A Flexible Forwarding System for Experiments in Information-Centric Networking

FAN YANG\(^1\), (Member, IEEE), TIANYUAN NIU\(^2\), (Member, IEEE)

\(^1\)State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China
\(^2\)Purple Mountain Laboratories, Nanjing 211111, China

Corresponding author: Tianyuan Niu (e-mail: arthurniu@bupt.edu.cn).

This work was supported in part by the National Key Research and Development Program of China (No. 2020YFB1804603).

ABSTRACT Information-Centric Networking (ICN) has drawn considerable attention as a new network architecture recently. However, the lack of experimental environments brings about the inconvenience of evolving ICN. As forwarding devices need to be updated once the protocols evolve, researchers spend a large amount of time developing forwarding engines rather than improving protocols. In this paper, we propose a flexible forwarding system called Content-oriented Experimental Forwarding System (CEFS) for experiments in ICN, aiming at alleviating the inconvenience of evolving ICN protocols. Combined with Software Defined Networking, CEFS uses southbound messages to manage the protocol, routing, and content in the forwarding system. Moreover, we design a generic packet processing module to forward ICN packets in CEFS, using Open vSwitch as the basis to implement it and verifying its feasibility. Evaluation data show that CEFS is better than Named Data Networking Forwarding Daemon and Protocol-Independent Packet Processors behavioral-model. The applications show that researchers can use simple messages to manage the protocol, routing, and content of ICN without updating the forwarding system, which significantly improves the efficiency of evolving ICN protocols.

INDEX TERMS Forwarding system, Information-Centric Networking, Software Defined Networking

I. INTRODUCTION

INFORMATION-Centric Networking (ICN) [1] is proposed as a new network architecture to overcome the shortcomings of host-to-host communication, which has attracted attention from industrial and academic communities.

Named Data Networking (NDN) [2] and Content-Centric Networking (CCN) [3] are relatively complete designs of ICN, but protocols in NDN and CCN are still evolving. Thus, the researchers need to conduct lots of experiments for feasibility verification. To support new protocols, it would take much time to patch forwarding devices, which results in the slow evolution of ICN protocols and makes the actual deployment of ICN unfeasible.

Named Data Networking Forwarding Daemon (NFD) [4], a network forwarder, is a core component of the NDN Platform. It is implemented and evolved along with the NDN protocol. When the NDN protocol evolves, researchers spend much time on the evolution of NFD, which makes the process extremely inefficient.

P4 is a high-level language for programming protocol-independent packet processors [5]. Although P4 is not designed for ICN, it provides a way to solve the problems mentioned above. P4 programs specify how a network device processes packets according to the requirements of users. This method can avoid re-developing packet processors when the protocols evolve, so it can effectively reduce the period of evolving protocol.

However, the format of ICN packet header is very complicated. In ICN architectures such as NDN and CCN, the packet header is encoded as a Type-Length-Value (TLV) [6, 7] format, so it is difficult to write P4 programs for ICN. Besides, the hardware device of P4 is expensive. It is not worthwhile to deploy the experimental environment through hardware devices. Moreover, the performance of software devices is poor and cannot meet the requirements of some ICN experiments.

In this paper, we propose a flexible forwarding system called Content-oriented Experimental Forwarding System...
(CEFS) for experiments in ICN to solve the above problems. We borrow the idea that separating the control plane from the data plane makes the forwarding devices controllable from Software-Defined Networking (SDN) [8]. One critical point for SDN is the southbound protocol between the control plane and the data plane. At present, OpenFlow [8], one of the mainstream southbound protocols in SDN, is designed for the TCP/IP network. Although this protocol has been updated continuously, it cannot support the TLV format in ICN. Besides, it cannot update protocols in the forwarding devices, restricting the development of the network. However, its idea is generic and can be applied to ICN, which can simplify the process of evolving protocols.

CEFS is a flexible forwarding system that can manage the protocol, routing, and content in ICN. We extended and modified the southbound protocol of SDN to support the ICN architecture, and used particular southbound messages to transfer the information of protocol, routing, and content to the forwarding system. Considering the specific features of ICN, we designed a generic packet processing module to forward packets in ICN and added some operations for caching and distributing content to the system. It makes ICN experimental deployment more flexible and efficient.

The main contributions of our paper are summarized as follows:

1) We design the southbound messages for ICN, which transfer the information of protocol, routing, and content to the forwarding system.

2) We design a generic forwarding system to forward arbitrary packets in ICN and add some operations for caching and distributing content to the system.

3) We modify Open vSwitch (OVS) [9] to implement the forwarding system and evaluate the feasibility and efficiency of it.

We introduce the background in section II and discuss the related work in section III. In section IV, we describe the design of CEFS and present how to support the protocol, routing, and content management of ICN. Then, we evaluate the forwarding performance of CEFS and demonstrate many applications of ICN in section V. Finally, we conclude this paper in section VI.

