Forwarding Programming in Protocol-Oblivious Instruction Set

Jingzhou Yu[*], Xiaozhong Wang[*], Jian Song[*], Yuanming Zheng[*], Haoyu Song[#]
Huawei Technologies CO., LTD
[*]Beijing, China [#]Silicon Valley, USA
yujingzhou@huawei.com

***Abstract*—Protocol-Oblivious Forwarding (POF) is an enhancement to OpenFlow-based SDN forwarding architecture. In this paper, we proposed a basic POF Flow Instruction Set (POF-FIS) which can be used to edit and forward packets as designed on the controller side. Working on the southbound interface of SDN, POF-FIS is independent of target platforms and northbound interfaces. To design the forwarding process on the controller side, users can take advantages of high-level programming languages or directly manipulate the POF-FIS using graphical or command-line user interface. High-speed execution of POF-FIS is very important for network elements, while eliminating the need of hard-coded protocol parsing and packet processing. We show that POF-FIS allows the forwarding capability of the flexible network elements to be fully released to achieve higher performance and more expressive forwarding behavior.**

Keywords—SDN, OpenFlow, POF, FIS

I. Introduction

Software-Defined Networking (SDN)[1] decouples the controller plane and the data plane. It brings possibility to move most network service specifications from network elements to controller, which can be deployed on separate servers. SDN enables a much more flexible network. OpenFlow[2], which prescribes a way to handle network elements from a controller and describes the communication protocol between controller and network elements, is a popular SDN southbound interface standard.

Numerous networking protocols exist today and many more emerge in the future. However, current network elements can only support a small number of protocols even with OpenFlow. Operators cannot implement services that use unsupported protocols on these network elements. The rapid development of information industry needs the network elements to be more flexible and extendable. Protocol-Oblivious Forwarding (POF)[3][4][5] is an enhancement to the current OpenFlow-based SDN forwarding architecture with the objective to improve the SDN programmability. POF enables SDN data plane to support new protocols and forwarding services without modifying any code on network elements. Bosshart et al. recently proposed a high-level language named P4[6]. P4 mainly focuses on programming protocol-independent packet processors.

The southbound interface between the SDN controller and network elements needs a unified flow instruction set. Any protocols, policies and services for today and tomorrow should be able to be realized using the combination of these flow instructions assembled by the SDN controller.

In this paper, we propose a basic flow instruction set, POF-FIS, which allows parsing, editing, and forwarding packets arbitrarily. POF-FIS can be used for the southbound interface communication between SDN controller and network elements.

II. POF Flow Instruction Set

Using POF to support new protocols, the operators only need to download some flow rules with associated instructions into the network elements. We call this instruction set POF-FIS. POF-FIS is an enhancement and extension of the instructions and actions defined in OpenFlow 1.x.

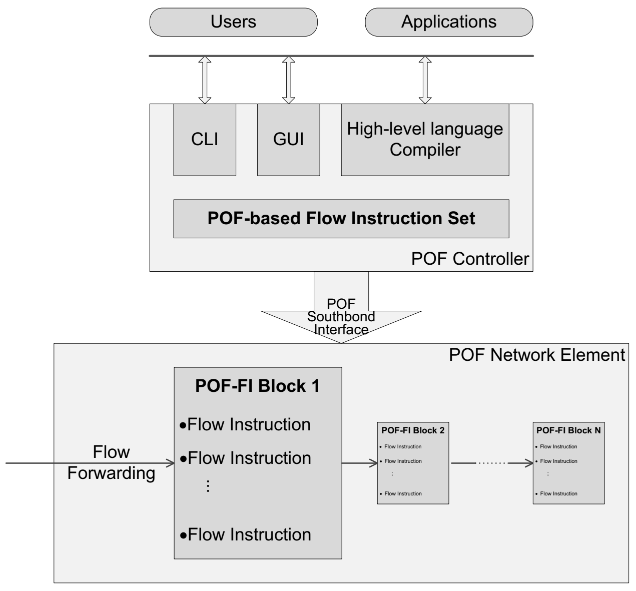


Fig. 1. The position of POF-based Flow Instruction Set in POF framework.

