Available online www.jocpr.com

Journal of Chemical and Pharmaceutical Research, 2014, 6(6):1507-1514

Research Article

ISSN : 0975-7384
CODEN(USA) : JCPRC5

The research and implementation of a method of validity guaranteeing in forces system

[1]Qiong Wu, [1,2]Ming Gao, [1]Weiming Wang and [3]Chen Chen

[1]College of Information and Electronic Engineering, Zhejiang Gongshang University, Hangzhou, P. R. China
[2]Department of Information Science & Electronic Engineering, Zhe Jiang University, Hangzhou, P. R. China
[3]National Defense University, Changsha, P. R. China

ABSTRACT

Gives the design and analyses the procedure of the method of validity guaranteeing of ForCES (forwarding and control element separation) LFB (logical function block) topology. To improve efficiency of the method, pattern-matching technology is also introduced. According to the character of LFB topology and comparing with various pattern-matching, we employ the multi-pattern matching and related algorithm for our implementation.

Key words: ForCES; LFB topology; validity guaranteeing; multi-pattern matching

---

INTRODUCTION

Along with the rapid expansion in the application field of computer network, new features and demands appear and evolve constantly. Thus sufficient flexibility is needed in equipments of NGN (next generation network) in order to respond quickly to the new business and requirements. Besides, sufficient openness should be provided so as to allow the users to make a combination of the open resources flexibly. In this way, different demand services of network can be afforded. At last, sufficient modular features is also needed. These modules should be standardized by standardizing organization. Thus, each module can be developed by different manufacturers and can be combined organically into a whole part.

In order to get rid of the bondage of traditional architecture of network equipment, ForCES (forwarding and control element separation) [1] working group which specializes in the architecture of equipment of NGN was founded by IETF (the internet engineering task force) in 2003. The key point of ForCES technology is that the forwarding and control plane are separated structurally in a network equipment. At the same time, the resources in forwarding plane are virtualized, modularized and standardized, thus realizes open reconfiguration flexibly. All these characteristics enable function modules be restructured by network operators just like building blocks. And then realize all kinds of new business by customization [2]

A ForCES system can be divided into two parts, FE (forwarding element) and CE (control element) both of which can have one or more. Redundant backup of control could be implemented by setting multiple CEs, while packets of different rates and protocols can be forwarded and processed by multiple FEs. According to different processing operations, FE can be composed of several LFBs (logical function blocks) [3], e.g. classification, scheduling and queue management LFBs, which are all defined in RFC 5812, i.e. ForCES FE Model [4].

As a logic module that can accomplish specific functions, a range of parameters should be provided to users for configuration and seeing. The parameters belong to LFBs; LFBs belong to FEs while FEs belong to ForCES NE. According to the logic hierarchical relationship, a tree map of ForCES NE resources can be constructed. Each parameter is a node which is unique in the tree. For each node, more describing information which we call LFB

Attribute should be defined, mainly are introduction information and data structure (e.g. the data structure of IP address is 32 bits integer) of the node.

LFB capability which is only for external inquiry reflects the capacity of LFB, e.g. the version number of LFB class, the optional features supported by LFB class, a maximum number of configurable output ports of a output port group and the additional constraints of LFB attributes .

The internal topology of ForCES system is constituted by the connection between LFBs in FE. Usually a topology corresponds to a function of ForCES system. The dynamic topology of LFB in ForCES system means that the internal topology is changed by users online and in real time so as to convert, update etc. system functions.

To sum up, the technology of dynamic topology is a key technology to realize a new generation of open architecture networking equipment. The premise is that the resulting dynamic topology is correct. Only under the condition of that, a new generation of open architecture networking equipment can really run into reality. Therefore, the method of validity guaranteeing of LFB topology discussed here is an important method to guarantee the normal and efficient running of ForCES system. Although the dynamic topology of LFB is researched by many domestic institutes, no effective mechanism has been announced by a research team so far. The institute of the author takes an active participates in the researching and standard setting work of IETF ForCES working group, which is an internationally recognized research group of ForCES technology.

As is known to all, the forwarding process of packets is completely controlled by a switch or a router. But the core idea of OpenFlow [5] is that the process is completed by an OpenFlow Switch and a Controller respectively, which is similar to the core idea of ForCES.

