International Journal of Future Generation Communication and Networking
Vol. 10, No. 10 (2017), pp.73-86
http://dx.doi.org/10.14257/ijfgcn.2017.10.10.07

CFCC: A Controller Framework Supporting Component Based SDN Applications Development

Dandan Qi[1] and Subin Shen[2]

[1]*Department of Communication and Information Engineering*
[2]*Department of Computer*
[1]*1011041127@njupt.edu.cn*, [2]*sbshen@njut.edu.cn*

Abstract

Software Defined Networking(SDN), especially, OpenFlow based SDN, has been widely aware that it facilitated creating new services and protocols, due to its programmable interface, via which programmers can program network control logic only on the controller instead of all the network devices. However, current controllers provide very low-level interfaces, leading to the high complexity involved in the process of programming SDN applications. In this paper, in order to reduce the complexity, we propose and implement a SDN controller framework, CFCC, which supports component based SDN applications development, that is, new SDN applications can be created by composing the existing component, thus reducing the complexity involved in the process of developing new SDN applications. We treat the SDN applications as a collection of interdependent components, which are higher-level functions implementations, and cooperate with each other to implement the whole function of SDN applications. Also, we demonstrate the feasibility of CFCC through developing and evaluating routing control applications upon the controller framework.

Keywords*: SDN application development, component based approach, SDN Controller Framework*

1. Introduction

Software Defined Networking (SDN) is a new networking paradigm in which the forwarding hardware is decoupled from control decisions. OpenFlow based SDN moves the control plane from the data plane into a logically centralized controller which controls behavior of the data plane and open the programmable interface, via which user can program or modify the control logic on the controller, such as routing control application [1, 2].

However, such a programmable interface is low-level, which offers basic features to developers, resulting in the high complexity involved in developing advanced SDN software applications. In this scenario, full development and deployment of such applications in staging and production environments remains a challenge for network operators [3].

Facing this challenge, this paper proposes component based approach for developing SDN applications. We choose component based approach for two reasons. First, the component based approach has the potential advantages in terms of, such as reducing development time, enhancing application quality [4], in the aspect of developing new applications, due to that it is reusing and composing the existing components instead of developing applications from scratch. Second, coincidentally, the component based approach is very suitable for developing SDN application, because we found that there existed shared functionalities across different SDN applications belonging to the same

Received (February 17, 2017), Review Result (September 5, 2017), Accepted (September 15, 2017)

ISSN: 2233-7857 IJFGCN

class. In this condition, if these shared can be encapsulated by the components to be reused across different SDN applications development, the effort to developing new applications will be reduced largely.

In this paper, a component-based controller framework, supporting efficient SDN application development, is presented. We treat the SDN application as a collection of interdependent components, which cooperate with each other to realize the whole function of the SDN application. Especially, we suggest that, for each class of SDN applications, such as routing control application, a corresponding components function encapsulating should be conducted, to guarantee a suitable granularity of components.

In order to further minimize the complexity involved in SDN application development, CFCC provide a 'drag and drop' based user-friendly environment for designing SDN applications, which allows users construct the component composition flow just by simply dragging and dropping, linking the components icons, and configuring the related parameters. CFCC converts the graphical workflow into the formal workflow based on XML, which, finally, are executed by CFCC. Such procedure is analogous to the automated web service composition, which first converts the user request into the formal composition workflow and then implements the composite service by executing the formal workflow [18].

The paper is organized as follows. In Section 2, we introduce the related work, and the differences and relations between them. In section 3, CFCC is introduced. Accordingly, in Section 4, a case study about routing control application development is presented, to demonstrate the feasibility of CFCC. Finally, the conclusion is presented in Section 5.

2. Related Work

Driven by the situation that network operators are facing high complexity involved in developing advanced SDN software applications, some researchers are trying to create high level programming language for SDN applications, such as Frenetic [5], Nettle [6], NetCore [7], Procera [8] and Pyretic [9], the main idea behind these works was raising the abstract level of the control function in the controller and then formulating abstract programming language based on these abstraction functions. For example, SDN language Pyretic [9] abstracted the inner details of the controller functions from the users, the users can use it express network policies, query network state and reconfigure networks. Compared with these works, our work enables more suitable function abstraction level, because for each class of SDN application, we conducted a specialized component function encapsulating. In addition, CFCC enables intuitively creating SDN application through dragging and dropping, and linking the components icons in the graphical composition interface provided by CFCC. In all, the suitable function abstraction level and the graphical component assembly approach make the SDN application development easier.

To the best of our knowledge, this paper is the first work for studying SDN controller supporting the component based SDN application development. However, actually, in the research community, the component based approach has been gaining popularity and interest in the form of Mobile Ad-Hoc Networks (MANETs) and sensor networks architecture for developing network control service. Paper [10] presented a component based methodology for modeling mobile ad hoc routing protocols. The component based approach provided two major contributions in protocol design and modeling. First, it allowed the modularity in protocol design. Compared with routing protocols implemented as large monolithic software, it was easy to adapt to varying environmental conditions by adapting component composition. Furthermore, the approach allowed the reuse of existing components across current and future protocols of the same class. Paper [11] proposed the component based approach for developing MAC protocols with the purpose of improving the flexibility of the protocol development and rapid prototyping protocols,

the experiment showed that the component get high reusability across different MAC protocol implementation. In addition to reusability and flexibility, component based framework had much potential in such as reaching intelligence by adding some intelligence plugin into systems. For example, paper [12] proposed the component based protocol stack design, based on which system can automatically suggest a composition protocol stack according to the current environment and user inputs, and also could adapt to environment change by dynamically reconfiguring the components or recomposing the components. All these works demonstrated the benefit, such as high flexibility, high reusability, rapid prototyping, brought by the component based approach to the network control application development, which also provided confidence for us to believe that employing the approach in developing SDN application is feasible and can bring benefit.

