International Journal of Future Generation Communication and Networking
Vol. 9, No. 3 (2016), pp. 57-68
http://dx.doi.org/10.14257/ijfgcn.2016.9.3.06

A Composition Method of Logical Function Block Chain[1]

JIN Rong[1, 2], HE Xiong-xiong[1] and Zhu Guang-xin[3]

1. College of Information, Zhejiang University of Technology, Hangzhou 310014, China
2. College of Information & Electronic Engineering, Zhejiang Gongshang University, Hangzhou 310018, China
3. OB Telecom Electronics Co., Ltd., Hangzhou 310012, China
E-mail: jinrong@mail.zjgsu.edu.cn; hxx@zjut.edu.cn; hans@obtelecom.com

Abstract

ForCES is a new open programmable architecture, it separates control and forwarding, it also abstracts resources of Forwarding Elements (FE) into some Logical Function Blocks (LFB), Control Element (CE) can reconfigure the forwarding function of FEs by recomposing their LFB chains. Firstly, A SDN architecture based on ForCES is proposed. Then, based on the formalization of the concept of LFB and the traditional I/O matching algorithm, an LFB chain composition method is proposed. This method can combine a series of LFBs to an LFB chain according to special application request. Based on the LFB chain composition algorithm, an improved algorithm is proposed, which can improve the composition efficiency. At last, an example is provided to illustrate how this method works, and a simulation is given to compare the efficiency of the base algorithm and the improved algorithm.

Keywords*: SDN, ForCES, Logical Function Block, Composition*

1. Introduction

With the emergency of new network applications, the new requirement is no longer a high performance of data forwarding, but an open flexible network to adapt to various business requirements. Under this background, many future network architectures are proposed [1-4], including Software Defined Network (SDN) [5]. The main idea of SDN is to accomplish a kind of dynamic management of network resources by a way like software definition. In SDN, user can construct different data networks by dynamically programming, and then SDN can meet all kinds of application requirements. Once SDN was proposed, it received wide attention, and it is regarded as the developing direction of future networks. But how to implement the architecture of SDN is still a controversial topic at present. Many technologies, such as OpenFlow，have been trying to implement SDN [6]. Forwarding and Control Element Separation (ForCES) technology also began to be used in the study of the implementation of SDN because of its separation characteristics and its control mechanism based on the model [7-9].

[1] This work was supported in part by a grant from the National Basic Research Program of China (973 Program) (No. 2012CB315902), the National Natural Science Foundation of China (No.61379120), Zhejiang Leading Team of Science and Technology Innovation (No.2011R50010-11, No.2011R50010-15). Zhejiang Provincial Key Laboratory of New Network Standards and Technologies (NNST)(No.2013E10012).Youth Foundation of Zhejiang Gongshang University(No.QZ13-8)

ISSN: 2233-7857 IJFGCN

ForCES technology is a kind of network element construction mode with the separation of Forwarding Element (FE) and Control Element (CE). As shown in Figure 1, a network element satisfying the ForCES protocol is known as a ForCES element. The number of CEs is usually only several (at least one), and the number of FEs is usually several hundreds. The communication protocol between CEs and FEs is ForCES protocol [2].

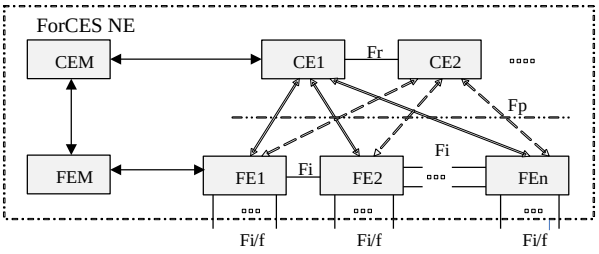


Figure 1. Forces Architecture

ForCES is a new open programmable architecture. IETF ForCES working group has developed a series of protocols after 10 years of research. Now ForCES is being studied for SDN implementation. ForCES implements the separation of CE and FE, and also implements the centralized control from CE to FE. This idea of separation and centralized control also happens to be promoted by SDN architecture. In addition, ForCES has a very good abstraction and definition of open resources, the resources of FE are abstracted into LFBs. CE can reconfigure the forwarding function of FE by recombining the LFB topology of FE. This kind of abstraction of resources provides good support for SDN to realize resource management. So we can extend ForCES architecture from Network Element to SDN. As shown in figure 2, a SDN architecture based on ForCES is proposed. SDN architecture includes three layers, and they are application layer, control layer, and infrastructure layer. The communication between control layer and infrastructure layer is ForCES protocol. The routing and forwarding infrastructure is described as Forwarding Element. The user of application layer can use API to control the reconfiguration of infrastructure layer.

