ICN-based Network and Service Controller on an OpenFlow Network testbed

Anu Mercian, Asit Chakraborti, and Ravishankar Ravindran

Abstract—Information-centric Networking (ICN) has gained popularity lately due to increase in importance for content/information in current days. It is thus easier and cost effective to route based on information than on location as it is done in IP-based network. The advantages of ICN network can be combined with software defined network, which separates the control plane from the data plane, thus reducing control level overhead in the network and much faster routing. We attempt to implement an ICN based service over an Openflow based network, thus transferring the control layer to the central Controller of the openflow based network. We explain the implementation level details.

Index Terms—Software Defined Networking (SDN), Information-Centric Networking (ICN), Cloud Orchestration

I. ICN OVER SDN

Information Centric Networking, although introduced in 1979, it has gained fame recently due to the many advantages. ICN is well known for its advantages in naming the content rather the location, thus making content, location independent and reducing the necessity to access the location upon each request, rather obtain the content cached at the intermediate routers. This also vouches for security as the location is maintained anonymous and issues such as man-in-the-middle can be inherently avoided. ICN also supports efficient caching techniques for content to reduce congestion and increase search speed. This correspondingly reduces the memory requirement at the end location. This is advantageous to current times where most traffic is from hand-held devices such as mobile phones, tablets etc, which do not have huge memory capabilities when compared to desktop systems.

Software Defined Networking is the concept of moving the control of the network to the centralized system, Controller. This is in efforts to reduce control overhead by separating the control plane from the data plane. Therefore, based on the control instructions from the Controller, the packets are transferred in the data plane. The advantage of this network is the network can be designed as per the needs set by the controller. Thus, the name, software defined, where the network connectivity can be designed by the controller based on centralized structure. The major advantages of SDN is managing multiple network services from the controller. One implementation of SDN is Openflow network, which send control as openflow messages from the Controller to the switch and the switch maintains a flow table that defines the set of flows set-up by the controller.

The efforts to implement ICN over SDN, collectively called Software-Defined Information Centric Network (SD-ICN), is being experimented in research to combine the advantages of two networks. Although each of the concepts are gaining fame individually, there are multiple test beds that are implementing ICN over SDN/Openflow network. We discuss many of the current implementations in the next section. ICN works by populating the Forward Information Base (FIB) table with the name-based routes in the network. By implementing ICN over SDN, the controller needs to populate and depopulate the FIB by using the openflow control messages. This requires the incorporation of suitable ICN and openflow based agents to communicate between them. In this paper, we discuss our efforts to implement these agents.

II. RELATED WORK

The work in this paper is the extension implementation of [1] on a test bed, which describes the basic set-up for ICN over SDN network for edge-cloud services and the authors also discuss a conferencing example of the edges nodes. This example has been demo-ed with our current test-bed as well. From simulation analysis, it is observed that the ICN based edge-cloud framework benefits more than peer-to-peer design. This paper describes the necessary components of the framework such as ICN Cloud Orchestrator having three important components: Compute Controller, Storage Controller and Network Controller. These components are explained in detail in the next section which discussing the implementation details.

There have been a few ICN over SDN implementations [2], [3], [4]. The difference of our implementation from the previous implementations are, the focus is on cloud orchestration. We reduce the complexity of intermediate agents and improve the overall service orchestration capability of the set-up. And the reuse of our implementation is simple and requires very few modifications. In [3], the authors have implemented ICN over SDN, named CONET, by utilizing the IPv6 options field. Based on the options field matching, the openflow decision is made. They also discuss interest handling requirements and routing operations. This implementation requires the introduction of an NRS (Name-routing service node) in the network. This requires modification at each packet level and accounts for packet processing time. This implementation is done on the OFELIA testbed [2]. The OFELIA testbed is a dedicated
testbed for Openflow ICN operations [2]. This testbed is being worked on a large scale and supports all ICN based applications. The testbed also implements using the options field for IPv6 in the packets.

ICN based implementations support new Caching techniques, as the apriori reasoning of ICN is that the requests access the cache more frequently than obtain content from the end location. In support of this field, a lot of literature is based on efficient caching techniques. The authors in [5], discuss an efficient caching technique for ICN over SDN by implementing three extra modules in the controller such as Measurement, Optimization and Deflection. The measurement module keeps track of the popular data and gives the flow statistics and the optimization module makes a decision based on the flow statistics and the deflection modules creates mapping of content based on the decision made. These features help efficient caching. Another paper that introduces efficient caching in an alternate content based network over SDN is [6].

