Mobile Networks and Applications (2020) 25:114–124
https://doi.org/10.1007/s11036-018-1194-9

WECAN: an Efficient West-East Control Associated Network for Large-Scale SDN Systems

Haisheng Yu[1] · Heng Qi[1] · Keqiu Li[1]

Published online: 3 January 2019

Abstract
Software-Defined Networking (SDN) has been proposed as a promising way for its centralized network control and management. However, the latest SDN research focuses on smaller network environments such as data centers and enterprises, which easily lead to single point of failure and unbalanced network load in large-scale network environments. One effective way to solve this problem is to establish a standardized mechanism between network entities such as data centers, enterprises and Internet service providers (ISPs). In this paper, we propose WECAN, an efficient West-East Control Associated Network for enabling communication between different SDN entities. WECAN has three complementary modules: Network Information Collection (NIC) module, Cross-domain Management (CDM) module and Controller Selection Management(CSM) module. NIC collects network information from a different set of controllers and generates a domain-wide network view. CDM collects domain information from other domains to generate a global network view. Base on the domain-wide network view and global network view, CSM selects the most efficient controller for each network flow in the network. To test WECAN, we develop a prototype system. Our experimental results show that WECAN can effectively control network entities to communicate, and WECAN has greatly improved network latency, network throughput and network reliability compared to a single controller-controlled network. Moreover, WECAN is very easy to use.

Keywords WECAN · Software-defined networking · OpenFlow

1 Introduction

Traditional network has been widely successful and becomes a basic infrastructure which national economies depend on. However, it has long been known that the architecture of traditional network has significant deficiencies, such as the coupling between control layer and data (or forwarding) layer, low-level configuration of individual components, complicated middle-boxes, and unfriendly Graphical User Interface (GUI). The above problem is not a cold fever, but deep-rooted in the traditional network of chronic illness, if not resolved will develop into a network of cancer. Many new network design principles and methods have been proposed to solve this problem.

Software-defined networking (SDN) is one of the most watched solutions. SDN not only simplifies network management, but also enables the network to meet specific end-to-end requirements. Unlike traditional network implementations, SDN recommends separating the control layer from the forwarding layer and centralizing the decision making for higher-level routing decisions to the control layer.

So centralized control is key to SDN.

However, operators only have limit degrees of control on SDN. There are a variety of SDN controllers, including Floodlight [1], Ryu [2], OpenDaylight [3], Beacon [4], ONOS [5], Pox [6], Trema [7], Nox [8] and so on. Each SDN controller provides their own API to manage the network and forwarding path computation. One key limitation of those controllers is that each SDN controller has its own communication mechanism that makes SDN networks very difficult to exchange information between

✉ Heng Qi
hengqi@dlut.edu.cn

Haisheng Yu
yuhaisheng1@gmail.com

Keqiu Li
keqiu@dlut.edu.cn

[1] School of Computer Science and Technology, Dalian University of Technology, No 2, Linggong Road, Dalian 116023, China

different domains. The domain can be a ISP network or an enterprise network, and it can also be a data center.

The goal of this paper is simple: without compromising or even improving the performance of the network, different networks and network controllers can communicate with each other. To provide the necessary context, we start by asking the following questions.

1. Does different SDN controllers need to exchange network information (e.g. topology and routing table) with each other?
2. How and What does different controllers communicate with each other?
3. Which controller should be responsible for processing the incoming network flow?

The rest of the paper is organized as follows. We describe the background and motivation and answer the question 1 in Section 2. We present the design principles and answer the question 2 in Section 3. We describe the implementation and application and answer the question 3 in Section 4. We evaluate the performance of WECAN in Section 5. Finally, we conclude it in Section 6.

2 Motivation and problem statement

Much research has been done on how to build SDN management layer [9–17] but these efforts are mainly focused on smaller, simply organized networks such as data center networks and enterprise networks.

In these proposals, SDN usually has three different functional layers, such as application layer, control layer, and data layer [18–20]. OpenFlow [21] is a standardized protocol between control layer and data layer, enabling SDN to evolve rapidly. However, there is no standardized protocol between controllers, which makes the development of SDN in the hybrid and complex network environment is hampered. Figure 1 provides a common SDN management architecture that shows the detailed components of the architecture.

In this architecture, How to use the network is determined by the application layer.

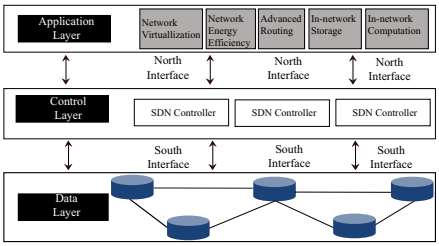

Fig. 1 Common SDN architecture has three layers: application layer, controller layer and data layer

The control layer counts the data packets and calculates the status of the network. Then, the control layer generates the forwarding rule and sends it to the data layer. The data layer processes a single data packet according to the rule.

Over the years, a large number of controllers were developed for this architecture: Beacon [4], Floodlight [1], NOX [8], POX [6], Ryu [2], Trema [7], OpenDaylight [3] and the nearest ONOS [5]. The 3-layer architecture successfully solves the problem of SDN southbound communication between the controller layer and the data layer. But it does not tell us how the northbound protocol between the application layer and the controller layer works. In this paper, the southbound communication refers to the network communication between the SDN controller and the data layer (forwarding device). The northbound communication refers to the network communication between the SDN controller and the network application. The East-west communication is the network communication between SDN controllers.

