Developing and Deploying a Carrier-class SDN-Centric Network Management System for a Tier-1 Service Provider Network

Saurabh Hote, Puneet Ghodasara, Tamal Das, Aniruddha Kushwaha, Sidharth Sharma, Sarvesh Bidkar, and Ashwin Gumaste
Indian Institute of Technology (IIT) Bombay, India.

***Abstract*—Software defined networking (SDN) has considerably shaken the telecommunications industry, with almost every major vendor announcing SDNizing of their product portfolio and all providers building use cases to inculcate the SDN concept. Significant collaborative activity is underway towards proposing a common set of SDN standards. However, with huge amounts of existing deployment of network gear, one question that remains is how to adopt SDN given the existing infrastructure? To this end, we have developed a well-standardized technology with minor tweaks and created a hardware paradigm whose forwarding plane conforms to carrier-class standards, but whose control plane caters to the SDN philosophy. This paper discusses our experience of building such a control plane and its subsequent deployment.**

We describe the design and implementation of a *network management system* (NMS) for carrier-class networks using Carrier Ethernet manifestations. The management system subscribes to the SDN philosophy, thereby facilitating user-control-based provisioning and service definitions. A centralized controller communicates to Carrier Ethernet Switch Routers (CESRs) that provision services based on multiple identifiers such as IPv6, IPv4, MAC, CTAG/STAG, port-based, etc. The design of the controller in the NMS and the control state-machine in the CESR, as well as their interactions are described. The paper details the concepts underlying the SDN system as well as its module-level and service-level implementation aspects. Our key contribution is that the CESR that we built along with the SDN NMS is put to test in a tier-1 provider network, thereby facilitating real-network performance measurement. A city-wide network was built and its results are presented in this paper.

***Index Terms*—Network management system, SDN, Carrier Ethernet, SDN whiteboxes.**

I. Introduction

Software defined networking (SDN) [1, 33, 35] is an emerging network philosophy that decouples the control plane from the data plane to dynamically enforce user-defined service features across a network. It brings in programmability into the network [2, 25]. SDN implements a software-based approach to configure the underlying hardware that enables routing and forwarding of data efficiently as per user-defined rules. It is a way to control and operate a network through a centralized software (called SDN controller) [3] to achieve benefits in *operation*, *administration*, *management* and *performance* (OAMP) [4] as well as service provisioning, agility and programmability (that are difficult to achieve with current switches and routers). SDN provides for orchestration of services and resources that facilitate programmability to be built-in; thus allowing users to define their unique service requirements [36]. *It is envisaged that through SDN, one may eventually be able to achieve extreme programmability of facilitating service chains* [5] *to be developed using either conventional network gear that works as white-boxes supporting SDN needs, or through the recently proposed NFV (network function virtualization) paradigm* [5]. In addition to abstracting the network, the SDN control architecture opens APIs to enable network services such as routing, multicast, security, access control, bandwidth-management, traffic engineering and QoS optimization, etc. to be mapped to user-defined needs [3, 42-48]. SDN enables automation in service provisioning to ease various operations of the network. SDN also enables network monitoring and facilitates pre-defined actions based on network events to administer the network. SDN can be helpful to optimally use the network resources to manage the network in a cost-effective way [6].

SDNs have so far been deployed in data-center environments or enterprise networks and studies [7] are largely focused on specific enterprise-class networks. Given the amount of existing investment in service provider networks, and how providers are particular about their carrier-class requirements, it is imperative to understand as to how SDN with its flexibility can be mapped into a service provider network [37].

A typical core/regional service provider network uses optical fibers at the physical layer with *wavelength division multiplexing* (WDM) technology that acts as a bandwidth multiplier [38]. Transport is achieved through either legacy *SONET/SDH* or the more recent *Carrier Ethernet (CE)* suite of protocols and the network layer in a provider network is dominated by *IP/MPLS* routers [26, 31, 39, 40]. Each of these layers has a complex set of functions to accomplish, and a well-defined set of services to provision. In such a scenario, we seek to investigate the role of SDN in a provider network. IP/MPLS, though packet-oriented is not carrier-class, while SONET/SDH and inherently WDM are both predominantly circuit-switched technologies with limited scope for SDNization [27]. To adhere to operational requirements of an SDN solution, we will focus on packet-oriented communication that is also carrier-class. Hence, our focus is on investigating *whether packet-oriented CE can be a good choice for SDN architectures.*

CE is a carrier-class version of Ethernet, characterized by lack of CSMA/CD, no MAC-learning and absence of spanning tree protocol. It is a traffic-engineered solution, where a centralized Network Management System (NMS) provisions all the services. Hence, a CE network is readily amenable to SDN philosophies, which also requires clear distinction between the control and forwarding (data) planes.

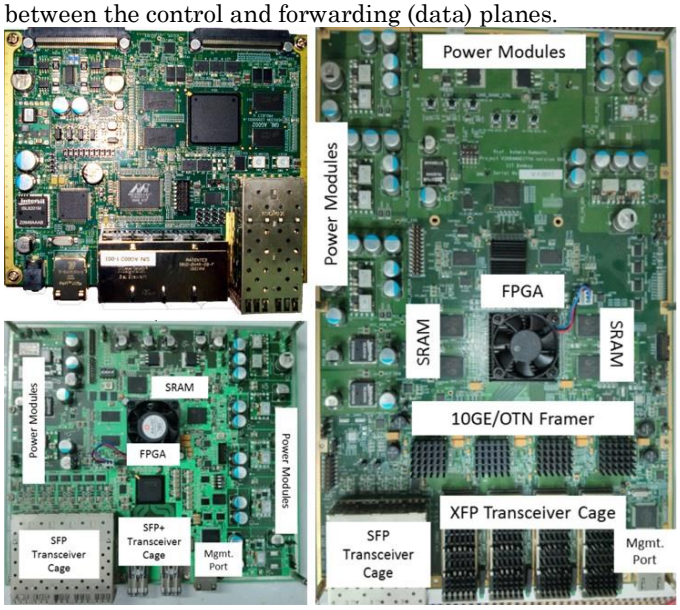

Figure 1: Customer Premise Equipment (top-left), metro edge device design (bottom-left), core device design (right) [9].

A. NMS Overview

We have designed a hardware manifestation of a *Carrier Ethernet Switch Router* (CESR, see Figure 1) [9, 24] that uses a concept called Omnipresent Ethernet (OE) [9, 11]. OE is a modification within the CE premise, adding routing capability to existing switching as well as facilitating user-defined forwarding parameters. To an extent, OE is similar to Fibbing [10] – whereby the network nodes are cloaked to artificially induce user-defined states. However, unlike Fibbing, which is primarily envisaged only at the IP layer, OE is particularly tied to the CE standards but yet uses IP, MAC, ports, tags as various identifiers on which the CE forwarding plane acts. The hardware architecture of the CESR is described in [9] and in this paper, we focus on a specific SDN implementation that is valid for the CESR as well as other carrier-class provider technologies.

In this paper, we describe the engineering of a control plane for CE networks that conforms to an SDN approach such that it separates the data/forwarding plane in the hardware from a software-based control plane at a centralized location (controller). The centralized controller (NMS) can plan, configure and monitor the network of OE-CESRs (henceforth termed just CESRs). The NMS opens out to a platform-independent Java GUI. The NMS is OpenFlow-compatible [8] though for the particular deployment in the tier-1 provider (called MTNL Mumbai, India), this requirement is relaxed. A shorter preliminary version of this paper, with only fewer results and no insights into the NMS design was presented in [12], while this paper details the working of the SDN-based NMS as well as adds new results from the field trials and extensive lab tests. An earlier prototype of our NMS that supported OpenFlow v1.2 was shown in [8].

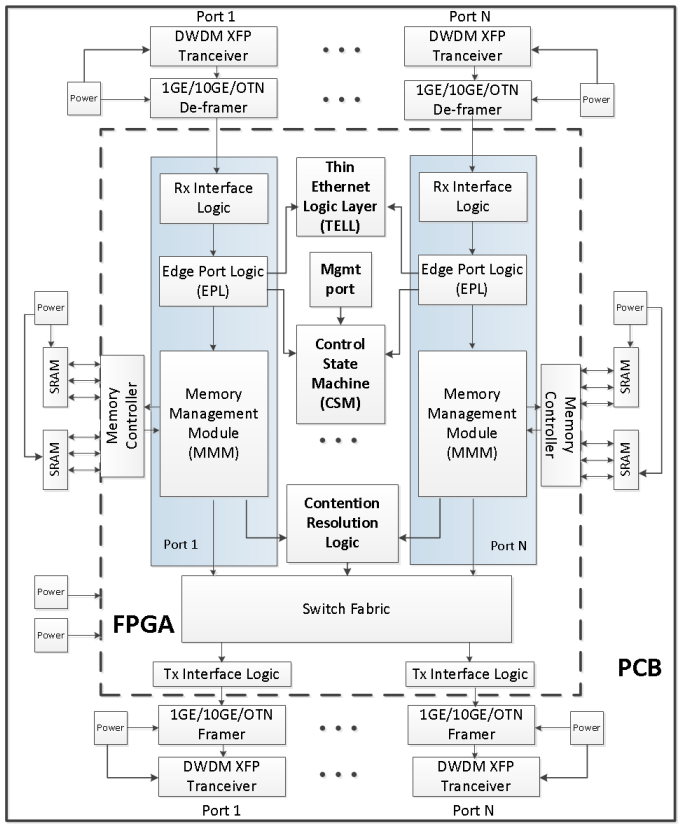

Figure 2: Architectural block diagram of $N \times N$ CESR.

Key NMS Controller Design Objectives: The following key objectives are inculcated in the design of the NMS controller (Figure 2):

- ✓ *Scalability*: The NMS controller facilitates the network to scale to at least 5000 nodes (CESRs) and at least a million enterprises/services. These numbers are the typical in a regional network. Our design considerations (described later) are such that we are able to achieve these numbers. See section VI for the results that justify scalability.
- ✓ *Open APIs*: A key goal of our NMS is to provide open APIs that can be used to describe new services, write scripts, as well as create new policies for the network to function. To this end, we have been able to achieve all the aforementioned goals. We have been able to describe new services using scripts as well as provide for ways to support ECMP that is otherwise not supported by the base CE standards.
- ✓ *Compatibility*: Our NMS is compatible with existing provider management frameworks. In fact, it is also possible to integrate with other NMSs and SDN controllers though this description is beyond the scope of this paper.
- ✓ *Carrier-Class Performance*: *A major goal of this work is to enable programmability in the network without losing carrier-class features*. To this end, we have engineered the NMS in a way that carrier-class features are never compromised while achieving programmability. In particular, the NMS provides for each service OAMP features along with <50 millisecond protection and restoration for a wide-variety of faults.
- ✓ *No storage on the box philosophy*: Unlike most existing NMS, which have an operating system such as Cisco IOS or Juniper's JUNOS that resides within the box

(hardware), our approach is different – we do not have an OS in the box. The hardware only supports tables and a control-state-machine, while the controller houses everything else, including the OS and the NMS application. Our argument is that this approach is more tuned towards the SDN framework as well as provides to make the forwarding plane simple and cost-efficient (no processor is required in the hardware).