Fig. 1 shows the position of POF-FIS in the overall POF framework. In the POF network element, network flows are handled by flow instructions in the form of POF Flow

Instruction Blocks (POF-FIB). POF-FIBs are deployed by a controller through the POF southbound interface. All the flow instructions in POF-FIBs are defined in POF-FIS. The POF controller can use a Command-Line Interface (CLI), a Graphical User Interface (GUI), or a high-level programming language compiler as the northbound interfaces to users and applications. In order to design the whole forwarding application, users can organize the POF-FIS to POF-FIBs using the northbound interfaces, and then download the POF-FIBs to the network elements.

The followings are the main categories of POF-FIS based on functionality:

TABLE I. CATEGORIES OF POF-FIS.

Category	Instructions
EDITING	SET_FIELD, ADD_FIELD, DEL_FIELD, ALG, CALCULATE_CHECKSUM, SET_FIELD_UPDATE_CHECKSUM, INC_FIELD, DEC_FIELD, OR_FIELD, SRL_FIELD, SLL_FIELD, AND_FIELD, XOR_FIELD, NOR_FIELD, NOT_FIELD
FORWARDING	GOTO_TABLE, COUNTER, OUTPUT, GROUP, MOVE_PACKET_OFFSET, SET_PACKET_OFFSET
ENTRY	SET_TABLE_ENTRY, ADD_TABLE_ENTRY, DEL_TABLE_ENTRY
JUMP	BRANCH, COMPARE, JUMP
FLOW	SET_FLOW_METADATA, GET_FLOW_METADATA, ORDER_ENFORCE

A. Editing

This kind of instructions are used to edit the packet data. Packet data editing is the most important part during the forwarding process. Almost all of the protocol rules need to edit the packet data by writing, storing, copying, calculating and so on.

SET_FIELD sets any field in packet data to any value, e.g. destination MAC address in Ethernet header. ADD_FIELD and DEL_FIELD can insert or delete a custom field into or from the packet data, e.g. tunnel label. These are three of the most useful instructions.

Using these three instructions, users can define a totally custom field for Operation Administration and Maintenance (OAM). Fig. 2 shows the topology of users' own local area network with the custom OAM field. When a normal IPv4 packet comes into users' own network, the OAM field can be inserted into the packet data by ADD_FIELD instruction at the ingress gateway. Inside the local area network, users can write the OAM field with any value. The OAM field can be deleted at the egress gateway if the packet is going to be send out of the users' own local area network to be a normal IPv4 packet. The use of the OAM field is quite wide, including firewall, label switching, priority match, statistics and so on. It's depends on the users. Using editing species POF-FIS, users are also able to freely decide where the OAM field locates, how long the OAM field takes.

The remaining editing species instructions, such as ALG, INC_FIELD, DEC_FIELD, CALCULATE_CHECKSUM and some other logical operations, all are kinds of calculating of the packet data. ALG can do some arithmetic like Hash. Time-To-Live (TTL) field in the IPv4 header can be decreased by 1 using DEC_FIELD instruction during the IPv4 forwarding. All of the bit-level operations can be handled by the logical instructions.

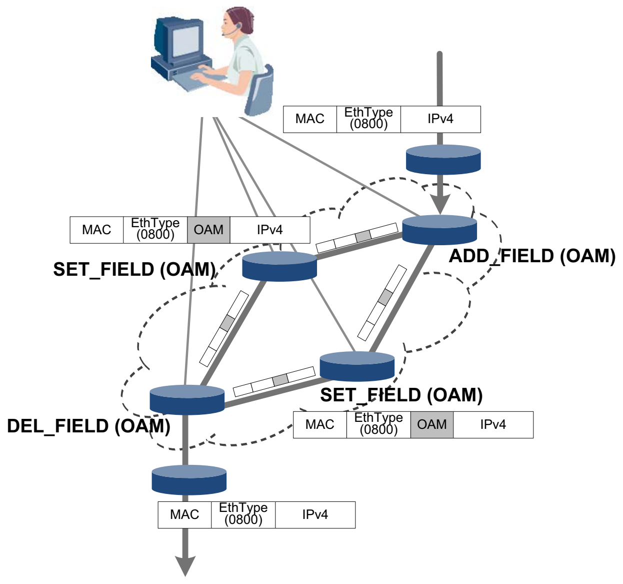


Fig. 2. Usage of POF-FIS in users' LAN with the custom OAM field.