II. BACKGROUND

In traditional host-to-host communication, users must get content from a specified IP address. As shown in Fig. 1(a), the desired content is completely coupled with the IP address, so the user needs to send the request to a specified host to get content. However, the content and address are decoupled in ICN. Users only need to care about what content they want rather than where to get it. Forwarding devices use content names as unique identifiers in place of IP address for data transfer. As shown in Fig. 1(b), the user sends a request message containing the name of a desired content, and any forwarding devices that possess the content can reply to the request directly. Users can get the content from the nearest device, thus significantly reducing the transmission time and utilizing the network bandwidth completely.

Numerous studies and surveys [10–12] consider that name parsing, caching and forwarding are the most concerning techniques in ICN. The characteristics of ICN require two capabilities for the ICN forwarding devices. First, using TLV format to encode a content name in ICN protocols means that the forwarding devices should process packet headers with variable length. Second, the forwarding devices should store and distribute content as well, so that a user can get the content they need from the nearest cached device.

The traditional forwarding devices in ICN are coupled with protocols. Although both NDN and CCN protocols adopt the TLV format, their forwarding devices are different and not compatible with any other ICN protocols. The packet processing in traditional forwarding devices is restricted, and it is urgent to develop a generic packet processor for ICN.

The principal packet processing in forwarding devices is shown in Fig. 2. First, after receiving a packet, the parser module extracts the key from the packet header. Then, the flow table module looks up a matched entry in flow tables through the key. Finally, the action set module executes the corresponding actions for the matched entry.

Currently, the parser and flow table modules in traditional forwarding devices are designed for a specific protocol. When this protocol evolves, researchers must redevelop these two modules to match the updated protocol. However, ICN is still a long-term evolutionary network architecture, and NDN and CCN protocols are constantly updated. Each protocol modification means that the parser and flow table modules in current forwarding devices need to be updated, which consumes a lot of time and resources. Therefore, we hope to propose a flexible ICN forwarding system whose parser and flow table modules are generic for all protocols and the action set module supports caching and distributing content.
It meets two basic requirements mentioned above and can be used for any ICN protocols, present or future designed.

III. RELATED WORK

There are enormous works [10–15] trying to enhance the performance of ICN forwarding, deploy ICN architectures on real hardware, or solve the bottleneck problem of ICN. In [13], the authors propose a self-learning based routing algorithm to utilize multiple routes fully and decrease deployment complexity. In [16], a hierarchical ICN caching system is proposed and facilitate the incremental deployment of an ICN structure in the traditional network. These two examples show that ICN’s research confronts the dilemma: it is not fully compatible with existing networks, which means it is difficult for ICN to evolve rapidly when confined with traditional network architecture. Therefore, a flexible system should be designed for ICN researchers, with no impediment from existing networks.

The combination of ICN and SDN has intrigued academia and industry. Several articles show that both of them will benefit from such a combination. In [17], the authors propose a method that the ICN fields are integrated into the option field of IP header to forward packets of ICN in the traditional network. In [18], the authors further use the OpenFlow switch to forward the integrated packets and support complex operations such as storage, routing, and security. They discuss the long-term and short-term approaches to combine SDN and ICN, and implement the short-term approach. In the long-term approach, they illustrate the necessity of supporting non-integrated ICN packets.

CCNx [19] is one of the typical implementations of CCN. In [20], the authors suggest using wrapper combined with CCNx daemon and OpenFlow switch. The wrapper is connected to their ports and maps the name of the content that parsed in the CCNx daemon to the fields that the OpenFlow switches can recognize.

In [21], the authors propose a way that uses the second communication channel in parallel with OpenFlow to communicate between the ICN forwarding modules and the SDN controller. The ICN-aware forwarding modules send the ICN packets to a specific application in the controller. Through the analysis result, the application configures all the ICN-aware forwarding modules and common forwarding modules in the flow path. In the common forwarding modules, the Ethernet and IP addresses of packets are used as matching fields. It is similar to create an IP tunnel for each ICN stream.

The methods mentioned above all support ICN based on the TCP/IP network. These designs may restrict the development of ICN. As described in [18], it is necessary to support non-integrated ICN. Therefore, our forwarding system is designed for non-integrated ICN. Besides, the researches mentioned above are designed for a specific ICN implementation, but we aim to provide a flexible and programmable ICN experimental network.

Authors in [22] describe a method using Berkeley Packet Filters (BPF) to forward arbitrary packets. They use Open-Flow Extensible Match (OXM) to carry a compiled BPF program. BPF can execute the program and return the matching fields that SDN switches can recognize. Authors in [23] introduce several modifications to overcome the shortcomings of an approach outlined in [22]. Instead of classic BPF, they use Extended BPF (eBPF) to deploy ICN. Meanwhile, they use the experimental messages and matching fields in OpenFlow to transfer eBPF program, which can be directly parameterized. However, forwarding packets by this switch depends on the implementation of eBPF, and it is arduous to develop eBPF programs for the intricate format of ICN packets.

On the contrary, we can configure our forwarding system with simple messages. Also, [23] is aimed to solve the problem of forwarding packets between different ICN domains in the long distance, which neglects the storage and distribution of content in ICN. Our purpose is to design a system that forwards packets in the internal network of ICN, which does not only support forwarding arbitrary packets but also provide operations with content.