The main contribution of our work is to take a telecom-class system (the CESR), and SDNize it. Further, we deploy the SDNized CESR system in the field and measure performance. The goal is to show that SDN certainly has a role to play in conventional service providers and can be brought out using technology that need not be just whiteboxes, but with boxes that are geared towards SDN adaptability. Though we must add that due to the specific technology focus, the SDN that we build is not as open as an SDN with whiteboxes. However, this comes with the premium of facilitating a carrier-class data-plane.

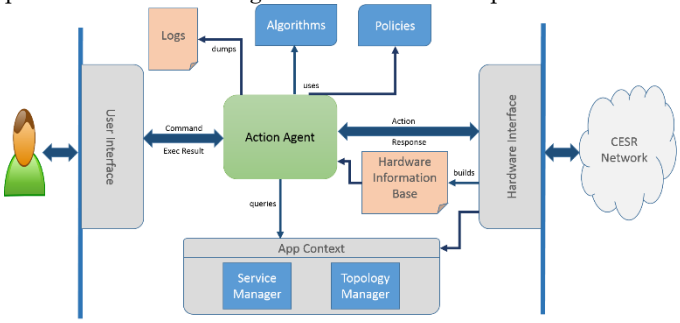

Figure 3: The NMS Architecture.

B. NMS Architecture

The SDN-based NMS controller architecture is shown in Figure 3. The NMS encompasses modular design and is designed to achieve specific service provisioning and management goals for network operators. The main goal of the architecture is to keep real-time consistency between the physical network and the logical network state maintained by the application. The NMS ensures consistency by maintaining a state automaton of all network elements and services, while periodically synchronizing the state of the virtual topology with the actual physical topology.

Our SDN-centric NMS offers multiple benefits over the conventional systems, namely:

- Our NMS keeps track of provisioned bandwidth on each port of the CESR. Whenever any new service is configured involving that port, the NMS checks available bandwidth at the port. In any case, it cannot exceed the maximum provisioning limit of the port, which guarantees the carrier-class nature of the CESR. This feature is hard to achieve in traditional switches because they do not offer centralized control on provisioning of services.
- The NMS also polls in all the CESRs for latency, jitter, packet loss and displays these statistics to the user on a per-service basis. Alarms, triggers and correction rules can be set to take appropriate action post an alarm or trigger.
- The convergence time of OSPF/CSPF running in the control plane of the conventional switches is high. Our path computation algorithms create paths between service end-points, which solves the convergence issues seen by OSPF/CSPF algorithms, as our algorithms are executed ahead-in-time (as compared to the service provisioning) and are centralized.
- Security of the network is an ever-evolving issue. The centralized control offers better security of the network because NMS monitors each CESR in the network and plug-and-play is impossible in the network. Each CESR is provisioned through the NMS thus implying that there is complete authentication of any new node that is added in the network. Similarly, each service is provisioned and authenticated through the NMS as well – implying no traffic can be added unless it is already provisioned by the NMS.

We note that backward compatibility with existing technology is important from a CapEx perspective. To this end, our CESR framework is completely compatible with existing gear. The advantages obtained by SDN such as agility and programmability should not come at the cost of incurring huge CapEx. In this case of CESR deployment – CESRs are standard packet-optical elements that are deployed in networks today and most of these manifestations come with a provisioning and management system – the NMS. In our case, we have made modifications to the NMS such that we facilitate a north-bound API that allows programmability and agility to be incorporated in.

C. Comparison with other SDN Approaches

Other SDN implementations such as ONOS [50], which is a Java-based distributed SDN operating system for managing networking components. The way in which ONOS provisions services and performs discovery of nodes in the network is analogous to our approach. Our NMS is designed specifically for CESRs, whereas ONOS is more generic and used for controlling a wider spectrum of networking equipment. Early use-cases for ONOS focused on new, innovative services for providers, which includes examples like Central Office Re-architected as a Data Center (CORD), Packet-Optical Convergence, SDN-IP Peering, and IP Multicast content distribution.

In our case, we map all services primarily to two base services – ELINE and ELAN – that are provisioned using IP, MAC, port-based, C/STAG-based identifiers. ONOS supports multiple configuration and control protocols such as Openflow, Netconfig, PCEP, etc. in the southbound interface. Our NMS specifically supports only the OE protocol. There are commonalities between the ONOS and our approach such as fault handling, SDN philosophy adoption and programmability.

This paper is organized as follows: Before exploring the high-level architecture of the NMS in Section III, we provide a brief primer on CESR itself in Section II. Section IV details the service provisioning framework in the NMS, while Section V delves into the various key NMS processes. Section VI presents results from a lab test-bed and the city-wide field trial, while Section VII concludes this paper.

II. Primer on the CESR

In this section, we describe the design of the CESR router, which manifests in three flavors – a core device, a metro-edge device and a customer premise equipment (see Figure 1). The CESR follows the OE concept [9, 11], using which the network topology discovered by the centralized NMS and is transformed into an auxiliary binary graph. In the Omnipresent Ethernet (OE) concept [9, 11], a network-graph

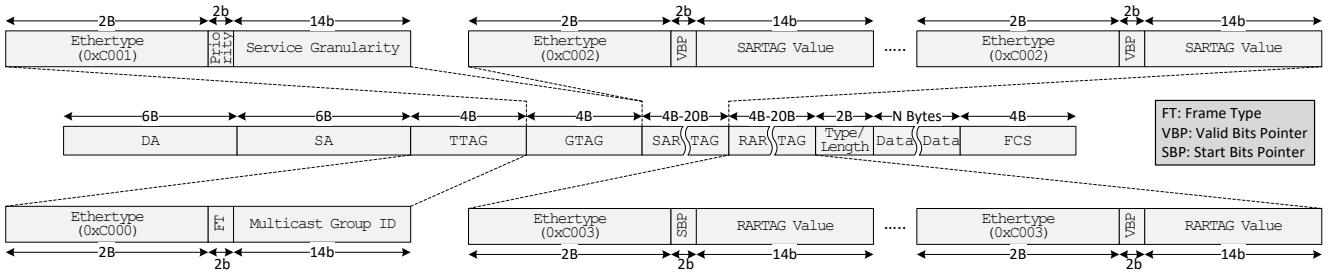

Figure 4: OE frame format PDU.

is built in a centralized *network management system* (NMS) and is converted to a binary-graph by ironing out nodes whose degree of connectivity is greater than 1×2, to nodes that have degree of connectivity either of 1×1 or 1×2. This process is described in [9, 11] and duplication is avoided here. Each physical node is periodically pinged to update the binary graph. The NMS then creates and stores routes (both working and protection paths) between every source-destination pair, which is periodically validated.

A. Port-to-Port Data Flow

The CESR hardware is developed using the methodology shown in [9, 11]. The port-to-port path followed by an OE-frame (as defined by the *protocol data unit* – PDU in Figure 4) through the CESR is as follows (see Figure 2): Any CESR port can be *classified* as an edge port (connected to client) or a core port (connected to other CESRs) for a service. Data arrives at the SFP/XFP ports (1Gbps/10Gbps) and is sent to a SERializer/DESerializer (SERDES) that converts the data into FPGA-readable format – at 4 lines of 3.125Gbps or 10 lines at 125Mbps. The heart of the CESR is the *Xilinx Virtex 6, 365T* FPGA. Once the data enters the FPGA, we perform 8b/10b or 64b/66b encoding/decoding depending on the line-rate. A GigE/10GigE MAC ensures conformity of the arriving data to the relevant Ethernet framing standard.

The data then enters a module called ***edge port logic* (EPL)**, which examines the header and infers whether the arriving data classifies the port as an edge port or a core port. This is done as follows: if the header already contains the explicit route to the destination (encoded as labels/tags as part of the header), then the arriving data is said to nomenclate the port as a core port. In contrast, if the arriving header contains no labels pertaining to how the data should be handled by the CESR, then the CESR port is an edge port. A packet is encoded with an `RARTAG` by the EPL.

Table 1: Structure of the TELL table

Service Type	Lookup Identifier	Primary Path RARTAG	Protection Path RARTAG	Rate Limiter Information (CIR, CBS)	QoS
MAC	00:00:EE:F2:2B:1C	01100001101	110000010100010	100, 3	3
IPv4	59.23.45.129	01000011110	1111111001	250, 2	4
…	…	…	…	…, …	…
…	…	…	…	…, …	…
IPv6	2470:ac2f:4	111110010	0111011100	500, 3	1
VLAN	27	00001100001	0000110010101	100, 3	3

If the port is an edge port, the EPL sends the packet to a ***reassembly engine***. This engine contains a series of SDN populated flow tables. These tables (are called *Thin Ethernet Logical Layer*, or TELL tables) are of the format shown in Table 1. Multiple identifiers in the incoming packet header are mapped against these tables. These identifiers could be IPv4, IPv6, MAC, CTAG, STAG or port IDs or any other flow identifiers that the user wants to define his service by. If a match occurs, then the labels/tags that correspond to the match are inserted in the packet.

A tag (called **`RARTAG`**) determines the binary route to the destination (see OE PDU in Figure 4). Since a tag is 32-bit long, of which 20 bits describe the route, we may use multiple tags concatenated together to describe the route to the destination. The OE PDU is an Ethernet frame with extra tags inserted (thus making the maximum transmission unit >1518 bytes). The tags or labels (depending on the model of operation) are distinguished by specific Ethertypes. If none of the identifiers of an incoming packet matches with any of the entries in the tables, the packet is stored (only for a few milliseconds) in a *new service buffer* and the corresponding request is sent to the controller. If the controller cannot provision the service corresponding to the packet's identifier, the packet and all such subsequent packets are dropped. However, if the service is provisioned by the controller, a new entry is created in the table and subsequent packets are treated as per this entry.

In case a port is an edge port for a service, or if an incoming packet already contains an RARTAG (i.e. if the port is a core port for that service), then the packet is sent to the next module within the FPGA – called the ***contention resolution logic* (CRL)**. The CRL is responsible for QoS management and scheduling traffic into the subsequent module – the ***switch fabric***. For an N-port CESR, we have $N(N-1).Q$ packet buffers in the CRL. The CRL is a vital component of the virtual output queue (VOQ) system that feed to the switch fabric. A buffer in the CRL uniquely corresponds to a particular QoS level (from among the user defined Q levels), and a particular output port (hence there are $(N-1).Q$ ports per incoming line.

The CRL is also connected to another block called the ***memory management module* (MMM)**, which interfaces the FPGA with the off-chip QDR (SRAM) memories. The goal is as follows: as far as possible, we want to keep all the data within the FPGA (for cut-through/express switching) and only when the data exceeds what the FPGA can store in its packet buffers, is the data offloaded to the off-chip QDR chips. The task of maintaining data in the QDRs, management of data structures in the QDR (which is an exact replication of the packet buffers with greater depth) is performed by the MMM.

The CRL schedules data to the $N(N-1).Q \times N$ port switch fabric using a simple ***weighted round-robin scheme***. To do so, the CRL reads the first RARTAG in the incoming packet. The most significant 20 bits correspond to the route, which are read along with a 5-bit pointer that indicates from where to begin reading the bits. The CRL then reads $2 \log N$ bits from the bit that the pointer indicates, which indicate the output port for that switch. The CRL then schedules the

packet and also increments the five pointer bits to point to the next location, so that it can be acted upon by the next CESR along the route. If the RARTAG has no more valid bits (due to the bits being exhausted along the route), then this tag is discarded and the next tag is read.