B. Forwarding

This kind of instructions are used for packet forwarding. The whole forwarding for one packet in the network element might contain multiple processes. Users can separate them into some different flow tables according to the functionality, e.g. Layer-3 Parse Table and Layer-3 Encap Table. When the processing in a flow table is complete, users can execute GOTO_TABLE instruction to send the packet data from the previous flow table to the next flow table. This instruction is very similar to the OFPIT_GOTO_TABLE instruction in OpenFlow[2]. There is more match field information about the next flow table in GOTO_TABLE. COUNTER instruction can count the number of packets which have already been handled, also can count the total length in byte unit. OUTPUT instruction sends the packet data out of the network elements through the specified network port. At the meantime, users can decide where the packet to be send starts from, and whether sends the metadata before the packet data or not. GROUP instruction is used for multicast.

MOVE_PACKET_OFFSET and SET_PACKET_OFFSET can move the packets' base pointers forward or backward or to a specified location. These two instructions are very useful to handle the packets on different layers. For example, using SET_PACKET_OFFSET instruction, users can set the packet offset to 112-bit, which is the start position of IPv4 header (layer 3) in a normal Ethernet packet. No matter which layer the protocol or the business located, even though layer 7, users are always able to handle the packet data on this layer using these two packet offset related instructions. As long as users know the offset of this layer, the packet base pointer can be moved easily to the position of the packet data on this layer.

C. *Entry*

This kind of instructions let the network elements operate the flow entry by itself. SET_TABLE_ENTRY instruction sets the parameter and the match information of flow entries. ADD_TABLE_ENTRY and DEL_TABLE_ENTRY can insert a new flow entry into a flow table, or delete a existent flow entry from a flow table.

Operation of the flow entry by network element is very useful for the protocol rules which need to study the information about the network, e.g. topology, routing and neighbors. For instance, MAC learning is one of the most common function in the network switches. Maintain one flow table for mapping of MAC address and network port number, and check the source MAC address field of the packet data, which is received by one input network port. If this source MAC address doesn't exist in the mapping table, users can implement the ADD_TABLE_ETNRY to insert the mapping information about the source MAC address and the input port number. Route learning also can be realized in the same way.

D. *Others*

JUMP and FLOW these two species are advanced instructions. The JUMP instructions are able to alter the packet data processing procedure. The FLOW instructions provide some operations about the global status of the data flow.

III. CHARACTERISTICS OF POF-FIS

A. *Flexibility*

Profited from the POF technology, POF-FIS is oblivious of any protocol. POF denotes any protocol field with the following structure:

```
field {
    offset;
    length;
};
```

The "offset" is the field's start position relative to the current protocol header. The "length" is the field's length in bit unit. The following example Fig. 3 shows the Ethernet protocol header format.

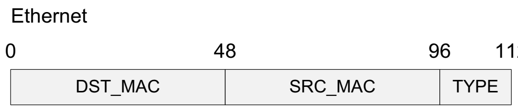


Fig. 3. Ethernet protocol header.

There are three fields: DST_MAC, SRC_MAC, and TYPE. They are denoted as follows:

- DST_MAC: {0,48}; /*offset is 0bit, length is 48bit.*/
- SRC_MAC:{48,48};/*offset is 48bit, length is 48bit.*/
- TYPE: {96, 16}; /* offset is 96bit, length is 16bit. */

Identify applicable sponsor/s here. If no sponsors, delete this text box (*sponsors*).

It is easy to see that any existing or new protocols can be denoted in the similar way. To support new services, users can assemble the POF-FIS freely to design the flow forwarding process. It brings full programmability to the network elements.

B. *Independence*

1) Independent of Northbound Interface

The southbound interface of a SDN controller is used for the communication between the controller and network elements. The northbound interface indicates the interaction between controller and users or applications. POF-FIS acts on controller, southbound interface, and network elements; therefore it is independent of the northbound interface. The controller can provide various types of northbound interface to users or applications. No matter what kind of northbound interface is used, the controller needs to translate the whole forwarding process into POF-FIBs, and then download them to the network elements.

Northbound interface independence allows much more flexibility and diversity of choice. POF-FIS does not stipulate any specific way on how to design service or packet forwarding process on controller. Users can freely choose one way to use according to their preference and demand. For example, users can directly manipulate POF-FIS through the graphical user interface or command-line interface of the controller, or they can take advantage of some high-level programming languages to define the forwarding process. This will be described in details in the next section.