As a new high-level language, P4 makes the data plane programmable. P4 provides a P4 compiler and a software device called P4 behavioral-model (BMv2). The P4 language can specify the format of packets and the FIB (Forwarding Information Base) lookup process, and then the P4 compiler generates the executable file, which enables BMv2 to forward packets. In [24], the authors introduce a method that BMv2 is used to forward packets in NDN. However, the limitation of the P4 language makes it hard to be implemented. Moreover, BMv2 cannot support operations such as the storage of content. Although P4 is increasingly developed, it can be foreseen that P4 will take a long time to improve itself.

A high-speed NDN forwarding architecture NDN-DPDK [14] is a competitor that also provides high performance for NDN forwarding. NDN-DPDK is the first complete implementation of a high-speed NDN forwarding system on real hardware by the authors’ claim, which achieved a throughput of over 100 Gbps and ran on commodity x86 hardware. Compared with our work, NDN-DPDK focuses on developing a fast prefix matching for NDN Interest and Data packet, which accelerates the process of PIT and Content Store lookup. However, our proposal focuses on providing a flexible forwarding system for the users and researchers who want to develop a new protocol like ICN, or verify the feasibility of a new algorithm deployed in ICN.

As discussed above, there are many methods and architectures proposed to improve the forwarding performance of ICN, and each of them focuses on different aspects to either enhance the performance or fix the critical problem. However, our work proposes a novel architecture aiming at offering a generic forwarding system, designed for anyone who wants to test and verify their own ICN protocols or algorithms. The main contribution of our work is providing this flexible forwarding system with relatively high performance, and the capability of supporting any ICN-like protocols.
IV. DESIGN OF CEFS

SDN brings us inspiration. In SDN, routes in forwarding systems can be managed through the interaction between separated control and data plane. Combined with SDN, our forwarding system can manage the protocol, routing and content in ICN by extending this interaction. We design an OpenFlow-based southbound protocol for ICN to satisfy content-based network requirements, and decouple the control and data plane by dividing functions in ICN and dispatching them into different planes. Based on this new southbound protocol, we propose a generic forwarding system, which can forward arbitrary packets with TLV format headers including the protocol, routing, and content information transferred from the controller.

The features converge to an architecture of CEFS, which is summarized in Fig. 3. It consists of a data plane packet processing module (DPM) and a southbound protocol packet processing module (SPM).

In addition to these three modules previously mentioned in the traditional packet processing, DPM includes the protocol module and the content cache module. The parser and flow table modules can process arbitrary packets through protocols stored in the protocol module. The action set module includes some actions such as forwarding, caching and distributing of content, which can meet the needs of ICN experiments.

SPM is designed for parsing and processing southbound messages from the controller. The existing southbound protocol, OpenFlow, is relatively complete, but its design is only appropriate for TCP/IP networks and cannot be used in ICN. For example, the FLOW_MOD message is one of the fundamental messages in OpenFlow, which can manage the routing in traditional forwarding systems. However, it can only match the fields belonging to TCP/IP stack. Moreover, OpenFlow lacks some messages to manage protocols and content in the forwarding system of ICN. Therefore, we retain some basic messages, such as the message that establishes the connection. Then we modify the FLOW_MOD message, and add the PROTOCOL_MOD and CONTENT_MOD messages, to manage the protocol, routing, and content in the forwarding system.

A. PROTOCOL MANAGEMENT

In traditional forwarding systems, protocol information is predefined and cannot be modified. During the processing of packets, forwarding systems can only parse packet headers according to some fixed protocols, which hinders the evolution of ICN protocols. Therefore, it is necessary to support protocol management in our system.

We design the PROTOCOL_MOD southbound message to modify ICN protocols in CEFS. The controller uses this message to send the newest protocol information to the forwarding system, while the protocol parser in CEFS can add, update and delete protocols by parsing this message. All protocols are stored in the protocol module and can be modified all the time.

The significant handicap for implementing the protocol management is how to efficiently define ICN protocols in the PROTOCOL_MOD southbound message. Each ICN packet header is encoded into a TLV format in general, such as NDN and CCN packets. TYPE numbers in TLV format are used to distinguish different levels of TLV, and these assigned numbers that belong to different protocols are distinct from each other. Therefore, the TYPE numbers considerably contain a quantity of information about ICN protocols, which are important to the forwarding system. Besides, an ICN packet comprises many levels of TLV inside the outermost top-level. Some TLV levels may contain subsidiary TLV levels and each subsidiary level may also be further nested [6]. As shown in Fig. 4(b), the VALUE of TLV top-level contains two sub-levels.

Therefore, an ICN protocol specifies the assigned TYPE numbers and defines the hierarchical relationship between these numbers. Because of the nested TLV levels, the relationship of different TYPE numbers in a protocol can be recognized as multiple binary trees or a forest. The number of trees in the forest depends on how many TLV top-levels are in the protocol. For example, there are two trees in an NDN forest because the TYPE numbers of TLV top-levels are 0x05(Interest) and 0x06(Data), respectively. Shown in Fig. 4(a), the Interest tree indicates that the TYPE numbers of the subsidiary TLV levels under the top-level must be 0x07, 0x21 and 0x1e.