The ***switch fabric*** in the FPGA is designed using multiplexers. We use cascaded multiplexers such that these facilitate the switch fabric to route from any of the ingress $N(N-1).Q$ ports to the egress N ports. Once the packets are sent to the egress ports, there is an egress MAC instantiation that ensures conformance to the 1GigE/10GigE Ethernet standard and sends the packets to the PHY (to be processed for 8b/10b or 64b/66b encoding) and then to a SERDES (for transmission to an XFP or an SFP).

To facilitate switch action, one of the key elements of the CESR is a ***control state-machine*** **(CSM)** module in the FPGA. The CSM module communicates with the NMS in one of two ways: (a) through a dedicated element management port – essentially a RJ45 interface on the CESR, or, (b) through the control plane. One or two (for redundancy) CESR(s) is (are) connected to the NMS through the RJ45 I/O port.

The ***NMS controller*** (Figure 1) is responsible for provisioning services, discovering the network, protection, restoration, etc. Control packets are generated by the NMS controller and sent to CESRs. Whenever a new service is provisioned, the two (or more in case of multipoint) ends of the service are earmarked as *management end-points* (MEPs). For every service, the MEPs periodically (say every 3.33 milliseconds, as defined in IEEE 802.1ag Continuity and Fault Tolerance Standard) exchange a `Hello` message. Loss of three consecutive `Hello` messages indicates to the receiver a service failure and to switch to the protection path.

Control packets are generated at the NMS controller and are differentiated by the `Ethertype` field in the header of the packet. Such differentiation is standardized in IEEE 802.1ag or ITU.T.1731 Y.1731. Control packets also have the QoS bits in their first tag (called `SARTAG`) set to the highest QoS level. The control traffic is embedded within the data plane, and the CESR always gives the highest priority to control plane traffic to ensure bifurcation between the two planes. The CSM populates the SDN tables for provisioning services in the CESR.

B. List of Hardware Tables

We now define some of the key hardware tables used in the hardware, which the NMS populates based on the SDN philosophy. All of the following reconfigurable tables (except CFM table) are used in EPL, and are managed and populated by the NMS. These tables are based on the RMT principle [13] and coded in VHDL in the FPGA. In the largest CESR that we built, we did have capability to store the TELL table into a TCAM that was separately controlled by a secondary Spartan 6 150T-3 FPGA.

Port SARTAG Table: This table is a subset of the TELL table and improvises the forwarding. For a port-based service, it is time-consuming to revisit the TELL table after a lookup miss for a given data frame. Hence, the NMS can directly point to a port index of the Port SARTAG table and get the forwarding information. If a service is of type *port-based service*, then the NMS configures the *Port SARTAG* table with parameters such as port number, primary path identifier (RARTAG), protection path identifier (RARTAG), and rate-limiting index. This is the default provisioning, whereby the destination address does not find a match. Hence, the packet is forwarded to a default port (possibly to a gateway router) with the goal that an eventual match at another CESR would occur.

Multicast Table: If a service is a multicast service, then the NMS configures a *Multicast table*, which maps a multicast group to its constituent CESR ports to which the packets are to be sent. For each multicast service, the NMS designates two multicast group IDs – one each for primary and protection trees. So at each intermediate CESR in the primary and the protection tree, the NMS configures an *output port vector* for that multicast service. The output port vector is a set of ports which are intermediate ports on that CESR to achieve reachability to all the leaves of the multicast tree.

Acceptable Port PDU Table: This table is used to configure the CESR ports to ensure standards-compliance of the incoming PDUs. The configurable fields in this table (i.e. port parameters) are: (1) default priority of port; (2) connected link type (Copper or Fiber); (3) port translation type (MAC, IPv6, IPv4, VLAN, MAC+IP, MAC+VLAN). Port translation type is determined by the service provisioned on a particular port. If a port only carries IPv4/IPv6 services, the port translation type would be IPv4/IPv6. If a port carries one or more MAC and IP services, the port translation type would be MAC+IPv4/IPv6; (4) whether a port carries any port-based service; (5) auto-negotiation status of the port; (6) whether a port is a core port (connected to other CESRs) or an edge port (connected to a client) or is disconnected.

***Rate Limit Table*:** At the ingress CESR, traffic needs to be bound as per the specified bandwidth granularity of the service. The NMS configures a *Rate Limit (RL)* table that contains the service rate limiter information. Each entry in the RL table contains the following fields: (1) Rate limiting index; (2) Bandwidth limit, and, (3) Committed Burst Size (CBS) limit.

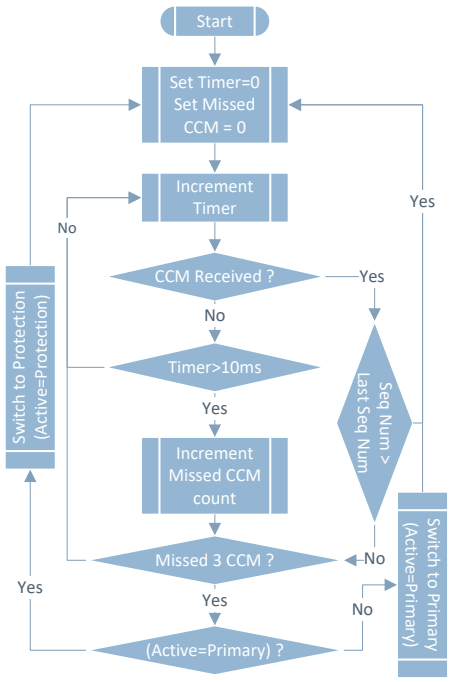

Figure 5: Flowchart of the CFM process

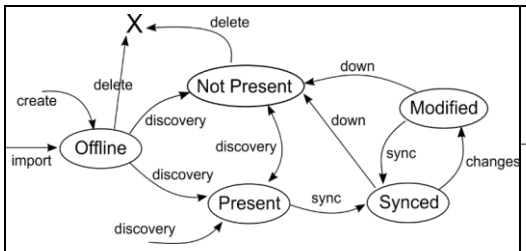

Figure 7: Node automata

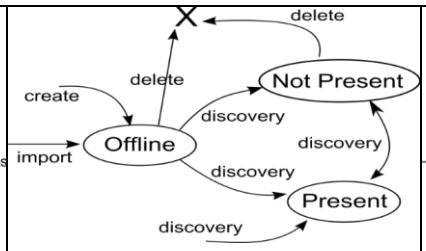

Figure 8: Link automata

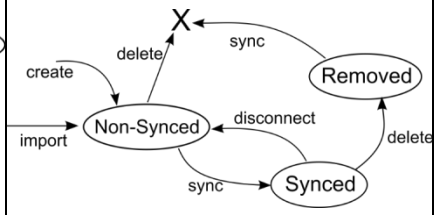

Figure 9: Service automata

Connectivity Fault Management (CFM) Table: To build CFM messages for heartbeat i.e. to ensure connectivity between two or more endpoints, the NMS configures an IEEE 802.1ag-compliant *CFM Table*, and is standardized for both PBB-TE [30, 32] and MPLS-TP implementations. The IEEE802.1ag encompasses all the aspects of OAM&P (operations, administration, maintenance and provisioning). The CFM table is used by the CESR hardware to monitor the status of the primary and protection paths. The important parameters in the CFM table are: (1) Offset number for the service (ID); (2) Route identifier of the CFM message (for unicast); (3) Multicast group ID, which results in a multicast vector at the CESR; (4) whether a corresponding service is multicast; (5) whether an entry is for an active path. Table 2 presents the logical view of the CFM table, while Figure 5 describes the CFM process.

The NMS configures two CFM table entries for each service, one for the primary path and another for the protection path. The 14th bit of the offset specifies whether the entry is for *primary* or for *protection*. For example, as shown in Table 2, for service #1, NMS has configured two CFM entries with offset number 1 and 16385. The `Active path` field of CFM entry with offset 1 is set (primary entry), while that with offset 16385 is not set (protection entry).

Table 2: Logical view of CFM Table

Offset	RARTAG#1	RARTAG#2	RARTAG#3	RARTAG#4	RARTAG#5	Global Service #	Remote Service #	# of endpoints	Start node output port	Multicast service	Active path	Entry Valid
1	0x3890	0x2480	0xC000	0xC000	0xC000	1	2	1	7	0	1	1
2	0x2380	0xE457	0x3900	0xC000	0xC000	2	5	1	3	0	1	1
…	…	…	…	…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…	…	…	…	…
16385	0x3790	0x2349	0xC000	0xC000	0xC000	1	16386	1	8	0	0	1

To configure any of the above mentioned tables, the NMS sends a `WriteFrame` to the corresponding CESR, specifying the table to be configured. The specified parameters are appended in the payload of the `WriteFrame`. The format of the `WriteFrame` is as shown in Figure 6.

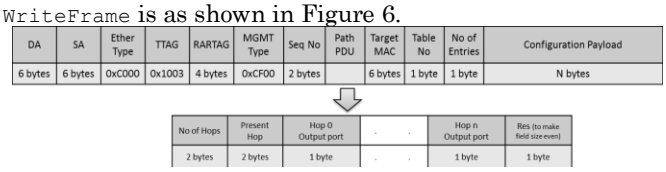

Figure 6: WriteFrame Format.

With the primer on the data forwarding plane, we are now in a position to describe the SDN-centric control plane.

III. SDN Control Plane-Based NMS Architecture

The NMS provisions services based on five parameters: IPv6, IPv4, MAC, port-based and C/S-TAG-based. It has a GUI in the front-end, and a back-end that is connected to a peer router (CESR). In each CESR, there is a control state machine that represents the backend of the NMS. A user views a palette in the front-end of the NMS in which the user sees the entire network topology; all the CESRs; and all the provisioned services as well as the backup paths. Through this palette, the user can setup services as well as maintain existing services. A user can setup services through the palette by right-clicking edge node entities for that service, and selecting the type of service that the user wants to setup, and configure bandwidth, QoS, protection-level etc. This information is then transferred to the appropriate CESRs. Work-path, protection-path selections and mapping of service identifiers are done by the NMS backend and communicated to the respective CESRs through the peer router.

The SDN-enabled hardware exposes an open API, using which the NMS configures the forwarding plane (a *Control to Data Plane Interface* (CDPI) in SDN terminology [14]). The CDPI facilitates the NMS to provide forwarding information as well as announce the hardware capabilities, network statistics reporting and event notifications to the CESRs.

The NMS is a platform-independent standalone GUI application that supports JVMs. It carries out management functions using the CDPI provided by the CESR devices. The NMS is connected to any of the CESRs present in the network and this becomes the default interconnection point. This CESR connected to the NMS is called the *peer router*. Each CESR has a dedicated management port to connect to the NMS, though only one (or two for redundancy) CESR(s) is (are) actually connected to the NMS. Communication between the NMS and all other CESRs is done through this interconnection point. The control-protocol is in-band with the data, only segregated by a control PDU. The NMS communicates with the network via the OE protocol [11], whose PDU is shown in Figure 4.

We used the `JnetPcap` library [15] to communicate with the hardware. It is a Java wrapper for `libcap` and allows communication with the SDN controller. The rest of this section describes different functional components of the architecture.

A. App Context

This module maintains global parameters required for the NMS operation. It comprises a *Topology Manager* and *Service Manager* sub-modules.