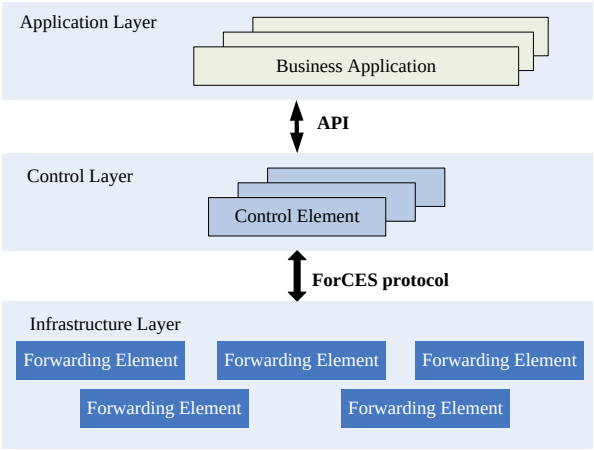


Figure 2. SDN Based on Forces

The internal resources of Forwarding Elements shown in Figure 2 are fine-grained abstracted to some Logical Function Blocks (LFB), as shown in Figure 3.The LFBs can be dynamically configured to form different LFB chains. The different LFB chains can constitute different functions of the FE. LFBs are the

abstraction of the resources of FE model in ForCES framework. LFBs are well defined and logically separated function blocks, they exist in FE and are controlled by CE through ForCES protocol. IETF ForCES working group has formed FE model [10] and LFB library [11]. The FE model defines data model, and the LFB library defines some common LFB classes. LFB classes provide typical routing and switching functions, these functions are classified into several LFB classes, and then these classes constitute a typical and flexible LFB library. The LFB library can adapt to various IP forwarding requirements. There is Ethernet packet processing LFB, IP packet authentication processing LFB, IP forwarding processing LFB, and so on. The LFB library is divided into core LFB and general LFB. The core LFB is currently mainly refers to the FE protocol LFB and the FE object LFB, the other LFBs are all belong to the general LFB. As shown in Figure 3, these LFBs can be flexibly combined to a LFB topology to achieve a typical routing and forwarding function.

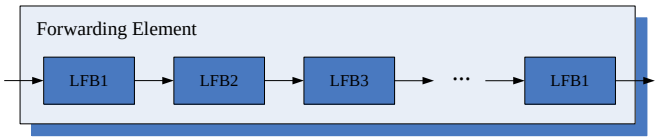


Figure 3. LFB Topology in FE

In SDN based on ForCES, SDN control layer should have the ability to dynamically combine LFBs into LFB chain according to the requirement of application layer. This problem must be solved in SDN based on ForCES. Therefore, this paper focuses on the study of the method of LFB chain composition. Based on the formalization of the concept of LFB and traditional I/O matching algorithm, an LFB chain composition method is proposed. This method can combine a series of LFBs to an LFB chain according to special application request. Based on the LFB chain composition method, an improved algorithm is proposed, which can improve composition speed. At last, an example is provided to illustrate how this method works, and a simulation is given to compare the efficiency of the base method and the improved method.

2. Conceptions and Definitions

Traditional I/O matching algorithm has been widely used in many areas, such as scheduling, web service etc [12-14]. Its principle is matching the input and output parameters of functions. This paper introduces the traditional I/O matching algorithm into the LFB chain composition area in SDN based on ForCES. But the input and output of LFBs are more complex than that of usual functions. So the traditional I/O matching algorithm must be improved, and the definition of matching relation between input and output must be redefined.