3. Framework

We treat the SDN application as a collection of interdependent components, which cooperate with each other to realize the whole function of the SDN application. And different combinations of components can form different SDN applications with various capabilities. Accordingly, the SDN controller must also be a component composition platform to support the creation of SDN applications through composing a suitable set of components deployed in the controller. CFCC consists of a *SDN application development environment* and a *SDN application execution environment*, as shown in Figure 1. We design that just by simply dragging and linking the component icon on the visual interface provided by the *SDN application development environment*, the application developer can fulfill the process of SDN application creating. CFCC employs the openflow protocol as the communication protocol between itself and the switches. The overall functioning workflow is described as following, as shown in Figure 2: the controller need analyze and validate the correctness of the graphical component composition plan from the application developers (implemented in the module *composition verification*), then translate the abstract plan into a formal composition workflow, and then deploy it on the controller (implemented in the module *formal workflow generation*). When SDN applications are requested to run, the formal workflow will be parsed to coordinate the invocation of the related components to implement the whole function of the SDN applications.

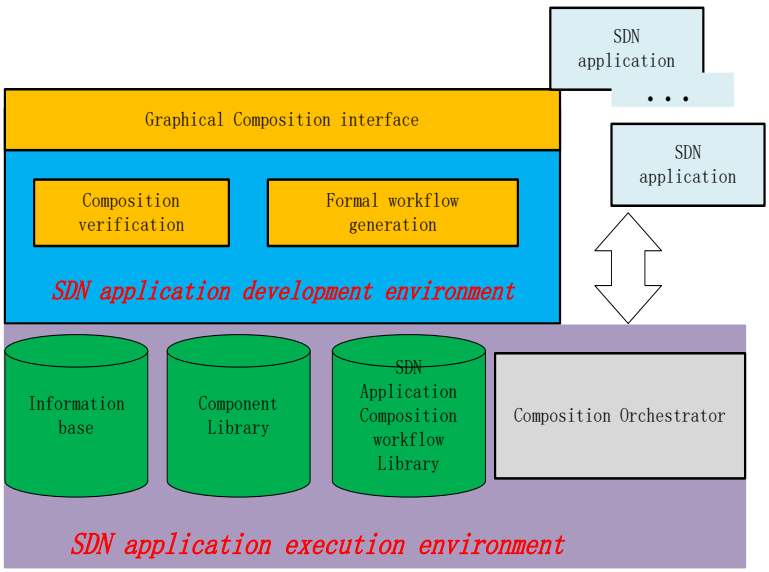


Figure 1. CFCC Controller Framework

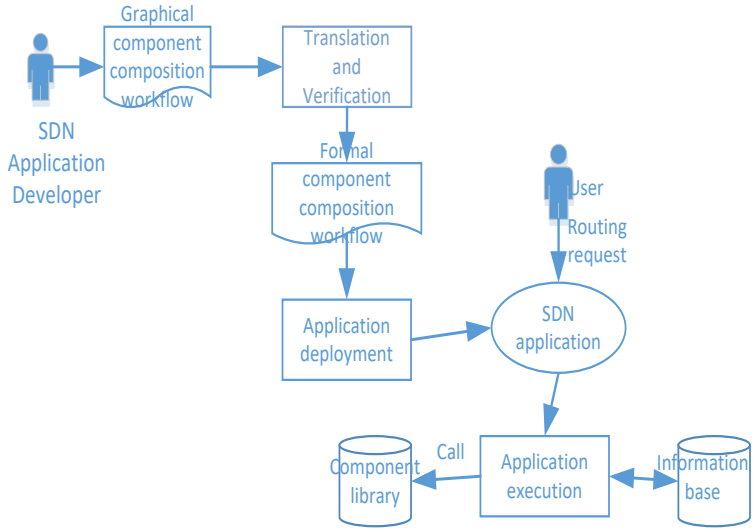


Figure 2. The Workflow of the Controller for SDN Application Creation

3.1. SDN Application Development Environment

The development environment provides application developers with useful information and tools to more easily create new SDN applications. To reach this goal, besides the suitable granularity of functions that the component encapsulates, we design the environment with consideration that the developers do not need to deal with much complex programming details, instead, the development environment automatically translate the abstract composition workflow into the formal workflow which can be executed by the running environment. The development environment consists of three modules: (1) *Graphical composition interface*, (2) *Composition verification*, (3) *Formal workflow generation*.

Graphical composition interface: The aim of the *graphical composition interface* is to enable developers to intuitively design the composition workflow, abstract the SDN application developers away from programming language concepts like variables or data types, just by dragging and dropping, linking the components icons, and configuring the related parameters. The interface will be presented in section 4.