The basic assumption of a 3-layer architecture is the southbound interface between the data layer and the control layer, and the northbound interface between the application layer and the control layer is standardized. In fact, the northbound interface has been changing. Worse, there are many types of SDN controllers that cause the northbound interface to be impossible to standardize. Each SDN controller has its own application interface. While all SDN controllers share a common basic interface set, when you dig deeper, you will find more and more different advanced features. For example, both Pox and Floodlight have an interface for getting the switch and traffic information, but if you want to delete a flow entry, it's easy to do in Floodlight and it's hard in Pox. Floodlight's web interface uses a static push interface to easily remove the flow entry, but it's hard to fix it in Pox. At the same time, different controllers have different functions. For example, OpenDaylight [3], Ryu [2] and Floodlight [1] all have an interface for getting switch and flow information, but a firewall only exists in OpenDaylight, a load balancer in Ryu and a monitoring application in Floodlight.

Another issue is the lack of scalability of the three-layer SDN architecture. An SDN controller has limited processing power: NOX is capable of processing 30K requests per second, while Floodlight can handle about 250,000 requests a second. Maestro [22] has the strongest processing power and can handle about 300,000 requests per second. In order to achieve a large-scale network environment and achieve a scalable control layer, many recent papers explored the architecture of a large-scale or global-wide SDN controller, including: [13, 14, 23, 24], and they focus on the necessary components to implement such an SDN controller. A key limitation of these controllers is that each SDN controller has its own communication mechanism that makes it difficult for the SDN network

to exchange information between different domains. The domain can be a data center network or an enterprise network, or it can be an ISP network.

In order to solve these problems, many efforts have been made to find an effective way to redesign the SDN architecture. In order to solve the single-controller failure problem, ElastiCon [25] proposed a dynamic protocol-based load balancing system using dynamic migration protocols between controllers. HyperFlow [23] proposes a logically centralized but physically distributed controller that updates the state of other physical machines on the network through the publish/subscribe system. By connecting switches to the closest part of the controller and passively synchronizing the OpenFlow controllerś entire network view, HyperFlow positions the decision to a separate controller to minimize the control-layer response time to the data layer. However, if a HyperFlow controller fails, all the switches connected to the failed controller must be reconfigured to connect to the new controller. In DIFANE [15], the ingress switch redirects the packet to an "authority" switch that stores all forwarding rules, while the ingress switch caches traffic flow entries for future use. The controller is responsible for splitting the rules to the authority switches. Onix [26] is the first true logical centralized physical distributed controller that enables global network views. To improve scalability, Onix supports creating new instances with new scopes by aggregating or partitioning. Because the new scope is limited to devices that are physically close to each other, Onix is a flat architecture that can handle only one level of process. In addition, it is not easy to define a sub-scope for a policy because it is the same application to resolve the conflict when it happens in the process of aggregation or partitioning. Onix and HyperFlow reduce the look-up overhead by enabling communication with local controllers, while still allowing the application to be written with a simplified central view of the network. The potential drawback in the distribution of the entire control layer is the trade-offs associated with consistency and stability, which may cause the application to believe that their accurate view of the network is not properly performed. For high availability and horizontal expansion, ONOS [5] uses a distributed architecture and provides a global network view of the application. Onix and HyperFlow reduce lookup overhead by enabling communication with local controllers while still allowing applications to be written with a simplified network view. The potential downside is that due to consistency and obsolescence trade-offs, the application may cause incorrect behavior because the application does not have an accurate view of the network when the network state is distributed in the control layer. FRESCO [27] uses independent modular libraries and assembles them to provide complex network functions. Our approach, on the other hand, tries to provide complex network functions based on existed controllers. Software-Defined Internet Architecture(SDIA) [11] is dedicated to breaking down Internet services into well-defined tasks and how to implement them in a modular fashion. As a network design principle, 4D project [28], starting in 2004, advocate a clean slate design. It propose three key principles, namely, direct control, network-level goals and network-wide view. It suggest to separate route decision logic from the protocol governing network element interactions. Meanwhile, it recommends giving the decision layer a global network view, serviced by the dissemination and discovery layer, to control the data layer for forward traffic. FlowBricks [29] considers the interaction between different controllers when the network size is small, and does not consider the interaction of different controllers in a large-scale network.

In this paper, We propose WECAN, which shares the 4D's view that decision, dissemination and discovery function should be dispersed to multi-layers, can make a clear separation between application layer and control layer, which makes network administrator much more smoothly manage a network. The detailed design can be found in Section 3. The goal of WECAN is to resolve communication problems between different controllers. In fact, our research has been recognized by industry and academia. The system is deployed in both CLOUDLAB [30] and CENI [31]. Compared with current controllers (such as floodlight, ryu and ONOS), WECAN can support multiple types of controllers, and WECAN is more convenient and easy to use. For example, it supports to send the flow table rules through the web interface and also supports the import and export of topologies. These features are not supported by other SDN controllers. The significance of WECAN is that it obscures the differences in network types, sizes and controllers that network managers have to focus on. We expect to make the communication platform between SDN domains easier to use through our work, and we hope this work can help to standardize the West-East communication protocol of SDN controller.

3 WECAN design

In this section, we present the design of the promising SDN architecture, WECAN for improving the SDN management. First, to support complicated networks, WECAN has to meet requirements for Privacy, scalability, Generality, and Compatibility. Based on the requirements, we introduce how to design Northbound API, domain relationship and the database in Section 2. Then, we describe the four layers in WECAN architecture and gives a detailed introduction into the decision layer which is the core component in WECAN.