2) Independent of Service and Application

POF-FIS is a generic instruction set which describes the basic packet processing primitives. It is not designed for any specific services or applications. Various services can be implemented through different combinations of the same set of instruction. Every instruction of POF-FIS can be used in the design of any services and the realization of any applications.

C. *Completeness*

1) Support All the Protocols

POF technology is protocol independent. Each field in the packet format can be described by offset and length. POF-FIS also uses this method to identify and manipulate any field in any packet format. In other words, any field in any packet protocol, whether existing or new, can be an object manipulated by POF-FIS.

We provides two examples to show this point. The Ethernet protocol uses the Ether Type field to identify the layer 3 protocol. The IPv4 protocol makes the forwarding decision based on the destination IP address. These protocols define the way to handle the packets including parsing, editing and forwarding, which are all covered by POF-FIS. There are five types of instructions in POF-FIS: EDIT, FORWARDING, JUMP, ENTRY, FLOW, which mean to cover all the possible operations on packets. For instance, EDIT includes all packet editing related instructions we believe ever needed, and FORWARDING and JUMP cover all the common packet processing related instructions.

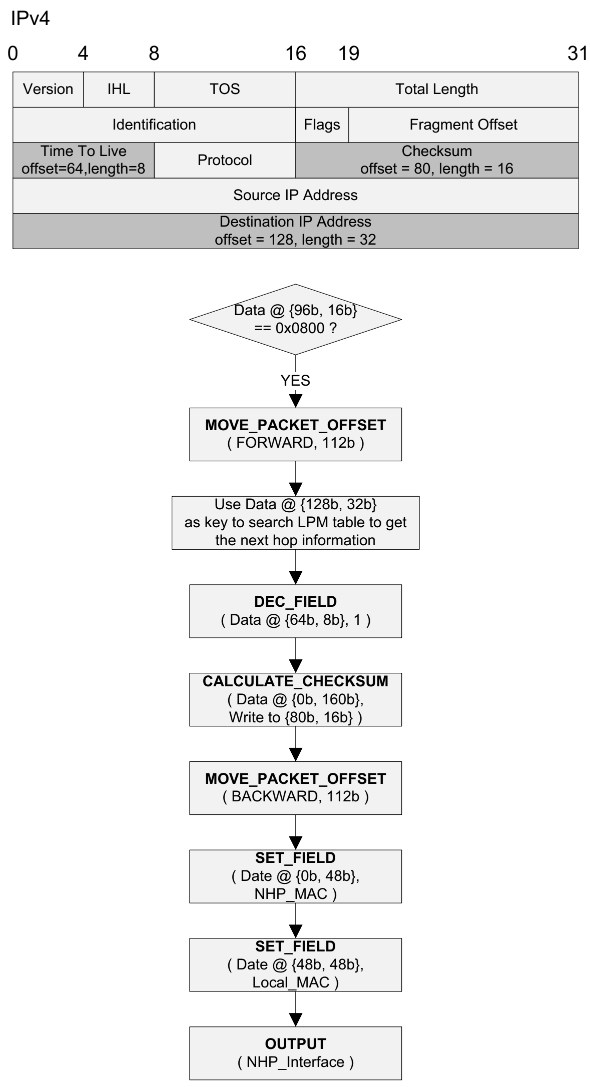

IPv4
0
4
8
16
19
31
Version
IHL
TOS
Total Length
Identification
Flags
Fragment Offset
Time To Live offset=64,length=8
Protocol
Checksum offset = 80, length = 16
Source IP Address
Destination IP Address offset = 128, length = 32
Data @ {96b, 16b} == 0x0800 ?
YES
MOVE_PACKET_OFFSET (FORWARD, 112b)
Use Data @ {128b, 32b} as key to search LPM table to get the next hop information
DEC_FIELD (Data @ {64b, 8b}, 1)
CALCULATE_CHECKSUM (Data @ {0b, 160b}, Write to {80b, 16b})
MOVE_PACKET_OFFSET (BACKWARD, 112b)
SET_FIELD (Date @ {0b, 48b}, NHP_MAC)
SET_FIELD (Date @ {48b, 48b}, Local_MAC)
OUTPUT (NHP_Interface)

Fig. 4. How to handle the Ethernet and IPv4.

