FestNet: A Flexible and Efficient Sliced Transport Network

Hao Jiang*, Nakjung Choi†, Marina Thottan† and Jacobus Van der Merwe*
*School of Computing
University of Utah, Salt Lake City, United States
Email: {j42672, kobus}@cs.utah.edu
†Network Intelligence and Control Systems Research Group
Nokia Bell Labs, New Providence, United States
Email: {nakjung.choi, marina.thottan}@nokia-bell-labs.com

***Abstract*—Network slicing was adopted as a solution for future networks to support various applications with diverse requirements. While active research has focused on this functionality, most of the work targets RAN or packet core slicing and leaves the transport network nearly untouched. With packet core functions moving to data centers and parts of RAN functions moving to edge clouds, the transport network will gain significantly more importance. To ensure stringent service level agreements (SLAs) and facilitate slice management, it is imperative for the transport network to support live slice mobility with no disruption of current services. In this paper, we present FestNet, a Flexible and Efficient Sliced Transport Network, achieved by our virtualized Programmable Data Plane (vPDP) and two-layer design. Not only conventional protocols but also stateless and stateful network functions (NFs) can be integrated as slices in FestNet. We implemented a FestNet prototype. Our evaluation shows that FestNet supports live slice migration with no packet loss and no state loss for stateful slices and that FestNet provides many times faster slice operations than implementations in related work.**

I. INTRODUCTION

It is envisioned that a significant number of diverse IoT devices will be simultaneously connected to future networks. The coexistence of user- and machine-centric applications implies a broad range of throughput, latency, and reliability requirements. Today's one-size-fits-all network is no longer able to accommodate such a diversity of use cases simultaneously. In order to tackle this problem, the Next Generation Mobile Networks (NGMN) Alliance adopted the concept of network slicing, which allows multiple end-to-end logical networks to run on top of the same physical infrastructure, providing a network-as-a-service (NaaS) and multi-tenancy model for differentiated use cases.

Network slicing has drawn considerable attention in both academic and industrial communities ever since its invention. Numerous technologies, such as SDN and NFV [1]–[3] have been proposed to realize this innovative network functionality. However, most of the literature has targeted either RAN or packet core slicing [4]–[6], and transport network slicing has remained largely unexplored [19]. With end-to-end network slicing being a standard in future networks, the transport network should also be able to operate as slices, and thus, the traditional transport network consisting of optical or IP routers/switches will no longer be suitable. Moreover, with packet core functions moving to data centers and parts of RAN functions moving to edge clouds, the role of the transport network becomes much more significant. To better facilitate connections among these data centers and edge clouds, the transport network should extend its functionality and be able to run not just transport protocols but also stateless and stateful network functions (NFs).

To guarantee stringent service level agreements (SLAs) and facilitate slice management, it is necessary for the transport network to support live slice mobility without disruption of current services. The authors of [7] show the necessity of live slice mobility in use cases where transport network slices are required to move with autonomous vehicles and drones to support their mobility with the consistency of services and allocated resources. Non-disruptive slice mobility implies not only the migration of a slice instance, but also the duplication of current states if the slice is running a stateful NF. Ensuring no state loss makes the transport network more resilient and elastic. When failover occurs, assuming perfect early detection [8], flows can be redirected to a new slice instance on another physical node with no penalty. In the scenario of scale-out, traffic flows can be distributed to newly started slice instances on other physical nodes without disruption of current services since the states of traffic flows are also copied to these new slice instances.

Live slice mobility also facilitates energy efficiency and slice management. At night when user traffic is expected to decrease substantially, slices can be migrated to a small section of the network, with unneeded physical nodes powered down or put into hibernation [22]. Since the energy consumption of network equipment is dominated by the transmission of pilots (when there is no service data transmitted) [9], a considerable amount of energy cost will be conserved every night. Over time, the deployment of slices in the transport network may develop unbalanced traffic loads, and hotspots may be generated in certain sections of the network. To avoid bottlenecks, some slices running on busy physical nodes can be moved to relatively idle ones. For scheduled maintenance, slices on physical nodes to be maintained can be moved to other physical nodes with no need for mistake-prone changes

978-1-6654-0522-5/21/$31.00 ©2021 IEEE

of routing configurations.