The validity of a topology in OpenFlow is reflected by the consistency of flowtables. A local flowtable is maintenance by each OpenFlow switch in order to forward packets. A flowtable mentioned above is a core data structure to decide forwarding strategy by the switch. An action is taken to deal with network traffic by the switch chip according to entries of a flowtable. Each entry is constituted by three fields, i.e. header field, counters and actions field. The header field will be first analyzed when packets enter in an OpenFlow switch, then look up to see whether there is a corresponding matching entry. If true, related actions will be executed. Otherwise, packets will be forwarded to the controller through a safe tunnel and then any actions to execute will be decided by it. It is based on this method to lookup a flowtable that ensures each packet to be able to execute corresponding operation correctly.

Click, which is a modular software router, is put forward by Eddie Kohler, a doctor in MIT University, American [6]. And it was developed by laboratory of parallel and distributed operation system of department of computer technology in MIT. A complete Click router system should contain three parts, i.e. the component elements, datagram structure and the connection way. The first part is the most basic composition unit. The function of process unit of a router can be implemented by each component element. Users can choose the elements that are needed and write configuration files and combine each element together according to their demands by using the modular design approach. The configuration file of Click is a directed graph which views an element as a node. Packets are forwarded and processed among elements. To this point, it is extremely similar to the LFB topology and its function discussed in this paper. The connection way of different elements can be divided into three kinds, i.e. push, pull and agnostic [7].

Data center network refers to network infrastructures, which connects a large amount of servers through high-speed links and switches [8]. The connection relationship among severs is provided by the topology of the data center network, i.e. the connection sequence of all the links and intermediate nodes between any two servers, by which the way of routing between servers can be determined. The structure of data center network is generally a tree structure composed of second layer or third layer routers and switches [9]. With the continuous development of the data center, more and more defects and insufficiencies gradually expose in the traditional structure. Some of the disadvantages are that it cannot guarantee the correctness of topology, also not easy to expand and upgrade.

In order to improve these problems, a Fat - tree concept is introduced. Fat - tree is an improved structure of the traditional tree. An intermediate node of a Fat-tree structure can have multiple parent nodes and therefore the link number between upper and lower layer set switches and between set and core switches is increased. However, the expansion and upgrading problem of data center network can't be fundamentally solved by Fat – tree. To this point, the VL2, which is a scalable and flexible structure of data center network is proposed by Albert Greenberg [10] based on the traditional tree structure.

FORCES FE MODEL AND A TYPICAL LFB TOPOLOGY

FORCES FE MODEL

The internal structure of FE in ForCES system is shown in Fig. 1. It can be divided into different kinds of LFBs according to various modules' functions in FE. LFB and its attributes can be controlled by CE which runs ForCES protocol, each LFB is connected by datapath. The connection relation (i.e. LFB topology) is also defined by ForCES protocol running on CE in order to form different LFB topology, thus to implement dynamic allocation of resources to accomplish a variety of different types of IP services.

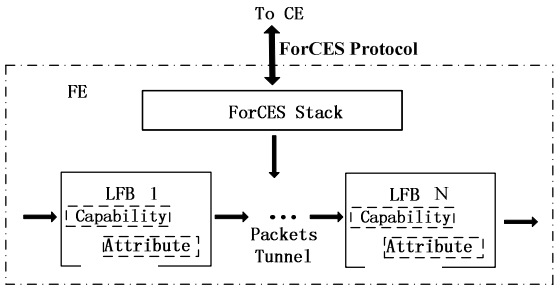

Fig. 1: The internal structure of Forwarding Element (FE) in ForCES system

The requirements that a FE model must meet is defined in RFC 3654, mainly include the following issues:Logically separable and independent packet forwarding operations should be provided by the datapath in FE. Possible topologies are created among all the LFBs and packets will be processed in a certain logical sequence. Possible operation abilities of LFB should be offered (e.g. capacity restriction, constraints, optional attributes and configuration granularity etc.). Configurable parameters of LFB should be afforded. Exchangeable metadata among LFBs should be provided.

It should be noted that a LFB can be divided into common LFB and core LFB. Except for FE Object LFBs and FE Protocol LFBs others are common LFBs, e.g. EtherPHYCOP，EtherMACIn，EtherClassifier，IPv4Validator，IPv4UcastLPM，IPV4NextHop，EtherEncap，ARP，EtherMacOut BasicMetaDataDispath etc. LFBs, which are mainly used for packets processing and all of them should be defined according to FE modeling method.