So firstly our LFB should be defined according to the LFB model. And some other conceptions should also be defined, such as connecting relation between I/O port, success relation between two LFBs, LFB chain requirement, and sequential composition of LFB and so on.

2.1. Def. 1 LFB Definition

LFB model defines LFB as an eight-tuple. It can be described as follows.

$$S = (n, t, v, I, O, A, E, C)$$
$$= (name, tagID, version, InputPorts, OutputPorts, Attributes, Events, Capabilities)$$

Where '*n*' is short for '*name*' and denotes LFB's name; '*t*' is short for '*tagID*' and denotes LFB's tag identification; '*v*' is short for '*version*' and denotes LFB's version number; '*I*' is short for '*InputPorts*' and denotes the input port set of LFB; '*O*' is short for '*OutputPorts*' and denotes the output port set of LFB; '*A*' is short for '*Attributes*' and denotes operable attribute set; '*E*' is short for '*Events*' and denotes event set; '*C*' is short for '*Capabilities*' and denotes capability set.

LFB defined as above eight-tuple can be illustrated as Figure 4. The input port set can contain several inputs ($I_1, I_2, ...$), and the output port set can also contain several outputs ($O_1, O_2, ...$). $I_i(n, p, M)$ denotes an input port, where '*n*' denotes port name, '*p*' denotes packet type and '*M*' denotes the metadata being waited to input this port; $O_i(n, p, M)$ denotes an output port, where '*n*' denotes port name, '*p*' denotes packet type and '*M*' denotes the metadata outputted by this port.

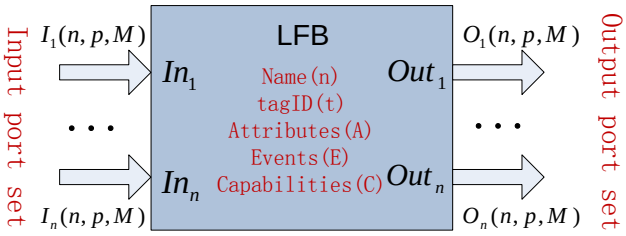


Figure 4. LFB Definition

2.2. Def. 2 Formal Description of LFB

In order to study the mechanism of LFB composition, LFB is formally described as follows.

$$LFB = (Inputport, Outputport)$$
$$Inputport = (Inputport_1, Inputport_2, Inputport_3, ...)$$
$$Inputport_i = (Packet, Metadata)$$
$$Outputport = (Outputport_1, Outputport_2, Outputport_3, ...)$$
$$Outputport_i = (Packet, Metadata)$$
$$Packet = (Packet_1, Packet_2, Packet_3, ...)$$
$$Metadata = (Metadata_1, Metadata_2, Metadata_3, ...)$$

An LFB contains several inputs and outputs, and each input or output consists of one or more '*Packet*' and '*Metadata*'.

2.3. Def. 3 Containing and Equivalence Relation between Two Concepts

Take '*packet*' for an example to explain the containing and equivalence relation between two concepts. (The '*metadata*' has no containing relation at present, and it only has equivalence relation.)

1) If $packet1$ and $packet2$ are same, then it is expressed as $packet1 = packet2$.

E.g. If $packet1$ is IPv4Unicast, and $packet2$ is also IPv4Unicast, then $packet1 = packet2 = IPv4Unicast$.

2) If $packet2$ contains $packet1$, then it is expressed as $packet1 \subset packet2$.

E.g. If $packet1 = IPv4Unicast$, $packet2 = IPv4$, then $packet1 \subset packet2$.

2.4. Def. 4 Containing and Equivalence Relation between two Concept Clusters

Take '*packet*' for an example to explain the Containing and Equivalence relation between two concept clusters. '*metadata*' is similar.

$$Packet1 = (packet1_1, packet1_2, packet1_3,...)$$
$$Packet2 = (packet2_1, packet2_2, packet2_3,...)$$

If $\forall\ packet1_i \in Packet1$, $\exists\ packet2_j \in Packet2$ and $packet1_i \subseteq packet2_j$, then it means $Packet2$ contains $Packet1$, which is expressed as $Packet1 \subseteq Packet2$.