The topology manager allocates a unique ID to each network element including CESRs, client nodes and links between nodes, and manages all additions, deletions and modifications in the network elements. To ensure consistency between the network and the NMS, the topology manger maintains an automata state of each network element. The automaton of a node is shown in Figure 7. A node can be in one of five states (*offline*, *present*, *not-present*, *synced*, *modified*), while a link can be in one of three states (*offline*,

present, *not-present*). A node starts in an *offline* state when created by a user (in the *planner mode* of the NMS) or imported from a file. After a user triggers an initial discovery, nodes that are present in the physical network switch to the *present* state and the remaining nodes (i.e. those from the planning palette that are not found in the network) switch to the *not-present* state. Subsequent node discovery processes toggle the state of node from the *not-present* to the *present* state and vice versa depending on changes in the physical topology. Only nodes in *not-present* state can be deleted from the NMS. Moreover, only nodes in the *present* state are configurable by the NMS and are switched to the *synced* state after a sync operation. A *sync* operation ensures that the characteristics of the node, as defined by the NMS, are transferred to the appropriate CESR. Any changes in these nodes push them to the *modified* state, which is crucial for service provisioning. In case of service provisioning, all the nodes requiring their hardware (HW) configuration to be updated are switched to the *modified* state by the NMS. If a node goes down, the NMS switches its state in its automata to the *not-present* state. Once a node goes to the *not-present* state, it needs to be synced again after it has been (re)-discovered. This ensures consistency with the HW configuration. A link automaton (see Figure 8) also works in a similar fashion.

The service manager is responsible for: (1) creation/deletion of services, (2) state maintenance, and, (4) monitoring QoS parameters. It also provides a globally unique ID to each service in the network. To satisfy our goal of real-time consistent view of services in the network, the *service manager* maintains a similar automata state (see Figure 9) for each of the services created. All the services are either created or imported, and start with a *non-synced* state. After a sync operation, a successfully configured service is switched to the *synced* state. If the NMS is disconnected or a service's transit node goes down, the NMS marks the service as *non-synced* again. *Synced* services cannot be deleted directly from the NMS states, as the traffic for the service is continued to be allowed in the physical network. Once a user deletes a service, the NMS changes the state of this service as *removed* and marks it as *to-be-deleted*. It is only in next sync operation, that the NMS removes the deleted service from the system.

B. Hardware Information Base (HIB)

Each CESR has HW configuration tables described in Section II. The NMS maintains the details about the contents of each table present at every CESR in the form of objects. This stored state is called as the *Hardware Information Base (HIB)*. Detail descriptions of such tables are given in [9].

The NMS forms a replica of the HW configuration that needs to be written in the form of local software objects. The NMS is thus aware of any table overflow, or any invalid entry. Hence, it can flash an error to the user as a user configures the network. The *HIB* helps the NMS to avoid the misconfiguration in the actual network.

When any node or service is updated, the NMS updates the *HIB*. During sync configuration, the NMS reads this *HIB* and builds OE frames to configure the relevant routers in the network. The NMS also captures the events from the hardware and maintains synchronization between actual hardware tables and software objects. Some of the examples of such events are link failure, node failure and instances when a service gets switched from primary to protection.

C. Hardware Interface (HI)

This module is responsible for communication between the NMS controller and the CESRs. The NMS controller uses the *HI* for discovering network topology, updating node configuration, configuring services, getting the status of a service, setting QoS values, capturing changes in service state, and getting acknowledgments of operations. The *HI* uses a *frame builder* module (in the NMS) to generate OE frames for each of the operations that it needs to perform. For example, to synchronize any CESR with the NMS state, the *HI* reads the hardware table information from *HIB* and sends it to the *frame builder,* which returns an OE frame (with requisite information) to be sent to a target CESR.

The *HI* uses a physical network interface to send Ethernet frames using the `JnetPcap` library [15]. It provides a time-bound response to a request, and indicates a failure after a fixed number of unacknowledged attempts. The *HI* also receives asynchronous OE frames such as `CFMNotificationFrame` and reports to the registered process. In case of `CFMNotificationFrame`, it invokes the `CFMDaemon` described in [16].

D. Logs, Policies and Algorithms

The NMS uses three kinds of logging (using apache `log4j` library [17]): (1) On screen notification, (2) file-log, and, (3) configuration frame log. All important events such as node or service-state modifications, success of user operations are reported on-screen to users. All other general events are logged in a separate file. The NMS also logs each configuration frame sent to the network in a separate format in a `configframe.log` file, which helps debug unexpected network behavior.

Policy and *Algorithm* modules are separated from the management modules to enable developers to plug their custom policies and algorithms such as path selection for a service, traffic engineering, etc. The NMS uses the Algorithm module to calculate the primary and protection paths of a unicast service and primary and protection tree for a multicast service.

E. Authorization

NMS access requires authorization, which is implemented using the Bouncy Castle Library [19]. However, in practical scenarios, network administrators and network supervisors are different individuals. Hence, our designed NMS supports a role-based access. An encrypted file is used to store and verify the user credentials. A standard RSA algorithm [18] for encryption and decryption and a SHA algorithm [18] to make random-key pairs are employed.

IV. Service Provisioning in the NMS

Since the premise of this work is to develop an SDN framework that also supports a service provider's existing service base, we adhere to the default CE service set as defined by the Metro Ethernet Forum (MEF) [20] and map all new services to some combination of MEF defined services wherever possible. The CESR hardware primarily allows two base types of services that are defined in the MEF standard: (a) an ELINE service (point-to-point service), and, (b) an ELAN service (point-to-multipoint service). These *vanilla*

services are not suitable to cater to all types of service needs by providers. A service provider needs a variety of services to fulfill the clients' traffic requirements, especially from the SDN service definition standpoint. The NMS provides the enlisted user-defined services in addition to a standard set of services as proposed by the MEF: (a) Bi-directional protected unicast service (ELINE); (b) Bi-directional non-protected unicast service; (c) Protected multicast service (ELAN); (d) Non-protected multicast service; (e) Broadcast service; and (f) IP service with ARP.

While we argue that most user-defined services can be mapped to the aforementioned services portfolio, we also provide an interface in the NMS for implementing custom service types. An example of such custom service can be an OSPF-enabled IP service or an IP video surveillance service (that uses one IP multicast service and many MAC unicast services), etc. The developer can extend the `AbstractUserService` class provided by the NMS API and register this new service type class to the `ServiceManager` module. In this paper, we have covered an example of such custom service and deployed it in a tier-1 service provider's network (as shown in Section VI).

We now describe the different steps involved in the service provisioning operation.

A. Adding a service in the NMS

A user can establish a unicast service by providing the source and destination clients for the service. For a multicast service, the user needs to provide more than two or more clients among which the user wants to create a service.

B. Finding the service path

The centralized SDN controller maintains the overall view and state of the network, and hence is in the best position to make a decision pertaining to path selection. The SDN controller can also incorporate policies fed by a network administrator to avoid or favor some links over others. The SDN controller can also automatically distribute the traffic load across the network to maximize network utilization.

The NMS finds a service path or a service tree for the ELINE and ELAN respectively, upon receiving a service provisioning request. It also attempts to find a protection service path or protection service tree for the given service. If the NMS fails to find such a protection path/tree, it asks for a user confirmation to allow to provision the service as a *non-protected service.*

The `ServiceManager` module calculates both primary path and protection path (if required and feasible) using `PathManager` utility module and stores the information as part of the `AbstractUserService` class.

Finding the primary and protection path for a unicast service: The `PathManager` module in the NMS finds a service path/tree for all provisioned services. The `PathManager` finds the shortest path between the source and the destination using standard Breadth First Search [21] for the primary path. To find the protection path, the `PathManager` simply ignores those links that are used in the primary path. If it does not find any feasible protection path, the `PathManager` uses standard Depth-First Search [21] with the caveat of using all the links of a primary path to be used at last, so as to find a maximal edge-disjoint path.

Finding the primary and protection tree for a multicast service: The `PathManager` uses a modified Kruskal's algorithm [21] to find a spanning tree structure as a primary tree for a multicast service. In our modified Kruskal's algorithm, we start with *N* sets consisting of edge-CESRs (a CESR that is directly connected with a client). We combine two sets that have the least distance between any of the member nodes. This process is continued till a single set is obtained that forms a spanning tree across all the edge CESRs. Two sets are combined in a way that does not result in a loop. We also consider those CESRs that are in a partially combined set into consideration to calculate the minimum distance between the two sets.

For generating a protection tree for a multicast service, the same approach is used. We avoid the links that are used in the primary tree to calculate the minimum distance between two sets. If no other edge-disjoint path between any of the two sets is found, we consider the path that is used in the primary tree. This way, a fully edge-disjoint tree, if possible, is found, else a maximal edge-disjoint tree is selected [21].

C. Registration of a service

The SDN controller has to keep track of the services that are provisioned in the network. This is essential for a variety of reasons: (1) to avoid service duplication, (2) to report the number of services and their health, and, (3) to isolate services post a failure.

A `ServiceManager` class is responsible to keep track of all the services that are defined for a network. Upon finding a suitable primary and protection path/tree for a service, the NMS registers the service with the `ServiceManager`. The `ServiceManager` registers these services and denotes a unique ID for each service. This unique service ID is used to generate a hardware configuration that will eventually populate CESR tables. The hardware configuration is described in Section II. If for whatever reason during the service provisioning process, the hardware configuration generation fails for any of the CESRs, we rollback the hardware configurations of all the other CESRs that are impacted by this service. This ensures successful end-to-end consistent service provisioning.

D. Service Configuration

For provisioning a new service in the network, the NMS needs to configure the forwarding plane of the CESRs. Figure 10 shows the general data flow for the service configuration process. The user registers the request to sync a particular service with the network topology at the `ActionAgent` process through the UI module. The `ActionAgent` then queries the service manager module (mentioned in Section IV) to obtain the configuration objects for the service and passes these to the `Frame Builder` process, which creates `WriteFrame` objects (shown in Figure 6) from the received configuration and sends it to the `Pcap Injector module` for sending the frame into the network. The `Frame Parser` process parses the packets received from the `Pcap listener` process into `WriteReplyFrame` objects and notifies the `Service Manager` process about the result which in turn forwards this update to the UI module, so that it can be displayed to the user.

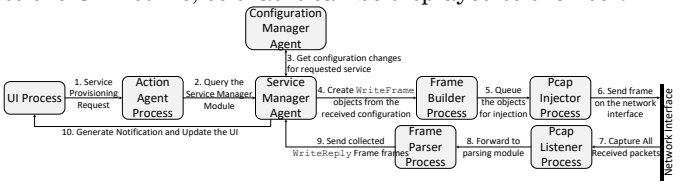

Figure 10: Service Configuration Event Flow.

The *configuration manager* generates the HW configuration for all the services. The HW configuration for a service depicts the modifications to be done to the CESR devices in order to provision the service in the network. It includes a lookup entry, a rate-limiter entry, a CFM entry and a multicast member entry (in case of multicast service only). Given below are the details of the different table entries that are generated as part of the configuration process for provisioning a particular service S.

Rate limit entry (RL Entry): This entry is used for rate limiting the traffic for a particular service. The NMS identifies the incoming ports of each CESR for a given service. It then searches for a common offset value of a free entry in the associated RL table of each of these identified ports. A 2-byte RLID is constructed by combining the common offset value and port number (4 bits of port number with 6 bits of offset value) for each port. The RL entry requires bandwidth and CBS parameters to be set that are part of the service information. This RLID is stored against the service ID S in a servRLmap data structure.