Composition verification: *Composition verification* is to analyze and verify the correctness of the composition workflow from the SDN application developers, before it is put into execution, to avoid the irreversible loss due to the execution of the wrong workflow. As far as we know, formal analysis and verification techniques are widely applied for the component composition verification (especially the web service composition verification), and there are many mature tools for formally modeling component and verification, among which we use CPN TOOLs [13] for verifying the correctness of the composition workflow from the SDN application developers. The CPN TOOLs is a colored petri net based modeling and verification tool, which can verify the reachability, boundedness and liveness of the component composition workflow. In order to realize automated verification in CFCC, the component composition verification need translate the graphical composition workflow, into XML-based petri net compatible with the CPN TOOLs, which then automatically analyze and verify whether the petri net is correct.

Formal workflow generation: We design that CFCC can automatically generate the SDN application composition workflow based on XML by parsing the graphical component composition workflow, then deploy it into the SDN application composition workflow library.

3.2. SDN Application Execution Environment

The *SDN application execution environment*, as its name implies, is the place where the applications are executed. The implementation is based on the J2EE (Java 2 Platform Enterprise Edition) platform and EJB (Enterprise JavaBeans) component model. It consists of a *component library*, an *information base* and a *SDN application coordinator*.

Component library: The *component library* stores and manages all the components and their description files for developing SDN applications, and we design that the component library can be extended for supporting more network control functions. The components are loosely coupled, can fulfill a basic task. Through composing such components, a more value-added service function can be implemented. It's important to note that a very coarse component function encapsulation restricts their reusability due to the increased inner complexity. For example, suppose the multipath routing protocol ECMP (Equal-cost Multi-Path) is implemented by only one component, called ECMP component. This certainly limits the possibility of sharing the ECMP component between different routing applications. On the contrary, very fine grained component function encapsulation leads to complex component interdependencies that result in complex composing process. Therefore, in order to make the component granularity suitable, for each class of SDN application, they should have their own component function encapsulation, rather than share the same one with other class of applications. For that, we first analyze a wide range of SDN application instances belonging to the same class, then identify and encapsulate their shared constituent functionalities and their own distinct functionalities using components, finally deploy the components into the component library for being invoked to implement the SDN application function. When new functionalities are needed for a new routing application, the corresponding new components are created and added into the component library, rather than start from scratch for the new routing applications. Take SDN routing application in SDN network for example, Routing, generally consists in three basic tasks or functions. The first one is to collect the state information (include the network and application flow) and keep it up to date. The second task is to find a (multiple) feasible path (paths) for a new connection based on the collected information. The third one is to configure the routing path into the switches. Accordingly, the generated components are summarized as four major categories, and some typical components are listed in Table 1.

Table 1. Routing Related Component

Component catergary	component	Component function
Network state information monitor	*GetTop*	Get network topology
	MonlinkBW	Monitor bandwidth of network link
	MonlinkDelay	Monitor delay of network link
	...	…
Application state information monitor	*MonflowBW*	Monitor bandwidth of application flow
	MonflowDelay	Monitor delay of application flow
	MonflowJitter	Monitor jitter rate of application flow
	MonQoEVideo	Monitor QoE of video application
	...	…
Calculating path	*CalSinglePath_DCLC*	Calculate a Single path employing the DCLC algorithm
	CalMultiPath_ECMP	Calculate equal cost multiple paths
	CalMultiPath_UCMP	Calculate unequal cost multiple paths
	...	…
Configuring flow table	*ConfFlowPath*	Configure forwarding behavior of switches.
	…	…

Information base: The *information base* is to store and manage information about network, users, applications and other information, serves as the information provider and storage for the running of SDN application.

SDN application composition workflow library: The library stores the workflows generated from the *SDN application development environment* for provisioning to *SDN application coordinator* to execute the SDN applications.

SDN application coordinator: When SDN applications are requested to be run, *SDN application coordinator* parses the corresponding component composition workflows stored in the *SDN application composition workflow library* to control the invocation of the related components. The interaction between the components is centrally controlled by the coordinator, which employs event-driven scheduling mechanism.

We design the interaction mechanism as follows:

1) As showed in Figure 3, there are the data channel, used to transmit the data about input and output, and control channel, used to control the component running by the coordinator, between each component and the coordinator.

2) For each SDN application, the member components pass the running result to the coordinator via the data channel.

3) The coordinator needs to register the trigger event of the member component operation for SDN applications to be run. When the trigger event arrives, or the running condition is met, the coordinator starts the corresponding component to run via the control channel.

For example, the *best effort routing service* can be realized by combining the following three components:

GetTop: the function is to get network topology.

CalSinglePath_SPF: the function is to calculate a routing path between the source node and the destination node employing the Dijkstra algorithm.

ConfFlowPath: the function is to configure the routing path into the corresponding switchs.

The invocation procedure by the coordinator through the control channel is as follows: the component *GetTop, CalSinglePath_SPF, ConfFlowPath* respectively registers the trigger event "*Routing request arrived*", "*Network topology was got*" and "*Routing path was got*" into the coordinator. When the event "*Routing request arrived"* arrives from the data plane, the coordinator invokes the component *GetTop*, which then generates the event "*Network topology was got*" to trigger the invocation of the component *CalSinglePath_SPF*. Finally, the event "*Routing path was got*" is generated by the component *CalSinglePath_SPF*, which triggers the invocation of the component *ConfFlowPath* to configure the routing path into switches. In the above procedure, the component *GetTop* generates and passes the network topology to the coordinator through the data channel, which in turn passes the network topology to the component *CalSinglePath_SPF*. In the same way, the component *CalSinglePath_SPF* generates and passes the routing path to the component *ConfFlowPath* indirectly through the coordinator.