2.5. Def. 5 Connectable Relation of Ports

According to LFB model, if the packet type of an input port of a subsequence LFB contains or equals the packet type of an output port of a precursor LFB, and the metadata type of the output port of the precursor LFB contains or equals the metadata type of the input port of the subsequence LFB, then this precursor output port can connect to this subsequent input port. The connectable relation of ports can be formalized as follows.

The precursor output port is expressed as $Outputport1 = (Packet1, Metadata1)$, and subsequent input port is expressed as $Inputport2 = (Packet2, Metadata2)$. If the following two conditions are satisfied, then it suggests that the precursor output port can connect to the subsequent input port.

1) $Packet1 \subseteq Packet2$
2) $Metadata1 \supseteq Metadata2$

The connectable relation of the two ports can be expressed as $Outputport1 \Xi Inputport2$.

2.6. Def. 6 Succession Relation of LFBs

According to LFB model, if every output port of precursor LFB can establish connection relation with one input port of subsequent LFB, the succession relation between the two LFBs is established. It is formalized as follows.

Precursor LFB is expressed as $LFB_1 = (Inputport1, Outputport1)$, and subsequent LFB is expressed as $LFB_2 = (Inputport2, Outputport2)$. If $\forall\ Inputport2_i \in Inputport2$, $\exists\ Outputport1_j \in Outputport1$ and $Outputport1_j \Xi Inputport2_i$, then a succession relation can be established between this precursor LFB and this subsequent LFB, which is marked as $LFB_1 \Theta LFB_2$.

2.7. Def. 7 LFB Chain Requirement

When user's application requirement arrives, the control layer of SDN based on ForCES will map the user's application requirement into a LFB chain requirement with special chain input ports and chain output ports. According the LFB chain requirement, control layer will dynamically compose some special LFBs with special functions into a special chain. And then control layer will tell infrastructure layer to reconfigure network nodes. The LFB chain requirement is denoted as *LFBR*, which is formalized as follows.

$$LFBR = (Inputport, Outputport)$$

2.8. Def. 8 Sequential LFB composition

We define that only the LFBs having succession relations can be composed into a sequential LFB chain. And this kind composition of LFBs is called sequential LFB composition.

Given a LFB set and a LFBR, sequential LFB composition can be deemed to discover a sequential LFB chain $LFB_1, LFB_2, ..., LFB_i, \ldots, LFB_n$ to satisfy the LFBR. This sequential LFB chain only has succession relation between any precursor LFB and its subsequence LFB, that is $LFB_i \Theta LFB_{i+1}$. And this sequential LFB chain must satisfy $LFBR.Inputport = LFB_1.Inputport$ and $LFB_n.Outputport = LFBR.Outputport$.

3. Implementation of Composition Method

A sequential composition method of LFB chain is proposed based on I/O matching algorithm. This method can build LFB chains according requirements automatically. Based on the definitions of section 2, in section 3, an implementation framework is proposed firstly, and then composition steps are introduced, and at last detail algorithm is given.

3.1. Implementation Framework

The LFB composition implementation framework is shown as Figure 5. The matching agent locates in the control layer of SDN based on ForCES. It consists of three parts, and they are mapper, combiner and selector. According to arriving *LFBR*, the mapper searches LFB library for a LFB set which could satisfy application requirement. By this time these LFBs are discrete and standalone. The combiner is responsible for combining these LFBs into one or more LFB chains according to special *LFBR* and then putting these chains into the LFB Chain Set. At last the selector will pick out the best chain as the final output.

This paper focuses on the second step, and proposes an atomic LFB chain composition method based on ForCES LFB model and traditional I/O matching algorithm. LFB chain composition can be divided into three types, sequential composition, branch composition, and hybrid composition, this paper studies on the sequential composition problems.

