Towards Internet Innovation: Software Defined Data Plane

Gaofeng Lv*, Zhigang Sun, Yijiao Chen, and Tao Li

School of Computer, National University of Defense Technology,
Changsha, Hunan, China 410073
{lvever,sunzhigang,yijiaochen,taoli}@nudt.edu.cn

Abstract. In order to support new network architectures, Openflow implements flows forwarding based on multiple tables via pipelines, which increases the difficulty of the implementation. With the advent of multi-core CPU, a software defined data plane, LabelCast, is proposed, which characterizes the ability of forwarding operations and processing services through the Label table and Cast table. Forwarding layer lookups based on fixed-length labels and schedules packets processing, including light-semantics action instructions of general process, which is easy to be realized and is denoted by the Label table, and protocol semantics or status-related service of special process, which could be enriched via opening resources within network devices and is arranged by the Cast table. LabelCast supplies a reliable and programmable data plane, and could load multiple network architectures, so as to facilitate Internet innovation.

Keywords: Processing in networks, network experiments, Labelcast.

1 Introduction

To promote network evolution, Stanford proposed Openflow protocol[1]. Openflow abstracts data-plane of networks, makes the control-plane programmable, and supports isolating experiment traffic from production traffic, which offers a powerful support for a new network protocol to be experimented in a real network environment. Openflow makes a compromise between academia and industry by abstracting the data-plane and opening the interface, so that it allows researchers to develop new protocols based on the unified interface. Openflow has been the de facto abstract layer of data-plane in the SDN (Software Defined Network)[2], and being applied to the WAN, like traffic optimization in Google's global data center and infrastructure construction of Internet2 etc.

Satisfying the vendor's demand for device closure and user's demand for isolation of packets promoted Openflow, which also placed restrictions on the Openflow's support for new network architectures. There are some network architectures which cannot run well based on the Openflow, especially which needs

* The work is supported by 863 project(2013AA013505) and NSFC project(61202485).

to deal with packets one by one, like NDN[3] etc. Openflow provides two ways to deal with packets one by one. One way is sending packets to Openflow controller, which poses a great burden on Openflow controller and is also contrary to the architecture of separation between data-plane and control-plane. The other way is sending packets to intelligent data processing platform, which directly connects with Openflow switches. Due to the need of vendors for device closure emphasized by Openflow, it just provides the data-plane abstract but hides the implementation details in data-plane. It is not conducive for researchers to extend Openflow dataplane function to support per-packet processing.

In order to support the packet-by-packet processing of network architectures such as NDN and etc., researchers use programmable hardware to extend the network forwarding-plane like NetFGPA[4], NetMagic and etc., build a software virtual router[5][6] based on general multicore processors[7][8] and deploy Middleboxes[9][10] in networks so as to enable more functions of in-network processing. Processing in networks provides an efficient implementation of innovative network functions, but the lack of a unified abstraction and control interface is not conducive to the development and deployment of new network protocols. We further enhance the scalability of the network equipment to support new network architectures by lending the vendor's need toward the device closure of the resources, which lets device vendors provide intelligence processing resources and development interface based on the network equipment resources. At the same time, the isolation of experimental traffic and production traffic in Openflow is a kind of scheduling based on packet options, which is unable to meet the demand of dealing with special message options in new network architectures. We directly distinguish network architectures by labels, which simplify packets classification and forwarding. Inspired by opening resources and unified labels, we propose a software defined data-plane named LabelCast. LabelCast identify network architectures with labels at the protocol level, while only making abstraction for the ability of computing, storage and forwarding of network devices, so that could support new network architectures.

In Labelcast, by dividing the data plane of the network into fast forwarding plane and intelligence service plane based on semantics of processing in networks, and Label tables and Cast tables are proposed to abstract forwarding resources and the computation and storage resources separately. LabeCast has the following advantage: (1) Simplifying hardware implementation of forwarding lookup based on fixed-length labels, (2) Enabling multiple user-defined applications running concurrently controlled by multiple Cast tables, (3) Achieving the application isolation leveraging the extensible resource container, (4) Loading multiple network architectures via LabelCast abstraction layer.