3.1 Design requirements of management layer

In this subsection, we want to highlight the importance of design requirements in a common SDN architecture. We insist that as long as a large-scale network works well it should satisfy the following requirements:

High Scalability. Since SDN controller can feasibly manage a limited number of devices, there's doubting that a reasonably large network should deploy more than one SDN controllers. When it comes a large network, there may be hundreds of thousands of controllers.
Enough Privacy. Operators may choose to implement different privacy policies in different SDN domains. For example, if most users in one SDN domain have their own privacy policies, the network information (eg, network topology) in this domain should not be exposed to external entities.
Great Generality. If SDN replaces the positions of the traditional network, a standard communication protocol is needed to deliver a necessary message between domains, such as accessibility, topology, and bandwidth.
Good Compatibility. To improve the compatibility with the traditional network, we should understand the relationship between traditional domains. Then we must create a communication protocol or use existing protocols to control the hybrid network.

3.2 Northbound API, domain relationship, and data storage

In this subsection, we want to introduce some key points in WECAN design.

Northbound API There are many successful solutions in resolving SDN southbound communication problem, but very few is proposed in resolving the northbound problem between application layer and controller layer. The main reason is that we treat the subject without ever clarifying this issue that the SDN routing problem needs to be decomposed into intra-domain relationship problem and inter-domain relationship problem.

To meet the above-mentioned requirements, WECAN add a new layer, decision layer, a new layer between the application layer and the controller layer, which manipulates different controller's interface and provides application layer with a standard interface. Decision Layer is the core component in WECAN, which acts as a standard communication protocol between application layer and controller layer. It centralizes all the network intelligence and network control, such as access control and routing decisions. In WECAN, the control layer is called the dissemination layer which function is to collect information and send it to the decision layer, and distribute the routing information delivered by the decision layer. The data layer accepts the routing information sent by the dissemination layer and forwards the data packets according to the routing. information.

Domain Relationship The concept of SDN domains was introduced to support the need for partitioning a network control layer among different controllers within an administrative domain. An SDN domain can be defined as the portion of the network being managed by a particular SDN controller.

For large-scale networks, only one controller is not sufficient due to the scalability and reliability requirements. For example, a SDN controller has limited processing power: NOX [8] handles about 30K requests per second while Floodlight can handle about 250K requests per second. In order to implement a scalable control layer in a large-scale network environment, it is necessary to allow multiple controllers to control multiple domains simultaneously. Meanwhile, WECAN suggest to decompose Internet service into well-defined tasks. According to the domain division, WECANś task is divided into two kinds of inter-domain tasks and intra-domain tasks.

Network View Network view is an important basis for the decision of network routing. Many SDN controllers provide a network view in the control layer. For instance, Onix use Network Information Base Details(NIB) to store network view. NIB has a set of network entities which stores the key-value pairs and identified by a flat 128-bit global identifier [26]. In the initial SDN architecture, the control layer is mainly realized by a single controller which can easily get the network view, so the developers can focus on features and functionality, as well as performance. However, with the diversification of network application requirements and the expansion of network size, it is difficult for one controller to control the entire network. What followed was a tremendous challenge to getting the network view.

WECAN partitions network view into two parts, namely global-wide network view and domain-wide network view. Domain-wide network view only stores information in a domain, whereas global-wide network view stores a domain's connection with other domains. According to the division of the network view, WECAN decomposes Internet service into interior domain tasks and exterior domain tasks. Interior domain tasks include collecting network information from a group of controllers with no interconnection and generating a domain-wide network view. Exterior domain tasks include collecting network information from other domains and generating a global-wide network view.

Table 1 Ten core tables in WECAN

Table	Explanation
w_controller_info	Controller type,such as ONOS,Floodlight
w_ip_domain	ip belongs to domain information
w_domain_info	Domain information
w_domain_relation	Relation between domain
w_flow_info	Flow entry information
w_host_info	Host information
w_link_info	Link connect switch
w_port_info	Port belongs to switch
w_port_stats	Port statistics information
w_switch_info	Switch information

Data Storage TO store network information, WECAN database have ten core tables. Table 1 gives WECAN database table names and explanations. Eight of the ten tables are used to represent information in a network domain: domain base information, controller, traffic, host, link, switch, port belongs to switch, and port statistics. Meanwhile, the other two of the ten tables describe the domain relationship. The w_ip_domain table shows which IP segments are included in a domain. The w_domain_info table shows the interrelationships between domains.

3.3 4-Layer SDN architecture

It is easy to understand how WECAN realizes a control platform if we know the features of each layer in the platform. In fact, WECAN has four layers which have very distinct roles (see Fig. 2). Compared to 3-layer platform, we represent significant principles and major challenges in designing WECAN.

