Abstract Model of SDN Architectures Enabling Comprehensive Performance Comparisons

Tatsuya Sato, Shingo Ata, and Ikuo Oka
Graduate School of Engineering,
Osaka City University,
3–3–138 Sugimoto, Sumiyoshi, Osaka 558–5858, Japan
Email: {sato.t@c.,ata@,oka@}info.eng.osaka-cu.ac.jp

Yasuhiro Sato
Faculty of Maritime Safety Technology,
Japan Coast Guard Academy,
5–1 Wakaba, Kure, Hiroshima 737–8512, Japan
Email: sato@jcga.ac.jp

***Abstract*—Software-defined Networking (SDN) is a new network architecture that decouples the control plane from the data plane. Scalability of the control plane with respect to network size and update frequency is an important problem that has been addressed by previous studies from a variety of viewpoints. However, the solutions found in these studies may be only locally optimized solutions. To find a globally optimized solution, a broader viewpoint is required: one in which various SDN architectures can be evaluated and compared. In this paper, we propose an abstract model of SDN architectures, which enables multiple SDN architectures to be compared under a unified evaluation condition, and discuss the modeling of SDN architecture and its variations to find the optimal design from a global viewpoint. We first propose a generic model of SDN architectures and derive variations in terms of composition unit (single or multiple), processing principle (sequential or parallel), or location (intra- or inter-node). We then show that existing SDN architectures can be represented as one of the variations of our abstract model with fitted parameters. Finally we discuss how variation of components affects performance and show, using message-driven simulations, that our model enables comprehensive performance comparisons of different SDN designs represented as parameterized models.**

I. Introduction

Software-defined Networking (SDN) has emerged as a new networking paradigm for realizing a flexible network management [1]. Fig. 1 shows the fundamental concept of SDN architecture. A key principle of SDN is the separation of the control and data planes from network devices, such as switches and routers. The control plane is logically centralized and provides programmable application-programming interfaces (APIs) for managing the physical layer. The data plane specializes in forwarding packets according to instructions from the controllers. In SDN, the controllers enable flexible management and unified control via programmable interfaces with a global view of the network status.

However, the approach of using a centralized controller brings up the important issue of scalability with respect to the network size [2]. The processing type of controller can be classified into two types: reactive (i.e., reacting to events of the data plane) and proactive (i.e., active control by the control plane). In reactive type, when a switch on the data plane receives the new flow, the switch informs the controller of the arrival of it. The controller then configures the forwarding rule for the flow based on the global network view. If the network becomes large, then the centralized controller may become overloaded or the control channel between the controller and switches may become a bottleneck. Consequently, the controller can become unable to deal with requests for new flows, and the switch can become unable to handle new flows even when the data plane is not congested. Thus, the reactive type processing on the control plane can have a larger influence on the stability of the network than the proactive one.

To address this concern about the scalability of the control plane, various methods have been proposed. For example, ElastiCon [3] divides the data plane into sectors and assign one or more controllers to each area. Although this approach has to consider how, when, and where the global information is synchronized among the sub controllers, the scalability of the control plane is certainly improved. In an alternative approach, Beacon [4] tries to improve the performance of the controller by using multi-thread technology. However, a single controller may not be enough to manage large-scale networks, even if the controller has a many CPU cores and much available RAM. Another approach is adopted by DevoFlow [5], which reduces the workload between switches and controller by dividing data flows into classes of flows thereby reducing the number of flow requests.

These previous studies have focused on specific aspects of the whole problem (e.g., improvement of machine performance or load balancing), so the solutions they propose may be only locally optimized. Focusing on a specific aspect is not sufficient to evaluate the performance of various SDN architectures because the network performance depends on various factors. In SDN architectures there are some functional elements, such as controllers, switches, and data stores. It is obvious that these elements can be combined in different ways (e.g., multiplexing and parallelization). Connections between functional elements are also subject to some kinds of variation (e.g., one-to-one and one-to-many), and these variations also affect the performance of SDN architecture. For instance, we cannot objectively clarify whether scalability is better improved by using multiple controllers with a single processor or by using a single controller with multiple processors. Therefore, to obtain the globally optimized solution, it is necessary to clarify all variations or extensions, and compare all possible combinations of these variations through unified evaluation criteria.

The final objective of our research is to find the global optimum solution of SDN architecture. As the first step for it, we propose an abstract model to enable comparative performance evaluation among existing SDN architectures

978-3-901882-77-7

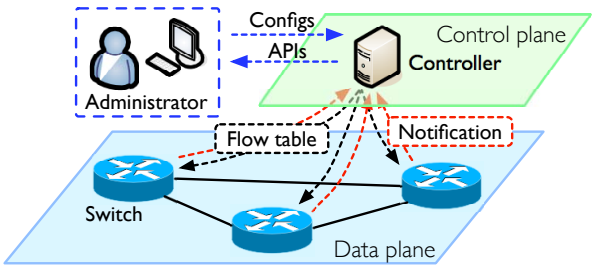


Fig. 1. Fundamental concept of an SDN Architecture

and validate it. Specifically, we extract fundamental functions from various SDN architectures and define them as abstracted function components. We also define processing models to express control processes of SDN. SDN architectures can be represented as variations of the proposed abstraction model, which enables a comparison of them in terms of a unified performance evaluation. To validate our model, we show that existing approaches can be expressed as specific cases (i.e., variations) of our abstract model. Moreover, our model can represent features such as multiplexing of function components and parallelization of processing, and by combining such variations, new SDN architectures can be characterized as a set of parameter values. By evaluating the scalability of all possible variations, the optimal SDN architecture for improving scalability can be found as one variation of our model. This idea can be applied not only to scalability but also to other problems, such as fault tolerance. Thus, we consider numerous patterns of variation of our model and discuss how the differences affect the performance of SDN architecture.