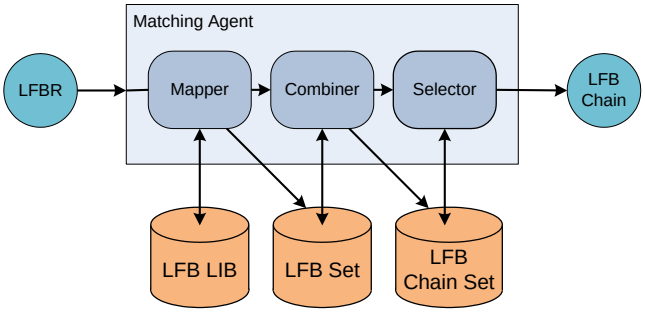


Figure 5. Implementation Framework of LFB Composition

3.2. Composition Steps

The proposed sequential composition method of LFB chain is implemented in the combiner. The detail steps of composition algorithm are given as follows.

LFBR denotes LFB chain requirement; *LFBSet* denotes the LFB set where LFBs are mapped out and have no connection relation with each other; *LFBChain* denotes an

LFB chain; *ChianLen* is an input parameter denoting length of a *LFBChain*; *LFBChainSet* denotes the set of *LFBChain*s, and it is the output of the algorithm.

Step1. Traverse *LFBSet* to find out every LFB whose inputport matches with *LFBR.Inputport*, and put them into *LFBChainSet* as the first LFB of respective *LFBChain*.

Step2. For each *LFBChain* of *LFBChainSet*, traverse *LFBSet* to search for a successor LFB matched with the last LFB of the current *LFBChain*, and then connect this LFB to the *LFBChain*. If there are more matched successor LFBs, then copy current *LFBChain* to form more *LFBChain*s and connect the different successor LFBs to them respectively. If there is no successor LFB we can find, then delete the current *LFBChain*.

Step3. Repeat step2 until the length of *LFBChain* is equal to *ChainLen*.

Step4. Check whether the output of each *LFBChain* matches *LFBR.Outputport*. If not, then delete the *LFBChain*. Then all the rest *LFBChain*s can meet the *LFBR*. If lastly the LFBChainSet is empty, it means there are no chains can be composed to meet the *LFBR*.

3.3 Composition Algorithm

The detail sequential composition algorithm of LFB chain is given as follows.

```
//initialization.
LFBChainSet←NULL
//find the first LFB.
for all LFB in LFBSet do
        if (LFB.Inputport == LFBR.Inputport)
        then { TempChain.add(LFB)
                LFBChainSet.add(TempChain)
                Templen←1
                }
//find successor LFB for each LFBChain.
while(TempLen<ChainLen)
        {for all LFBChain in LFBChainSet
                {FoundFlag=false
                TempChain=LFBChain
                for all LFB in LFBSet do
                        {if ((LFBChain.lastLFB Θ LFB) and
(LFBChain.lastLFB!=LFB) and FoundFlag=false)
                        then {LFBChain.add(LFB)
                                FoundFlag=true
                                }
                        if ((LFBChain.lastLFB Θ LFB) and
(LFBChain.lastLFB!=LFB) and FoundFlag=true)
                        then {TempChain.add(LFB)
                                LFBChainSet.add(TempChain)
                                FoundFlag=false
                                }
                        }
                if(FoundFlag=false)
                then LFBChainSet.delete(LFBChain)
                }
         TempLen++;
        }
//output check.
```

for all LFBChain in LFBChainSet
{if (LFBChain.lastLFB.Outputport!=LFBR.Outputport)
then LFBChainSet.delete(LFBChain)
}

3.4 An Improved Algorithm

According to the composition algorithm described above, when *LFBSet* becomes numerous and *LFBChain* becomes long, the efficiency of the composition process will become low, because the composition method needs to search successor LFB one by one. We proposed an improved method as following to improve the efficiency.

The idea of the improved algorithm is to compose an LFB chain from both ends to middle. Firstly, it finds out the LFB which match with the input port of *LFBR* as the first LFB of the *LFBChain*, and finds out the LFB which match with the output port of *LFBR* as the end LFB of the *LFBChain*. Then it scans the whole *LFBSet* to find a successor LFB matched with the first LFB, and then connects this LFB to the *LFBChain*. At the same time, it scans the whole *LFBSet* to find a precursor LFB matched with the end LFB, and then also connects this LFB to the *LFBChain*. In this way, It gets two half chains with the half length of *LFBChainLen*. At last it connects the two half chains into a whole *LFBChain*.