The structure of the paper is showed as follows, section 2 introduces the evolution of network architectures, section 3 proposes a software defined data plane, LabelCast, and the key mechanisms, such as the mapping and allocation of labels, service atoms, section 4 provides the application of LabelCast, section 5 is the testing and analysis of the performance of the prototype, and the last section is the conclusion of the work.

2 Related Works

Openflow utilizes the three tuple <Matching, Instructions, Statistics> to denote the abstraction layer of network forwarding. In the evolution progress, the matching field extended from 12-tuples of Ethernet, IP and TCP in Openflow specification 1.0 to 15-tuples in Openflow specification 1.1, and from fixed length rules to the variable-length matching field based on TLV <Type, Length, Values> in Openflow specification 1.2. The complexity processing of matching implemented in Openflow switches try to support new network architectures. However, just enlarging the matching field would increase the difficulty of the implementation and could not satisfy unknown network protocols. The processing of Openflow extends from one stage process to multiple stage pipelines in order to solve the combination blast of multiple tables, which would increase the complexity of mapping the special processing of the content-centric or service-centric network architectures to the multiple pipelines of Openflow. The functions of Openflow switches are continuing to develop, but they are limited to the processing of packet basic options, such as modification and replacement, and still could not support the requirements of new network applications, such as data cache.

To solve the complexity of the implementation of Openflow multi-stage flow tables, based on current router functions of forwarding and controlling, Openflow+[15] made an extension of Openflow on the aspects of the interface between the forwarding and controlling layers and flow tables, and then proposed the presentation of openflow rules based on TLV and the mechanism of the mapping Openflow flow tables to ACL and FIB of routers. The scheme provides a simple method to implement Openflow on commodity routers, simplify the implementation of Openflow and enhance the deployment of Openflow.

Juniper enlarges the forwarding layer by customization services, and then proposes the service layer[16]. By integrating computing and storing resource with network elements Juniper service layer provides the platform and SDK for services from the third-party, which could support multiple services of network managements and has good scalability. Due to the capability of SDK Juniper service layer runs services mainly including network management, which could not support packets processing in depth in the key data path. On the other hand, Juniper service layer only admits authorized partners developing new network applications on their platform, which lacks of openness.

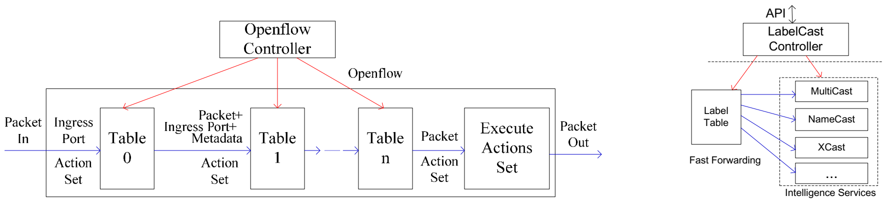
Middleboxes[11] provide a novel method of patching to add new functions to Internet, which could implement general packet forwarding and advanced packet processing, such as NAT, firewall, proxy, IDS, etc., in order to enhance the ability and security of Internet. On the other hand, the widespread deployment of middleboxes and other network appliances has primarily resulted in some challenges and criticism due to poor interaction with higher layer protocols.

Middleboxes also brought a lot of problems, which utilize the method of redirecting or DNS mechanism for targeting network traffic to Middleboxes. The method would increase the complexity of the network control, and add packet processing delay. On the other hand, Middleboxes are short of scalability, which could not meet the requirements of high-performance flows processing, and lacks of programmability, which could not support the new network protocol experiments, only act as the rapid deployment of mature network protocols.

Future Internet, such as XIA[12], NDN[13], Nebula[14], proposes different forwarding layer of networks. NDN abstracts the forwarding layer by the content store table(CS), the pending interest table(PIT) and the forwarding information table(FIB). CS provides the index of data storing, PIT records the pending interest packets to eliminate redundancy requests and the interest packets are forwarded according to FIB. Forwarding layer of NDN implements packets forwarding and supports data storing, which could not be implemented on Openflow switches. XIA adopts directed-graph to identify the path between the source and destination nodes, in which all elements are identifiers of entities (e.g., addresses, identifiers of services, identifiers of contents). The target contents could be retrieved at intermediate nodes of XIA when the present of the fallback path in the directed graph, which could support the end-to-end and non-end-to-end communication model and could not be realized in Openflow.