The paper is organized as follows. A survey of related work is given in Section II. A detailed description of our abstract model of SDN architecture is given in Section III. We discuss variations of our proposed model in Section IV. By using the model, we evaluate and compare some existing SDN architectures as variations of our abstract model in Section V, and then a comparative evaluation is briefly performed in Section VI. Finally, we conclude this paper in Section VII.

II. Related work

SDN separates the control plane from the data plane, and a centralized controller manages all switches on the data plane. However, such centralized control may become unable to respond to demand as the size of network increases. The scalability of the control plane is thus one of the important problems standing in the way of the adoption of SDN architectures for large-scale networks [2].

Previous research into improving the scalability of SDN can be categorized into three approaches: (1) improving the performance of the controller itself; (2) designing a distributed architecture for the control plane; and (3) reducing the workload of the controller.

In the first approach, there are proposals for improved SDN controllers, such as Maestro [6], NOX-MT [7], Beacon [4], and McNettle [8]. Maestro adopts parallelism in single-threaded controllers and achieves linear scalability on an 8-core machine. NOX-MT is a multi-threaded successor of NOX, a single-threaded controller written in C++, that can handle 1.6 million requests per second on an 8-core machine (NOX handles 30 thousand requests per second). Beacon is a Java-based controller with multi-threading technology and I/O batching, which can handle 12.8 million requests per second with a 12-core machine. McNettle is a library for writing control programs in Haskell that can be executed effectively on multi-core NUMA servers. In [8] it was shown that a controller written using McNettle scales up through 46 cores, and can handle 14 million flows per second with 46 cores.

In the second approach, several distributed control plane architectures have been proposed. HyperFlow [9] tries to ensure scalability by increasing the number of SDN controllers in the management domain. In this approach, network events sent to a distributed controller are shared with the other controllers by using a publish/subscribe messaging model to update their local states. Onix [10] is a distributed controller system, operated on a cluster of one or more servers, that uses a distributed hash table to maintain the consistency of the global network status. ElastiCon [3] is intended to address one of the problems arising from distributed control planes: an imbalance of processing workload among distributed controllers as a result of configuring a static mapping between switches and controllers. As a solution, a dynamic control method that changes the mapping and the number of distributed controllers depending on the network workload is proposed. In [11], a database-oriented management method to maintain the consistency of the whole network by taking advantage of a relational database system was proposed. In this method logical network information, such as high-level network configurations, is separate from the physical network information so that network administrators do not have to care about the physical information to manage the whole network. Pratyaastha [12] provides an optimal way of distributing SDN controller instances and partitioning application states. In this method, the application state related to a switch and the management of the switch are assigned to the same controller instance. The authors of this paper also discussed a suitable partition granularity that minimized inter-controller communication and the number of dependencies between SDN switches and a specific partition.

The third approach reduces the workload of the controller by extending the functionality of switches on the data plane. In DIFANE [13], some of the forwarding rules managed in controllers are pre-installed into some switches, called authority switches. Using the pre-installed rules, the switches can handle corresponding flows without requests to controllers. DevoFlow [5] divides data flows into classes and adopts a fuzzy matching approach to determine the forwarding rule applied to each flow class. This approach shows that the workload of the controller is reduced by making the fuzzy matching rules looser.

Each of the approaches described above addresses the problem of scalability from a particular viewpoint and derives a solution to accomplish the specific objective. However, these solutions might be only locally optimal because they focus on only one of various variations of SDN architecture. We cannot clarify which approach is the optimal solution for a specific problem because these approaches are evaluated differently in various situations. To obtain the globally optimized

solution for a specific problem, a unified evaluation scheme for SDN architectures that enables the comparison of various approaches using a single set of criteria is required. Furthermore, though individual solutions provide their evaluations to show the effectiveness or contribution of the study, the performance measures used in the evaluation are different by every literature. The motivation of our study is also to provide comparative evaluations with a metric of which some studies didn't provide the evaluation in the original literature.

There are a number of studies proposed modeling methods that can represent various types of SDN architectures as one of their models and uniformly compared various controller platforms. The authors in [14] proposed an OpenFlow [15]-based network model to estimate the packet sojourn time and the packet loss rate in a system. However, they assumed a simple model consisting of one switch and one controller, not a generic model that can be used to express a variety of SDN architectures. The authors in [16] also proposed model for an OpenFlow network based on Jackson network, and it can be extended to multi-node data plane. The authors in [17] proposed a network calculus-based mathematical model of SDN switch and controller to capture delay and buffer length. They express the control plane as a single-queue model, does not consider modeling of processing or functions of the control plane. The authors in [18] focused on the architecture of the control plane, and categorized some previous work into three types of structures: centralized, decentralized, and hierarchical. They also discussed how the differences among these types affect the scalability of the control plane. Although the objective of this research is similar to ours, their evaluation does not take into account variations (e.g., multiplexing and parallelization of processing) of the controller itself. The objective of this paper is to represent various types of SDN architectures through our abstracted model, not to compare our model with existing modeling methods in this paper.

We propose an abstract model to describe each function of SDN architecture, and express some of the models of previous studies as variations of our model. We believe that expressing various approaches proposed in previous studies via a single model enables a unified evaluation. Furthermore, by exploring all possible variations of our model, new variations that have not been considered in previous studies may be discovered to be the best solution for a specific problem.