Lookup entry (TELL Entry): TELL is a lookup engine to enable mapping of the edge port packet to the OE domain. The first free TELL entry index of an ingress router is found by querying the *configuration manager*. We store an offset of the TELL entry against the service ID of S in a map. This step is repeated for an egress CESR of the same service S. For a multicast service, a common offset value across the TELL table of every CESR is computed. This offset value is required for service protection as explained later.

The `RARTAG` field in the TELL entry is acquired from an assigned service path for service S. In case of a multicast service, we put a *multicast group ID* instead of the RARTAG. The *configuration manager* provides a unique ID for each multicast service. We store the primary multicast group ID and tree protection multicast group ID against the service ID of S in the `servPrimMcastmap` and a `servProtMcastmap` data-structures, respectively.

The RLID pointer field is obtained from a hash-map of `servRLmap` by providing service ID S. The RLID pointer-field is used to restrict the exceeding traffic for the service corresponding to this TELL service entry.

PortSARTAG entry: The OE protocol provides an option for port-based service provisioning. A user can define a default port-based service in case all other translations fail, i.e., no match is found. Service information of such port-based services is stored in another table to enhance efficiency of the lookup. It is expensive (due to limited memory available in the FPGA) to re-lookup TELL entries after a match has failed. The port-based service is defined against an incoming port and hence a lookup entry can be indexed against an incoming port in a different table with an $O(1)$ lookup complexity.

Only one port-based service is provisioned for a particular port. The required fields for a port-based service are: the `PortSARTAG` entry, an `RARTAG` and a `RLID`, and these are obtained in the same way as a TELL entry. The `PortSARTAG` table has predefined fixed positions for each port, where the entries for primary and protection `RARTAG` are maintained.

Multicast member entry (MCast Table Entry): For a multicast service, we need to configure all the CESR ports that are part of either the primary or the protection tree. A uniquely identified multicast group ID for each service is supplied by the *configuration manager* while generating the TELL table entries.

Protection entry (CFM entry): The CESRs send *CCMs* between each pair of service end-points. In case of a fault in any service path/tree, the service end-points stop receiving CCMs and all the service end-points are triggered independently to switch the service to a protection path. A CFM entry is required to enable the CCMs for a service. The NMS populates two CFM entries – one for the primary path/tree and one for protection path/tree – for each service at the edge CESRs.

In case of a unicast service, the primary CFM entry offset is the same as the TELL offset for that service. The `servTELLmap` provides a TELL offset for a service S. The CFM entry for a protection path/tree is also maintained in the same CFM table with an appropriately scaled offset value. If the CFM offset of the primary entry is n, then the CFM offset of the protection entry will be $n + 16384$ (14^{th} bit in CFM offset field would be 1). The remote offset field in the CFM table is generated by the CFM entry that is used to validate the service at the egress router. As a multicast service has multiple end-points, we provide the same remote offset (and hence TELL offset). To protect a port-based service, we need to have two CFM entries for each TELL entry – one reserved and one for each of the port-based service.

If the NMS controller fails to find a free TELL, RL or Multicast entry, it reports a failure in table generation to the `ActionAgent` module.

V. Key NMS Controller Processes

The SDN controller incorporates a series of well-defined processes to govern the network. Some of the processes are to provide a global view of the network, provisioning services in the network and highlight a fault. The SDN controller is also responsible to define the process of configuring a network.

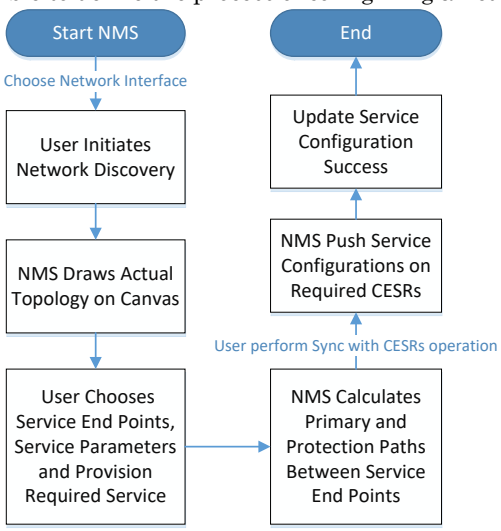

Figure 11: The NMS flowchart

Apart from service provisioning, the NMS allows several miscellaneous operations such as state discovery of the network, service monitoring for latency, fault detection in the network, peer router connectivity check, router firmware diagnostics, etc. This section details each of these processes. The NMS flowchart is presented in Figure 11.

A. *Network Discovery*

The network discovery operation identifies all the CESRs in the topology and the connectivity between them (1Gbps/10Gbps lines). Figure 12 shows the general flow of the discovery process carried out by the NMS.

The OE protocol uses `HelloFrame` and `HelloReplyFrame` for network discovery. The Discovery process after receiving the request from the UI module queries the *Frame Builder* process for the creation of `HelloFrame` object. This frame is then passed onto a *Pcap Injector* process that sends the frame to the connected CESR (peer router). The peer router replies to this `HelloFrame` with a `HelloReplyFrame` and forwards a copy of a `HelloFrame` to all its connected neighbors. All adjacent routers reply with a `HelloReplyFrame` to the router that sent a `HelloFrame`.

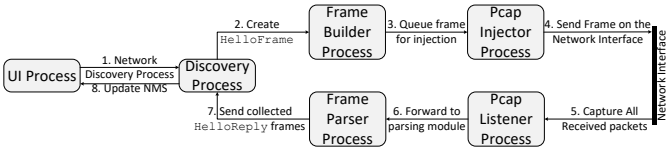

Figure 12: Network Discovery Process Flow.

The *Pcap listener* process collects all the received frames and forwards them to the *Frame Parser* process. The parser process then identifies all the `HelloReplyFrame` objects received during the discovery process and sends them back to the *Discovery* process. The *Discovery* process then analyses the received list of reply frames to build a virtual topology and updates the UI.

A `HelloReplyFrame` includes the MAC address of a CESR along with hardware capabilities of the CESR such as port attributes. Port attributes include port status (whether it is connected/edge/core), MAC address of a connected CESR (if a port is connected), remote port number of a connected router (if port is connected) and auto-negotiation information (if any).

Each CESR stores sequence numbers of the last *two* `HelloFrames` that it has received to avoid loops.

When a user triggers a network discovery, the NMS will also synchronize the on-screen topology with the actual physical topology.

One of the major requirements of any service provider is to get a global view of their network. The NMS maintains the real-time network status by continuously synchronizing the physical network topology to the on-screen topology. The NMS generates an updated on-screen topology from the `HelloReplyFrames` collected from the network after a network discovery process. The NMS identifies each node uniquely by a pre-configured MAC address and indicates an error, if nodes with a MAC address conflict are discovered.

The NMS considers fiber-plant as part of its optimization tool for service provisioning. Fiber losses are first computed for the shortest path. Thereafter, amplifiers are added leading to OSNR computation. If the OSNR is above the achievable-BER threshold, then we select the appropriate wavelength. If, however, the OSNR is not within the requisite threshold, then we add OEO conversion sites based on a heuristic model. Subsequently, wavelengths are selected between OEO spans to provision the traffic. The procedure is repeated for the protection path that is chosen as node and edge disjoint from the work path.

Shown in Figure 13 are snapshots of the optical layer provisioning of a service. In particular, note the fiber loss and OSNR computation for the primary and protection paths.

B. *Service Monitoring*

For a service, we monitor its path (either primary or protection), and, its QoS parameters (such as latency, jitter etc.). At the time of discovery, the NMS sends a `ReadFrame` for each of the services configured with the value of the *read operation field* as *service status* to the ingress and egress CESR. The control plane processes in each CESR reads the `ReadFrame` and returns a `ReadReplyFrame`. A *pathStatus* field in `ReadReplyFrame` tells the NMS controller about the current route used by the service. The route status needs to be same in reply of the ingress and the egress CESR as per the MEF standard. The NMS flashes an error if an inconsistency is observed. This operation is also carried out when there is a link failure.

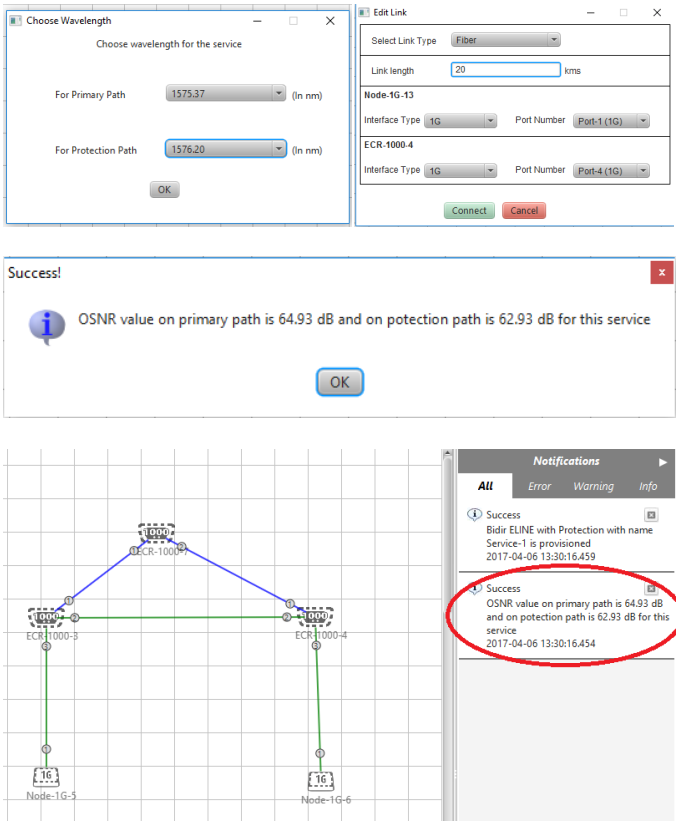

Figure 13: Snapshots of optical layer provisioning of a service: Notification Popup Window (middle); Wavelength selection for primary and protection paths (top-left); Setting link characteristics (top-right); Notification in the pallette sidepanel (bottom)

The NMS also supports real-time *latency monitoring* as a part of QoS monitoring. When a user begins to monitor a service, the NMS generates a special `ReadFrame` based on the service path. The NMS sends this `ReadFrame` to the ingress node of that service. The control plane of that node accepts the `ReadFrame` and creates a data frame with the data field empty. It also adds an appropriate OE header, so that the data frame chooses the existing service path (using the RARTAG in the OE-header). An ingress CESR gets the OE header information from the `ReadFrame`. The *control plane state machine* at the egress CESR returns this dummy data frame back to the ingress CESR. By calculating the difference between the sent and received frames, the ingress node

computes the latency of a service path and reports it backs to the NMS with a `ReadReply` frame. Note that all intermediate CESRs treat the frame at the highest QoS (#7) level.

The NMS captures the `ReadReply` frame and matches the sequence number with the `ReadFrame`. If it is a valid reply frame, it fetches the latency value provided in the `ReadReply` frame. The NMS periodically sends `ReadFrame`s to capture the real time latency.