Fig. 4 shows the steps to handle the Ethernet and IPv4.

- Check the packet data with 96-bit offset and 16-bit length (i.e. EtherType field). If it equals to 0x0800, go to the next step.
- Execute the MOVE_PACKET_OFFSET instruction to move the packet base pointer forward with 112-bit to the IPv4 header.
- Use the packet data with 128-bit offset and 32-bit length (DIP field) as a match key to search the LPM table in order to get the next hop information.
- Execute the DEC_FIELD instruction to decrease the value of the packet data with 64-bit offset and 8bit length (TTL field) by 1.
- Execute the CALCULATE_CHECKSUM instruction to calculate the checksum of the packet data with 0-bit offset and 160-bit length (IP header), and write the checksum result to the packet data with 80-bit offset and 16-bit length (Checksum field).
- Execute the MOVE_PACKET_OFFSET instruction to move the packet base pointer backward with 112-bit to the Ethernet header.
- Execute the SET_FIELD instruction to set the value of packet data with 0-bit offset and 48-bit length (DMAC field) to the MAC address of the next hop.
- Execute the SET_FIELD instruction again to set the value of packet data with 48-bit offset and 48-bit length (SMAC field) to the local MAC address.
- Execute the OUTPUT instruction to send the packet out through the interface connected to the next hop. The simple IPv4 forwarding procedure is finished.

Four flow instructions are used for the IPv4 packet forwarding. The operands and the parameters of these instructions are described by a universal format with offset and length of the field in the packet, instead of using hard coded fields for predefined protocol formats.

Since POF-FIS is designed to cover all the possible operations on packets and the object to be handled can be any field in any part of a packet, it is capable of realizing any packet protocols even those nonexistent today.

2) *Designable Instruction Set without Using other Tools*

Users can take advantage of high-level programming language to program the services. The program can be translated or compiled into POF-FIBs by some tools, to complete the design of the whole forwarding process. This is certainly convenient but not the only way. If one has sufficient understanding about the networks and is quite familiar with POF principle, he or she is able to design the whole forwarding process by directly assembling the POF flow instructions on the controller without any compiler or interpreter.

The compilers or interpreter may not be intelligent enough to achieve the optimal POF instructions which are efficient and support best network element performance. If this is the case, users are better to directly manipulate the POF flow instructions without using other tools to achieve better device adaptation and greater code efficiency.

IV. Realization

A. Direct Manipulation of POF-FIS

Above the POF interface, any network forwarding application needs to be converted to the POF flow instructions first. One way to do it is to directly use Graphical User Interface (GUI) or Command-Line Interface (CLI) for interactive data plane programming. This is like programming in assembly language. Although needing to handle flow level details, this method is fast and direct. The GUI/CLI can be used to handle fast updates and can also be used to directly download compiled applications to network elements. Reference [4] have already implemented an open source GUI and CLI to support this programming method.

1) Graphical User Interface

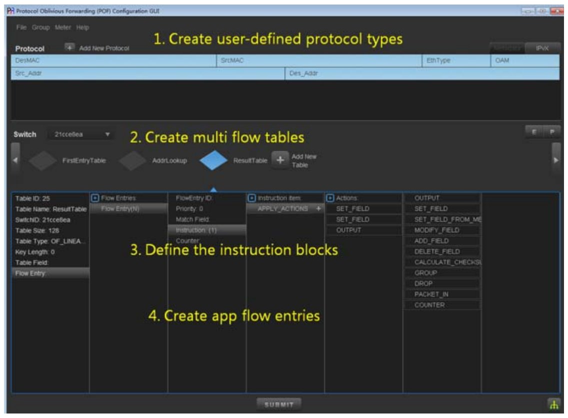

1. Create user-defined protocol types
2. Create multi flow tables
3. Define the instruction blocks
4. Create app flow entries
SUBMIT