III. Abstract model of SDN architecture

Here, we describe our abstract model of fundamental processing elements in SDN architecture. Our abstract model consists of two submodels: a *function model* and a *processing model*. The function model describes all functions of SDN controller and the details of each of its components. The processing model describes two different types of processing procedure.

A. Function model

An SDN controller has various functions: switch management, forwarding rule calculation, resource management, and so on. The basic principles of these functions are almost the same, though the processing content of each function is different. Fig. 2 shows the function model used to express the fundamental architecture of SDN. We categorize the controller functions into three fundamental function components, which are *Datastore*, *Information Processing*, and *Control*, and define a function model consisting of these three components.

The Datastore component manages all information for controlling the network. The information stored in this component is the topology of the physical network, network configurations described by administrator, resource information collected by SNMP agents, and so on. This component is usually composed of databases and distributed file systems, and provides minimal functional interfaces to respond to data requests from the information processing component. For example, ONOS [19], one of the real SDN controller, uses uses graph database and key value store for keeping network wide view.

The Information processing component generates the network information that is needed to configure switches appropriately, according to configurations and policies described by the network administrator. In particular, this component provides path calculation, resource management, packet filtering (e.g., acting as a firewall), and acquires the necessary information for the datastore component. Moreover, the administrator (or users if permitted) can develop arbitrary network applications in this component to provide flexible network services (e.g., network virtualization, traffic accounting, and encryption).

The Control component manages connections with the switches placed on the data plane. This component invokes appropriate processing components to handle network events notified from switches, and configures switches according to the processing result. For instance, if a switch receives the first packet of an unknown flow, the switch notifies that event to this component via the control channel between the switch and the controller. The component receiving the notification invokes the routing application of the information processing component, which is described in policies specified by network administrator, and sends forwarding rules to the corresponding switches.

The data plane consists of physical switches and links that connect each switch. In SDN, a physical switch is limited to packet forwarding according to its flow table because routing and other control mechanisms are assigned to the control plane. The switches maintain a connection to the control component to receive configuration commands, such as updates of the flow table, and to send notifications of the arrival of packets in flows that do not exist in the flow table.

B. Processing model

We define the processing model to express various processes of SDN architectures. The control processes of SDN architectures can be classified into two types: *reactive* (on-demand) and *proactive* (pre-configured).

Reactive control is a passive control style triggered by network events that occur in the data plane. Fig. 3(a) shows the sequence diagram of reactive control. When a network event, which includes the arrival of unknown packets and network failure of the switch or connecting links, occurs at a switch, the switch sends a notification to the control component connected with it to make the controller handle the network

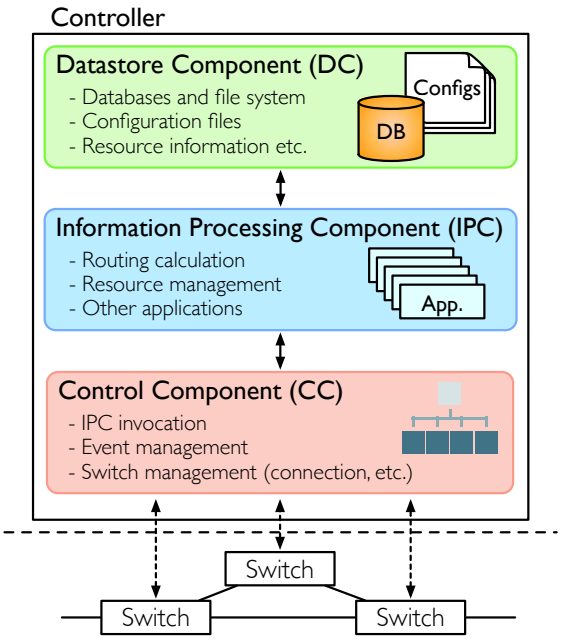
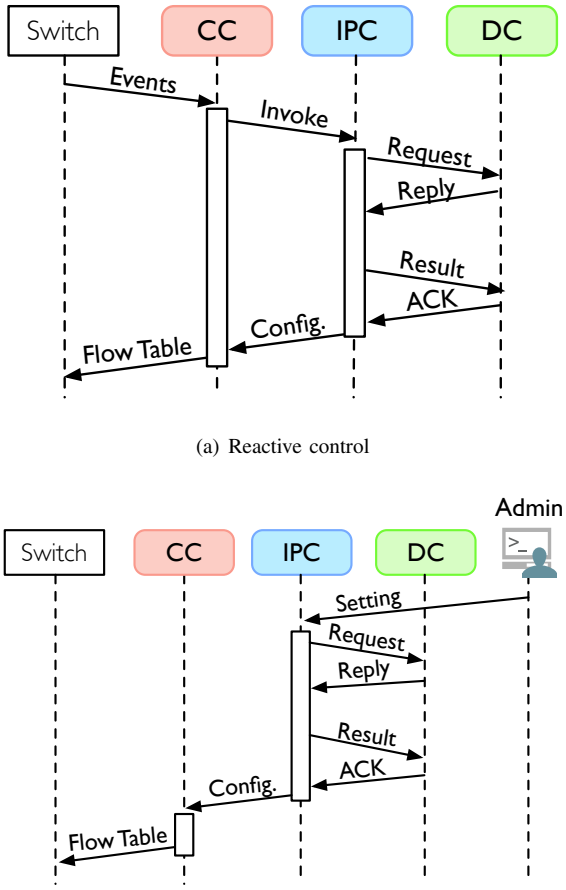


Fig. 2. Three fundamental components of the function model



(a) Reactive control

(b) Proactive control

Fig. 3. The sequence diagrams of the processing model

event. The control component that received the notification contacts information processing components to generate an appropriate configuration for responding to the notification. Then, the information processing components acquire network information, such as the topology of the physical network and host information, and generate the appropriate configuration by performing path calculation, resource allocation, and so forth. Based on the processing results, the control component creates configuration commands to be sent to the switch.