4. Case Study

As we know, the current internet has carried a diversified of applications with different QoS requirements, which, in conjunction with the customized requirement from end users and network owners, forces the arising of various routing control functions. Take for example the videoconferencing application, videoconferencing is an interactive multimedia application which requires a strict end-to-end delay and packets loss ratio [14],

or if the video is SVC encoded, it demands that the video in the base layer should be streamed without any packet loss or minimized delay variation, regarded as level-1 QoS flows, and the video in the enhancement layer can be regarded as either level-2 QoS flows (if capacity is available) or best-effort flows [15]. Therefore, single-path QoS routing is required for the first case, multi-path QoS routing for the second case is considered, which calculates QoS guaranteed path for transferring the base layer flow of the video, best-effort path for the enhance layer flow of the video, thereby guaranteeing a high level of video quality and reasonable cost at the same time.

There are more different routing functionalities, to solve the specific problems occurring in different situations. We have implemented a prototype of CFCC and deployed several components related to routing control function on it, and chose two routing control applications that are suitable for the above mentioned application scenario, named RCA1 and RCA2, as the case study to demonstrate the feasibility of CFCC.

In the following sections, we demonstrate the feasibility of CFCC by implementing and evaluating RCA1 and RCA2 based on CFCC.

First, we have implemented RCA1 and RCA2 based on CFCC exploiting graphical component composition interface. Second, we put them into running over an openflow network which is emulated using mininet [17], in order to validate the correctness of CFCC based RCA1 and RCA2, at the same time, to evaluate their performance by comparing with that of the Non-CFCC based RCA1 and RCA2.

4.1. The Creation of Two Kinds of SDN Routing Control Application

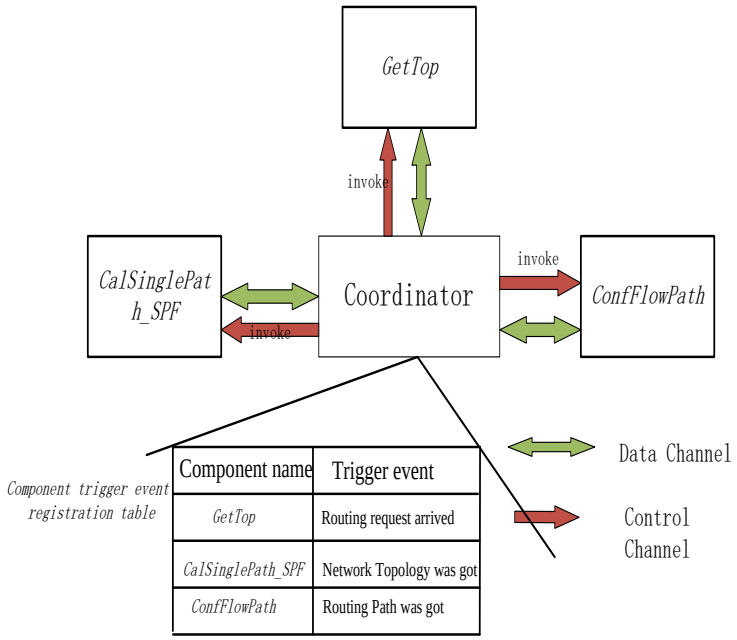
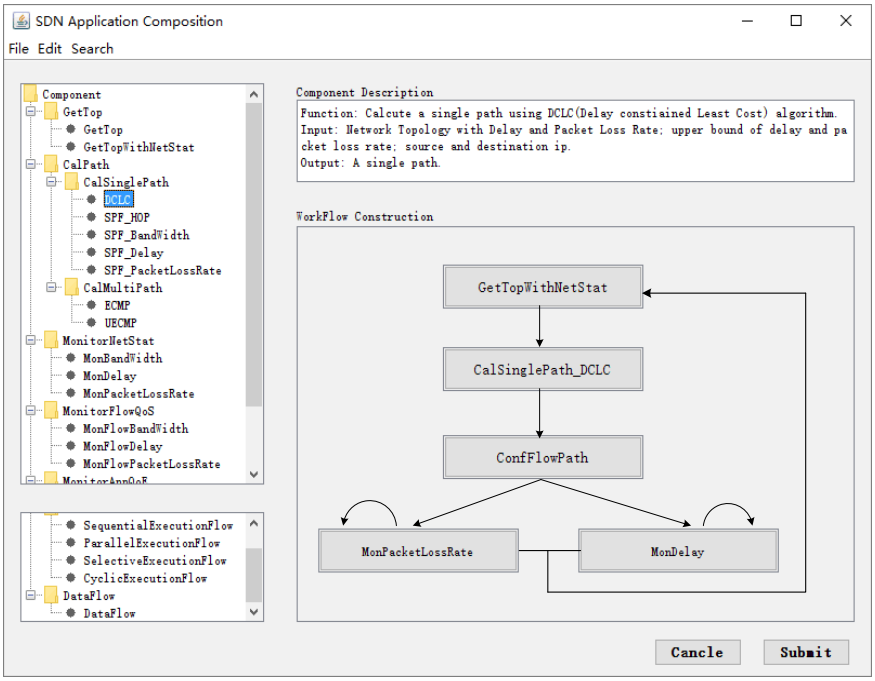
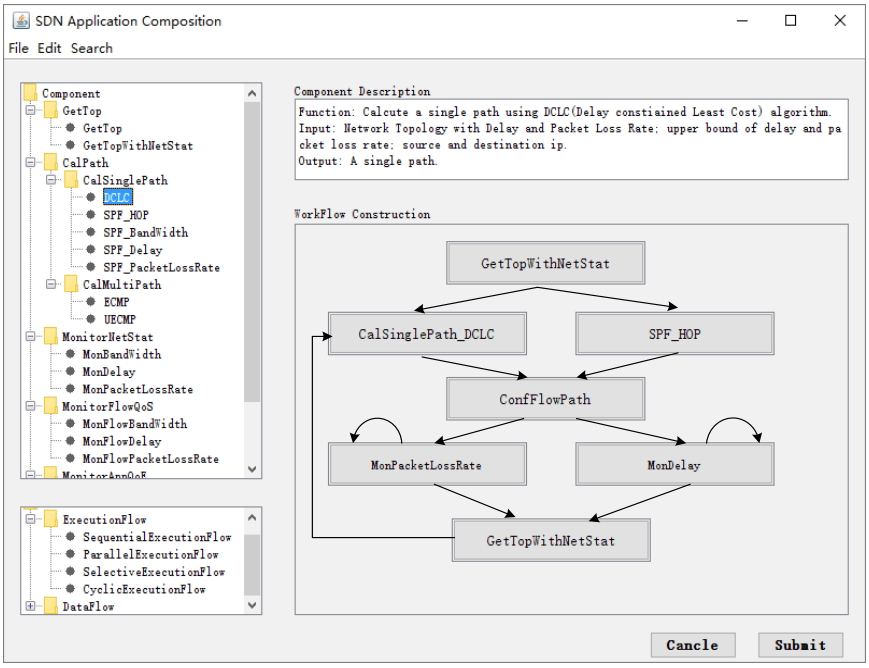