In this paper, we present *FestNet*, a **F**lexible and **E**fficient **S**liced **T**ransport **Net**work. A slice in FestNet could run "conventional" protocols such as TCP/IP, Segment Routing, NDN, or other to-be-invented protocols as well as stateless and stateful NFs such as NAT, firewall, IDS, and load balancer. FestNet not only realizes network slicing but also supports live slice mobility without disruption of current services.

In designing FestNet, we face the following major challenges: (i) How to ensure there is no packet loss when moving a slice; (ii) How to move a slice without having to change routing configurations of slices running on other nodes, especially neighboring ones; (iii) How to move states with the slice during the live migration process when the slice being moved is running a stateful NF and ensure the consistency of states.

To solve these challenges, our key insight is that architecting FestNet with a strong separation between the link layer and the network function layer, and *managing those layers separately*, can be exploited to realize isolated and movable slices. FestNet assumes that the future transport network will be realized through Programmable Data Plane technology [10], and further assumes a composable and virtualized Programmable Data Plane (vPDP) to realize the two-layer strategy [11]–[13]. Within this context, FestNet realizes its two-layer design principle through a *virtual Link Layer (vLL)* and a *virtual Network Function layer (vNFL)*. vLL runs underneath vNFL to provide link-layer support by establishing virtual links and tunnels connecting virtual network functions in the vNFL layer across physical nodes. The use of a tunneling mechanism ensures that live slice migration destinations do not have to be adjacent physical nodes. Instead, they could be anywhere in the network as long as reachable by the tunnels. The strong layer separation allows FestNet to be *agnostic* to the exact functioning at the vNFL layer. I.e., vLL deals with "link management" while vNFL realizes the functionality of different network functions. FestNet achieves no packet loss and no configuration changes of neighboring slices by manipulating forwarding databases (fdb) in the vLL layer. vNFL network functions are realized with different data plane programs, e.g., using P4 [10]. Each vNFL program, together with virtual links connecting to it, form one distinct slice on a physical node in FestNet. The use of data plane programmability not only allows customization of programs running in a slice but also the migration of NF states. There are two types of states: static (e.g., firewall policy table) and dynamic (e.g., firewall flow table). Static states can be realized via P4 runtime commands, and they will be loaded into the P4 program when a slice is created. Dynamic states can be realized by using stateful objects `counter` and `register` in P4, with which most if not all popular stateful NFs can easily be programmed. APIs can be developed to facilitate fast read and write of `counter` and `register` arrays in a specific P4 target (the device that runs P4 programs), which enables dynamic state migration. Since handling static states is straightforward in FestNet, within the context of this paper, if not specified, we mean dynamic states when referring to stateful NFs.

Our contributions are threefold. First, we present the FestNet architecture and show that the two-layer design facilitates fast creation, deletion, and non-disruptive live migration of both stateless and stateful slices. To the best of our knowledge, FestNet is the first transport network model that simultaneously supports network slicing and live slice mobility. Second, we implemented a FestNet proof-of-concept prototype with five slices to illustrate typical use cases, ranging from popular protocols suitable for future networks to stateful NFs. We also implemented a controller application that controls slice operations and management. Third, we evaluated our FestNet prototype and showed that FestNet indeed achieves non-disruptive live slice mobility to adjacent and non-adjacent physical nodes, and slice operations can be completed within hundreds of milliseconds, much faster than implementations of related work, which cost hundreds of seconds to provision services.

II. RELATED WORK

FestNet is related to network virtualization and slicing in general [14], slicing in future networks [1], [15], [16], and slicing of the transport network [7], [17]–[20]. The survey in [14] and [15] provides a thorough overview of network virtualization, including early network slicing concepts. Several other efforts focus on specific technologies that enable network slicing and its use cases [1], [16].