The difference between a core LFB and a common LFB is listed here: firstly, no input and output ports are included in a core LFB. It is not involved in a topology of packets processing. Secondly, an instance and only one instance of core LFB has already been existed before establishing a connection (the instance ID is 0x1). There is no need to instantiate again.

A FE Object LFB is an abstract of global functions of FE (e.g. LFB topology). The class ID of the LFB is 1. Only one LFB instance can exist in a FE. Its ID is defined as 1.

The ForCES working group has completed the definition of FE Object LFB class and XML modeling language [11] is chosen in the process of definition to describe the formal structure definitions. This is because both human and machines are easy to read XML and it supports broad programming tools. The definition of reusable data types and their own class is included In FE Object LFB definition. The core part is the latter. The definition of information structure of Data type is associated with the class definition, and used by the class definition.

The main function of FE Object LFB is that the LFB classes, instances and instances of output and input information is obtained via the capabilities defined by a manager. The connection relation between LFB instances in a FE is reflected by the information.

Many capabilities and attributes are defined in a FE Object LFB, e.g. ModifiableLFBTopology, SupportedLFBs, LFBSelectors etc. Here only two capabilities and attributes associated with management of LFB topology are discussed detailedly

The legality verification of topology information is closely related to the capability. The LFB classes supported by FE are provided by SupportedLFBs, the type of which is array. Each array element contains all the information of LFB class supported by FE. The type of the array element is SupportedLFBType, which includes the following contents: LFBName, name of LFB class. LFBClassID, ID of LFB class. LFBClassID and LFBName are global unique. LFBVersion, the version of LFB class supported by the current FE. LFBOccurrenceLimit, the maximum

number of instances of LFB class supported by FE. PortGroupLimits and PortGroupLimitType. The information of port group supported by LFB class is provided by PortGroupLimits, the type of which is array. The element type is PortGroupLimitType, which includes PortGroupName (i.e. name of a port group), MinPortCount (i.e. the minimum number of ports allowed by the port groups), MaxPortCount (i.e. the maximum number of ports allowed by the port groups). CanOccurBefores, which describes the current instance that can be deployed before which LFB class instances. The type is array and the element type of which is LFBAdjacencyLimitType just like CanOccurAfters. Unlike CanOccurAfters the ViaPort in LFBAdjacencyLimitType refers to the list of input ports that can be connected by adjacent LFBs. UseableParentLFBClasses, the array of which holds all the IDs of LFB parent (whether inheritance directly or indirectly) supported by FE.

LFBTopology is an optional content, which shows the connection relationship between LFB instances in FE. The data type of LFBTopology is array and the type of element in the array is LFBLinkType. Each array element stores information of each connection endpoint. LFBLinkType includes the following contents: FromLFBID, starting point LFBID of a connection. Its type is LFBSelectorType which contains LFB class ID and LFB instance ID. FromPortGroup, name of an output port group of a starting point in a connection. FromPortIndex, a serial number of the starting point in the output port group in a connection. ToLFBID, ending point LFBID of a connection. ToPortGroup, name of an output port group of an ending point in a connection. ToPortIndex, a serial number of the ending point in the output port group in a connection.

From the above it is not hard to find that LFB topology is described by FE Object in a way of directed connection. FromLFBID corresponds to source LFB and ToLFBID corresponds to destination LFB. Then combine with FromPortGroup, FromPortIndex, ToPortGroup and ToPortIndex, an output port of a source LFB and an input port of a destination LFB can be uniquely identified by any directed connection.

A TYPICAL LFB TOPOLOGY

A LFB topology which supports IPv4 unicast routing forwarding is depicted in Fig. 2, which contains several LFBs like EtherClassifier LFB, IPv4Validator LFB, I IPv4UcastLPM LFB, IPv4NextHop LFB, EtherEncap LFB, layer 2 table look-up LFB, Address resolution protocol LFB.