3 LabelCast: Software Defined Data Plane

With the development of microprocessors, the performance of CPU upgrades continuously and could implement more and more complexity processing in networks. At the same time, in the evolution of next generation architectures some new architectures are proposed, which requires high performance and more flexibility of network elements. To resolve the problem and extend the functions of forwarding layer of network platforms, a software defined data plane, *LabelCast*, is proposed based on network devices with multi-core processor.



(a)Openflow Model (b)LabelCast Model

Fig. 1. Openflow vs. LabelCast

In LabelCast, services are running on the computing and storing resources within network elements to implement protocol-related special processing in data path, which could be added easily according to the requirements of new network experiments compared with mapping the special processing of new network protocols to multiple pipelines of Openflow, as illustrated in Fig.1(a), and could run concurrently via utilizing the parallelism of multicore processors, as illustrated in Fig.1(b). Logical central controller of Labelcast provides the running platform for network application controllers, maps the network protocols to labels and provides unified configuration interface of forwarding layer to support the designing and testing of new network experiments in future.

3.1 Labelcast Scheme

The abstraction layer of LabelCast provides five tuples <*Label, Instructions, Service, Status, Restrictions*> as a unified and simple management interface for LabelCast controller, which abstracts the forwarding operations and processing services of the forwarding layer. In rules the label field includes label value and options, which is the complementary of labels and indicates the method or precondition of packet processing. The instruction field indicates forwarding operations(e.g., lookup and modification of the base options of packets). The service field indicates the functions of processing packets including the sequence of service atoms(e.g., matching and modification of any packet fields, storing and loading data).

a) *Instructions*: Modifying basic options of packets, and controlling packet in/out(forwarding operations) in the fast forwarding plane.
b) *Services*: The network protocols semantics or status-related special processing functions in the intelligence service plane based on the computing and storage resources within network devices. Services abstract the characterizes of computing and storage resources and supply the development library for users to design applications, which is denoted by *atomService*, *atomService* = app (compResource, storeResource, pkt).

LabelCast utilizes *Label table* and *Cast table* to describe and manage packet forwarding operations and services of packets processing based on the computing and storage resources within network nodes.

c) *Label Table*: Indicates the index of packets forwarding instructions or processing services on the forwarding plane, the entry contains the label and option domain, the action command domain and service index ID field, which is denoted by <Labels, Instructions, ServiceID>.
d) *Cast Table*: Indicates the forwarding plane services handling packet deeply, the entry contains the service domain, the state domain and resource constraints domain, which is denoted by <Services, Status, Restrictions>.

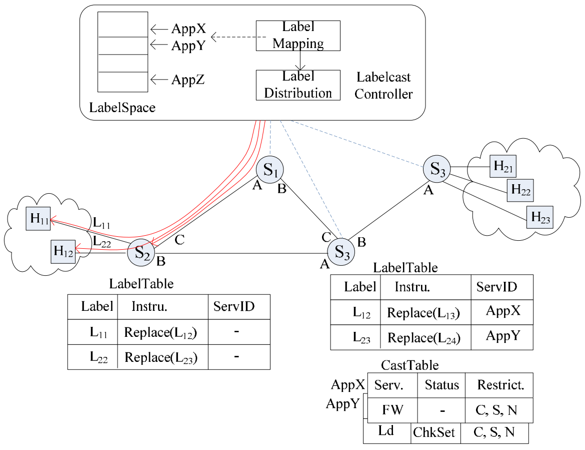
After forwarding operations based on the Label table, packets could be redirected to the target Cast table to be further processed based on protocols and status of networks. Unlike Openflow model of pipeline, Labelcast adopts parallel processing model controlled by multiple Cast tables, which could utilize the ability of multi-core processors and exhibit the simplicity of run-to-completion model. Labelcast controller manipulates the behavior of forwarding layer by configuring the label table and the cast table with the rules.

In the rules, the label field is described by OLV <Offset, Length, Value>. The offset field is the offset of the fixed-length labels in packets. OLV could increase the flexibility of rules exchanged between the Labelcast controller and the forwarding layer of network devices.

3.2 Label Table

The label table includes labels, instructions and services. The matching field is labels of packets, and the instruction field indicates packets forwarding actions, and the service field indicates the index of the method to process packets.