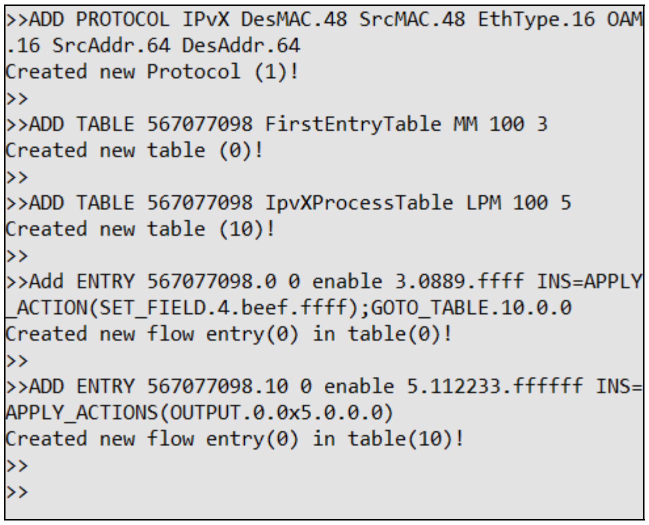
Fig. 5. GUI of POF Controller.

Fig. 5 shows the GUI of POF controller. There are four parts in the GUI.

- Create user-defined protocol types. Any packet format can be defined here using offset and length of each field.
- Create multiple flow tables for the forwarding process. These flow tables can be designed to operate in parallel or pipeline.
- Create flow forwarding entries for each flow table. Every flow entry includes match and instruction block.
- Add POF flow instructions to each flow entry.

All the actions on the GUI can be selected and parameterized. Users just need to click the mouse buttons with a few keyboard inputs to define the whole forwarding process.

2) Command-Line Interface



```
>>ADD PROTOCOL IPvX DesMAC.48 SrcMAC.48 EthType.16 OAM
.16 SrcAddr.64 DesAddr.64
Created new Protocol (1)!
>>
>>ADD TABLE 567077098 FirstEntryTable MM 100 3
Created new table (0)!
>>
>>ADD TABLE 567077098 IpvXProcessTable LPM 100 5
Created new table (10)!
>>
>>Add ENTRY 567077098.0 0 enable 3.0889.ffff INS=APPLY
_ACTION(SET_FIELD.4.beef.ffff);GOTO_TABLE.10.0.0
Created new flow entry(0) in table(0)!
>>
>>ADD ENTRY 567077098.10 0 enable 5.112233.ffffff INS=
APPLY_ACTIONS(OUTPUT.0.0x5.0.0.0)
Created new flow entry(0) in table(10)!
>>
>>
```

Fig. 6. CLI of POF Controller.

Fig. 6 shows the CLI of the POF controller. There are four commands illustrated in the screenshot. ADD PROTOCOL, ADD TABLE, and ADD ENTRY are used for the design of forwarding process. Users can check the detail information of tables and entries using DISPLAY TABLE command.

B. High-Level Language Description

Another method to convert the services or applications to POF flow instructions is to compile or interpret programs in some high-level languages. The high-level language provides another layer of abstraction that supports modularity and composition[7]. With the help of a high-level language, the developers can focus on what the application really wants to achieve rather than dealing with particular network element architecture and conducting tedious and error-prone flow-level match-action manipulations. Quite a few such languages have been proposed[6][7][8]. The following examples briefly show the source code using P4, C, and Java.

1) P4[6]

P4 is a new high-level language for programming protocol-independent packet processors. The following is an example to design the forwarding process using P4.

```
header   mTag {
    fields {
        up1 :           8;
        up2 :           8;
        down1 :         8;
        down2 :         8;
        ethertype :     16;
    }
}

action add_mTag(up1, up2, down1, down2, egr_spec) {
    add_header(mTag);
    // Copy VLAN ethertype to mTag
    copy_field(mTag.ethertype, vlan.ethertype);
    // Set VLAN's ethertype to signal mTag
    set_field(vlan.ethertype, 0xaaaa);
    set_field(mTag.up1, up1);
    set_field(mTag.up2, up2);
    set_field(mTag.down1, down1);
    set_field(mTag.down2, down2);

    // Set the destination egress port as well
    set_field(metadata.egress_spec, egr_spec);
}
```