Because the improved algorithm composes LFB chain from both ends to the middle at the same time, the efficiency of the improved algorithm will be improved greatly than the base algorithm, especially when the LFB chain become long. Specific simulation comparison will be discussed in section 5.

4. Application Example and Analysis

Assuming there is an *LFBR* as follows.

$$LFBR = (Inputport, Outputport), \text{ and } ChainLen = 3.$$

Where

$$Inputport = (Inputport_1) = (Packet1, Metadata1)$$
$$= ((packet1_1), (metadata1_1)) = ((Arbitrary), (Null))$$

The input port set has only one port $Inputport_1$. $Inputport_1$ has only one packet $Packet1$ and one metadata $Metadata1$. $Packet1$ is $Arbitrary$, and $Metadata1$ is null.

$$Outputport = (Outputport_1) = (Packet1, Metadata1)$$
$$= ((packet1_1), (metadata1_1, metadata1_2, metadata1_3))$$
$$= ((IPv4Unicast), (L3PortID, NextHopIPv4Addr, MediaEncapInfoIndex))$$

The output port set has only one port $Outputport_1$. $Outputport_1$ has only one packet $Packet1$ and one metadata $Metadata1$. $Packet1$ is $IPv4Unicast$, and $Metadata1$ is $(\text{L3PortID}, \text{NextHopIPv4Addr}, \text{MediaEncapInfoIndex})$.

LFBSet is shown in Table 1. There are three LFBs, IPv4Validator, IPv4UcastLPM and IPv4NextHop. Here we take IPv4Validator for example to introduce, and the others can refer to RFC 6956 [11]. The function of IPv4Validator is to verify IPv4 header. It has one input port, the needed input packet is arbitrary, and no input metadata is needed. IPv4Validator has four output ports. The packet type of the first output port is IPv4Unicast and the metadata is null. Actually the first output port is the

special output port for IPv4 unicast packet. The packet type of the second output port is IPv4Multicast and the metadata is null too. Actually the second output port is the special output port for IPv4 multicast packet. The packet type of the third output port is IPv4 and the metadata is ExceptionID. Actually the third output port is the special output port for exception. The packet type of the fourth output port is IPv4 and the metadata is ValidateErrorID. Actually the fourth output port is the special output port for validating error.

Table 1. LFB Set of an Application Example

LFB Name	Input Packet	Input Metadata	Output Packet	Output Metadata
IPv4Validator	Arbitrary		IPv4Unicast	
			IPv4Multicast	
			IPv4	ExceptionID
			IPv4	ValidateErrorID
IPv4UcastLPM	IPv4Unicast		IPv4Unicast	HopSelector
			IPv4Unicast	ExceptionID
IPv4NextHop	IPv4Unicast	HopSelector	IPv4Unicast	L3PortID, NextHopIPv4Addr, MediaEncapInfoIndex
			IPv4Unicast	ExceptionID

The processing of the algorithm is described as follows.

1) Traverse LFBSet to find out every LFB matched with *LFBR.Inputport*. Consequently IPv4Validator is the satisfactory LFB.
2) Traverse LFBSet to find out the successor LFB matched with IPv4Validator. Consequently IPv4UcastLPM is the satisfactory LFB and can connect to IPv4Validator to get $IPv4Validator \Theta IPv4UcastLPM$.
3) Traverse LFBSet to find out the successor LFB matched with IPv4UcastLPM. Consequently IPv4NextHop is the satisfactory LFB and can connect to IPv4UcastLPM to get $IPv4Validator \Theta IPv4UcastLPM \Theta IPv4NextHop$.
4) Verify whether the output of the chain $IPv4Validator \Theta IPv4UcastLPM \Theta IPv4NextHop$ matches LFBR.Outputport. As shown in Table 1, the output of IPv4NextHop can match the output of LFBR. Thus LFB chain $IPv4Validator \Theta IPv4UcastLPM \Theta IPv4NextHop$ meets LFBR. The processing of the composition algorithm ends.