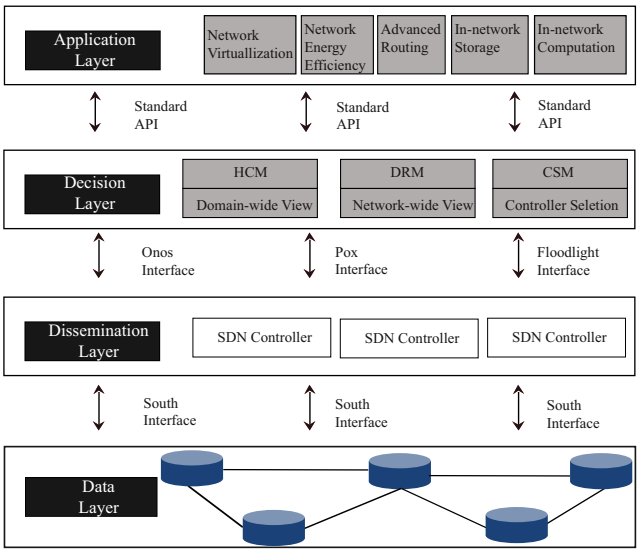

Fig. 2 WECAN architecture has four layers, decision layer is a newly established layer between application and control layer

Application Layer: Customized Demand How a specific application use a network is determined by the the application layer. These applications provide network virtualization, network efficiency, advanced routing, network storage, network computing and other functions. As the operating system has a graphical user interface (GUI) desktop, WECAN also provides a GUI that allows users to more easily understand and operate the network, that is, the application layer. However, the current application layer of the SDN controller generally provides simple network information browsing functions, which does not provide flow entry addition and modification operations, and does not provide communication between different types of SDN controllers. To address this issue, WECAN set up a convenient and easy to use web control interface based on FLEX, which can control many different types of SDN controllers.

Decision Layer: Control Logic The core component of WECAN is decision Layer. The decision layer consists of multiple servers called decision elements that connect directly to the network. Compared to three-layer SDN architecture, decision layer in WECAN is a newly established layer between SDN controllers and applications. Decision layer makes all decisions driving network control, including interface configuration , security, load balancing, accessibility and access control. Instead of traditional network's management layer, topology decision layer operates in real time on a network view of the resource limitations, the capabilities, and the traffic and of the physical devices. The decision layer uses algorithms to turn network-level objectives (e.g., reachable matrix, load-balancing goals, and survivability requirements) directly into the packet-handling rules (e.g., Routing rules, queuing parameters, forwarding table entries, packet filters) that must be configured into the data layer. Decision layer in WECAN has three complementary modules: Network Information Collection (NIC) module, Cross-domain Management (CDM) module and Controller Selection Management(CSM) module. NIC collects network information from a different set of controllers and generates a domain-wide network view. CDM collects domain information from other domains to generate a global network view. Base on the domain-wide network view and global network view, CSM selects the most efficient controller for each network flow in the network.

Dissemination and Discovery Layer: Cache Server Dissemination Layer in WECAN acts as an intermediary between potential hundreds of switches and remote application servers by congregating requests from switches into various servers. In the process, a SDN controller frequently requests resources to avoid contacting the server repeatedly for the resource and the route information (flow table) has not changed. The dissemination layer provides an efficient and

robust communication substrate that connects forwarding devices with decision layer. Although control information may traverse the same set of physical links as the data packets, the dissemination paths are maintained separately from the data paths. Only in this way can the control information be traversed without the need to configure or successfully establish a path in the data layer. In contrast, control information is transmitted over data paths in traditional networks, requiring the establishment of transmission routes prior to transmission of control information. In WECAN, the dissemination layer does not create control information itself, so it only need to moves the control information created by the decision layer to the data layer and sends the network state information identified by the discovery layer to the decision layer.

Data Layer: Forwarding Data The data layer is the carrier of SDN network, includes network switches, and any other network elements that support an interface allowing Dissemination Layer to read and write the state controlling the element's behavior (such as forwarding table entries). The function of the data layer is mainly confined to packet forwarding and simple processing. However, it is necessary to build a flexible and easily configured SDN data layer in order to adapt changing demands placed on the application by the end users, the new personalized requirements of data center network and other network application scenarios.

4 Implementation and application

To prove the solution, we design and implement a prototype system that use Floodlight and Maestro as SDN controllers.

4.1 WECAN implementation

The core component in WECAN is decision layer which has three sub-modules: Network Information Collection (NIC) module, Cross-domain Management (CDM) module and Controller Selection Management(CSM) module.

Network Information Collection (NIC) Network Information Collection is responsible for managing different controllers, such as Floodlight and Maestro. NIC gathers information from these controller's API and other event source systems, such as network monitoring system. NIC only considers the flow entries export policies in a domain, while policies between domains are resolved by Cross-domain Management (CDM).

The NIC has four components, including data storage, dissemination and discovery, routing decisions, and network display (see Fig. 3). The network display are responsible for displaying and managing inbound, traffic and topology information through front-end tools, which may be Web sites or software clients. In our prototype, the network display is a Web site and is written in flex language. The purpose of routing decisions is to generate new network configurations and new forwarding rules based on changes in network topology and statistics. Dissemination and discovery send network topology and statistics to routing decisions and distribute new forwarding rules to forwarding devices.

Domain Relationships Management For a large-scale network, it is generally known that different domains should exchange information. Before design a large-scale SDN network, it is very important to define inter-domain relationships. The knowledge of inter-domain relationships has many applications and usage in routing decision. First, it is crucial in network service management decision including the optimal placement of controllers and switches. Second, it can help domain administrators to achieve load balancing, congestion avoidance and fault tolerance. Third, it can aid domain administrators in planning for future contractual agreements. Fourth, it can help domain administrators to reduce the effect of the misconfiguration and to debug switch configurations.