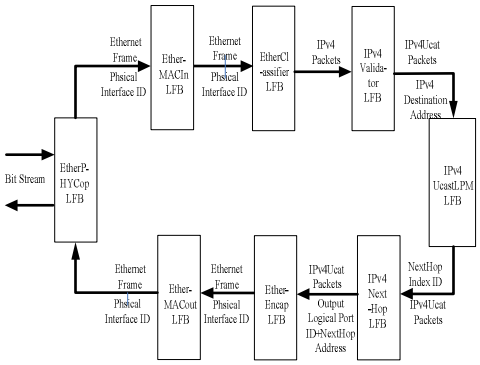

Fig. 2: A typical LFB topology which supports IPv4 service

1) EtherClassifier LFB, although it is called Ethernet classifier, actually it implements the two functions, i.e. encapsulation and classification of header of MAC frame. After de-encapsulating, the type field is extracted from the frame header in order to distinguish IPv4 and ARP packets.

2) IPv4Validator LFB, a header verify module which works in IP layer. It verifies IP packets one the one hand, and classifies packets of unicast, multicast, broadcast, etc. All the suspending outputs in Fig. 2 mean sending signals to CE. Then it will be processed by CE.

3) IPv4UcastLPM LFB, routing prefix will be matched according to destination IP address of IPv4 unicast packets. The item with maximum matching length [12] will be chosen. Then the index of Next hop table can be obtained.

4) IPv4NextHop LFB, looking up the MAC address corresponding to the next hop IP address in layer 2 cache table. If it exits in the table, the MAC address will be exported. If not, the ARP procedure will be started to get the address.

5) EtherEncap LFB, encapsulating Ethernet frame after getting the destination MAC address.

6) Address resolution protocol LFB, which is used to implement ARP.

The label above each connection in Fig. 2 is frame, the label below is metadata, e.g. EtherClassifier LFB has one input and two outputs. The frame required by the input is Ethernet frame, the metadata is the physical interface ID. The frame generated in output one is IPv4 packets and no metadata generated. The frame generated in output two is ARP packets and the metadata is the physical interface ID and the source MAC address. It is not difficult to understand one of the conditions a legal directed connection must meet is that the output of a source LFB must be consistent with the input of a destination LFB, the input and output of a LFB should be clearly defined in RFC 5812.

A basic LFB library is provided in the document [13] according to FE model, ForCES protocol specification and some typical routing function modules in a router. The LFBs used for Ethernet processing include EtherPHYCop, EtherMACIn, EtherClassifier, EtherEncap and EtherMACOut. The LFBs used for IP packets verifying contain IPv4Validator and IPv6Validator. LFBs for IP packets forwarding are IPv4UcastLPM, IPv6UcastLPM, IPv4NextHop and IPv6NextHop. LFBs for redirecting are RedirectIn and RedirectOut. LFBs for common use include BasicMetadataDispatch and GenericScheduler. The identification number of the LFBs range from 3 to 17 in a sequence listed above.

AN IMPROVED METHOD OF VALIDITY GUARANTEEING OF LFB TOPOLOGY

Several steps are proposed to design the mechanism and the flow chart of the configuration of LFB topology is shown in Fig. 3.