If a service is disrupted while monitoring is in progress due to link or node failure, the NMS captures the *CFM notification* and updates the `ReadFrame`. The updated `ReadFrame` would have the requisite information to follow the protection path instead of the primary path that was disrupted. The NMS alerts the user about the active path of a service whether it is a primary path or a protection path.

C. Fault Detection

One of the advantages of having a SDN controller-based network is to automatically detect and counteract in case of a failure. The SDN controller is aware of the global state of the network and it can re-route the traffic if necessary or report an alarm or trigger a pre-defined process post a failure event.

In terms of faults, we assume all types of faults – fiber cuts, port failures, node failures, pluggable device failures and service layer faults. The NMS facilitates restoration after all types of faults by the deployment of IEEE 802.1ag – the connectivity fault management standard that facilitates service level protection, which takes care of all types of underlying faults. As part of the IEEE 802.1ag standard, we assume faults are at a service-level – which means that all types of faults that affect a service, i.e. fiber cuts, node/equipment failures, port failures and pluggable device failures are included. The way 802.1ag works is that we define maintenance end-points at the two or more (in case of ELAN) ends of a service. Between these two end-points, we exchange heart-beat messages at 3.33 ms interval. Loss of 3 consecutive heart-beat messages signal a failure to the receiver which then switches from the primary path to the protection path. As part of our design, we take into consideration fiber routes, wavelength planning etc. as well as ensure that the work and protection path are both node *and* edge disjoint. The work in [49] describes specific algorithms that converge in real-time to obtain a protection path for every ELINE and ELAN that is edge and node disjoint when possible. In this way, we are able to take into consideration all types of faults by abstracting these to service level faults. We point out that this type of fault rectification is commonly used in other Carrier Ethernet, MPLS and OTN manifestations.

Carrier class networks require network traffic to be rerouted within 50 milliseconds to the next alternate path post a failure. The controller cannot compute and re-configure the network in such a short span. Hence, the NMS pre-computes and provisions the protection path. The ingress and egress CESRs of the impacted path switch the traffic to the alternative path when a fault occurs.

Additionally, the NMS highlights the type and location of the failure. The OE framework has implemented ITU-T Y.1731 [22] and IEEE 802.1ag [16] standards for fault detections. CESRs report back with `CFMNotificationFrame` to the NMS when a fault occurs. The NMS needs to capture these events and process accordingly. Service disruption may happen due to link or node failure in a network.

The CESR sends Connectivity Check Messages (CCMs) among end-points of a service. The duration between two consecutive CCMs is 3.33 milliseconds. If any of the edge routers does not get *three* consecutive CCMs, it switches service traffic to the pre-provisioned protection path and generates a `CFMNotificationFrame` and sends it to the NMS. This `CFMNotificationFrame` includes the service number of the service that is disrupted and the newer active path (primary path or protection path).

When the NMS boots up, it starts a *CFM capture daemon process* along with other processes. This daemon continuously looks for `CFMNotificationFrame` from the network and adds it to the processing queue. The core manager of the NMS picks the `CFMNotificationFrame` from this queue and detects number of the services that are disrupted. The NMS also changes the data carrying path status of the disrupted service from primary to protection path or vice versa as per the information received in `CFMNotificationFrame`.

However, the NMS has not yet diagnosed the cause of this disruption as the `CFMNotificationFrame` messages do not give information of the type of failure. To diagnose failure, the NMS triggers a new discovery process. On completion of the discovery process, the NMS reports the type and location of a failure (by a method of elimination).

In the CESR based network, we provide protection on a per-service basis. While it may be best to abstract shared risk link group (SRLGs) in practice, however, due to the packet nature of CE traffic and the deployment of 802.1ag as a connectivity and fault management mechanism, we are able to better protect traffic – at the services level. Hence, the way we consider protection in this network is at a much finer level – at the service level as opposed to protection at a SRLG level which is more coarse. It may initially appear that such kind of per-service protection is more difficult to achieve, but the distributed nature of data-plane along with 802.1ag and Y.1731 markers for end points allows the scheme to scale to even a million+ services in a metropolitan domain.

Despite the fact that the CESR network primarily works on service level protection it is imperative to point out that we must take into consideration reservation of resources in fibers as well as in equipment (involved in protection). To this end, it is assumed that there is an orchestration layer which takes into consideration network-level protection and associated capacity planning at nodes, specifically computing the number of line-cards, cross-connects etc. requirement for restoration. Finally, the SDN controller, must take into consideration overall network availability amidst failure of entire links. To this end, our tool takes care not to route work and protection paths in the same fiber and across the same set of equipment (node and edge disjoint). However, achieving such degree of node and edge disjoint situation may not always be feasible, and hence the tool triggers an alarm whenever such routing of the protection path is not fully node/edge disjoint with the work path, though the tool allows provisioning of traffic in such a situation with the risk that a common node/edge that goes down will take down both the work and protection path with it.

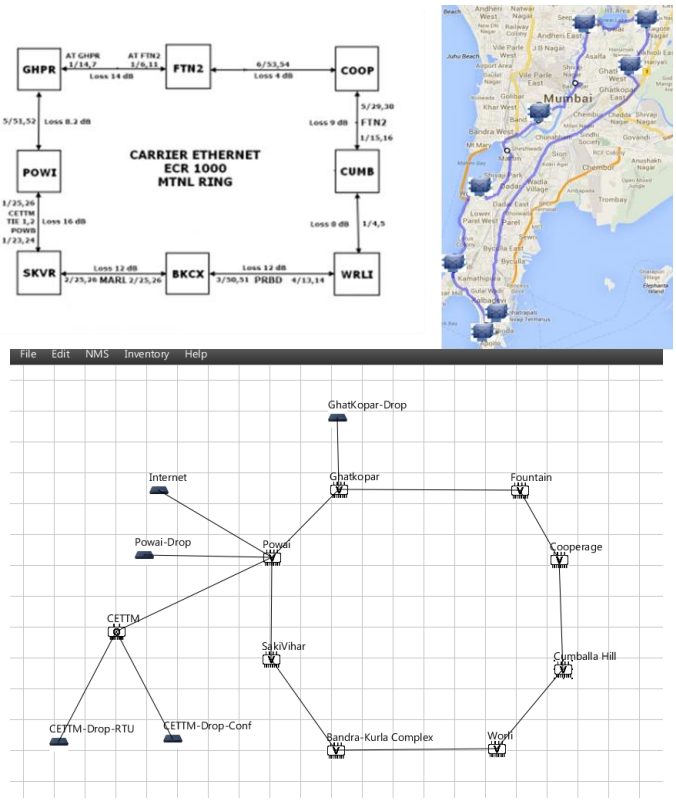

Figure 14: Network topology (top) and its geographical map (bottom).

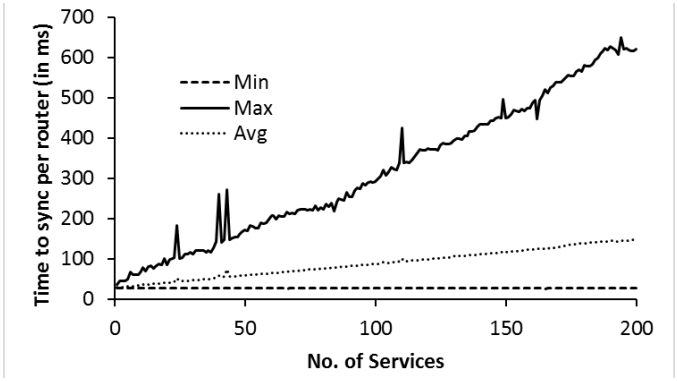

Figure 15: Sync time for various numbers of services.

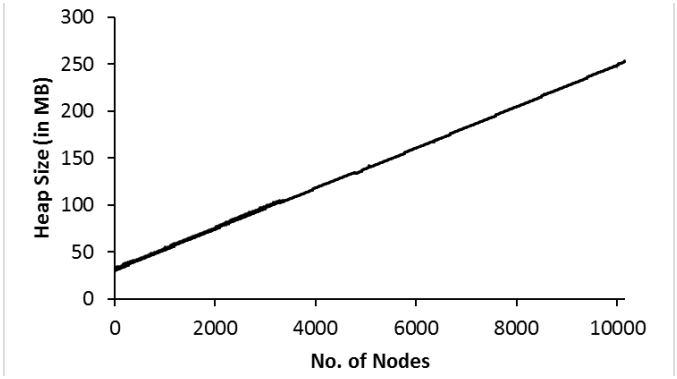

Figure 16: Heap size for various network size.

D. *Peer Router Status Check*

The NMS configures the service and other information in a network using a `WriteFrame`. The NMS sends a `WriteFrame` with appropriate configuration payload to configure any CESR. If that CESR is not directly connected with the NMS, it needs to append route information in the `WriteFrame`.

The NMS builds the required configuration `WriteFrame` and sends it to the connected CESR (peer router). The peer router forwards the frame based on the path PDU specified in the `WriteFrame` to the next connected CESR and so on. To accomplish this task, the NMS: requires the information of the peer router, and, requires the path PDU from peer router to the router where the configuration needs to be written.

The `PathPDUAgent` can formulate the path PDU between any two routers based on the path between them. The path PDU is a vector of output ports at each node in the path to a destination CESR. The NMS uses a `PeerDaemon` process to know the peer router.

The `PeerDaemon` process continuously sends a `HelloFrame` on the connected link with a time-to-live (TTL) field set to unity. The TTL value of 1 ensures that a CESR router does not forward the frame to other routers and hence prevents unnecessary flooding of frames. The NMS receives the `HelloReplyFrame` and detects the MAC address of the connected router and marks that router as peer router in the topology. All path PDU calculations are done considering the peer router as source router for any destination router.

There are two other advantages of the peer daemon: (1) to check whether NMS is connected to a network and, (2) to check the change in the NMS interconnection point (peer). When a user disconnects the existing NMS and reconnects it to a new CESR – which now becomes the peer router, the NMS auto-detects this change in the default interconnection point and re-marks the CESR as the peer router.

E. *Remote Router Upgrade*

The CESRs are FPGA-based and hence after new feature additions there might be a case where routers need to be upgraded to a new firmware. It is cost-effective to have a facility by which the NMS can upgrade all the routers in a network remotely.

A user selects the CESR type (from the three types that we built) and the board-type (based on the target FPGA). The NMS shows list of CESRs which conforms to the applied filter along with their status. The user needs to supply a firmware file which is then securely sent to all the target CESRs, one-by-one.

The NMS matches the board-type of the existing router firmware with the provided firmware file (called as bitfile) to avoid any inconsistency. Any mismatch in board-type is reported and the firmware upgrade process is aborted by the NMS. The NMS also validates the signature header of the provided bitfile before passing it onto the hardware for upgrade processing.

The NMS first erases all the sectors in the CESR containing older firmware and then updates it to a newer version. The NMS also keeps notifying the user with the progress of firmware upgrade process. The NMS uses `WriteFrame` with value of *table number* field equal to 0×05 to upgrade the CESR device firmware.

Table 3: List of Signals between NMS and CESR

Signals	From	To	Remarks
`Hello Frame`	NMS	CESR	To discover CESRs in the network.
`Hello Reply frame`	CESR	NMS	To inform CESR's capabilities and connections to NMS.
`Write Frame`	NMS	CESR	To write configurations to different tables/flash in a CESR.