Proactive control is an active control style triggered by the control plane, but not in reaction to network events, which is used when the network administrator wants to change the network configuration. According to instructions from the network administrator, the control component invokes information processing components to generate the necessary information. The rest of the procedure is basically the same as for reactive control. Some forwarding rules are pre-installed in switches by using this type of control to reduce the number of notifications from switches.

IV. Variations of our Abstract Model

In this section, we discuss variations via combination of the function components to express existing SDN architectures. Fig. 4 shows some examples of model variations for the SDN controller architecture. Each of the function components can operate independently of other components, and consists of one or more internal processing units, which can execute in parallel with other units in the same component. In particular, the information processing component corresponds to the network application provided in the controller domain, and controllers are typically equipped with multiple components. We should consider not only variations of our function model, but also the architecture of the internal processing units.

A. *Parallelism of the internal processing unit*

Parallelism of the processing unit in one component corresponds to the use of multi-threading technology, and can reduce the processing time taken to complete the task of the component. The parallelized processing units are executed on the same component, and share the computational resources of the component. A typical example of this variation is shown in Fig. 4(a). Though a controller with multi-threading technology is expected to achieve near-linear scalability with the number of the threads, efficient resource consumption for threads should be considered.

B. *Multiplexing of a function component*

Multiplexing of a function component is a common way to ensure the scalability of the control plane. This variation corresponds to multi-core CPU architectures that enables well behaved multiprocessing and multiple or distributed controllers with a single-core processor. In addition, parallelism of the internal processing unit in the function component can be adopted. Each function component has only one role and operates independently of other components. For instance, a cluster of multiple datacenters can be expressed by multiplexing of the datastore component. Network applications such as path and resource calculations are implemented as an information

processing component. Control of switches is placed in the control component, and multiple controllers can be realized by multiplexing of the control component. Figure 4(b) shows a model variation consisting of a single controller with a multi-core environment. Fig. 4(c) shows a distributed controller architecture in which a number of multi-threaded controllers divide up the control plane. Finally, Fig. 4(d) shows an example of a distributed controller system using datacenters, that is, clusters of a huge number of servers.

C. Processing type of multiplexing

Multiplexing of the processing function component comes in two types: parallel and sequential. In the parallel type, multiple distinct components can operate at the same time regardless of the progress of other components. In contrast, the sequential type is used for the preparation of forwarding rules to be sent to switches, because invoking this procedure is the last phase of various network applications.

D. Location of the function component

Basically, all function components on the control plane are placed at the same node as a controller of the plane. However, it is possible that function components are placed at different nodes which are physically separated from other components. For example, some existing architectures manage network information in databases that are located at another nodes. In this case, the datastore component operates at a different node from the one where the control and the information processing components are located.

E. Processing timing

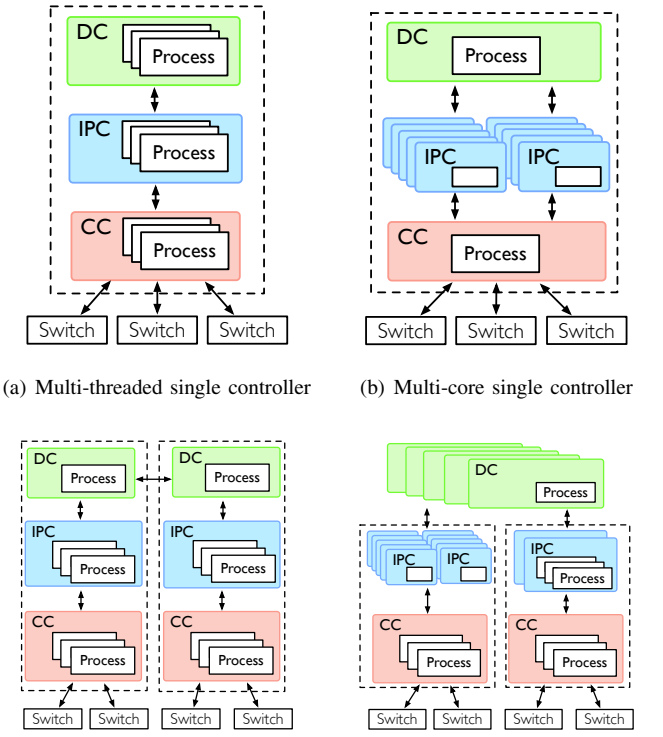
When multiple function components operate together at the same time, timing among the components is very important for reliable operation. We should consider two types of processing timing modes: synchronous and asynchronous. In the synchronous mode, the operation of each component is dependent upon other components. Though any component operates independently of others in the asynchronous mode, the consistency of the information that multiple components may access has to be maintained by adopting suitable procedures, such as an exclusive lock mechanism.

F. Connectivity

The connectivity, that is, the communication style between function components, is basically dependent on the number of multiplexed components. It is a very simple case, one component connects to another directly. However, if both function components are multiplexed, various connection forms can be considered. To distribute incoming requests or messages to multiple components, we introduce a buffer attached at the front end of the multiple components as one variation of our model.