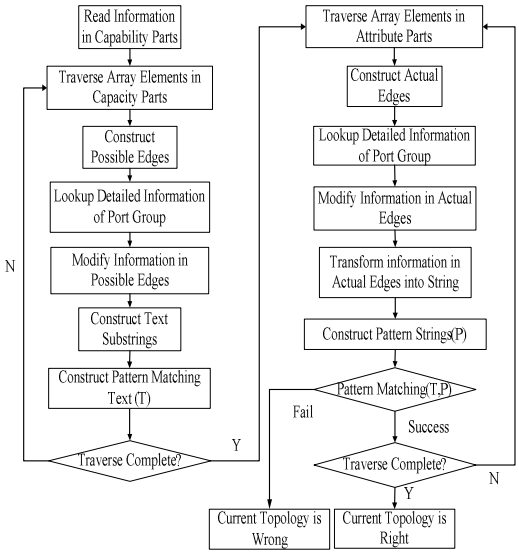

Fig. 3: The flow chart of the configuration of LFB topology

Firstly, read information contained in the capability parts from FEObject LFB to create a chain table of topology capabilities. Each data element in the table is a directed edge allowed in a LFB topology, i.e. possible edge. The data type of FEObject LFB is array and the element type is struct, each struct corresponds to a LFB. Its existence shows that the LFB can be instantiated. The restrictions of instantiation of the corresponding LFB and possible adjacency relation are described in a struct. Members of the struct contains class name, class ID, maximum number of instances, port restrictions and adjacency relations with previous and next LFBs of the current LFB. Read the description of the adjacency relations from an element struct. Then exhaust and construct all the possible edge. Here the starting point of a possible edge constructed by the adjacency relation of a previous LFB is the previous LFB. The ending point is the next LFB. While the starting point of a possible edge constructed by the adjacency relation of a next LFB is the current LFB and the ending point is the next LFB. Message format for the possible edge contains the class ID and the name of the output port group of a starting point LFB and the class ID and the name of the input port group of an ending point LFB. The adjacency relationship with neighbor LFBs discussed above is just a kind of permission, which means it may not exist in an actual LFB topology.

Secondly, traverse each possible edge in the table mentioned in step 1. Use the name of a port group as a keyword to find detailed information of the port group in the LFB definition file. And then use the detailed information to replace the group name. Thus make possible edge information contains the class ID and the output information of a starting point LFB and the class ID and the input information of an ending point LFB.

Thirdly, traverse all the possible edges in the table after the replacement mentioned in step 2. Then transform the information included in the edges into string, called a text substring (T). The format is show as follow, " the class ID of a starting point LFB; the output information of a starting point LFB; the class ID of an ending point LFB; the input information of an ending point LFB ". The character":" is a segmentation character. Join all the text substrings together into one larger string as the main text of pattern matching. The format is, "text substring 1 & text substring 2 & …", "&" here is a segmentation character.

Then read the attribute parts to gain the current configuration topology from FEObject LFB. The data type of the configuration topology is array and the type of an array element is struct. Each data element in the table stands for a directed edge which actually exists in a LFB topology, i.e. actual edge. Information contains in an actual edge is the class ID, the instance number, the name of an output port group and the serial number of an output port group of a starting point LFB and an ending point LFB.

Next, traverse all the actual edges in the current configuration topology mentioned in step 4. Use the name of a port group as a keyword to find detailed information of the port group in the LFB definition file. And then use the detailed information to replace the group name. Thus make actual edge information contains the class ID and the output information of a starting point LFB and the class ID and the input information of an ending point LFB.

Traverse all the actual edges in the table after the replacement mentioned in step 5. Then transform the information included in the edges into string, called a pattern string (P). The format is show as follow, " the class ID of a starting point LFB; the output information of a starting point LFB; the class ID of an ending point LFB; the input information of an ending point LFB ". The character":" is a segmentation character.

At last, execute pattern matching [14], use the main text described in step 3 as the target in the pattern matching. Use the strings mentioned in step 6 as the pattern and then match one by one. Only when each string matches successfully, then returns success, otherwise returns failure; Here success mean the current LFB topology is correct while failure means not correct.

We can determine that whether each edge could be found in the capability library by a pattern matching algorithm after constructing the capability library and the actual edges. There are a lot of pattern matching algorithms. Classical single pattern matching algorithms include Knuth - Morris - Patt (KMP), Boyer Moore (BM) and the Quick Search algorithm (QS). Classical multiple pattern matching algorithms contain Aho Corasick (AC) algorithm and the Wu - Manber (WM) [15] algorithm. An improved algorithm based on AC algorithm is proposed in this paper to shorten the time of traversal.

String pattern matching are defined as follows, a text string, called text, whose length is n, a string called pattern is used for matching, its length is m, generally m < n, characters in a string come from the extended ASCII, if a string is found in the text which completely match the pattern, we call it is a successful string pattern matching. If no strings match the pattern, we call it a unsuccessful string pattern match.

All the keywords are incorporated into a collection by AC algorithm and all the strings in the collection are accepted simultaneously according to the theory of finite automaton. Each prefix can be marked by a unique state as the automaton is structurized. Even those prefixes of multiple patterns are also the same. When the next character in a text is not the one expected by a pattern, it will turn to a state represented by the longest prefix of a pattern, which is the corresponding suffix of the current state [16]. The time complexity of AC algorithm for pattern matching is O (n). The total time complexity of AC algorithm is O (M + n) when considering pretreatment time. Here M stands for the length of all the string patterns. The implementation of AC multiple pattern matching algorithm can be divided into two phases, i.e. preprocessing and matching. In the preprocessing phase, a finite state machine will be generated according to the pattern groups waiting for matching. While in matching phase, the state can be changed according to the input text. When reaching a state, if it has a matching pattern, the match is successful. Three functions are established respectively by AC algorithm in preprocessing phase, i.e. the goto function, fail function and output function, which form a finite automaton of tree form. In searching phase, using the above three functions alternately, and then scanning the text, which can locate all the keywords appeared in the text.