Figure 3. Interaction between Components

As showed in Figure 3, the graphical component composition workflow of RCA1 and RCA2 are constructed on the graphical interface of CFCC by dragging and dropping, linking the components icons, and configuring the related parameters. Obviously, our approach can largely reduce the burden for programmers than the traditional monolithic implementation approach. According to the graphical component composition workflow showed in Figure 4(a), the corresponding formal workflow of RCA1 are generated and showed in Figure 5.



(a) RCA1 Workflow Construction



(B) RCA2 Workflow Construction

Figure 4. Graphical SDN Applications Workflow Construction

```
<sequence seqM>
<!-- Async invoke of the GetTop component and
wait for the callback-->

<invoke partnerComponent="GetTop"
portType="rt:GetTopPT"
operation="GetTop"/>
<receive partnerComponent="GetTop"
portType="rt:GetTopCallbackPT"
operation="TopCallback"
variable="Top" />
<!-- Async invoke of the CalSinglePath_DCLC component and
wait for the callback-->
<invoke partnerLink="CalSinglePath_DCLC"
portType="rt:CalSinglePath_DCLCPT"
operation="CalSinglePath_DCLC"
inputvariable top="Top"
inputvariable s="SourceNode"
inputvariable d="DesNode"
/>
<receive partnerLink="CalSinglePath_DCLC"
portType="rt:CalSinglePath_DCLCCallbackPT"
operation="SinglePathCallback"
outputvariable="SinglePath" />
<invoke partnerLink="ConFlowPath"
portType="rt:CalSinglePaConFlowPathPT"
operation="ConFlowPath"
inputvariable="SinglePath"/>

<!--loop invocation of the MonPacketLossRate and
MonDelay component-->

<while>
<sequence seqCP>
   <invoke partnerLink="MonPacketLossRate"
    portType="rt:MonPacketLossRatePT"
    operation="MonPacketLossRate"
    inputVariable="SinglePath"
    inputVariable="FlowID" />
   <receive partnerLink="MonPacketLossRate"
    portType="rt:PacketLossRateCallbackPT"
    operation="MonPacketLossRate"
    variable PLR="PacketLossRate" />
<sequence seqCM>
   <invoke partnerLink="MonDelay"
    portType="rt:MonDelayPT"
    operation="MonDelay"
    inputVariable="SinglePath"
    inputVariable="FlowID" />
   <receive partnerLink="MonDelay"
    portType="rt:DelayCallbackPT"
    operation="MonDelay"
    variable de="Delay" />
<if condition="plr &gt; 0.04 &amp; de &gt; 180"
<!--break of of the monitoring loop, recalculate
routing path-->

   <GoTo gotoNode="seqM">
   </GoTo>
</if>
</while>
/sequence>
```

Figure 5. Formal Composition Workflow of RCA1

RCA1: it is to choose a single path with guaranteed delay and packet loss rate for videoconferencing application, which can be implemented by assembling the following component:

GetTopWithNetStat gets the network topology with information about the bandwidth and packet loss rate.

CalSinglePath_DCLC uses the algorithm Delay-Constrained Least-Cost (DCLC) to select a path with the least cost and the delay less than a specified value.

ConfFlowPath configures the forwarding behavior of the switches.

MonPacketLossRate monitors the packet loss rate of the flow.

MonDelay monitors the delay of the flow.