V. Validation of our abstract model

In this section, we validate our model by expressing some existing SDN architectures using the model. We have to confirm that our abstract model can express SDN architectures proposed in previous studies and simulate the results shown in the studies. Moreover, our simulated results should be accurate enough so that a unified evaluation can be achieved using the model. We evaluate model by metrics of the performance of the control plane, which include the response time for new flow requests and the throughput of the controller.



(a) Multi-threaded single controller (b) Multi-core single controller

(c) Distributed controllers (d) Distributed controller with datacenters

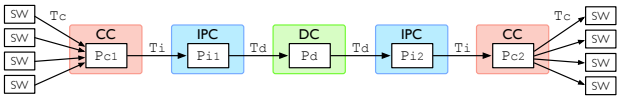
Fig. 4. Model variations for SDN controller architecture

Our validation step starts with the construction of a model variation based on our abstract model. By specifying various sets of parameters, we express various existing architectures in a single model. The model parameters are estimated using the values obtained for existing architectures. Finally, we compare the result of our variations with the original results from the previous studies.

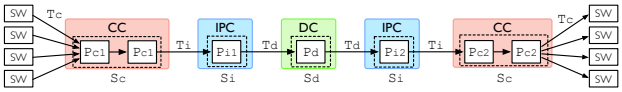
A. Construction of model variations

Fig. 5 shows some structural examples of model variations expressing typical controller architectures. To illustrate the construction of variations of our abstract model, we consider a variation of the model in which the procedure is of the reactive control that is basically used to handle network events that occur in switches. We first describe the model parameters which characterize the structure of the model.

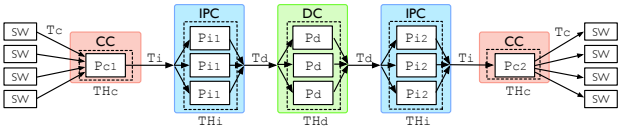
The processing delay is the time taken to handle the message that a component receives, and varies depending on the processing content of the component. The number of parallelized threads affects the delay if the component is multi-threaded. We represent the process delay as P_c on the control component, P_i on the information processing component, and P_d on the datastore component. The delays P_c and P_i



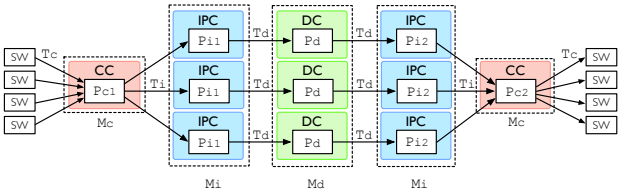
(a) Single controller



(b) Single sequential controller



(c) Single multi-threaded controller



(d) Single multi-core controller

Fig. 5. Validation architectures of our reactive-type model

are divided into two parts according to the direction of the processing: P_{c1} and P_{c2} for P_c; P_{i1} and P_{i2} for P_i. For instance, P_{i2} corresponds to the process of routing calculation or other applications because this component operates after some information is received from the datastore component. We assume a generic processing task, such as routing calculation, as the processing content of each function component, and also that the processing delay follows a probability distribution.

The transmission delays between each function component are given by T_c, T_i, and T_d. If all components are implemented in the same node, these transmission delays are negligible. However, there are model variations that placed these component at physically separated nodes.

Each function component can operate the internal processing unit sequentially, or execute parallel processing by some technique, such as multi-threading. The numbers of sequenced processing units for each function component are given by S_c, S_i, and S_d. Moreover, the numbers of parallelized threads in each function component are given by TH_c, TH_i, and TH_d. Figs. 5(b) and 5(c) show examples of the basic structure for this variation.

Fig. 5(d) shows a model variation for a single controller with multiplexed information processing and datacenter components. The parameters M_c, M_i, and M_d denote the degree of multiplexing of the respective function components. Generally, this multiplexing degree corresponds to the number of controllers and the number of cores in a controller.

B. Expressing existing SDN architectures using our model

By tuning the model parameters described in the previous section, we can express some existing SDN control architectures using our abstract model. In order to validate our model, we targeted six architectures that evaluation results about performance have been presented in the literatures: NOX, NOX-MT, Maestro, Beacon, FlowN, and ElastiCon.

NOX works as a single-threaded controller, and can be expressed as a basic configuration of the model. The degree of multiplexing and number of threads of all components in NOX are set to 1. NOX-MT is a multi-threaded successor of NOX and has a parallel processing unit for received messages. Thus, the information processing component of NOX-MT consists of multiple threads. Moreover, the processing delay in the control component of NOX-MT is comparatively short because this architecture reduces I/O overhead by applying I/O batching and Boost Asynchronous I/O. Maestro is multi-core, single-threaded controller. In Maestro, a received message is assigned to a core processor, and processed independently of other messages, that is, the information processing component is multiplexed. Beacon is also a multi-threaded server; its I/O threads can handle the processing content for each switch independently. FlowN provides several virtual networks on a physical network. The management application of each virtual network executes on a single controller. FlowN distributes requests from the real network to the management applications, and manages the mapping between the request and the application. Therefore, the information processing component of FlowN has two sequential processing units. ElastiCon is a distributed controller system, which has multiple controllers. Thus three components in the control plane are multiplexed according to the number of controllers. Moreover, it runs on a multi-core server and the number of threads of each component determines the multiplexing in each core processor.