After searching and matching, there is only one LFBChain in LFBChainSet. So the result is as follows.

$$LFBChainSet = \{(IPv4Validator \Theta IPv4UcastLPM \Theta IPv4NextHop)\}$$

5. Simulation and Test

C language is used in our simulation, the simulation implements the LFB chain composition method proposed in this paper. The simulation implements both the base algorithm and the improved algorithm to compare their efficiency. Here a composition

time in milliseconds is selected as the performance parameter. The simulation adopts the ForCES LFB library defined in RFC6956 [11].

The test measured the time of the base algorithm and the improved algorithm with gradually increased chain length. The input of the algorithm is the application request donated by LFBR and the LFB chain length donated by ChainLen. The output is the LFB chains satisfied with the application request.

The test results are shown in figure 6.

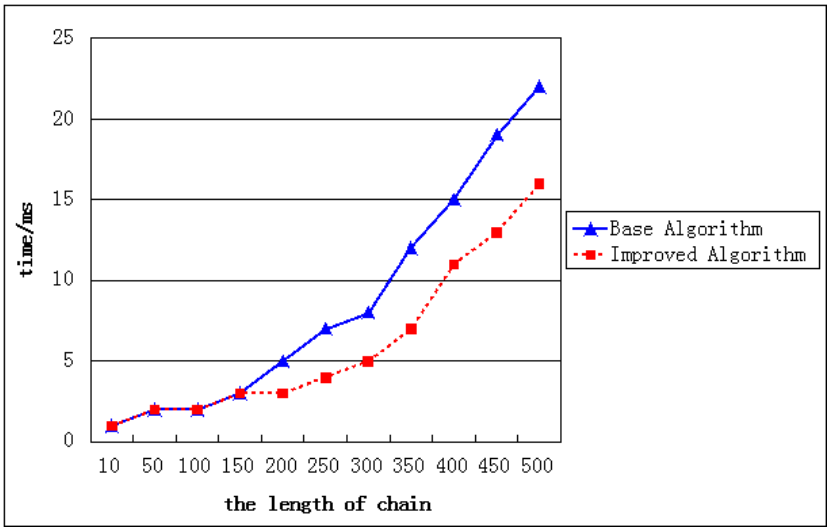


Figure 6. Comparison of the Base Algorithm and the Improved Algorithm

The x-axis in figure 6 shows the LFB chain length, we use the LFB chain length from 10, 50, to 500 for tests. The y-axis in figure 6 shows the costing time of LFB chain composition in milliseconds. The solid line curve represents the base algorithm, and the dashed curve represents the improved algorithm. The graph shows that the improved algorithm spend less time than the base algorithm as the chain length increases, so the improved algorithm has better efficiency than the base algorithm.

In theory, assuming there are n LFBs in LFBSet, the optimal efficiency of the base algorithm is $O(n)$, and the worst efficiency is $O(n!)$. The optimal efficiency of the improved algorithm is $O(n/2)$, and the worst efficiency is $O((n/2)!)$. The test result basically agrees with the theoretical analysis.

6. Conclusion

ForCES has a good Characteristic of separation and centralized control. In addition, ForCES has a very good abstraction and definition of open resources. So ForCES can provide good support for SDN. We proposed a SDN architecture based on ForCES.

In SDN based on ForCES, resources are abstracted into LFBs, and control layer can reconfigure LFB chain to manage FEs. Based on the formalization of the concept of LFB and the improved I/O matching algorithm, a LFB chain composition method is proposed. This method can combine a series of LFBs to a LFB chain according to special application request. Based on the LFB chain composition algorithm, an improved algorithm is proposed, which can improve the composition efficiency. Traditional I/O matching algorithm just specifies the matching between input and output parameters. Our improved I/O matching algorithm specifies not only the matching of I/O packets, but also the matching of I/O metadata, and the definition of matching must be according to the RFC definition of LFBs.

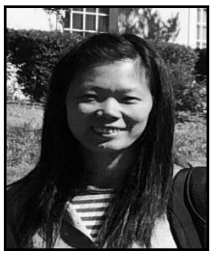
An example is provided to illustrate how this method works, and a simulation is given to compare the efficiency of the base algorithm and the improved algorithm, the result shows that the improved algorithm has better efficiency than the base algorithm.