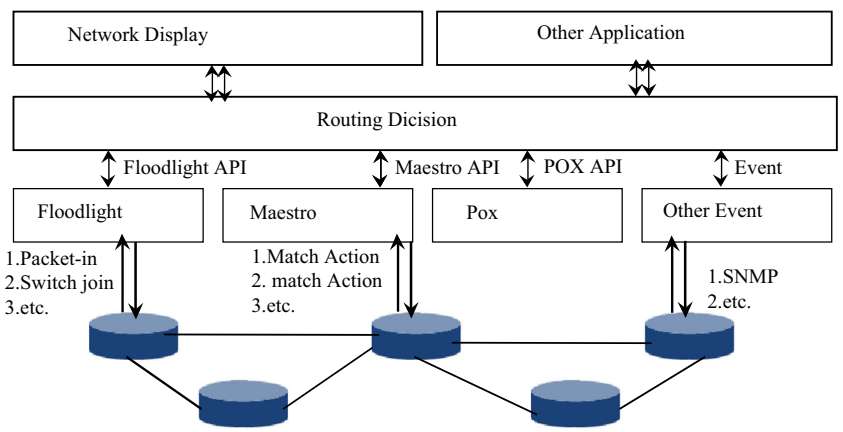

Fig. 3 There are four components in NIC, such as data storage, dissemination and discovery, routing decision and network display

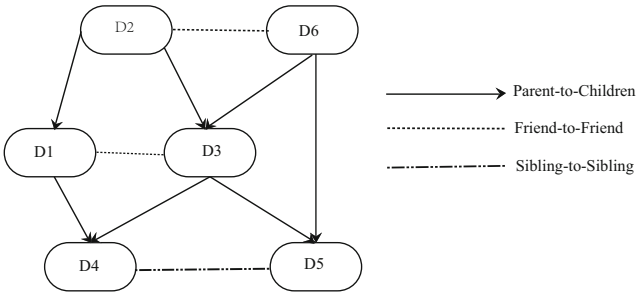

Fig. 4 Route paths (D1,D2,D3) and (D1,D2,D6,D3) are valley-free while Route paths (D1,D4,D3) and (D1,D4,D5,D3) are not valley-free

To address this problem, we propose a controller graph representation that classifies domain relationships between children-to-parent (c2p), friend-to-friend (f2f), and sibling-to-sibling (s2s).

In the c2p category, a children pays a parent for any traffic sent between themselves. In the f2f category, two domains freely exchange traffic between themselves and their children, but do not exchange traffic from or to their parents or other friends. In the s2s category, two domains administratively belong to the same organization and exchange traffic between their parents, children, friends, or other siblings. Figure 4 gives a example of domain relationships. Our solution is rather standard, borrowing heavily from long standing Autonomous System (AS) deployment practices.

Table 2 shows common relationships between domains and the export policies associated with them. Parents provide transit to children; friends exchange only traffic that is sourced and sinked by them, their children or their siblings; and siblings provide mutual transit.

Controller Selection Management In a distributed SDN system, there may be multiple controllers handling traffic at the same time. So how to choose the right controller to improve system processing efficiency?

Here we describe the four-step improvement of the controller selection algorithm.

Step 1 Random Selection: Random selection means that when a batch of requests arrives, the coordinator randomly selects a controller for a request. When the system has some switches and few controllers, random selection is sometimes more efficient. When a random selection is made, the cost of acquiring controller load information is zero, so the controller selection cost is also lower.

Step 2 power of two choice: Random selection is very effective when the network is small but decreases when the network becomes large or the controller load is unbalanced. For example, when the network has two controllers A and B, the random selection algorithm chooses a full-load A controller to handle a request. Since the idle B controller is not used, the average controller efficiency decreases at this time. To prevent this from happening, we use the power of two choice algorithm [32] to solve this problem. Under the power of two choice algorithm algorithm, the coordinator first selects two or more controllers for a request and then distributes the requests to the controller with the lowest load.

Step 3 Batch-Sampling:In the power of two choice algorithm, because each task need to get the selected two controller load information, which undoubtedly increase the cost of the controller selection. So how to reduce this cost? Here we use the batch sampling method. When a batch of m tasks arrive, the system no longer handles m tasks at once. But gets the load information of $2m$ controllers and selects m controllers from the $2m$ controllers to process the m tasks. We call this algorithm batch-filling. The batch-filling algorithm helps to reduce the overhead of getting the controller load.

When using the power of two choice algorithm, since each task needs to get the load information of the two or more controllers, this undoubtedly increases the cost of controller selection. So how to reduce this cost? Here we use the batch-filling algorithm which helps reduce the overhead of getting the controller load. When a batch of m requests arrives, the coordinator does not process the requests one by one but instead gets the load information from $2m$ controllers and selects the m controller from the $2m$ controller to handle the m requests.

When using the power of two choice algorithm, the cost of controller selection increases undoubtedly as each task needs to obtain load information for two or more controllers. So how to reduce this cost? Here we use the bulk sampling method. When a batch of m requests arrives, the coordinator does not process the requests one by one, but instead takes the load information from the $2m$ controller and selects the m controller from the $2m$ controller to handle the m request. The bulk-filling algorithm helps reduce the overhead of getting the controller load.

Table 2 WECAN export policy

Route export	Export policy
Children to parent	Only routes received from children and sibling
Friend to friend	Only routes received from children and sibling
Parent to children	All routes
Sibling to sibling	All routes

Step 4 Batch-Filling: The batch-sampling algorithm first probes the load of $2m$ controllers and then selects m lower-load controllers to handle m tasks, meaning that each controller handles a task. But sometimes this algorithm is inefficient, for example, m controllers have two controllers A and B, A load of 10%, B load of 90%. Here we assume that handling each task will increase the controller by 10% of the load. Obviously, if the task assigned to the B controller is transferred to the A controller, the efficiency will be higher. This is like pouring water into the cups of m different water levels. First, water is required to be injected into the cup with the lowest water level until the water level of the cup is aligned with the penultimate cup of the water level. And then continue with this process. We call this algorithm batch-filling. The batch-filling algorithm can increase the load balancing of the system and be more efficient.