For the processing delay, the exponential distribution is commonly used for queueing theory. However, the simulation results with that distribution have less variations compared to the original ones. Therefore we decided to basically use a log-normal distribution that has a heavy tail in all function components. In the case of existing architectures with multiplexing technology, allocation of control messages to each component in a certain manner is required. Thus we consider an assignment time that is proportional to the degree of multiplexing, n. We use a natural log of n and an exponential distribution to estimate the assignment time. For the transmission delay, we consider that it does not affect our results significantly because we compare the performance caused by the architectural difference of each method under the same condition. Therefore we set all transmission delays to zero at this time.

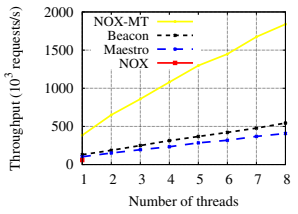
Tables I and II show the model parameters used to express the six targeted architectures in our model. Based on these parameters, we build simulation by using OMNeT++ [20], which is an event-driven network simulator and carried out measurement along the original experiments. Then we compare the results provided in each study with the simulated results from our model with these parameters.

TABLE I. PARAMETERS IN THE ABSTRACT MODEL: PROCESSING AND TRANSMISSION DELAYS

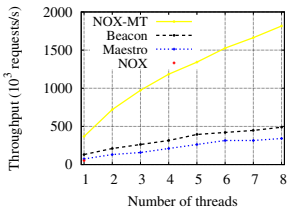
Parameter	P_{c1}	P_{i1}	P_d	P_{i2}	P_{c2}	T_c	T_i	T_d
NOX	LN(-12.0, 1)	0	LN(-14.1, 1)	LN(-13.6, 1.5)	LN(-12.0, 1)	0	0	0
NOX-MT	Exp(0.8us*log(N))	0	LN(-15, 1)	LN(-13.6, 1.5)	LN(-14.1, 1)	0	0	0
Beacon	Exp(5us*log(N))	0	LN(-14.1, 1)	LN(-13.6, 1.5)	LN(-12.23, 1)	0	0	0
Maestro	Exp(7us*log(N))	0	LN(-14.1, 1)	LN(-12.92, 1)	LN(-12.0, 1)	0	0	0
FlowN	0	LN(-9.808,0.005), 0	Exp(1us)	LN(-9.808,0.005), LN(-7.562, 0.005)	Exp(1us)	0	0	0
ElastiCon	Exp(85us*log(N))	0	LN(-14.1, 1)	LN(-10.125, 1.5)	LN(-14.1, 1)	0	0	0

TABLE II. PARAMETERS IN THE ABSTRACT MODEL: THE DEGREE OF MULTIPLEXING AND PARALLELISM

Parameter	M_c	M_i	M_d	TH_c	TH_i	TH_d	S_c	S_i	S_d
NOX	1	1	1	1	1	1	1	1	1
NOX-MT	1	1	1	1	N	1	1	1	1
Beacon	N	N	N	1	1	1	1	1	1
Maestro	1	N	1	1	1	1	1	1	1
FlowN	1	1	1	1	1	1	1	2	1
ElastiCon	N	N	N	M	M	M	1	1	1

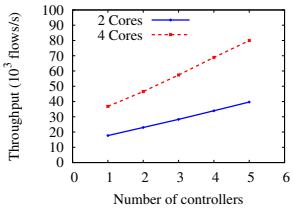


(a) Our modeling

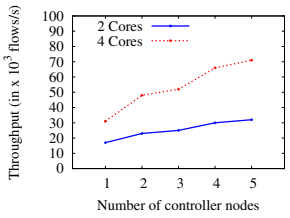


(b) Original work (in Fig.1 of [7])

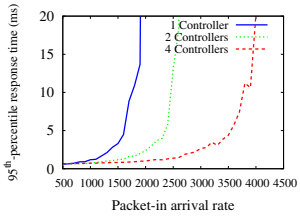
Fig. 6. Comparison of our model with various controllers



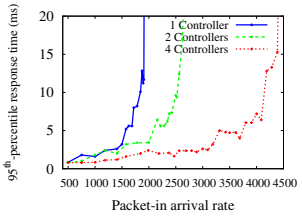
(a) Our modeling



(b) Original work (in Fig.7(a) of [3])

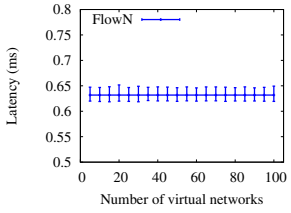


(c) Our modeling

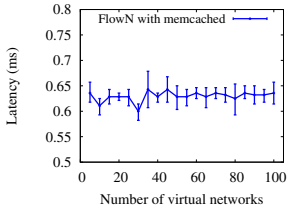


(d) Original work (in Fig.7(b) of [3])

Fig. 7. Comparison of our model with ElastiCon



(a) Our modeling



(b) Original work (in Fig.2 of [21])

Fig. 8. Comparison of our model with FlowN

C. Comparison results

Fig. 6(b) shows the average maximum throughput of 4 controllers with different numbers of threads. The result of our simulation in Fig. 6(a) shows a very similar performance to that of existing controllers.