The implementation process of the CFCC based RCA1 is: when receiving the event "*Routing request arrived*"from the data plane, the *Coordinator* executes the component *GetTopWithNetStat* to get the network topology with delay and packet loss rate information. Then the component *CalSinglePath_DCLC* is called to calculate a suitable path, according to which, the component *ConfFlowPath* will configure forwarding behavior into the corresponding switches. The component *MonPacketLossRate* and *MonDelay* are then executed constantly to obtain the delay and packet loss rate value of the flow, when the value of the metrics do not meet the requirement of users, the *Coordinator* re-start the above invocation process to rerouting the flow to a suitable path.

RCA2: It can calculate multi-paths for transferring the SVC encoded video flow. Compared with that of RCA1, the component composition workflow of RCA2 is relatively complex. The main difference lies in the task of routing path calculation, which need respectively calculate paths with guaranteed delay and packet loss rate for the base layer flow, and best-effort path for the enhancement layer flow.

4.2. Feasiblity Test

We take 2 groups of experiments over the same simulated openflow network, respectively for validating the behavior of CFCC based RCA1 and RCA2, at the same time, comparing the performance result of the CFCC based RCA1 and RCA2 and Non-CFCC[2] based RCA1 and RCA2.

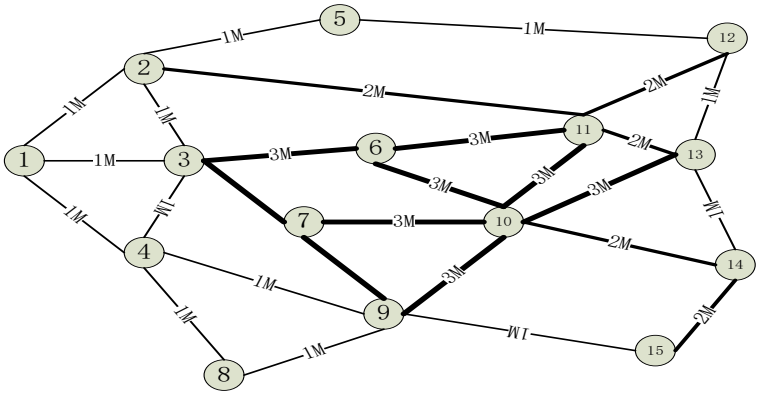


Figure 6. MIRA Topology

The experiment environment: In simulations we used MIRA topology, adopted from the literature dealing with the correlated routing problem [16], as showed in Figure 6, we set the simulation time 330s.

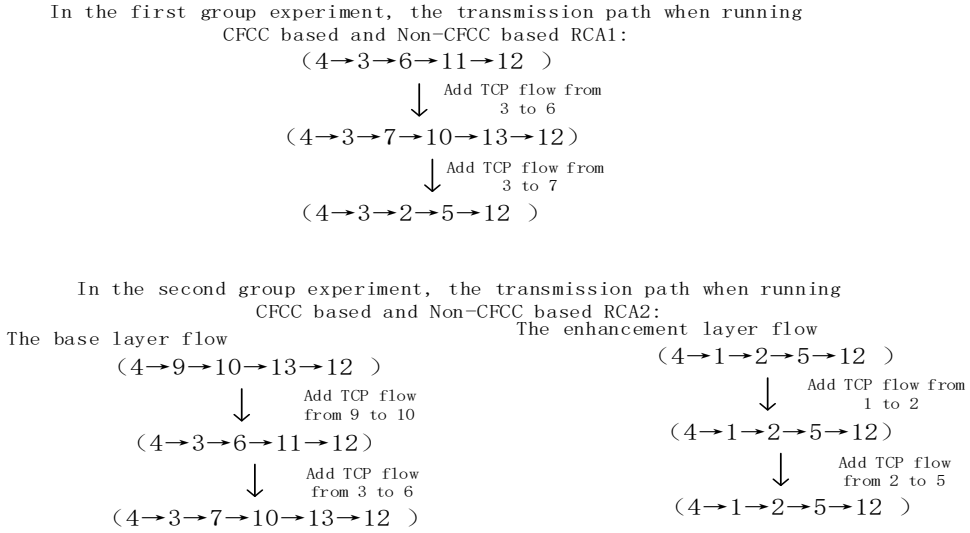


Figure 7. The Transmission Path of Videoconference Flow in the Two Group Experiment

- The first group of experiment running the CFCC and Non-CFCC based RCA1

In the first group of experiment, we respectively run the CFCC and Non-CFCC based RCA1 according to the following experiment procedure:

1) Node 4 and 12 generated the udp flow to represent the videoconference flow. At the beginning of the experiment, there is no background flow taking up the bandwidth resource on the link that the videoconferencing flow goes through.

[2] Non-CFCC means the application are implemented in traditional approach.

2) While after 100s, we manipulated that the tcp flow was generated to make congestion for the videoconference flow.

3) Again, at the time 200s, we manipulated that the tcp flow was generated to make congestion for the videoconference flow.

We have observed the same transmission path of the videoconference flow, when applying CFCC and Non-CFCC based RCA1, and summarized that in Figure 7. First, before the TCP flow occurs between the node 3 and 6, the videoconference flow passes through the path (4->3->6->11->12), while, when the TCP flow is generated to make the link (3->6) congested, the path of the videoconference flow is changed to the unobstructed path(4->3->7->10->13-12), again, in the same way, the path the videoconference flow is turned into the path (4->3->2->5->12) when the TCP flow is added from the node 3 to 7.