Moreover, both TYPE and LENGTH may have a variable
size in different protocols. Considering these features as described above, the PROTOCOL_MOD message contains the protocol forest and the size of TYPE and LENGTH, with which a protocol can be defined. The protocol information embodied in the PROTOCOL_MOD message is stored in the protocol module and used in the processing of packets.

The parsing algorithm of PROTOCOL_MOD message is shown in Algorithm 1. When receiving a PROTOCOL_MOD message, the forwarding system determines what action will be executed according to mod_id, which is a field in the PROTOCOL_MOD message header.

When the operation is Add, the parsing module of PROTOCOL_MOD will create a protocol node to store protocols and fields in the PROTOCOL_MOD header. Then, the module will parse each protocol tree in the PROTOCOL_MOD message and restore the tree structure based on pre-order traversal and marker bits. After parsing the message, the parsing module will distinguish if the number of protocol trees conforms with the PROTOCOL_MOD message header. If not, the protocol trees are recognized as abnormal and deleted, an ERROR message will be reported concurrently. If the protocol is correct, the created protocol tree will be stored in the protocol module. If there are other protocols in the protocol module, the existing protocols will be deleted and replaced by new ones.

When the operation is Delete, a different decision will be made depending on whether the message matches the corresponding information or not. If the message does not match other information, the protocol in the protocol module is deleted directly; if it matches the protocol information, the next move depends on whether the protocol information in the message is consistent with its counterpart in the module. First, some fields like child number and length are distinguished if they are consistent with their counterparts in the protocol node. Second, the module will determine whether all subsequent protocol trees are consistent. If all the information is consistent, the protocols currently stored in the forwarding system will be deleted, otherwise, an ERROR message will be reported.

### B. ROUTING MANAGEMENT

Inspired by SDN, we also use the FLOW_MOD message to control the routing in the CEFS effectively. The FLOW_MOD message can carry multiple entries, and each entry consists of some matching fields and actions. However, the matching fields and supported actions in OpenFlow are all designed for the TCP/IP network and cannot be used in ICN. For this reason, we re-design the flow entry in the FLOW_MOD message.

| ID | Match Fields |
|----|--------------|
| 1  | (type=0x05),(pre=0x05,type=0x07),(pre=0x05,type=0x21,value=0x4567),(pre=0x07,type=0x08,value=0x1234) |

**TABLE 1: Example of match fields**

**Algorithm 1** PROTOCOL_MOD Parsing Algorithm

```
Require: PROTOCOL_MOD message P_M;
if Mod_id = 0 then
    // Add a protocol;
    Add a protocol node;
    while Protocol p_i exists in P_M do
        ConstructProtocolTree(p_i);
        p_i ← p_i+1
    end while
    if CheckProtocol(P_M) ≠ 0 then
        // Check if protocol trees newly constructed are normal;
        // IF the function CheckProtocol does not return 0,
        // protocol trees are abnormal;
        Delete protocol trees;
        Report error;
    end if
else
    // Delete a protocol;
    if Protocols exist in P_M then
        if CheckConsistency(protocolnode, P_M) = 0 then
            if CheckConsistency(protocoltrees, P_M) = 0 then
                Delete protocols;
            else
                Report error;
            end if
        else
            Report error;
        end if
    else
        Delete protocols;
    end if
end if
```

The relationship of TYPE numbers in each packet must strictly conform to the protocol definition, but not all the assigned numbers are contained in the packet. Therefore, the relationship of TYPE numbers in a packet header are like a sub-tree subordinating to one of the protocol trees mentioned above. The lowest level VALUE of TLV, whose TYPE number is the leaf node of a binary tree, is suitable for matching, while others only include subsidiary TLV levels. As shown in Fig. 4(b), it is an available packet header that conforms to the protocol definition shown in Fig. 4(a). The binary tree of this packet is shown in Fig. 4(c), which is a sub-tree of the binary tree in Fig. 4(a). The matching fields of the flow entry are similar to the packet tree for being matched efficiently. The matching fields include the full path of the protocol sequence in the binary tree.
binary tree and the lowest level VALUE, as shown in Table 1. When a flow entry is matched, the actions of this entry will be executed. The supported actions of entries are shown in Table 2. They can meet the requirements of various ICN experiments.

The flow parser module can parse the FLOW_MOD message to add, update and delete entries. These entries are stored in the flow table and used in the lookup process. In addition, the idea of multiple tables in SDN can significantly reduce the number of entries, so each entry contains a table number that informs the flow parser to address the entry in the correct table.

The parsing algorithm of FLOW_MOD message is shown in Algorithm 2. This parsing module does not finish working until all the table entries are processed successfully. For an entry, CEFS will determine the exact operation on account of the command field in the FLOW_MOD header.

When the operation is Add an entry, the basic structure of this entry is generated by the information contained in the FLOW_MOD header. Then, the ICN_match and action fields are parsed to compose the intact entry. After parsing, CEFS will determine how to insert this entry to a table.