Figs. 7(a) and 7(b) show the throughput of ElastiCon with 2 and 4 cores for different numbers of distributed controllers. As the number of controllers increases, it seems that the value of throughput simply doubles. However, Fig. 7(b) shows that the throughput is not doubled by a doubling of the number of controllers. A distributed controller depends on cooperative processing when, for example, sharing the network view. Therefore the throughput of a distributed controller does not scale up linearly. In our simulation, we set this processing delay to be proportional to the number of controllers by an appropriate choice of model parameter. Comparing these figures, we see that the change of the rate of throughput by increasing the number of controllers on our simulation follows the actual result closely. Therefore, our model can reproduce ElastiCon well. Fig. 7(c) shows the 95th percentile response time of different number of ElastiCon controllers with various packet-in message arrival rates in our simulation. As seen in Fig. 7(d), the results with 1 and 2 controllers have a similar trend to the results of real ElastiCon. Although the response time in our model becomes large slightly earlier than for a real ElastiCon network with 4 controllers, it almost reproduces the actual result.

Fig. 8(b) shows the average latency of FlowN with different numbers of managed virtual networks. The result of our simulation in Fig. 8(a) remains close to 0.63 ms, which falls within the error bar range of Fig. 8(b). From these results we conclude that our model can satisfactorily represent various types of SDN architecture.

VI. UNIFIED EVALUATION OF EXISTING ARCHITECTURES

In this section, we perform a comprehensive performance comparison of SDN architecture by using represented model

TABLE III. PARAMETERS FOR UNIFIED EVALUATION: PROCESSING AND TRANSMISSION DELAYS

Parameter	P_{c1}	P_{i1}	P_d	P_{i2}	P_{c2}	T_c	T_i	T_d
FlowN	0	LN(-9.308,0.005), 0	Exp(1.65us)	LN(-9.308,0.005), LN(-7.06, 0.005)	Exp(1.65us)	0	0	0
ElastiCon	Exp(5us*log(N)+2.7us*log(M))	0	LN(-14.1, 1)	LN(-13.6, 1.5)	LN(-12.23, 1)	0	0	0

TABLE IV. PARAMETERS FOR UNIFIED EVALUATION: THE DEGREE OF MULTIPLEXING AND PARALLELISM

Parameter	M_c	M_i	M_d	TH_c	TH_i	TH_d	S_c	S_i	S_d
FlowN	1	1	1	1	1	1	1	2	1
ElastiCon	M	M	M	N	N	N	1	1	1

variations in the previous section.

In order to compare various architectures fairly, we have to take into account differences of the evaluation environment (i.e., CPU and RAM) of each approaches. We estimated each set of parameter values of which can express each architecture proposed in the original literatures respectively. However, it is not fair and inadequate to compare these approaches by using these parameters, because each approach has been realized in distinct environment. Therefore, we should revise a set of parameter values, especially the processing delay in our function model, to perform a comprehensive performance evaluation. We cannot obtain precise environment settings of each original work or literature. We adjust each set of parameters according to the structure of each approach because we consider that a similar architecture has a similar set of parameter values.

Fig. 9 shows the throughput of six existing architectures with different numbers of threads. In the original literature [7], only 4 controllers are evaluated using this metric. However, we can evaluate all SDN architectures with the same metric using our abstract model because our model can represent all of them. On tuning parameters of ElastiCon, the original literature does not clarify the environment of implementation and evaluation. Therefore we used Beacon's delay parameters for ElastiCon because these two approaches are designed on the basis of Floodlight, the architecture of which is similar to each other. Since ElastiCon is a distributed controller system, we include the number of controllers in the processing delay of P_{c1}. For FlowN, we simply focus on CPU clock rate of the machine on which the controller application is running. The original literature of FlowN indicated that the controller operating FlowN has 3.3 GHz CPU, while the other results are obtained with a controller with 2 GHz CPU. Hence, we set all of the parameters for FlowN to 1.65 (3.3 GHz/2 GHz) times larger than the original ones.

Table III and IV show the parameter sets tuned for FlowN and ElastiCon to perform a unified evaluation. To obtain the emulated results of ElastiCon, we set the number of controllers to 1, 2, and 4, as the same as the original work. Moreover, we can obtain the throughput of FlowN in case that the number of threads is 1 because FlowN is a single-threaded controller. In Fig. 9, NOX-MT shows high throughput, over 1.5 million messages can be served per second with 8 threads, that close to ElastiCon with 4 distributed controllers though it is single controller. This is mainly because NOX-MT optimizes I/O processing (i.e., I/O batching and ASIO) while ElastiCon doesn't support. This result shows that controller's I/O processing is quite important for performance. The FlowN architecture focuses on managing virtual networks and doesn't attempt to improve processing performance. Moreover, FlowN executes a mapping between physical and virtual networks, which causes extra processing delay compared to NOX; hence the throughput of FlowN is also low.

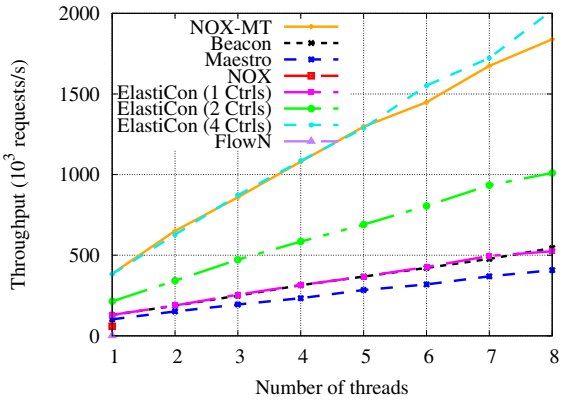


Fig. 9. Results of unified evaluation. We add evaluation of ElastiCon and FlowN to Fig. 6(a).

As described above, we can evaluate the performance of each architecture from a new unified viewpoint. In the future we will compare all architectures using various metrics.