4.2 Applications

In this section, we discuss some application being built on top of WECAN. We make a prototype based on WECAN and successfully integrate two different controllers: Floodlight [1] and Maestro.

Use Case1: Access Control In order to ensure network security, network devices need to filter traffic in the network, which requires the use of Access Control (ACL). Software defined networks (SDNs) make it easier to build access control lists (ACLs) than traditional networks. However, how to build an effective and efficient ACL application in SDNs needs a lot of work. So far, there has been some research on the use of ACLs in SDNs, but most of the existing work uses passive methods to implement ACLs so that new ACL updates can not be done immediately. Based on WECAN we propose CLACK, a user-driven centralized ACL method. In CLACK, users can take the initiative to implement ACL, that is, users can directly add ACL list to the network through the front-end of WECAN. The new rules added by these users are automatically updated to the network through Dissemination and Discovery Layer in real time. CLACK can avoid additional delay, save controller's resource, and also ensure network security.

Use Case2: Setup Forwarding Rules To simplify the process of setting forwarding rules, we create a web form with authentication in the front-end of WECAN. The routing decision translates user-entered rules into OpenFlow flow entries and sets the forwarding rules.

Use Case3: Network Topology Backup and Recovery Network diagnostics is very important for reproducing and diagnosing networks when something goes wrong. But how to back up the network topology and reproduce the network operation has been a big problem. In WECAN, we back up the network topology and network health into scripts and restore the network topology in a test environment. By backing up and restoring the network operating environment, WECAN offers great convenience for network diagnostics.

5 Evaluation

In this section, we evaluate WECAN in two ways: with a test application, designed to test a single WECAN instance's performance, and with cbench, used to verify WECAN's performance as a general platform. We test two key aspects of a single WECAN instance: request completion time and memory usage, and two key scalability-related aspects of WECAN: throughput and latency.

5.1 Performance metrics

A number of metrics must be determined carefully to accurately reflect the system performance. Thus, there are a number of performance metrics that evaluate our system performance, term as, response time, bandwidth, throughput and initializing time.

Response Time: The average of response time is computed to evaluate the overall system performance as follows. The response time is related to the speed at which the controller handles network traffic.

Throughput: The throughput as regarded has two aspects that the one is the throughput of WECAN and the other is the throughput of the network. Here is the WECAN throughput, which reflects the processing capacity of the WECAN, which also contains two aspects. The one is the throughput of the traffic processed by the WECAN, the other is the supporting throughput of

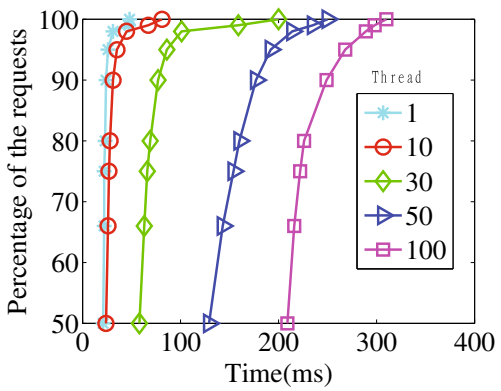

Fig. 5 Thread modification percentage of the requests complete in a certain time(ms)

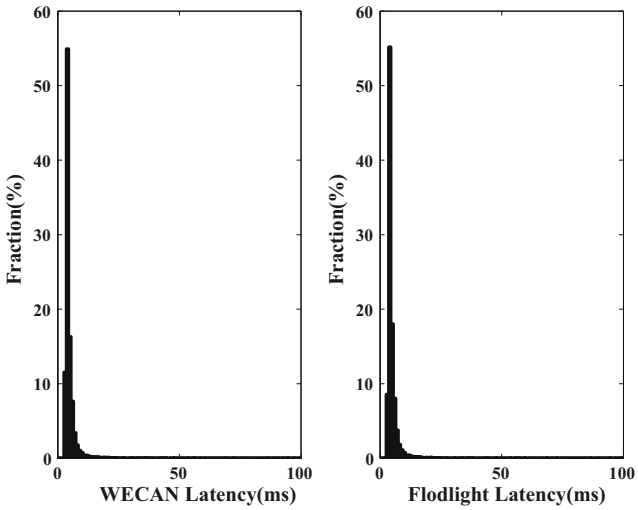

Fig. 6 Request completion time compare between WECAN and Floodlight

the JSON RPC provided by the WECAN. And network throughput reflects the performance of the network.

Memory: Memory reflects the resources WECAN occupied, when the memory usage is low, means that in the same time WECAN can process more user requests.

Initializing time: Initializing time reflects the coordinator initializing time and the consumption time of link controller.

5.2 Demonstrating the advantages of WECAN

Each WECAN instance has to connect the controllers it manages. To stress this interface, we connected three controllers to a single WECAN instance and ran apache benchmark to test WECAN's performance as a single instance. To measure this throughput, we ran an apache benchmark which repeatedly acquired exclusive access to the WECAN.

Figure 5 shows that thread modification percentage of the requests complete in a certain time. When there is only one thread, WECAN finishes 95% of the requests in 5 ms. When the threads increases, WECAN uses much less time to complete a request. As you can see from Fig. 6, it wastes no more than 3 ms to complete 95% of the requests.