When the operation is Delete an entry, the table number and priority in the FLOW_MOD header will be stored temporarily to select the entry that needs to be deleted. Moreover, if the ICN_match field exists in the FLOW_MOD header, the field will be split to generate an temporary structure to lookup the entry that needs to be deleted for convenience.

### C. CONTENT MANAGEMENT

Content management is indispensable in ICN experiments. The caching strategies are diverse in all kinds of ICN experiments. To support different strategies, we provide two operations for managing the content. The CONTENT_MOD southbound message is used to manage the content directly in the CEFS, which can be identified as the direct content operation, as shown in Fig. 5(a). Moreover, the actions Save_content and Read_content in the flow entries can store and distribute content. Only when packets match the entries, the actions will be operated, which can be identified as the indirect content operation, as shown in Fig. 5(b).

The content parser module and action set module are related to the content management. The content parser module parses the CONTENT_MOD message and then directly manages the content through the analysis result. The action set module contains two content operations. In order to let the controller know the locations of all contents the network, CEFS will report the information to the controller through the CONTENT_INF southbound message after executing the content operations.

When the content parser module receives a CONTENT_MOD message, it proceeds with the Mod_id field in the CONTENT_MOD header, as shown in Algorithm 3.

When the operation is Add a content, the content name and content data are parsed successively. The number of content should be consistent with the number field in the CONTENT_MOD header, and if it does not, the message is proven wrong and CEFS will report an error to the controller. When parsing a content name, the total_length field in the CONTENT_MOD header is parsed to acquire the length of this content name. Then, each level of this content name is parsed successively. When parsing the content data, the same action is executed as above to acquire the length of content data, thus accessing and storing it in the forwarding system.

| Action        | Function                        |
|---------------|---------------------------------|
| Output        | Forward the packet to one or more ports |
| Controller    | Forward the packet to the controller |
| Drop          | Drop the packet                  |
| Goto table    | Setup the next table in the lookup cache |
| Modify field  | Modify the packet fields         |
| Save content  | Save the content in the content cache |
| Read content  | Read the content from the content cache and forward it |
| Record path   | Record the path of request host  |
| Del path      | Delete the path of request host  |

**TABLE 2: Action set**

**Algorithm 2** FLOW_MOD Parsing Algorithm

```plaintext
Algorithm 2 FLOW_MOD Parsing Algorithm

Require: FLOW_MOD message F_M;
1: while Entry e_i exists in F_M do
2:   if command = 0 then
3:     // Add an entry;
4:     Parse FLOW_MOD header and ICN_match;
5:     Generate basic information of an entry;
6:     Generate matching fields of the entry;
7:     if ProtocolConformity() = 0 then
8:       // Check if entries conform with protocols;
9:       // Function ProtocolConformity() returns 0 when entries conform with protocols;
10:      Parse action;
11:      Insert e_i;
12:      e_i ← e_i+1;
13:   else
14:     Delete e_i;
15:     Report error;
16:     Exit;
17:   end if
18: else
19:   // Delete an entry;
20:   if ICN_match exists in F_M then
21:     Delete entries partially according to table number and priority;
22:     Exit;
23:   else
24:     Delete entries partially according to table number and priority;
25:     Exit;
26: end if
27: end if
28: end while
```

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/
When the operation is Delete a content, a different decision will be made based on the content name that the CONTENT_MOD message contains. If the message does not contain any content names, all the content will be deleted. If the message contains a content name, CEFS will lookup this content name and delete the corresponding content data. Once lookup fails, an ERROR message is reported.

D. DATA PLANE PACKET PROCESSING MODULE

DPM is a generic module that can use the protocol and routing information to forward any arbitrary packets in ICN. It includes the parser module, flow table module, and action set module. The process of addressing packets is shown in Fig. 6. When the parser module receives a packet, the first step is to find a matched protocol tree stored in the protocol module. Then, the parser module converts this packet to a packet tree and several inner value fields through the protocol tree. Next, the flow table module looks up multiple flow tables and finds a flow entry that matches the packet tree. The structure of matching fields in this entry must be a subtree of the packet tree. The matching function of the lowest level VALUE supports both the exact match and the longest prefix match. Finally, the action set module executes the actions in this entry. These actions are described in Table 2, where the last four items are designed for ICN. Because the Read_content and Save_content actions are explained in the section above, we only explain the Record_path and Del_path actions here.

Since ICN is a content-based network, CEFS does not know the source and destination during the data transmission. Typically, CEFS receives the request packet and reports it to the controller. Then, the controller selects a path to the target content and sets the host-to-content and content-to-host entries to the data plane so that the content can be correctly returned. In order to reduce the number of packets sent to the controller, we also provide the Record_path action in CEFS. CEFS only sends the first packet to the controller when receiving the same request packets from different ports. Meanwhile, CEFS can record or drop the path of other packets. After acquiring the target content, CEFS sends it to all request paths. Then these recorded paths should be deleted to avoid multiple responses, and the Del_path action is executed.

V. EVALUATION

To evaluate the feasibility and efficiency of CEFS, we use OVS as the basis to implement it. We keep the core forwarding processor and change the southbound interfaces of OVS.