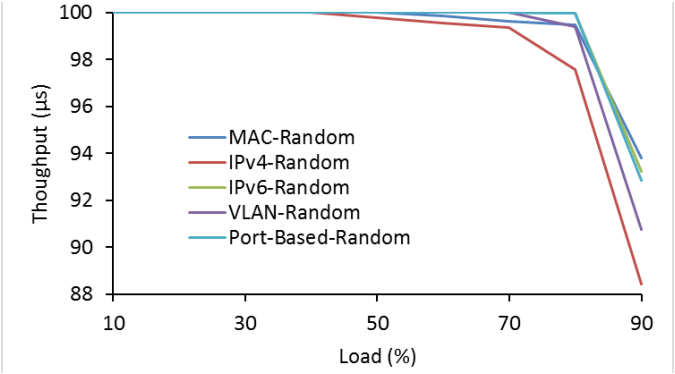

Figure 17: Throughput of SDN services.

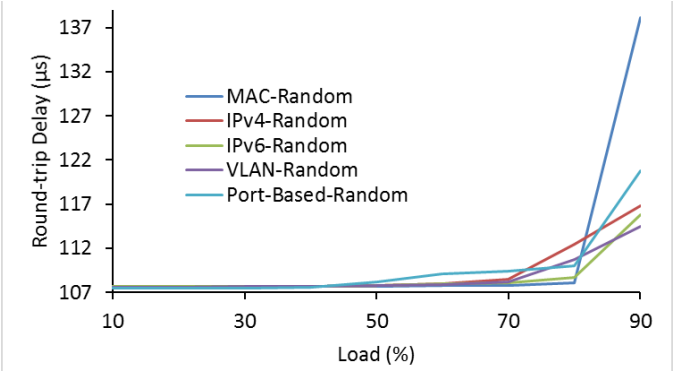

Figure 18: Latency of SDN services.

Read Frame	NMS	CESR	To query path status (primary/protection) and latency monitoring of a service.
Write Reply Frame	CESR	NMS	When configuration is successfully done, CESR sends this frame as an acknowledgment.
Read Reply Frame	CESR	NMS	CESR acknowledges NMS with the latency or the path information of the service with this frame.
CFM Notification frame	CESR	NMS	In case of any faults in the service CESR reports back to NMS with this frame.

NMS to CESR commands are specific and can be considered to be evolved after considering both OpenFlow and Protocol Oblivious Forwarding (POF) [51]. While OpenFlow 1.5 has a large number of commands that the controller uses towards the devices (SDN-compatible), POF has a subset of 3-4 commands that facilitates all interactions such as set, reset, get and delete. However, none of these protocols consider the carrier-class behavior, i.e. deterministic forwarding, protocol oblivious support at the data-plane and 50ms protection as a default. We consider each of the aforementioned carrier-class traits and this is what makes our approach different and in some way more suited towards provider networks.

A list of key commands and signals is postulated in Table 3.

VI. Simulation and Results

To evaluate the NMS, we adopt two test methodologies – a lab based test-bed with fully loaded traffic scenarios that includes a partially created dummy test-bed to augment the field-trial shown in Figure 14 that evaluates scalability of the NMS controller.

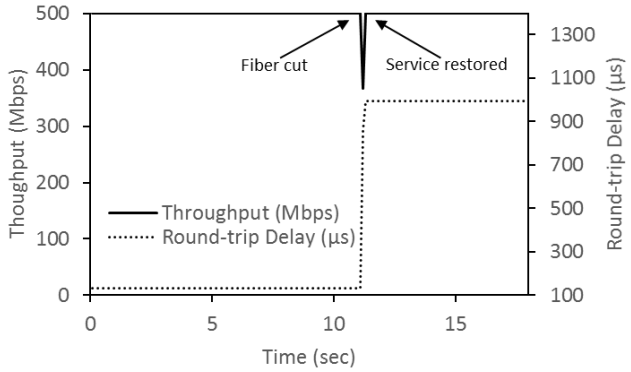

Figure 19: Sub-50 ms service protection and restoration.

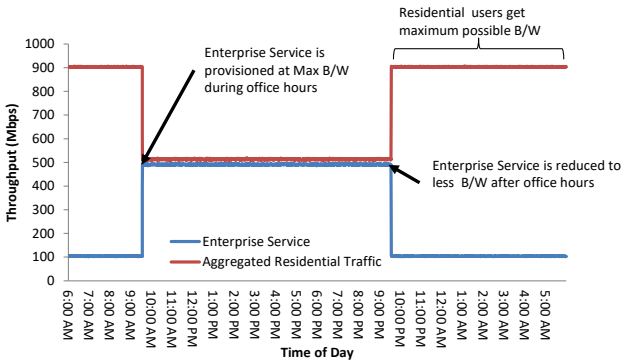

Figure 20: BoD service traffic results.

A. Lab-Based Test-bed

We have setup a 27-node COST [23] topology and connected 10 clients that pump in 1.5 Tbps of traffic (shown in Figure 21) using JDSU testers [29]. A script was written to provision services and configure the network repeatedly. Each service is provisioned between two randomly chosen clients among the 10 clients in the network.

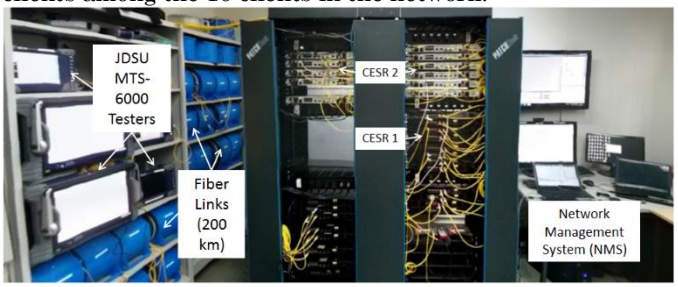

Figure 21: Wide angle view of the network set-up.

We measured the time to configure the network. As shown in Figure 15, *Min* shows the lowest time required to configure any CESR in the network. This indicates the time required to configure the CESR that is connected to the NMS (peer router). *Max* shows the maximum time required to configure any CESR in the network. This might be the time required to configure the farthest router in the network but it can also be the CESR that needs highest number of configuration frames. *Avg* shows the time required to configure the network divided by the number of CESRs in the network. The results indicate conformance to what a service provider expects in terms of provisioning time. Service set up is less than 100 ms on average for simultaneously setting up large number of services.

We also measured the memory required by the NMS for various sizes of the network. This result is important to predict scalability of the NMS controller. To calculate the

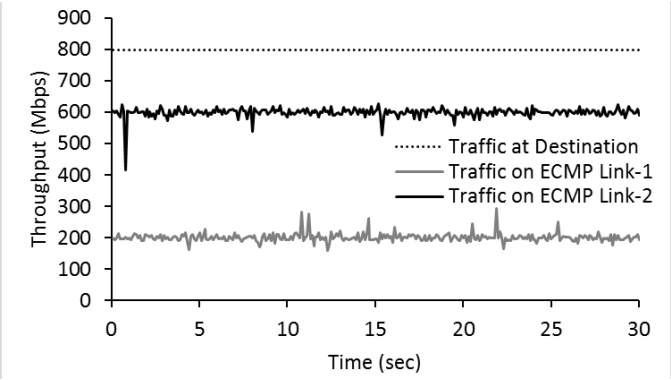

Figure 22: ECMP load balancing as a result of perfect CESR provisioning.

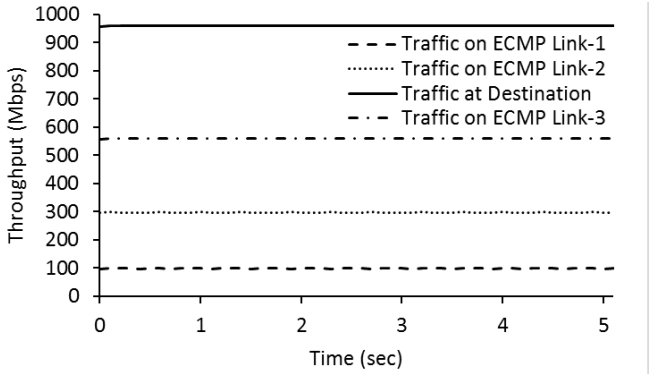

Figure 23: ECMP load balancing as a result of perfect CESR provisioning.

heap size consumed by the NMS controller we have used Java's in-built *runtime* functionality. Figure 16 shows that memory requirement is linear with respect to network size even for a network of 10,000 nodes and a million services. Each additional node in the network increases 200KB heap size as we store all hardware configuration offline in HIB.

B. Deployment Results

We have deployed our SDN-based NMS with 8 CESRs in MTNL – a tier-1 service provider in the city of Mumbai, India (see Figure 14). Our goal was to facilitate the service provider to simplify service provisioning, reduce cost and enable the creation of a new services portfolio.

Shown in Figure 14 is the ring configuration of the deployed CESR based network across the city. As can be seen a 2-fiber ring with 8-nodes in each ring has been built across the city of Mumbai with a total fiber layout of 130 km. Much of this fiber is old single mode fiber with attenuation as high as 0.5-0.6dB/km. The backbone can support 4×10 Gbps bandwidth using colored optics plugged using simple couplers and filters over dark-fiber, but with no optical amplifiers. This means that CESRs are connected to their adjacent nodes in the ring in a daisy chain structure. Each CESR can support up to 96 Gbps traffic and all the CESRs are connected to the centralized NMS controller that runs our NMS application for service provisioning through a peer CESR. The NMS accepts service requests from the network and provisions these as standard ELINE or ELAN services as per MEF specification [20] and opens scripts that the user can use to configure a service as per his requirement.

Shown in Figure 17 and Figure 18 are graphs for round-trip delay and throughput for 5-types of ELINE services set up by the NMS controller based on MAC, IPv4/v6, VLAN and Port based identifiers for ran dom frame sizes (MTU=1518 bytes). The ELINE services are all provisioned at cumulative bandwidth of 10Gbps and increments of 10Mbps.

Figure 19 shows the results of sub-50 ms service protection and restoration. An ELINE service is configured at 500 Mbps between consecutive nodes in the ring network. The fiber towards the shortest path of the service was removed to trigger protection path switching. Detection of service failure is implemented using periodic Connectivity Check Messages (CCMs) as per IEEE 802.1ag standard. Since the NMS pre-configures the protection path for every service, protection switching is hardware based and does not require any computation at the source node leading to sub-50 ms service restoration time. The minor service disruption as seen by reduced throughput in Figure 19 is due to the time required to detect the service failure whereas after detection of service failure, protection switchover takes only a few 10s of micro-seconds.

In addition, to regular services, we provisioned next generation SDN-enabled services on the deployed network that can lead to efficient utilization of the network infrastructure. To this end we proposed two new services: (1) Bandwidth on demand (BoD) and automated bandwidth sharing between enterprise and residential users, and, (2) ECMP-based load balancing.

For the first service we assume two types of customers – (1) enterprise customers that require guaranteed bandwidth during certain period of the day (or day of the week), and, (2) residential broadband users that are provided a best-effort service with a bandwidth limit (e.g. up to 10Mbps). The enterprise customers do not use the peak bandwidth after office hours and hence may not want to pay for the large bandwidth pipe. Similarly, residential users may not be using high-bandwidth applications (e.g. video on demand, video calls, etc.) during the office hours but may use them after-office hours. This kind of a scenario is common in a metropolitan area like Mumbai. In this scenario, the service provider can accommodate many residential customers on the same bandwidth pipe which is to be shared between enterprise users and residential users spread across the network. With the SDN-based CESR network, the service provider can dynamically provision guaranteed bandwidth requirements for enterprise customers as per their requirement and remaining bandwidth is used for the best-effort service provided to the residential users.