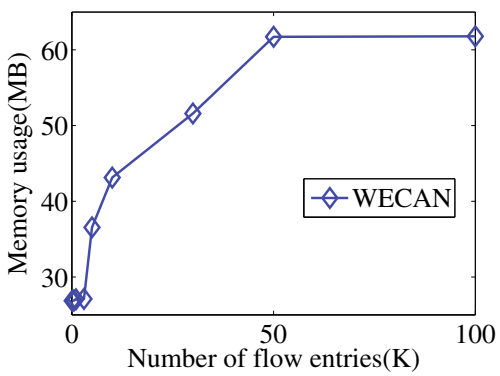

Fig. 7 Memory usage to add 1 million flow entries

Table 3 PC disposition

Product Name	ThinkPad T440
CPU	Intel(R) Core(TM) i5-4200U
Standard Memory	DDR3L-1600 8 GB
Operating System	Windows 7 Professional

Because decision layer in WECAN is a newly established layer between applications and controllers, it is necessary to compare the request time before and after decision layer established. To test the request time, we write an application which sends 100000 flow table entry update requests and monitors the time used in processing the requests. Figure 6 shows that request completion time between WECAN and Floodlight is less than 5%. WECAN requestś completion time includes Floodlight completion time, so the former spend longer time than the latter. More than 60% of the requests in WECAN is finished in 5 ms, and more than 95% of the requests finished in 10 ms, so does Floodlight. The result of this is identical to that of we do in the previous test. In the worst case, a flow table entry updates requests in WECAN can be finished in 100 ms.

Figure 7 describes the memory usage status of WECAN when adding flow entries. It shows that WECAN needs at most 60MB memory to add 1 million flow entries.

5.3 Experimental setup

To test the WECAN processing performance in a real network environment, we built a partial 4-radix FatTree. A PICA8 3297 switch is divided into seven switches. Each domain has an IP Range or an IP Range set. In our test, we construct two domain: Domain A and Domain B. Domain A's IP Range is 192.168.0.1/24, and Domain B's

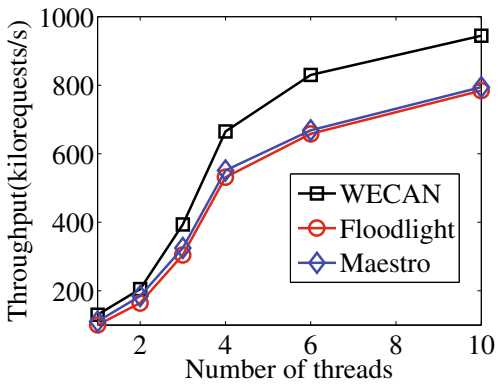

Fig. 8 The throughput of each controller connected 16 switches changes with the number of thread

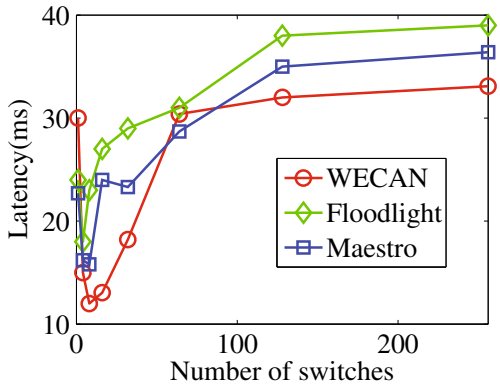

Fig. 9 Response time varying the number of switches for runs with 1 threads

IP Range is 192.168.1.1/24. We do this test on two PCs with same disposition: one host in Domain A and another host in domain B. Table 3 summarizes the parameters of the PC. Figure 8 describes the throughput of each controller connected 16 switches changes with the number of thread. Floodlight and Maestro can process 700 kilo requests per second at most. WECAN can process 900-kilo requests per second. Floodlight and Maestro do routing based on network-wide view while WECAN does routing based on simplified network wide view.

Figure 9 describes response time (milliseconds) varying the number of switches for runs with 1 threads. At the beginning, adding more CPU beyond the number of switches improve a little latency, however serving a far larger number of switches than available CPU results in a noticeable increase in the response time.

6 Conclusions

This paper presents the first effort that uses SDN four-layer platform to solve multi-SDN controller problems. The proposed solution, WECAN, might provoke interesting discussions on the research community and open the door to a range of innovation opportunities. In a word, we expect to see a new generation of SDN that is versatile, flexible, and easy to manage.

This paper makes the following contributions:

1. We propose a new layer between application and controller and design a distributed control framework, which makes heterogeneous controllers work together.
2. We have designed an initial WECAN architecture, which is capable of making kinds of controllers constitute a large control layer.
3. We proposed a state-of-art controller selection algorithm, to select the fastest and reliable controllers to handle traffic from the switches.
4. To verify WECAN, we design and implement a prototype system. In our prototype, there are two different SDN controllers: Floodlight and Maestro. Our experimental evaluation clearly indicates WECAN has higher performance than Floodlight and Maestro.

Acknowledgements This work was supported in part by the State Key Program of National Natural Science of China under Grant 61432002, in part by the NSFC under Grant 61772112, Grant 61672379, and Grant 61702365, in part by the Da lian High-level Talent Innovation Program under Grant 2015R049.