A. FORWARDING PERFORMANCE

Fig. 7 shows the topology of our setup for evaluating network performance. We use three machines with 4-core Intel
1.6GHz CPUs and 8GB RAM, and each of them runs the Ubuntu 14.04 LTS. All machines are equipped with 10 Gbps interfaces. To generate and count packets, we configure two machines with the high-performance packet generator called Moongen [25].

To derive the baseline performance of CEFS, we set a simple entry to forward the specific NDN packets to a statically configured port in the Single-Flow experiment. We evaluate the throughput and packets-per-second for different packet sizes. Then, we compare CEFS to NDN.P4 [24], NFD [4] and NDN-DPDK [14]. In NDN.P4, the authors use BMv2 to forward NDN packets. To overcome some functional limitations in P4, they use Data Link Layer to encapsulate the packets and set multiple entries to forward them. Therefore, we set the same number of multiple entries as NDN.P4 to forward the encapsulated packets in the Multiple-Flows experiment. NFD is a kind of NDN simulator and constructed by C++ programming language, widely used by ICN and NDN researchers. As for NDN-DPDK, it first deploys NDN on hardware based on DPDK and achieves high performance. Therefore, we chose these three methods as the comparisons of CEFS because they are either state of art works or widely used.

Fig. 8 shows the network throughput that our setup achieved with different packet sizes in comparison to three methods mentioned above. CEFS can forward 75k packets per second for the minimum-sized packet in the Single-Flow experiment approximately. The network throughput decreases in the Multiple-Flows experiment with more entries, but CEFS is still superior to BMv2 and NFD. In Fig. 9, CEFS achieves the throughput of 750Mbps for the maximum-sized packet, NFD can achieve 700Mbps under the same circumstance, and BMv2 can only achieve 350Mbps in NDN.P4 experiment. The throughput of BMv2 can achieve 450 Mbps with the stress experiment, an example provided by P4. Due to the complex process of NDN and the increasing number of entries, it is reasonable that the throughput decreases in NDN.P4 experiment. However, CEFS is inferior to NDN-DPDK in the aspect of throughput and forwarding rate because NDN-DPDK is based on DPDK, a high performance forwarding platform. Our work focuses on the flexibility and scalability for ICN experiments, and we will be dedicated to improving the forwarding performance of CEFS in the future.

Fig. 10 shows the delay of CEFS and the other three methods with different packet sizes. Delay is another essential indicator measuring network performance. Experiment results show that the delay of all four methods during one way end-to-end transmission is proximate, but CEFS has a lower delay compared with the other three methods. Especially, the delay of NFD deteriorates with the maximum-sized packet, and the possible reason is that NFD is a software-based forwarding system and it hits the bottleneck.

In [23], the authors evaluate the network throughput in two NDN test-beds built on NFD. The data show that the testbeds can only process 44.3 packets per second and the throughput is 64kbps. It proves that NFD is inefficient. Our forwarding system is better than NFD and BMv2 because the design of generic packet processing module conforms to the characteristics of ICN, which can effectively improve the forwarding performance.

B. APPLICATION

By deploying the experimental environment with CEFS, researchers can use simple messages to manage the protocol, routing, and content of ICN without re-developing the forwarding system, which remarkably reduces the complexity of examining a new protocol or other advanced mechanisms for ICN.
We design four experiments to verify the feasibility of CEFS, and use a real topology derived from the NDN testbed. Hosts use the packet generator to send request packets, and CEFS is deployed on all switches.

Initially, all switches are connected to the controller that can know the global topology. We use the \texttt{PROTOCOL\_MOD} message to define the NDN protocol in all switches. Then we send content A to S30 through the \texttt{CONTENT\_MOD} message. S30 saves content A and sends a \texttt{CONTENT\_INF} message to the controller.

1) Benchmark experiment
H1 sends the request packet to S44 to request the content A. S44 does not know how to forward the packet and sends it to the controller. The controller selects the shortest path according to the target content location and global topology. Then it sets the entries to switches through the \texttt{FLOW\_MOD} message. The selected path is H1-S44-S20-S10-S2-S8-S29-S30, as shown in Fig. 11(a). All switches in this path save the content A and report to the controller. After that, H2 requests the same content. S44 responds by itself because it saved the content A. The path is H2-S44, as shown in Fig. 11(b). We measure the round-trip time (RTT) of the content request and response as shown in Table 3. After H1 requests, the RTT of H2 request is tremendously reduced due to the cache of switches.

2) Routing management experiment
H1 requests the content A, and the process is identical to the benchmark experiment except the controller selects a path with minimum delay. The selected path is H1-S44-S46-S43-S12-S8-S29-S30, as shown in Fig. 12(a). In Table 3, by comparing this experiment with the benchmark experiment, the RTT of H1 reduces about 14% because the controller selected a path with minimum delay. This experiment shows that CEFS can change entries easily through the \texttt{FLOW\_MOD} message. It is useful to validate routing strategies in some experiments.