- The second group of experiment running the CFCC and Non-CFCC based RCA2

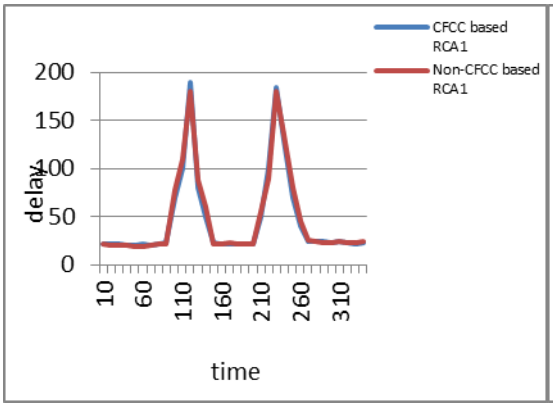
Similarly, we respectively run CFCC and Non-CFCC based RCA2 following the below experiment procedure for the second group experiment:

1) We generate two kinds of udp flow between Node 4 and 12, respectively representing the base layer flow and enhancement layer flow. At the beginning of the experiment, there is no background flow taking up the bandwidth resource on the link that the videoconferencing flow goes through.

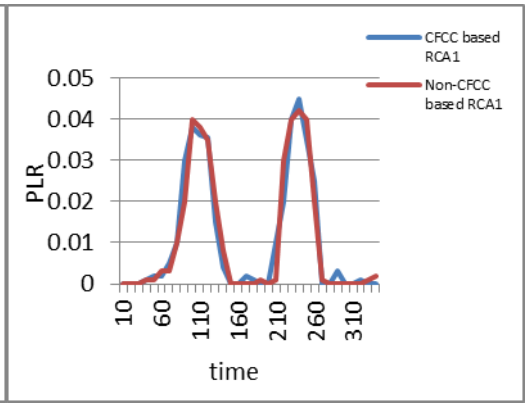
2) While after 100s, we respectively manipulated that the tcp flows were generated to make congestion for the videoconference base and enhance layer flow.

3) Again, at the time 200s, we respectively manipulated that the tcp flows were generated to make congestion for the videoconference base and enhancement layer flow.

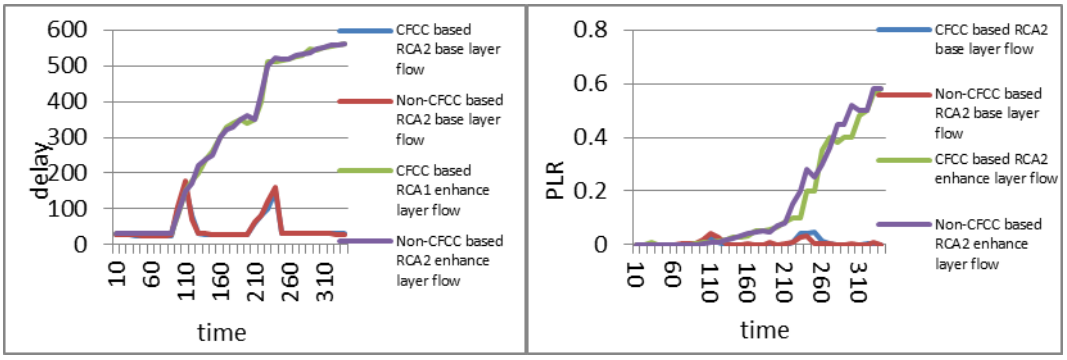
Also, from Figure 7, we can see the same transmission path of the videoconference flow when applying CFCC and Non-CFCC based RCA2. The base layer flow first passes through the path (4->9->10->13->12) , then turns its path into the path (4->3->6->11->12) when the background flow is generated to interferer its QoS, and in the same way, the path (4->3->6->11->12) is changed into (4->3->7->10->13->12). On the contrary, the enhancement layer flow is always unchanged regardless of whether the background flow is generated to interferer it.



(A) The Dealy Result of CFCC Based and Non-CFCC Based RCA1



(B)The Packet Loss Rate Result Of CFCC Based and Non-CFCC Based RCA1



(c)The Delay Result of CFCC based and Non-CFCC based RCA2

(d)The Packet Loss Rate Result of CFCC based and Non-CFCC based RCA2

Figure 8. The Performance Comparison Result of CFCC based and Non-CFCC based Routing Control Applications

We have also observed that the performance characteristics of CFCC based routing application closely resemble to their Non-CFCC counterparts in terms of the delay and packet loss rate of the videoconference flow in all the cases. Figure 8 shows the delay and packet loss rate comparison for CFCC based routing application and their Non-CFCC counterparts at different network traffic. From Figure 8(a), we can obtain that the delay of the videoconference flow increases sharply at the time 110s and 210s, and then fall back at the time 140s and 240s, it may be caused by the transformation of the flow path at these times in the case of RCA1 is applied, which also incurs the increase and decrease of the packet loss ratio, indicated in Figure 8(b). For RCA2, Figure 8(d) and (c) indicate that the delay and packet loss rate of the base layer flow increase and decrease in the same way as that of the videoconference flow in RCA1(see Figure 8(a) and (b)), while the delay and pack loss rate of the enhancement layer flow always increase from the time 110s, the reason for that is it applies the best-effort routing which does not adopt any remedial methods when the QoS performance decrease.