References

1. Floodlight project. http://www.projectfloodlight.org/floodlight/
2. ryu project. http://osrg.github.io/ryu/
3. Medved J, Varga R, Tkacik A, Gray K (2014) Opendaylight: towards a model-driven sdn controller architecture. In: 2014 IEEE 15th international symposium on, pp 1–6
4. Beacon project. https://openflow.stanford.edu/display/Beacon/
5. Berde P, Gerola M, Hart J, Higuchi Y, Kobayashi M, Koide T, Lantz B, O'Connor B, Radoslavov P, Snow W et al (2014) Onos: towards an open, distributed sdn os. In: Proceedings of the third workshop on hot topics in software defined networking, pp 1–6
6. pox project. http://www.noxrepo.org/pox/about-pox/
7. trema project. http://trema.github.io/trema/
8. Gude N, Koponen T, Pettit J, Pfaff B, Casado M, McKeown N, Shenker S (2008) Nox: towards an operating system for networks. In: ACM SIGCOMM computer communication review, pp 105–110
9. Casado M, Koponen T, Shenker S, Tootoonchian A (2012) Fabric: a retrospective on evolving sdn. In: Proceedings of the first workshop on Hot topics in software defined networks, pp 85–90
10. Reitblatt M, Foster N, Rexford J, Walker D (2011) Consistent updates for software-defined networks: change you can believe in! In: Proceedings of the 10th ACM workshop on hot topics in networks, p 7
11. Raghavan B, Casado M, Koponen T, Ratnasamy S, Ghodsi A, Shenker S (2012) Software-defined internet architecture: decoupling architecture from infrastructure. In: Proceedings of the 11th ACM workshop on hot topics in networks, pp 43–48
12. Sherwood R, Gibb G, Yap K-K, Appenzeller G, Casado M, McKeown N, Parulkar G Flowvisor: a network virtualization layer. OpenFlow Switch Consortium, Tech. Rep.
13. Jain S, Kumar A, Mandal S, Ong J, Poutievski L, Singh A, Venkata S, Wanderer J, Zhou J, Zhu M et al (2013) B4: experience with a globally-deployed software defined wan. In: Proceedings of the ACM SIGCOMM 2013 conference, pp 3–14
14. Hong C-Y, Kandula S, Mahajan R, Zhang M, Gill V, Nanduri M, Wattenhofer R (2013) Achieving high utilization with software-driven wan. In: Proceedings of the ACM SIGCOMM 2013 conference, pp 15–26
15. Yu M, Rexford J, Freedman MJ, Wang J (2011) Scalable flow-based networking with difane. In: ACM SIGCOMM computer communication review, pp 351–362
16. Monsanto C, Reich J, Foster N, Rexford J, Walker D et al (2013) Composing software defined networks. In: NSDI, pp 1–13
17. Qiu T, Zhao A, Xia F, Si W, Wu D (2017) Rose: robustness strategy for scale-free wireless sensor networks. IEEE/ACM Trans Netw PP(99):1–16

18. Nunes BAA, Mendonca M, Nguyen XN, Obraczka K, Turletti T (2014) A survey of software-defined networking: past, present, and future of programmable networks. IEEE Commun Surv Tutor 16(3):1617–1634
19. Wang Y, Matta I (2014) Sdn management layer: design requirements and future direction. In: 2014 IEEE 22nd international conference on network protocols (ICNP), pp 555–562
20. Qiu T, Qiao R, Han M, Sangaiah AK, Lee I (2017) A lifetime-enhanced data collecting scheme for the internet of things. IEEE Commun Mag 55(11):132–137
21. McKeown N, Anderson T, Balakrishnan H, Parulkar G, Peterson L, Rexford J, Shenker S, Turner J (2008) Openflow: enabling innovation in campus networks. In: ACM SIGCOMM computer communication review, pp 69–74
22. Cai Z (2011) Maestro: achieving scalability and coordination in centralized network control plane. PhD thesis, Rice University
23. Tootoonchian A, Ganjali Y (2010) Hyperflow: a distributed control plane for openflow. In: Proceedings of the 2010 internet network management conference on Research on enterprise networking, pp 3–3
24. Qiu T, Qiao R, Wu D (2018) EABS: an event-aware backpressure scheduling scheme for emergency internet of things. IEEE Trans Mob Comput PP(99):1–1
25. Dixit A, Hao F, Mukherjee S, Lakshman T, Kompella RR (2014) Elasticon: an elastic distributed sdn controller. In: Proceedings of the tenth ACM/IEEE symposium on architectures for networking and communications systems, pp 17–28
26. Koponen T, Casado M, Gude N, Stribling J, Poutievski L, Zhu M, Ramanathan R, Iwata Y, Inoue H, Hama T et al (2010) Onix: a distributed control platform for large-scale production networks. In: OSDI, pp 1–6
27. Shin S, Porras PA, Yegneswaran V, Fong MW, Gu G, Tyson M (2013) Fresco: modular composable security services for software-defined networks. In: NDSS
28. Greenberg A, Hjalmtysson G, Maltz DA, Myers A, Rexford J, Xie G, Yan H, Zhan J, Zhang H (2005) A clean slate 4d approach to network control and management. In: ACM SIGCOMM computer communication review, pp 41–54
29. Dixit A, Kogan K, Eugster P (2014) Composing heterogeneous sdn controllers with flowbricks. In: IEEE international conference on network protocols, pp 287–292
30. cloudlab project. http://www.cloudlab.us/
31. Ceni project. http://www.fnic.cn/
32. Mitzenmacher M, Richa AW, Sitaraman R (2001) The power of two random choices: a survey of techniques and results. Handbook Random Comput 11:255–312