3) Content management experiment
H1 requests the content A, and the process is identical to the benchmark experiment except that only switches whose degree is greater than 8 save the content. The degree of a switch is the number of connections it has with other switches. Therefore, only S10 saves the content A. We use the indirect content operation to implement this caching strategy.
TABLE 3: RTT of different experiment

| Host | H1 | H2 | H1 | H2 |
|------|----|----|----|----|
| RTT(s) | 4.09 | 0.16 | 3.49 | 1.756 |

TABLE 4: RTT of protocol management experiment

| Host | H1 | H2 |
|------|----|----|
| Content | A | B | C | A | B | C |
| RTT(s) | 4.57 | 2.76 | 4.61 | 0.26 | 0.73 | 1.24 |

We set entries with the Save_content action to S10. After that, H2 requests the same content and S10 responds to it. The path is H2-S44-S20-S10, as shown in Fig. 12(b). Comparing this experiment with the benchmark experiment, the RTT of H2 in the benchmark experiment is shorter due to the different content caching strategies. This experiment shows that custom caching strategies can be easily implemented using content operations. It makes deploying various cache strategies easy and fast.

4) Protocol management experiment

We add a new TYPE number to the NDN protocol and define this new protocol through the PROTOCOL_MOD message. The controller selects the shortest paths for three different content requests, as shown in Fig. 13(a). The controller uses this new TYPE to identify different contents and sends different entries to switches. All switches save content A, switches whose degree is greater than 6 save content B, and switches whose degree is greater than 8 save content C. Therefore, S10 and S20 save content B, while only S35 saves content C. After that, H2 requests the same three contents. The nearest switches who possess these contents response. The new content distribution and paths are shown in Fig. 13(b). Table 4 shows the RTT of different content requests and responses. For H1, the RTT of content A is the same as C, while the content B is shorter, and it is proportional to the delay on each path. Because switches use the new TYPE to identify different contents and selectively save them, the RTT of content A reduce about 94%, and the RTT of content B and C reduce about 73% for H2. This experiment shows that users can update ICN protocols in CEFS, which can effectively speed up the evolution of ICN protocols.

VI. CONCLUSIONS

In this paper, we summarized some methods that combine SDN and ICN, and some related works to make the forwarder programmable in ICN, which are useful in reducing the complexity of experiments and accelerating the evolution of ICN protocols. However, all these methods have some shortcomings. Therefore, we propose a flexible forwarding system called CEFS for experiments in ICN. We modified the southbound protocol of SDN to support the ICN architecture, thus rendering the management of the protocol, routing, and content in ICN easier. We described the details of our design and used OVS to implement it. The feasibility of our proposal is verified, and the efficiency is shown in our experiments.

FIGURE 13: Paths of protocol management experiment

However, our proposal still has a lot of limitations and needs more improvements in the future. First, CEFS does not support TCP/IP-based overlay ICN architecture because the protocols in CEFS are defined by users and all the configurations are determined by those user-defined protocols. Second, our work is implemented in Open vSwitch, whose performance is better than some methods as shown above but worse than the real network device. Under this circumstance, CEFS cannot support high throughput experiments, thereby restricting the use of CEFS. In summary, our future work includes two main goals detailed below:

- Support heterogeneous network experiments which allow users to define the control plane and data plane separately;
- Improve the forwarding performance of CEFS to support higher throughput experiments.

After conquering these two limitations of CEFS, users can experiment with their self-defined ICN protocols with high fidelity and performance, regardless of any restrictions in present work. Meanwhile, CEFS will be deployed in CENI [26], which is an open, sustainable and at-scale networking infrastructure in China, for more researchers to conduct ICN experiments conveniently.

ACKNOWLEDGEMENT

Fan Yang and Tianyuan Niu thank State Key Laboratory Subject: Research on Network Protocol Universal Forwarding Technology under Grant NST20190301 and National key R & D plan: Research on Architecture and Key Technologies of Next Generation Network Processor under Grant 2018YFB1800602.
REFERENCES

[1] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica, “A data-oriented (and beyond) network architecture,” in Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications, pp. 181–192, 2007.

[2] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, K. Claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, “Named data networking,” ACM SIGCOMM Computer Communication Review, vol. 44, no. 3, pp. 66–73, 2014.

[3] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, “Networking named content,” in Proceedings of the 5th international conference on Emerging networking experiments and technologies, pp. 1–12, 2009.

[4] “Nfd: Named data networking forwarding daemon.” [Online]. http://named-data.net/doc/NFD/current/.

[5] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, et al., “P4: Programming protocol-independent packet processors,” ACM SIGCOMM Computer Communication Review, vol. 44, no. 3, pp. 87–95, 2014.

[6] “Ccnx packet format specification.” [Online]. https://na-med-data.net/project/specifications/.

[7] “Cnx messages in tlv format.” [Online]. https://datatracker.ietf.org/doc/draft-irtf-icnrg-ccnxsemantics/.

[8] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “Openflow: enabling innovation in campus networks,” ACM SIGCOMM Computer Communication Review, vol. 38, no. 2, pp. 69–74, 2008.

[9] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Sheller, et al., “The design and implementation of open vswitch,” in 12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15), pp. 117–130, 2015.