III. IMPLEMENTATION DETAILS

A. Components

In order to implement ICN based edge cloud service framework, there needs to be some standard modules to process key functions. Some of the key components are as per [1]:

- **ICN Cloud Orchestrator** – The function of the cloud orchestrator is to control the functionalities of all ICN services from the Controller. For an example scenario of function of cloud orchestrator would be, if many edge nodes are requesting to join a conference, the cloud orchestrator would bring up the conference platforms in each of the edge nodes and populate the FIB tables of the edges to be able to communicate with the requested nodes. Since the cloud orchestrator holds the responsibility for almost entire functionality, it should be able to control the other components of the network such as the Controller and router switches etc. Openstack software [7] is one such software that is available to achieve the functions of Cloud Orchestrator. The Orchestrator will house the controller that will function to send the control messages to each of the edge router/switch.

- **ICN Network Controller** – The Controller, although receives control information from the Cloud orchestrator, it works independently with that information. The presence of the controller introduces the software-defined nature to the network. The most popular SDN, Openflow network is implemented in this paper. For the purpose of the implementation, Floodlight Controller [8] is used as it is popular with Industry and has a good support group. And the openflow network is implemented by OpenVSwitch [9] installed at the edge nodes. The basic property of the network controller is topology discovery. In the case of the ICN Network Controller, the controller must be aware of the ICN nodes/edges in the network and register their location details for mapping.

- **ICN Service Controller** – In case of the service controller, this module should accept the incoming service requests from edge nodes and perform service discovery, mapping nodes to service maps. Along with service discovery, it is necessary to perform the ICN action from the controller. The necessary step towards ICN communication is populating the Forwarding Information Base (FIB) table and maintaining the Pending Interest Table (PIT).

B. Software Introduction

The testbed was implemented on four desktops, converting one to the controller and the other 3 desktops as the ICN nodes connected in the openflow network. The controller adopted for this experimentation is Floodlight Controller [8], which is JAVA-based software and based on sockets, it creates a channel between the controller and each node for Openflow messages exclusively. Each of the ICN node, inorder to enable openflow networking, has OpenVSwitch [9] installed on it. The openvswitch introduces a kernel module such that it introduces a virtual bridge switch that is physically connected with one of the ethernet interfaces on the system such as the “eth0”, “eth1” etc and establishes connectivity with the controller. The architecture set is shown in figure . For our base implementation, we assume that the each of the desktops are ICN nodes, but as we develop on the test-beds, the ICN nodes will be moved onto virtual instances, and therefore, we can show the working of exponential virtual ICN nodes which could demonstrate the use of the implementation on handheld devices. Each of the ICN nodes have CCNx-0.8.0 [10] implemented on them as well to demonstrate the use of ICN applications.

a) **OpenVSwitch**: It is important to understand the working of openvswitch to understand the implementation. The openvswitch maintains a flow table that is populated with rules that are either inserted by the controller or by the switch itself. Each flow rule has a actions field and has a set of standard action set, which are:

1) **output:port** - Action outputs the packet to the port which is supposed to be an Openflow port number.
2) **normal** - Action subjects to normal L2 or L3 processing.
3) **flood** - Action sends the packet on all switch physical ports other than the one it was received from and the ports which have disabled flooding.
4) **all** - Action sends packets to all ports except the one it was sent from. This is different from flood command, because this cannot be disabled at each switch.
controller - Action sends the packet as a "Packet-In" message to the controller. Every new packet in the network, that is, one which does not have a prior rule in the flow table of the switch, will be sent to the controller as a packet-in message.

We utilize the controller action of the flow rule to implement our test-bed. In order to understand how this done, it is useful to know the different openflow messages the controller processes to make a decision:

1) OFPacketIn - This packet is created based on the incoming packet, and along with it, it sets field of Inport, Length and Switch ID.
2) OFPacketOut - This packet is a response packet to the packet-in message and also sets an actions field.
3) OFFlowMod - When a flow rule has to be set at the switch, this message is used. It has an actions field, based on which decisions are taken.

IV. DESIGN

In this subsection, we discuss the design details of the implementation. We split this section into Controller Implementation details and Switch Implementation details. We need to enable suitable middle-agents that would be compatible between CCNx and OpenVSwitch. We extend Floodlight controller to include our specific application capability as well.

A. Controller-side implementations

In the Floodlight Controller, we introduce two new modules, ICN Network Controller for the ICN node discovery and ICN Service Controller for implementing the conferencing application. The former can be kept general for all ICN-based applications and the ICN Service Controller can be further extended to expand on the ICN applications. We show our implementation within the Floodlight architectural modules in the figure . Floodlight is a JAVA-based controller and the ICN extension modules are written in JAVA as well.