Several patterns $P_1, P_2, P_3, \ldots Pq$ are matched in $T[1,2,\ldots n]$ at one time. Here q stands for the number of patterns [17]. When q equals to 1, multiple pattern matching will degenerates into single one. The length of pattern Pi is m_i, i.e. $P_j[1\ldots m_j](1 \le j \le q)$. minlen is denoted as the minimum length of patterns, i.e. $minlen = \min\{m_j(1 \le j \le q)\}$.

An Improved Algorithm of Multiple Pattern Matching

Preprocessing Phase

Firstly the pattern set will be transformed into reverse finite automaton and the goto function and output function can be obtained according to the character distribution of the set. Then f_1 and f_2 functions will be constructed. The value of $f_1(c)$ is the distance between the char c occurred in the rightmost and the end of a pattern. However, the value of $f_1(c)$ should not exceed $\min(l)$. Or the shortest pattern is likely to be leaved out. The definition of $f_1(c)$ is shown as follow.

$$f_1(c) = \begin{cases} \min\{m\_j\text{-}i \mid P\_j[i]=c\}, c \in P \\ \min(l), \quad \text{else} \end{cases} \tag{1}$$

The value range of i and j is $m_j - \min(l) \le i < m_j, 1 \le j \le q$. The calculation of $f_1(c)$ is divided into two steps. Firstly let $f_1(c) = \min(l)$, here char represents each character in a string set. Then let $f_1(P_j[i]) = \min\{f_1(P[i]), m_j - (i+1)\}$, here $P_j[i]$ $(0 \le i \le m_j - 1)$ is a character in pattern P_j. Only when $c = P_j[m_j - 1]$, $f_2(c)$ function works. Here is the equation.

$$f_2(c) = f_1(P_j[m_j - 1]) \tag{2}$$

Matching Phase

The algorithm proposed in this paper is a reverse matching from the right side to the left of a pattern set. Before matching, let $f_1(P_j[m_j - 1]) = 0$. While matching, the procedure begins from $k = \min(l) - 1$, T[k] and match the last character of a pattern. If the procedure fails, the text pointer moves $f_1[T[k]]$ units to the right until it finds the last character of a pattern, here $k = k + f_1[T_k]$. Then match to the left according to goto function. If the matching succeeds, let $i = k, i = i - 1$ and match the next T[k]. If it fails, the text pointer moves $f_2[T[k]]$ units to the right and restart a matching process. Here $k = k + f_2[[T_k]]$. During the procedure, the pattern position in the text will be output if the pattern is matched. All the pattern positions can be found out when the whole text is processed by the mechanism.

RESULT AND DISCUSSION

The test environment is Intel(R) Core(TM)2 Quad CPU Q9500 2.83GHz,4GB RAM and Microsoft Windows XP Professional operating system. The compile environment is Microsoft Visual C + + 6.0.

Suppose the size of the capacity library is 3Mbits , the average length of the actual edges is 40 bytes, and then we can see from Fig. 4 that as the number of actual edges increases, the matching time increases either. However, the improving algorithm is more efficient and needs less matching time when compared with the initial AC algorithm.

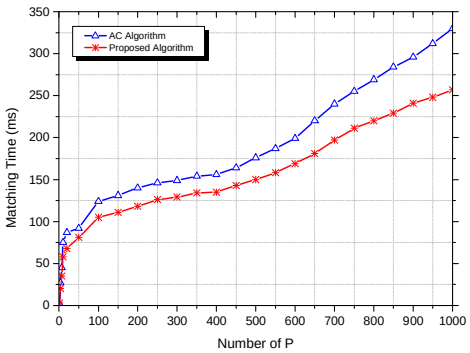

Fig. 4: Relation diagram of matching time and number of actual edges

Suppose the number of actual edges is set to 50 in advance, then with the increasing size of the capacity library, the relation of matching time and size of capacity library of the two algorithms is shown in Fig. 5. With the extending of the capability library, matching time in general is on the rise, but the time lag between the improved algorithm and the classical AC algorithm becomes larger gradually.