Labels are allocated by the LabelCast controller. The first packet of flows are sent to LabelCast controller due to missing against the Label table. Labelcast controller analyzes the packet and dispatches it to the target network application, which registers to process the protocol type of the packet. The network application allocates the label from the subspace of labels and assigns the label to the flow, and make the process policy for the flow. Based on the label and the policy the Labelcast controller generates rules of Label table, and notify the rules to the upstream and downstream nodes, as illustrated in Fig.2. At the same time, the Labelcast controller notify the label to the source node of the flow. After the configuration, the host sends the following packets of the flow with the label, which are marked by the Labelcast adapter of hosts.



LabelTable

Label	Instru.	ServID
L_{11}	Replace(L_{12})	-
L_{22}	Replace(L_{23})	-

LabelTable

Label	Instru.	ServID
L_{12}	Replace(L_{13})	AppX
L_{23}	Replace(L_{24})	AppY

CastTable

	Serv.	Status	Restrict.
AppX	FW	-	C, S, N
AppY	Ld	ChkSet	C, S, N

Fig. 2. Label Allocation and Distribution

In the forwarding layer lookup of the fixed-length labels of packets against the label table gets the method of packet processing, such as action instructions of light semantics or pointing to service atoms. The forwarding plane packet processing is usually implemented by multi-stage processing, including simple packet option modifications, such as TTL minus 1, look-up and output, as well as label replacement. Multi-stage processing are decomposed into the flow processing in the coarse-grained, firstly light semantics-related modifying options and matching against rules, then the semantics-related packet processing, and finally the light semantic output control. In Labelcast, modifying basic options and output control of packet forwarding are denoted by instructions and implemented by forwarding hardware, intermediate stage of packet processing as a network service based on the computing and storage resources. The instruction is constituted by a number of basic actions, such as output and drop action, which implement stateless packet forwarding operations.

3.3 Cast Table

Services are the sequence of service atoms and are scheduled by the Cast table. Labelcast could extend the service based on the resource of computing and storage. With the repaid increase of processors performance and memory capacity, network nodes has a strong ability of computing and storing in addition to the general forwarding hardware, which could provide the running platform for service to implement network protocol semantics and status-related services of packets processing in depth in the key data path.

Service atoms are implemented through application programs running on the platform, and provide advance functions related to network protocols, such as modification or replacement of any packet fields, forwarding based on rules and storing and loading of data. On the other hand, service atoms could modify the content of rules in forwarding layer, which could impact the process of subsequent packets, and could assimilate packets and produce new packets. Service atoms are developed by network vendors and could be extended by upgrading programs.

a) *Buffer primitives*: The allocation operation of the shared buffer or dedicated buffer memory for applications, which are denoted by *bufferAlloc*, *atomService* = *bufferAlloc* (restrictions) to abstract the allocation operation of storage resources required by the service applications.
b) *Threading primitives*: The operations for the creation of threads, which are denoted by *createThd*, *atomService* = *createThd* (restrictions) to abstract the allocation operation of computing resources applied by services.
c) *Registeration primitives*: The operation of adding user-defined function to a thread, which are denoted by *registerFun*, *atomService* = *registerFun* to dynamically load service functions.

Packets process involving a variety of services, for example, IP over MPLS process includes the options-specific modification of packets, in stack and out stack operations of labels, and the output control operations. Multiple service primitive perform in order, which constitutes a processing service, and pass the intermediate processing results through Metadata, which recorded in the state field of the Cast table entries, to implement more complex processing in networks.

Service atoms are application programs based on the computing and storage resources within network elements and implement protocol-related packet processing in depth on the key data path. Service atoms could be upgraded to implement new services and to support new network protocol-related packet processing, which could avoid the complex upgrade of hardware of forwarding layer and increase the flexibility of Labelcast. At the same time, network researchers could design new services of packet processing through reconfiguring the sequence of service and developing new service atoms, which could support new network architectures, based on opening resources within network devices.