[10] K. Yu, S. Eum, T. Kurita, Q. Hua, T. Sato, H. Nakazato, T. Asami, and V. P. Kafle, “Information-centric networking: Research and standardization status,” IEEE Access, vol. 7, pp. 126164–126176, 2019.

[11] Z. Li, Y. Xu, B. Zhang, L. Yan, and K. Liu, “Packet forwarding in named data networking requirements and survey of solutions,” IEEE Communications Surveys & Tutorials, vol. 21, no. 2, pp. 1950–1987, 2018.

[12] K. Yu, M. Arifuzzaman, Z. Wen, D. Zhang, and T. Sato, “A key management scheme for secure communications of information centric advanced metering infrastructure in smart grid,” IEEE transactions on instrumentation and measurement, vol. 64, no. 8, pp. 2072–2085, 2015.

[13] T. Liang, J. Pan, M. A. Rahman, J. Shi, D. Pesavento, A. Afanasyev, and B. Zhang, “Enabling named data networking forwarder to work out-of-the-box at edge networks,” in 2020 IEEE International Conference on Communications Workshops (ICC Workshops), pp. 1–6, IEEE, 2020.

[14] J. Shi, D. Pesavento, and L. Benmohamed, “Ndn-dpdk: Ndn forwarding at 100 gbps on commodity hardware,” in Proceedings of the 7th ACM Conference on Information-Centric Networking, pp. 30–40, 2020.

[15] G. Ravikumar, D. Ameme, S. Misra, S. Brahma, and R. Tourani, “Icasm: An information-centric network architecture for wide area measurement systems,” IEEE Transactions on Smart Grid, vol. 11, no. 4, pp. 3418–3427, 2020.

[16] L. Zeng, H. Ni, and R. Han, “An incrementally deployable ip-compatible-information-centric networking hierarchical cache system,” Applied Sciences, vol. 10, no. 18, p. 6228, 2020.

[17] A. Detti, S. Salsano, and N. Blefari-Melazzi, “Ipv4 and ipv6 options to support information centric networking,” Internet Draft, draft-detticonet-ip-option-02, Work in Progress, 2011.

[18] L. Veltri, G. Morabito, S. Salsano, N. Blefari-Melazzi, and A. Detti, “Supporting information-centric functionality in software defined networks,” in 2012 IEEE International Conference on Communications (ICC), pp. 6645–6650, IEEE, 2012.

[19] “The ccnx project.” [Online]. https://wiki.fd.io/view/Ccnx.

[20] X. N. Nguyen, D. Saucez, and T. Turletti, “Efficient caching in content-centric networks using openflow,” in 2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), pp. 67–68, IEEE, 2013.

[21] N. L. van Adrichem and F. A. Kuipers, “Ndflow: Software-defined named data networking,” in Proceedings of the 2015 1st IEEE Conference on Network Software Definition (NetSoft), pp. 1–5, IEEE, 2015.

[22] S. Jouet, R. Cziva, and D. P. Pezaros, “Arbitrary packet matching in openflow,” in 2015 IEEE 16th International Conference on High Performance Switching and Routing (HPSR), pp. 1–6, IEEE, 2015.

[23] P. Zuraniecki, N. van Adrichem, D. Ravesteijn, W. Intema, C. Papadopoulos, and C. Fan, “Facilitating icn deployment with an extended openflow protocol,” in Proceedings of the 4th ACM Conference on Information-Centric Networking, pp. 123–133, 2017.

[24] S. Signorello, R. State, J. François, and O. Festor, “Ndn p4: Programming information-centric data-planes,” in 2016 IEEE NetSoft Conference and Workshops (NetSoft), pp. 384–389, IEEE, 2016.

[25] P. Emmerich, S. Gallemüller, D. Raumer, F. Wohlfart, and G. Carle, “Moongen: A scriptable high-speed packet generator,” in Proceedings of the 2015 Internet Measurement Conference, pp. 275–287, 2015.

[26] “Ceni.” [Online]. https://ceni.org.cn.
FAN YANG, a lecturer at Beijing University of Posts and Telecommunications, graduated from Beijing University of Posts and Telecommunications with a Ph.D. His research interests include Software Defined Network and high-performance routing and switching technologies. He has participated in the National 973 Planning Project “Research on Service-Oriented Future Internet Architecture and Mechanism”, National 863 Project “Service-Oriented SDN Architecture and Key Technology Research”, “Future Network System and Structural Research for Service-Oriented Resource Intelligent Scheduling” funded by National Key Laboratory, etc., presided over “German Telecom & Huawei’s next-generation IP routing equipment research”, “Huawei ultra-high-speed network processor search algorithm research” and other enterprise projects. He has published more than 30 SCI/EI papers and national invention patents. He has won the 2013 National Technology Contribution Individual Award, the Huawei Golden Network Award, and the China Communication Society Science and Technology First Prize.

TIANYUAN NIU received the B.S. degree of Information and Communication Engineering from Beijing University of Posts and Telecommunications, Beijing, China. He is currently pursuing a Doctorate degree at Beijing University of Posts and Telecommunications. His research interests include future network architecture, Named Data Networking, cache mechanism in future network device and P4.