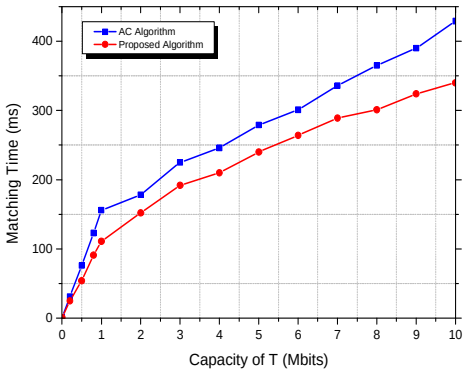

Fig. 5: Relation diagram of matching time and size of capacity library

CONCLUSION

This paper first introduces the model of ForCES FE and proposes a method of validity guaranteeing in LFB topology based on pattern matching algorithm which can effectively check the correctness of the current LFB topology. Here the concepts of possible edges and actual edges are raised for the reason of respectively describing information in capability and attribute parts of FEObject LFB. Then an improved pattern matching algorithm is proposed based on AC algorithm. In the end, a test environment is built to compare the matching efficiency of the two algorithms. We find that when the capacity library or the actual edges are set to a fixed size, the improved algorithm needs less matching time and is more efficient compared to AC algorithm.

Acknowledgments
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.61102074, 61170215), Zhejiang Leading Team of Science and Technology Innovation (No.2011R50010), Zhejiang Sci & Tech Project (No. 2012C33076), and Zhejiang Provincial Natural Science Foundation of China (No. Y1111117).

REFERENCES

[1] L. Yang, R. Dantu, T Anderson, et al. Forwarding and Control Element Separation (ForCES) Framework, *IETF RFC* 3746, April, **2004**.
[2] Weiming Wang etc. Forwarding and Control Element Separation (ForCES). Hangzhou: Zhejiang University Press, **2010**.
[3] Ligang Dong, Fengen Jia, Weiming Wang. Definition and Implementation of Logical Function Blocks Compliant to ForCES Specification [J], *ICON*, **2007**.
[4] J. halpern, et al. Forwarding and Control Element Separation (ForCES) Forwarding Element Model [EB/OL], *http://tools.ietf.org/html/rfc5812*, **2010**.
[5] OpenFlow website: *http://www.openflowswitch.org.*
[6] Kohler E, Morris R, Chen B, et al. *ACM Transactions on Computer Systems*, 18(3): 263 - 297, **2000**.
[7] Daoping Sun, Hongke Zhang, Huachun Zhou, *Computer Applications and Software*, Vol.28 No5, **2011**.
[8] Chuanxiong Guo, Haitao Wu, Kun Tan, Lei Shi, et al. DCell: A Scalable and Fault-tolerant Network Structure for Data Centers [C]//*Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM'08)*, Aug 17-22,**2008**.
[9] AL-FARES M, LOUKISSAS A, VAHDAT A. A Scalable, Commodity Data Center Network Architecture. *[C]//Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM'08)*, 63-74, **2008**.
[10] G. A, H. J R, J.N, et al. VL2: A Scalable and Flexible Data Center Network SIGCOMM, Aug17-21, **2009**.
[11] Christopher League, Kenjone Eng. *Journal of Computers*, Vol 2, No 10 (2007), 9-17, Dec **2007**.
[12] Ming Yu, Dongju Wang. *Journal of Computers*, Vol 8, No 10 (2013), 2724-2729, Oct **2013**.
[13] ForCES Logical Function Block (LFB) Library, *http://www.ietf.org/id/draft-ietf-forces-lfb-lib.txt*, ***2013***.
[14] Sung-Hwan Kim, Chang-Seok Ock, Hwan-Gue Cho. *Journal of Computers*, Vol 8, No 7 (2013), 1804-1809, Jul **2013**.
[15] Yafeng Yao, Yi jiang. *Nantong vocational college journal*, 25 (4): 98-100, **2011**.
[16] Xihong Wu, Feng Zeng. *Journal of Computer Engineering*, 38(6): 279-281, **2012**.
[17] Xiaoyan Cai, Guanzhong Dai, Libin Yang. *Journal of Computer Applications*, 27(6): 1415-1417, **2007**.