2) C

C is one of the widely used programming languages of all time[10]. The following is an example to design the forwarding process using C.

```
struct   Metadata_L3 {
    uint8       L3Stake;        // L3 Offset
    uint16      VpnID;          // VPN ID
    uint16      RealLength;     // Packet Length
    uint16      SqID;           // QOS Queue ID
};
```

```
struct     Table_Portinfo {
      uint16          VpnID;                // VPN ID
      uint16          SqID;                 // QOS Queue ID
};

struct     IPV4_HEADER_S {
      uint4           Version;
      uint4           HeaderLength;
      union {
            uint8     TOS;
            uint6     DSCP;
            uint3     Precedence;
      };

      uint16          TotalLength;
      uint16          FragReAssemID;
      IPV4_FRAG_HWORD_S             FragHWord;
      IPV4_TTL_PROT_HWORD_S         TtlProtWord;
      uint16          Checksum;
      uint32          SIP;
      uint32          DIP;
};

(Metadata_L3 *) p_metadata;

(Table_Portinfo *) p_table;
p_metatada->VpnID = p_table->VpnID;
p_ipheader = p_packet + 14;
Goto_Table( TableID, p_metadata->VpnID, p_ipheader->DIP );
```

3) Java

The java programming language is a general-purpose, concurrent, class-based, object-oriented language[11]. The following is an example to design the forwarding process using Java.

```
class      Metadata_L3 {
      short           CurPos;               // Current Position
      short           VpnID;                // VPN ID
      short           RealLength;           // Packet Length
      short           SqID;                 // QOS Queue ID
};

class      Table_Portinfo {
      short           VpnID;                // VPN ID
      short           SqID;                 // QOS Queue ID
};
class      IPv4Header {
      byte            version;
      byte            headerLength;
      byte            diffServ;
      short           totalLength;
      short           id;
      byte            flags;
      short           fragmentOffset;
      byte            ttl;
      byte            protocol;
      short           checksum;
      int             srcIPAddress;
      int             desIPAddress;
};

public     void process () {
      metadataL3.VpnID = portInfo.VpnID;
      metadataL3.CurPos += 14;
      keyList.add ( metadataL3.VpnID );
      keyList.add ( ipv4Header.desIPAddress );
      OFAction.gotoTable ( TableID, keyList );
}
```

Each of these three high-level languages could be the users' choice according to the users' requirement and the availability of compiling tools.

V. Conclusions

POF-FIS is flexible to implement protocol rules and deploy services rapidly, whether existing or new. As the major southbound interface component in SDN, POF-FIS is independent of target platform, northbound interface, and the high-level programming language. Users can choose to use P4, C, Java, and any other high-level languages to design the forwarding process. Alternatively, users can also directly manipulate the POF-FIS to assemble the whole forwarding process.

We believe POF-FIS can become an important part of the OpenFlow 2.0 standard. The generic POF-FIS significantly improves the flexibility of network elements. POF-FIS allows the forwarding capability of the flexible network elements to be fully released to achieve higher performance and more expressive forwarding behavior.

References

[1] ONF. (2012, April 13). *Software-Defined Networking: The New Norm for Networks* [Online]. Available: https://www.opennetworking.org

[2] ONF. (2013, Oct. 15). *OpenFlow Switch Specification (1.4.0)* [Online]. Available: https://www.opennetworking.org

[3] H. Song, "Protocol-oblivious forwarding: Unleash the power of SDN through a future-proof forwarding plane," in SIGCOMM HotSDN Workshop, Aug. 2013.

[4] (2013). *Protocol Oblivious Forwarding* [Online]. Available: http://www.poforwarding.org

[5] H. Song, J. Gong, H. Chen, J. Dustzadeh, "Unified POF programming for diversified SDN data plane," in Unpublished, 2014.

[6] P. Bosshart, D. Daly, M. Izzard, N. McKeown, J. Rexford, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "Programming Protocol Independent Packet Processors," in Unpublished, 2013.

[7] N. Foster, M. Freedman, A. Guha, R. Harrison, N. P. Katta, C. Monsanto, J. Reich, M. Reitblatt, J. Rexford, C. Schlesinger, A. Story, and D. Walker, "Languages for Software Defined Networks, " IEEE Communication Magazine, vol. 51, pp. 128-134. February 2013.

[8] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker, "Frenetic: A Network Programming Language," in ACM SIGPLAN ICFP, 2011.

[9] A. Voellmy and P. Hudak, "Nettle: Functional Reactive Programming of OpenFlow Networks," in PADL, 2011.

[10] (2014, July). *TIOBE programming community index* [Online]. Available: http://www.tiobe.com

[11] J. Gosling, B. Joy, G. Steele, G. Brancha and A. Buckley. (2014, March). *The Java language Specification (Java SE 8 Edition)* [Online]. Available: http://www.oracle.com