VII. CONCLUSION

We have proposed an abstract model of SDN architecture based on function and processing sub-models. This abstract SDN model allows comprehensive performance evaluation and a unified comparison of various SDN architectures. We have validated our model by showing that various previously proposed architectures can be expressed as variations of our model. Future work includes a comparative evaluation of proposed methods for improving performance using various metrics. We will also evaluate the state of the art controllers and consider new configurations of the SDN architecture by combining multiple variations of our model. Moreover, We will perform the statistical test for simulation results with appropriate confidence intervals to clarify the validity.

ACKNOWLEDGMENT

This work was supported by a KAKENHI grant (No. 15H02694) from the Japan Society for the Promotion of Science.

REFERENCES

[1] "Software-Defined Networking: The New Norm for Networks," White Paper, Open Networking Foundation, April 2012.

[2] S. H. Yeganeh, A. Tootoonchian, and Y. Ganjali, "On Scalability of Software-Defined Networking," *IEEE Communications Magazine*, vol. 51, pp. 136–141, February 2013.

[3] A. A. Dixit, F. Hao, S. Mukherjee, T. V. Lakshman, and R. Kompella, "ElastiCon: An Elastic Distributed SDN Controller," in *Proceedings of the 10-th ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS 2014)*, Los Angeles, CA, October 2014, pp. 17–28.

[4] D. Erickson, "The Beacon Openflow Controller," in *Proceedings of the 2013 ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN 2013)*, Hong Kong, China, 2013, pp. 13–18.

[5] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: Scaling Flow Management for High-performance Networks," in *Proceedings of the ACM SIGCOMM 2011 Conference (SIGCOMM 2011)*, Toronto, Canada, August 2011, pp. 254–265.

[6] Z. Cai, A. L. Cox, and T. S. E. Ng, "Maestro, A System for Scalable OpenFlow Control," *Rice University Technical Report TR10-08*, December 2010.

[7] A. Tootonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On Controller Performance in Software-Defined Networks," in *Proceedings of the 2nd USENIX conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE 2012)*, San Jose, CA, April 2012, pp. 1–6.

[8] A. Voellmy and J. Wang, "Scalable Software Defined Network Controllers," *the ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 289–290, August 2012.

[9] A. Tootoonchian and Y. Ganjali, "HyperFlow: A Distributed Control Plane for OpenFlow," in *Proceedings of the 2010 Internet Network Management Workshop/Workshop on Research on Enterprise Networking (INM/WREN 2010)*, San Jose, CA, April 2010, pp. 1–6.

[10] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. R. Y. Iwata, H. Inoue, T. Hama, and S. Shenker, "Onix: A Distributed Control Platform for Large-scale Production Networks," in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation (OSDI 2010)*, Berkeley, CA, USA, October 2010, pp. 1–6.

[11] Y. Kawai, Y. Sato, S. Ata, D. Huang, D. Medhi, and I. Oka, "A Database Oriented Management for Asynchronous Reconfiguration in Software-Defined Networks," in *Proceedings of IEEE/IFIP Network Operations And Management Symposium (NOMS 2014)*, Krakow, Poland, May 2014.

[12] A. Krishnamurthy, S. P. Chandrabose, and A. Gamber-Jacobson, "Pratyaastha: An Efficient Elastic Distributed SDN Control Plane," in *Proceedings of the 2014 ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN 2014)*, Chicago, IL, August 2014, pp. 133–138.

[13] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable Flow-based Networking with DIFANE," in *Proceedings of the ACM SIGCOMM 2010 Conference (SIGCOMM 2010)*, New Delhi, India, 2010, pp. 351–362.

[14] M. Jarschel, S. Oechsner, D. Schlosser, R. Pries, S. Goll, and P. Tran-Gia, "Modeling and Performance Evaluation of an OpenFlow Architecture," in *Proceedings of the 23rd International Teletraffic Congress (ITC 2011)*, San Francisco, USA, September 2011, pp. 1–7.

[15] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, pp. 69–74, April 2008.

[16] K. Mahmood, A. Chilwan, O. N. Osterbo, and M. Jarschel, "On The Modeling of OpenFlow-Based SDNs: The Single Node Case," in *Proceedings of the 6th International Conference on Networks and Communications (NeCoM 2014)*, Deira, Dubai, UAE, November 2014.

[17] S. Azodolmolky, R. Nejabati, M. Pazouki, P. Wieder, R. Yahyapour, and D. Simeonidou, "An Analytical Model for Software Defined Networking: A Network Calculus-based Approach," in *Proceedings of the 2013 Global Communication Conference (GLOBECOM 2013)*, Atlanta, GA, USA, December 2013, pp. 1397–1402.

[18] J. Hu, C. Lin, X. Li, and J. Huang, "Scalability of Control Planes for Software Defined Networks: Modeling and Evaluation," in *Proceedings of IEEE 22nd International Symposium of Quality of Service (IWQoS 2014)*, Hong Kong, China, May 2014, pp. 147–152.

[19] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, and G. Parulkar, "ONOS: Towards an Open, Distributed SDN OS," in *Proceedings of the 2014 ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN 2014)*, Chicago, IL, USA, August 2014, pp. 1–6.

[20] "OMNeT++," http://omnetpp.org/.

[21] D. Drutskoy, E. Keller, and J. Rexford, "Scalable Network Virtualization in Software-Defined Networks," in *Proceedings of the IEEE Internet Computing Vol.17*, March-April 2013, pp. 1–6.