3.4 LabelCast Processing

The Labelcast rules of five tuples include the restriction field of resources, which indicates the resources that network applications could use. Each LabelCast node allocates computing resource, storage and network bandwidth to network applications based on the LabelCast rules issued by LabelCast controller, which realize the resources sharing between network experiments. LabelCast nodes adopt eXtensible Linux Container to manage resources and allocate computing time, storage size and network bandwidth to the container based on the restriction field of rules, which provides the dedicated resources for network applications.

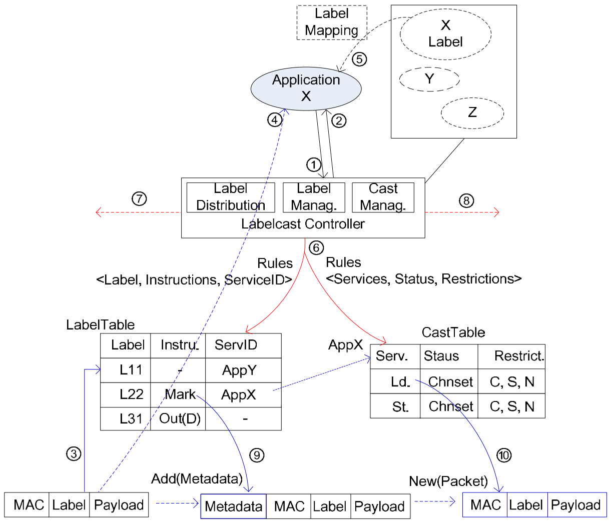


Fig. 3. Processing Model of Labelcast

The network application registers the network protocol of packets to the Labelcast controller firstly, and the Labelcast controller allocates the subspace of the label space to the application, as illustrated in Fig.3. When packets arrived at the Labelcast node, the Label table are lookup against the label in Labelcast options of packets. The first packet of flows with default label could be sent to the Labelcast controller due to unmatching against the Label table. The Labelcast controller dispatched the packet to the network application, which register the network protocol type. The network application allocates the label from the subspace and assigns the label to the flow, and make the process policy for the flow. Based on the label and the policy the Labelcast controller generates rules and configure the Label table and Cast table, and then notify the rules to the upstream and downstream nodes. At the same time, the Labelcast controller notify the label to the source node of the flow. After the configuration, the host send the following packets of the flow with the label, which are marked by the

Labelcast adapter of hosts. The following packets match the Label table and are modified according to the instructions, and then are forwarded to the Cast table and are processed by the target service.

4 NDN Based on Labelcast

We design NDN based on Labelcast, which demonstrate that processing in networks could be developed based on the software defined data plane efficiently.

For NDN, the prefixes of structural names are mapped to the fixed length label, so the interest packets for different chunks of the same content are assigned the same label, and are processed by the same method, which implements the aggregation of interest packets with the same names. Labelcast controller notify hosts the label, which would be inserted in the following interest packets for the content. Hosts requesting the same content behave alike, so they are assigned the same label, which could increase the characteristic of the flow and utilize the cache of data efficiently.

NDN controller running on Labelcast controller computes the path of interest packets, for simple, ingress data packet and egress interest packet are assigned the same label, which make data packets returning as the original path. Labelcast controller uses the label and the service of storing provided by the forwarding plane to configure Label table, and configure the policy of storage, which determines whether storing the data or not, and determines interest packet accessing data cache or not.

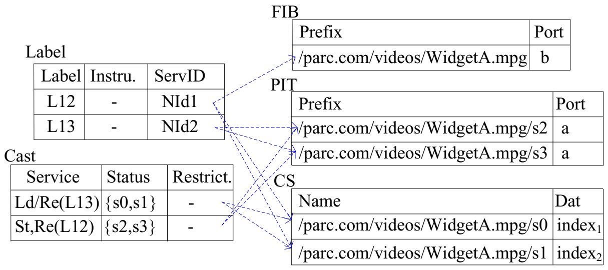

Label
Label | Instru. | ServID
L12 | - | NId1
L13 | - | NId2
Cast
Service | Status | Restrict.
Ld/Re(L13) | {s0,s1} | -
St,Re(L12) | {s2,s3} | -
FIB
Prefix | Port
/parc.com/videos/WidgetA.mpg | b
PIT
Prefix | Port
/parc.com/videos/WidgetA.mpg/s2 | a
/parc.com/videos/WidgetA.mpg/s3 | a
CS
Name | Dat
/parc.com/videos/WidgetA.mpg/s0 | index1
/parc.com/videos/WidgetA.mpg/s1 | index2