In addition, the composition method of LFB chain can be divided into three types. They are sequential composition, branch composition and hybrid composition. This paper proposed a sequential composition method of LFB chain. Further research will deep into branch composition method and hybrid composition method.

References

[1] 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 Communications, Review, **(2008)**.
[2] A. Doria, J. Hadi Salim, R. Haas, H. Khosravi, W. Wang, L. Dong, R. Gopal and J. Halpern, "IETF RFC 5810: ForCES Protocol Specification", http://tools.ietf.org/search/rfc5810, **(2010)**.
[3] A.P. Manu, B. Rudra, V. Kumar and O.P. Vyas, "Broker's Communication for Service Oriented Network Architecture", International Journal of Future Generation Communication and Networking, vol. 5, no. 4, **(2012)**.
[4] P. Huss, N. Wigertz, J. Zhang, A. Huynh, Q. Ye and S. Gong, "Flexible Architecture for Internet of Things Utilizing an Local Manager", International Journal of Future Generation Communication and Networking, vol. 7, no. 1, **(2014)**.
[5] N. Feamster, J. Rexford and E. Zegura, "The Road to SDN: An Intellectual History of Programmable Networks", ACM Queue, **(2013)**.
[6] B.A.A. Nunes, M. Mendonca, X.N. Nguyen, K. Obraczka, T. Turletti, "A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks. Communications Surveys & Tutorials", IEEE, vol. 16, no. 3, **(2014)**
[7] E. Haleplidis, O. Cherkaoui, S. Hares and W. Wang, "Forwarding and Control Element Separation (ForCES) OpenFlow Model Library", https://tools.ietf.org/html/draft-haleplidis-forces-openflow-lib, vol. 03, **(2013)**.
[8] S. Hares, "Analysis of Comparisons between OpenFlow and ForCES", http://tools.ietf.org/html/draft-hares-forces-vs-openflow-00.txt, **(2012)**.
[9] S. Hares, "Forces vs. ONF, IETF 84 Meeting Presentation", http://www.ietf.org/proceedings/84/slides/slides-84-forces-3.pdf, **(2012)**.
[10] J. Halpern and J.H. Salim, "Forwarding and Control Element Separation (ForCES) Forwarding Element Model", http://tools.ietf.org/html/rfc5812, **(2010)**.
[11] W. Wang, E. Haleplidis and K. Ogawa, "IETF RFC 6956: Forwarding and Control Element Separation (ForCES) Logical Function Block (LFB) Library", http://tools.ietf.org/html/rfc6956, **(2013)**.
[12] D.R. Zheng, P. Yi and B.Q. Wang, "Journal of Information Engineering University. Matching Algorithm with Multiple Output Ports for Input Queued Switches", vol. 2, **(2009)**.
[13] Y.N. Fu, L. Liu and C.Z. Jin, "Journal on Communications. Service chain-based approach for Web service composition", vol. 7, no. 28, **(2007)**.
[14] S.P. Liu, D.Y. Liu, H. Qi and J.Q. Wang, "Journal of Jilin University:Eng and Technol Ed. Service community chain based approach for web service composition", vol. 1, no. 1, **(2010)**.

Authors



Rong Jin, She received her B.S. degree in physics from Shaanxi Normal University, China in June 2000 and her M.S. degree in communication and information system from Zhejiang University of Technology, China in March 2003. She is currently working towards his Ph.D. degree in control theory and control engineering at Zhejiang University of Technology, China. Her current research interest includes open programmable networks and Software Defined Network.

Xiongxiong He, He received the M.S. degree from Qufu Normal University in 1994 and the Ph.D. degree from Zhejiang University in 1997. From 1998 to 2000, he held a postdoctoral position in Harbin Institute of Technology. He joined Zhejiang University of Technology in 2001, and since then he has been working as a professor in the College of Information Engineering of ZJUT. His research areas include control theory and signal processing.

Guangxin Zhu, He received his Ph.D. degree in communication and information system from Zhejiang University, China in December 2010. He is working as a senior engineer in OB Telecom Electronics Co., Ltd. His current research interest includes open programmable networks and signal processing.