To implement this bandwidth on demand (BoD) scenario, we assume ELINE-based services across the network provisioned from a common gateway (controller) which dynamically varies resource usage based on an SDN API that is fed time-sensitive reactive scripts. Figure 20 shows the bandwidth received by enterprise and residential users based on the time of the day. The enterprise customer's service is configured at 450 Mbps during office hours and it is reduced to 100 Mbps after office hours. The residential users share the remaining bandwidth and obtain high bandwidth (close to the bandwidth limit of their service) after office hours. This script can be further extended to compute traffic requirements of the enterprise customers through bandwidth monitoring such that more residential customers can be packed into the best-effort bandwidth pipe. The impact of this SDN-based service is that it facilitates the over-provisioning ratio in the access part of the residential network to be

healthy when most required, i.e. in the morning and evening/night.

ECMP is an important feature for service provider networks to achieve inverse multiplexing and avoiding bandwidth bottlenecks within the network. ECMP balances traffic across given paths to optimally utilize the network. ECMP features of CESR are enhanced by SDN to perform various traffic engineering tasks. Results in Figure 22 showcase the performance of an ECMP load balancing service which is pre-configured to split traffic (in the ratio of 75:25) across two ECMP paths. Figure 25 shows the performance of similar services where the traffic is split in the ratio 60:30:10 across three ECMP paths. The traffic at the receiver has minimal acceptable jitter and shows stability in the result. Figure 20, Figure 22 and Figure 23 show the advantages in terms of service delivery with SDN that also leads to significant reduction in operational overhead due to the centralized NMS controller and the programmable SDN APIs. Note that conventional CE does not support ECMP, and it is only through the use of the SDN API that ECMP is facilitated in a CESR network.

VII. Conclusion

In this paper, we have described an SDN-based NMS that we built to support a tier-1 provider. The NMS is developed in conjunction with hardware that was reported in [9]. The NMS is deployed in the service providers' network with improvement of the services portfolio, due to the SDNization of the network. The NMS is described in this paper in detail. The work is important because it hyphenates existing network gear (CESRs) to the SDN philosophy thus making an important series of services use cases that inculcate standardized technology. We describe the working of the NMS, its internal functioning, some of the key data structures that facilitate the working of the NMS as well as dwell upon key design choices. Results from a testbed for scalability are included, in addition to results from an actual network.

References

[1] B.A.A. Nunes, et al. "A survey of software-defined networking: Past, present, and future of programmable networks," IEEE Commun. Srvys. & Tuts., Vol. 16, No. 3, pp. 1617-1634, 2014.

[2] N. Feamster, J. Rexford, and E. Zegura. "The road to SDN: an intellectual history of programmable networks," ACM SIGCOMM, Vol. 44, No. 2, pp. 87-98, 2014.

[3] Software-Defined Networking (SDN) Definition [Online], Open Networking Foundation. Available: https://www.opennetworking.org/sdn-resources/sdn-definition

[4] M. Subramanian, "Network management: principles and practice", Pearson Education India, 2010.

[5] B. Han, et al. "Network function virtualization: Challenges and opportunities for innovations," IEEE Commun. Mag., Vol. 53, No. 2, pp 90-97, Feb 2015.

[6] I. F. Akyildiz, et. al., "A roadmap for traffic engineering in SDN-OpenFlow networks," Elsevier Computer Networks, Vol. 71, pp 1389-1286, October 2014.

[7] D. Levin, et al. "Incremental SDN deployment in enterprise networks," ACM SIGCOMM, Vol. 43, No. 4, pp. 473-474, ACM, 2013.

[8] S. Bidkar, C. Vaishampayan, and A. Gumaste, "Demonstrating OpenFlow over a Carrier Ethernet Switch Router (CESR) – A Services Perspective," *Invited paper*, Networking and Optical Communications (NOC), Valencia Spain, June 2012

[9] S. Bidkar et. al, "On the Design, Implementation, Analysis, and Prototyping of a 1- µs, Energy-Efficient, Carrier-Class Optical-Ethernet Switch Router," IEEE/OSA Journ. of Lightwave Tech. Vol. 32, Issue 17, pp. 3043-3060 (2014).

[10] S. Vissicchio, et al. "Central control over distributed routing." Proceedings of the ACM Sigcomm, 2015.

[11] A. Gumaste et al. "Omnipresent Ethernet—Technology choices for future end-to-end networking." IEEE/OSA Journ. of Lightwave Technology, Vol. 28, No. 8, pp. 1261-1277, 2010.

[12] S. Bidkar, et al. "Field trial of a software defined network (SDN) using Carrier Ethernet and segment routing in a tier-1 provider," IEEE GLOBECOM, Dec. 2014.

[13] P. Bosshart, et al. "Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN," ACM SIGCOMM Computer Communication Review, Vol. 43. No. 4, pp. 99-110, 2013.

[14] OpenFlow v1.5 Specification [Online]. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf

[15] jNetPcap Library [Online]. Available: http://jnetpcap.com/

[16] IEEE 802.1ag - Connectivity Fault Management [Online]. http://www.ieee802.org/1/pages/802.1ag.html

[17] Apache Log4j 2 [Online]. http://logging.apache.org/log4j/2.x/

[18] N. Ferguson, Bruce Schneier, and Tadayoshi Kohno. *Cryptography engineering*. John Wiley & Sons, 2010.

[19] Bouncy Castle Library [Online]. https://www.bouncycastle.org/

[20] MEF Technical Specifications [Online]. https://www.mef.net/carrier-ethernet/technical-specifications

[21] T. H. Cormen, et. al, *Introduction to algorithms*. MIT press, 2009 3rd ed..

[22] Y.1731: OAM func. and mechanisms for Ethernet based networks [Online]. https://www.itu.int/rec/T-REC-Y.1731/en

[23] I. Tsirilakis, C. Mas and I. Tomkos. "Cost comparison of ip/wdm vs. ip/otn for european backbone networks," Proc. of 7th Int'l Conference on Transparent Optical Networks, 2005.

[24] ECIL Routers [Online]. Available: http://www.ecil.co.in/press_releases/Press_Clippings_Routers_in_MTNL2013.pdf

[25] E. Haleplidis, et al., "Network Programmability with ForCES," IEEE Commun. Srvys. & Tuts, Vol. 17, No. 3, pp. 1423-1440, 2015.

[26] S. Das, G. Parulkar, N. McKeown, "Rethinking IP core networks,"IEEE/OSA Journ. of Opt. Comm. Networking, Vol. 5 No 12.,pp 1431 - 1442, Dec. 2013.

[27] A. Gumaste and S. Akhtar, "Evolution of Packet-Optical Integration in Backbone and Metropolitan High-Speed Networks: A Standards Perspective," IEEE Commun. Mag., Vol. 51, No. 11, pp. 105-111, Nov. 2013.

[28] M. Tang, G. Zhang and J. Liu, "Space-Stretch Trade-off Optimization for Routing in Internet-Like Graphs," in Journ. of Communication and Networks, Vol. 14, No. 5, October 2012.

[29] JDSU MTS-6000A Compact Network Test Platform [Online]. Available: http://www.jdsu.com/en-us/test-and-measurement/products/a-z-product-list/Pages/mts-6000a.aspx.

[30] IEEE Standard 802.1Qay-Sept. 2009, "Amendment 10: Provider Backbone Bridge Traffic Engineering".

[31] IETF RFC 5921: A Framework for MPLS in Transport Networks.

[32] D. Bhamare, A. Gumaste, M. Krishnamoorthy and N. Dayama, "Optimal and Heuristic Techniques for the Backbone VLAN Identifier (BVID) Allocation in 802.1Qay Provider Backbone Bridged – Traffic Engineered Networks," IEEE Trans. on Network and Service Management, Vol.11, No.2, pp.172-187, June 2014.

[33] SDN Architecture. [Online] Available at: https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/SDN-architecture-overview-1.0.pdf., Open Networking Foundation.

[34] A. Singla, et. al. “Scalable routing on flat names,” In Proceedings of the 6th International Conference on emerging Networking EXperiments and Technologies (Co-NEXT), 2010.
[35] K. Diego, et al. "Software-defined networking: A comprehensive survey," Proc. of the IEEE, vol.103, no.1, pp.14-76, Jan. 2015.
[36] J. Dix, “Clarifying the role of software-defined networking northbound APIs,” May 2013. [Online]. Available: http://www.networkworld.com/news/2013/050213-sherwood-269366.html
[37] SDN Migration Considerations and Use Cases [Online], Open Networking Foundation, November 21, 2014. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/solution-briefs/sb-sdn-migration-use-cases.pdf
[38] A. Gumaste and T. Antony, DWDM Network Design and Engineering Solutions, Ciscopress, McMillan Ed 1, Dec. 2002.
[39] G. Suwala, and G. Swallow. "SONET/SDH-like resilience for IP networks: a survey of traffic protection mechanisms," IEEE Network Mag., Vol. 18, No. 2, pp. 20-25, 2004.
[40] L. Fang, R. Zhang, and M.Taylor, "The evolution of carrier Ethernet services-requirements and deployment case studies," IEEE Commun. Mag., Vol. 46. No. 3, pp. 69-76, 2008.
[41] M. Jarschel, et al. "Interfaces, attributes, and use cases: A compass for SDN," IEEE Commun. Mag., Vol. 52, No. 6, pp. 210-217, 2014.
[42] OpenDayLight, http://www.opendaylight.org
[43] Indigo: Open source openflow switches [Online]. Available: http://www.openflowhub.org/display/Indigo/.
[44] Ankur Kumar Nayak, et al. "Resonance: dynamic access control for enterprise networks," Proceedings of the 1st ACM workshop on Research on enterprise networking, 2009.
[45] B. Raghavan, et al. "Software-defined internet architecture: decoupling architecture from infrastructure," Proceedings of the 11th ACM Workshop on Hot-Topics in Networks (HoTNets), 2012.
[46] Controller performance comparisons [online]. Available: http://www.openflow.org/wk/index.php/Controller Performance Comparisons.
[47] Advait Dixit, et al. "Towards an elastic distributed SDN controller," ACM SIGCOMM Computer Communication Review, Vol. 43, No. 4, pp. 7-12, 2013.
[48] V. Gurbani, et al. "Abstracting network state in Software Defined Networks (SDN) for rendezvous services," IEEE International Conference on Communications (ICC), 2012.
[49] S. Joshi, S. Mehta, T. Das and Ashwin Gumaste, "On Control Plane Algorithms for Carrier Ethernet Networks: Unicast, Multicast provisioning and Control Traffic Reduction," in Optical Switching and Networking Journal, Dec. 2016.
[50] P. Berde et al., “ONOS: towards an open, distributed SDN OS,” In Proceedings of the 3rd ACM Workshop on Hot topics in Software Defined Networking, August 2014.
[51] H. Song, "Protocol-oblivious forwarding: Unleash the power of SDN through a future-proof forwarding plane," In Proceedings of the 2nd ACM SIGCOMM Workshop on Hot topics in Software Defined Networking, pp. 127-132, 2013.