Fig. 4. Rules mapping between Labelcast and NDN

In NDN network, interest packets are forwarded based on FIB, while data packets are forwarded by PIT, which is generated according to the ingress port of interest packets. The lookup in NDN is longest prefixes matching, the publisher registers the name of content, and maintains the content on source nodes. In LabelCast, when interest packets arrive, the result of lookup based on the fixed-length labels indicates actions instructions of the general processing of the base options or services to process packets according to the name. Services update the status of storing by adding chunk number to the member set of labels and ports, which is implemented by Bloomfilter[17]. When data packets arrive, services modify labels, choose the output port, and delete the chunk number from the member set of labels and ports. The Label table and the Cast table implement the forwarding of interest packets and the generation of data packets together, which correspond the functions of PIT and CS in NDN, as illustrated in Fig.4.

5 Prototype of LabelCast

5.1 Implementation of the Prototype

Based on the general-purpose multi-core processor FT1000 and Network Processing Engine, NPE, we implement a prototype system to support LabelCast. The Label table is implemented in NPE to schedule packet forwarding operations and packet output control instructions in data plane. While taking the advantage of the computing and storage resources within FT1000 connected through the system bus PCI-E, we design the Cast table to implement a protocol and status related packet processing services.

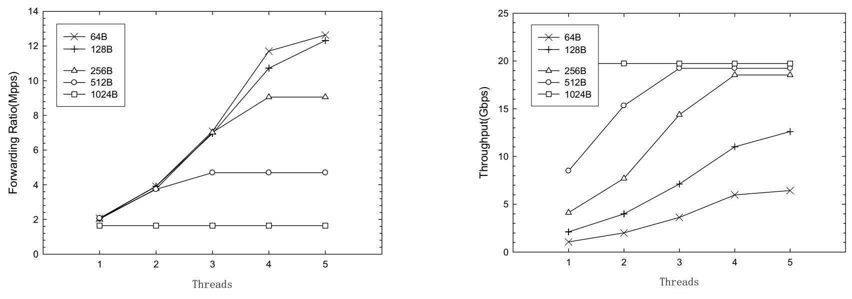
In the prototype system, the eXtended Linux container, XLC, is designed to implement the virtualization and allocation of computing and storage resources, which simplify the development of user-defined services. Based on a scalable resource container, researchers could design and implement the custom packet forwarding and processing services, and in the development of services the application could call the system components and libraries provided by the Labelcast prototype system to accelerate applications development.

5.2 PacketDirect IO Performance

Labelcast data plane provides the basic operations of packet forwarding and services of packets processing. Basic forwarding operations in data plane are implemented in NPE, to provide the light semantics actions of general packet processing, including look-up table, modifying the basic packet options and output control. Packets processing are implemented by the services running on FT1000 being tightly coupled with the NPE, to implement network protocol semantics or the state-related deep packet processing services in data path, such as the replacement of specific fields of packets, packets caching and the calculation of route based on the matrix of the network layer reaching information. Therefore, in the prototype system the system bus between NPE and FT1000 becomes the key to improve the system performance.

In the design and implementation of the prototype system, an efficient PacketDirect mechanism was designed and implemented to improve system throughput of the transport mechanism between NPE and FT1000. In experiments network tester transmitting packets to NPE via two 10Gbps ports, testing the PacketDirect IO performance of the prototype in the case of different number of threads and packets size, the results are shown in Fig.5. With the number of forwarding threads increasing, the performance of processing has a certain upgrade under different packets size; 1 thread implements line-rate forwarding of

1024-byte packets, 3 threads arrive at the line-rate forwarding of 512 bytes packets, 4 threads reach the highest forwarding rate, the maximum bandwidth of the tester's ports, 20Gbps, for 256 bytes packets. 64 bytes packets forwarding rate is relatively high, because in the case of the same internal packet buffer size, the number of cached small packets is relatively high. For the system throughput, under the same number of threads the larger packet size is, the higher the throughput is, for the system throughput is proportional to the length of the packet under the same packet forwarding rate. For the packet size of 1024 bytes single-thread could achieve the maximum rate of the tester, and for other packets sizes as the increasing of the number of threads (processing power), the system throughput increases. The results show that the PacketDirect mechanism of the prototype meets the performance requirements of the system throughput.



