, and Rui Miao. 2013. Software Defined Traffic Measurement with OpenSketch.. In *NSDI*, Vol. 13. 29–42.