From our understanding, the series of s.eps implemented by Floodlight are firstly the Floodlight controller initiates a channel at a port (default, 6633) that is used for listening to OF Messages alone. It possesses a OFMessageListener module that captures the packet as it is and reads the corresponding fields of the generic TCP/UDP packet and the OF specific fields such as inport, length and actions, if applicable. Then the message is read as an OFMessage and then categorized as either a ICMP, LLDP, ARP or Data packet etc. The configuration file of Floodlight can be designed such that a set of modules can be loaded at run-time. In most general case, the Forwarding module is loaded that responds to the OFPacketIn message. The Forwarding module sends OFPacketOut for a corresponding OFPacketIn message with the action field set and in some cases, even the OFFlowMod message to set a flow rule at the Switch is sent.

b) Network Controller: The network controller has to detect the OFPacketIn messages that indicate the presence of ICN nodes. To support this, we introduce a set of Service Messages that will be used to indicate the presence of ICN nodes and ICN services. We show the set of service messages in Table . The Network Controller recognizes the service messages 1 & 2 and creates a Node List for all the ICN nodes. This is helpful to gain knowledge of the ICN nodes and their IP and MAC addresses, that can be used for content mapping. A description of the processes are shown in a polling diagram in the figure .

c) Service Controller: The service controller is used to detect service messages 3 and 4 and send out messages 5 and 6 based on decisions. The basic concept of Service controller is to identify the specific ICN Service and distribute resources for its purposes. We explain the ICN chat service which is similar to a conferencing service. But the implementation concept can be expanded to other ICN services as well. The fundamental requirement by the controller is to be able to modify the FIB table in the ICN node, this will help content mapping and
control from the Controller.

On receipt of message 3 and 4, this module will obtain the ICN prefix that identifies the chat service, along with the ICN node details. Similar node lists can be created for different chat services. When the number of chat applicants for a single chat service prefix increases more than one, then we need to distribute resources between the applicants. This is done by sending the message 5, that indicates a FIB entry has to be added to the FIB table and similar when an applicant closes the chat application, message 6, a message to delete that specific FIB entry has to be sent. Again, a polling diagram is provided in figure to explain the steps involved in implementing the service controller.

Another important aspect as a part of the Controller implementation is the building the OF Packet Structure. It is important to send the service message from Table as Openflow messages, only then the purpose of implementing ICN over SDN is achieved. The OVS, on the switch, takes the responsibility of converting a normal UDP packet into an OF message, but on the controller side, it is required to logical program to create a OF packet. This was achieved by encapsulating the data message of desired string, within an UDP IP packet, again within an IPv4 Ethernet packet. The s.e.s of which are shown in figure.

### B. Switch-side implementations

We tried to implement all logical components at the controller and reduce the computationality at the switch side. The switch side required an agent that would be able to communicate between the ICN services and the OVS flow tables. We achieved this by designing a local Python-based agent that would dynamically run continuously and send service messages when ICN services instantiate. We call this agent, ICN Host Manager as it will work at the ICN host. The pseudo algorithm for the ICN Host manager is show in the algorithm 1.

#### Algorithm 1: ICN Host Manager

**Ensure:** agent runs on ICN nodes

```python
while i do
    createUDPsocketandbindtoport
    addflowrulatosendmessagesfromagenttocontroller
    if ccndisup then
        sendCCND_UP
    end if
    if ccnchatisup then
        icn_prefix = conference_name
        sendCCN_CHAT_UP+ < icn_prefix >
    end if
    else {ccnchatisdown}
        sendCCN_CHAT_DOWN+ < icn_prefix >
    end if
    else {ccndisdown}
        sendCCND_DOWN
    end if
    fib_mod = recvfrom(controller)
    if fib_modisFIB_ADD then
        addFIB < icn_prefix >/ < icn_node >
    else {fib_modisFIB_DEL}
        delFIB < icn_prefix >/ < icn_node >
    end if
end while
```

The Controller and Switch side implementations can be shown at a high level in figure and in-depth in figure.

### V. CONCLUSION

In this article, we describe the implementation details of ICN-over-SDN network. We utilize the floodlight controller...
and introduce additional functionality to incorporate content-based routing. Similarly we utilize CCNx protocol for ICN implementation. This is to demonstrate that the overlap application can be done with third-party software, by upgrading software applications, thus avoiding any upgrade in hardware propriety. The output of the implementation was observed by causing the controller to flood the Forwarding Information Base (FIB) table with the ICN-based client applications and the content messages assigning flow actions in the OpenvSwitch Flow tables. Future work of this implementation is importing this functionality within an Openstack framework and using Neutron to generate virtual machines as ICN client applications and using this implementation within the Controller module.

REFERENCES