(a)Forwarding (b)Throughput

Fig. 5. PacketDirect IO Performance of Prototype

6 Conclusion

Labelcast provides and abstracts computing and storing resource within network devices for researches, and support running user-defined service based on extensible resource container, which provides a reliable and programmable data plane and support new network architectures. By dividing the data plane of the network into fast forwarding plane and intelligence service plane based on semantics of processing in networks, Label tables and Cast tables are proposed to abstract forwarding resources and the computation and storage resources separately. Action instructions based on fixed-length labels in Labelcast is easily to be implemented in hardware and service atoms of special processing could be dynamically extended compared with Openflow. NDN are designed on the NPE platform with Labelcast model, which support a large scale of parallel requests on the same content and packet by packet processing. The implementation of the prototype promotes the evolution of network architectures.

References

1. McKeown, N., Anderson, T., Balakrishnan, H., et al.: OpenFlow: Enabling innovation in campus networks. ACM SIGCOMM Computer Communication Review 38, 69–74 (2008)
2. Gude, N., Koponen, T., Pettit, J., et al.: Nox: towards an operating system for networks. ACM SIGCOMM Computer Communication Review 38, 105–110 (2008)
3. Van Jacobson, Smetters, D.K., Thornton, J.D., et al.: Networking Named Content. Communications of the ACM 55(1), 117–124 (2012)
4. Naous, J., Gibb, G., Bolouki, S., McKeown, N.: NetFPGA: reusable router architecture for experimental research. In: Proc. of ACM PRESTO, NY, USA (2008)
5. Argyraki, K., Baset, S.A., Chun, B.-G., et al.: Can software routers scale. In: Proc. of PRESTO, Seattle, USA (2008)
6. Egi, N., Greenhalgh, A., Handley, M., et al.: Towards high performance virtual routers on commodity hardware. In: Proc. of ACM CoNEXT, Madrid, Spain (2008)
7. Dobrescu, M., Argyraki, K., Iannaccone, G., Manesh, M.: Controlling Parallelism in a Multicore Software Router. In: Proceedings of the ACM PRESTO, Philadelphia, USA (2010)
8. Guo-Han, L., Rui, M., Yong-Qiang, X., Chuan-Xiong, G.: Using CPU as a Traffic Co-processing Unit in Commodity Switches. In: Proc. of the HotSDN, Helsinki, Finland (2012)
9. Gibb, G., Zeng, H., McKeown, N.: Outsourcing network functionality. In: Proc. of HotSDN (2012)
10. Walfish, M., Stribling, J., Krohn, M., Balakrishnan, H., Morris, R., Shenker, S.: Middleboxes no longer considered harmful. In: Proc. of OSDI, NY, USA (2004)
11. Sekar, V., Egi, N., Ratnasamy, S., Reiter, M., Shi, G.: Design and Implementation of a Consolidated Middlebox Architecture. In: Proc. of NSDI, NY, USA (2012)
12. Anand, A., Dogar, F., Han, D., et al.: XIA: An Architecture for an Evolvable and Trustworthy Internet. In: Proc. of the Hotnets, Cambridge, MA, USA (2011)
13. Ghodsi, A., Koponen, T., Rajahalme, J., et al.: Naming in Content-Oriented Architectures. In: Proc. of SIGCOMM ICN, Toronto, ON, Canada (2011)
14. Ghodsi, A., Koponen, T., Raghavan, B., et al.: Information-Centric Networking: Seeing the Forest for the Trees. In: Proc. of the Hotnets, Cambridge, MA, USA (2011)
15. OpenRouter: OpenFlow extension and implementation based on a commercial router. In: Proc. of the ICNP, Vancouver, BC, Canada (2011)
16. Kelly, J., Araujo, W., Banerjee, K.: Rapid Service Creation using the JUNOS SDK. ACM SIGCOMM Computer Communication Review 40(1), 56–60 (2010)
17. Hao, F., Kodialam, M., Song, H.: Fast dynamic multiple-set membership testing using combinatorial bloom filters. IEEE/ACM Transactions on Networking 20(1), 295–304 (2012)