In all, the experiments demonstrate that same behavior and performance of the CFCC and Non-CFCC based routing applications, which indicates the correctness of CFCC.

4.3. Evaluating Overhead

We compare the flow setup time when using the CFCC based routing application and their Non-CFCC counterparts. To measure this, we capture packets between the CFCC controller and the OpenFlow switches, and measure the round trip required to submit routing request of the flow and receive a corresponding flow routing configuration. We observe that the component based routing applications require additional setup time in the range of 0.4 milliseconds to 9.8 milliseconds, which is negligible compared to the benefit generated by the component based routing application development.

5. Conclusion

Facing with the challenge involved in the process of SDN applications development based on the low-level programmable interfaces provided by the current controllers, the paper proposed a controller framework CFCC to support the component based SDN application development with the features that the components provide more suitable functions abstraction level than the programmable interfaces of the current controllers, meanwhile, have high reusability across different SDN applications belonging to the same class, thus can support efficient SDN application development. We have verified the

ISSN: 2233-7857 IJFGCN

validity of CFCC by creating the routing control applications in the section 4, and our evaluation demonstrate that CFCC introduces negligible overhead on the flow setup time and that it enables rapid creation of the popular routing control applications. As the future work, we will add more components into the component library to create more other SDN applications, to further evaluate and improve the performance of CFCC.

Acknowledgments

This research was supported by Future Network Foresight Research Program of Jiangsu Province under Grant No. BY2013095-1-08; the College Graduate Research and Innovation Projects of Jiangsu Province under Grant No. CXLX12_0487; the Project of BUPT National Key Laboratory (SKLNST-2008-1-13).

References

[1] B. Astuto, A. Nunes, M. Mendonca, X.N. Nguyen, K. Obraczka and T. urletti, "A Survey of Software-Defined Networking: Past,Present, and Future of Programmable Networks", IEEE Communications Surveys & Tutorials, vol. 16, **(2014)**, pp. 1617-1634.
[2] N. McKeown and T. Anderson, "OpenFlow: enabling innovation in campus networks", ACM SIGCOMM, USA, vol. 38, **(2008)**, pp. 69–74.
[3] F.A. Lopes, M. Santos, R. Fidalgo, S. Fernandes and S. Member, "A Software Engineering Perspective on SDN Programmability", IEEE Communications Surveys & Tutorials, vol. 18, **(2015)**, pp. 1255-1272.
[4] S. Mahmood, R. Lai and Y.S. Kim. "Survey of component-based software development", IET Software, vol. 1, **(2007)**, pp. 57–66.
[5] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story and D. Walker, "Frenetic: a network programming language", SIGPLAN Not, USA, vol. 46, **(2011)**, pp. 279-291.
[6] A. Voellmy and P. Hudak, "Nettle: taking the sting out of programming network routers", in Proceedings of the 13th international conference on Practical aspects of declarative languages, USA, **(2011)**, pp. 235–249.
[7] C. Monsanto, N. Foster, R. Harrison and D. Walker, "A compiler and run-time system for network programming languages", SIGPLAN Not., vol. 47, **(2012)**, pp. 217–230.
[8] A. Voellmy, H. Kim and N. Feamster, "Procera: a language for highlevel reactive network control", in Proceedings of the first workshop on Hot topics in software defined networks, ser. HotSDN. USA, **(2012)**, pp. 43–48.
[9] J. Reich, C. Monsanto, N. Foster, J. Rexford and D. Walker, "Modular SDN Programming with Pyretic", USENIX ;login, vol. 38, **(2013)**, pp. 40-47.
[10] E. Paraskevas and J. S. Baras, "Component Based Modeling of Routing Protocols for Mobile Ad Hoc Networks", CISS, MD, **(2015)**, pp. 1-6.
[11] J. Ansari, X. Zhang, O. Salikeen and P. M. Onen, "Enabling Flexible Medium Access Design for Wireless Sensor Networks", 2011 Eighth International Conference on Wireless On-Demand Network Systems and Services. Bardonecchia, **(2011)**, pp. 158-163.
[12] J. Ansari , E. Meshkova, W. Masood, A. Muslim , J. Riihij?rvi and Petri M?h?nen, "CONFab: Ontology and component based optimization of WSN protocol stacks with deployment feedback", Computer Networks, vol. 74, **(2014)**, pp. 89–108.
[13] K. Jensen, "Coloured Petri Nets: basic concepts analysis methods and practical use. Springer", **(1997)**.
[14] D.E. henni, A. Ghomari and Y. Hadjadj-Aoul, "Videoconferencing over Openflow Networks: an Optimization Framework for Qos Routing", 2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing, **(2015)**.
[15] Z. Bai, S. Li, Y. Wu, W. Zhou and Z. Zhu, 'Experimental Demonstration of SVC Video Streaming using QoS-Aware Multi-Path Routing over Integrated Services Routers", IEEE ICC 2013 - Next-Generation Networking Symposium, **(2013)**.
[16] S. Tomovic, I. Radusinovic and N. Prasad, Performance comparison of QoS routing algorithms applicable to large-scale SDN networks.
[17] http://mininet.org/
[18] D. Berardi, G. D. Giacomo and M. Mecella, "Basis for automatic service composition, in: Tutorial at the 14th International World Wide Web Conference (WWW'05)", **(2015)**.