Efforts related to the slicing of the transport network include enabling multi-tenancy through slicing [17], the use of SDN and NFV in transport network slices [18], and the challenges associated with transport slices [19]. More closely related to our work are more recent efforts that, similar to FestNet, suggest the migration of transport slices [7] and touch on slice management through orchestration [20]. FestNet is different from these earlier efforts in proposing the use of vPDP technology to realize transport slices, in defining clear architectural principles (strong layer separation, independent management of these layers, and live slice migration) to address challenges associated with transport slices, and in providing a proof-of-concept implementation illustrating the feasibility of our approach.

FestNet is also related to various node migration approaches [21], [22]. Work on live virtual machine (VM) migration [21] is likely the earliest work related to migrating nodes, albeit in this case, not network nodes per se. More recent work on using SDN and NFV to realize a network architecture follows a similar approach to migrate virtualized network elements (i.e., in their realization, virtualized network functions are realized as VMs that can be live-migrated). FestNet is more closely related to earlier work on the migration of network nodes [22], which uses conventional transport network technologies to realize link-layer management and migrates conventional IP router functionality between different physical router instances. FestNet makes use of vPDP technology, which affords it the ability to realize a similar approach using a single base technology, allowing for slice

differentiation and rapid evolution (enabled by data plane programmability).

[8] and [23] propose new network architectures and methods to reliably transfer states when moving a stateful NF in scenarios of failover, scale-in, scale-out, etc. However, their solutions demand a change of network architectures and modification of existing NF code. FestNet achieves the easier and faster transfer of states for stateful NFs, with no need to alter the network architecture or NF code.

III. FestNet Architecture

Slices running in FestNet are isolated, able to be assigned with different levels of resources, and movable without impacting on current traffic.

We separate FestNet into two layers: vLL and vNFL. As shown in Figure 1(a), FestNet assumes a composable vPDP [11]–[13] as the basis for the FestNet node architecture. In this vPDP framework, FestNet's link and network function layers are realized as separate data plane programs, i.e., one vLL and several vNFL instances, with one vNFL for each slice. Figure 1(b) depicts the FestNet network architecture, showing physical links between nodes, logical links associated with the "yellow" slice, and the FestNet controller with its links.

a) The Two-Layer Design: In FestNet, P4 is used to program the network function running in a slice. A single data plane provided by P4 can only have one network function running on top of it. P4 itself does not enable the data plane to perform multiple network functions simultaneously. To solve this problem, vPDP has been proposed by further extending the P4 abstraction [11]–[13]. The use of vPDP indeed allows a physical node to be divided into slices of logical nodes. However, vanilla vPDP is not flexible enough to support live slice migration. When moving a slice (a P4 program in this case), runtime commands of this P4 program have to be modified to adapt the new link-layer environment on the target physical node. Moreover, runtime commands of the P4 programs on the immediate upstream and downstream physical nodes also have to be updated to be able to redirect traffic since links are not moved with slices. These runtime command modifications will undoubtedly suspend user traffic. Is it possible to move a slice without having to modify any P4 runtime commands? FestNet's separation of the link layer and the network function layer in vPDP is designed to solve this issue.

To bolster smooth live slice migration, the vLL layer can create multiple isolated virtual Ethernet networks across the same shared physical infrastructure by tunneling layer-2 messages over layer-3 networks. This tunneling mechanism allows slices to be moved to not only adjacent physical nodes, but also non-adjacent ones. Both slice migration patterns will be shown in section V. When a vLL virtual interface is created, a remote tunnel endpoint (a physical interface) is required to be specified so that this virtual interface can choose the physical route to tunnel its layer-2 messages. As shown in Figure 2, although the virtual interface of the blue slice and that of the orange slice have the same destination, they can establish different tunnels by specifying different remote endpoints on the destination physical node. To contribute immediate traffic redirection when moving a slice, we adopt the notion of "soft local and hard remote" in configuring tunnel endpoints for vLL virtual interfaces. Although it is required to specify the remote tunnel endpoint when creating a vLL virtual interface, it is allowed that the local tunnel endpoint is left unspecified. The blank configuration for the local tunnel endpoint allows vLL to choose automatically whichever local tunnel endpoint that has a reachable route provided by the physical network to the remote tunnel endpoint. This feature is essential for traffic redirection in live slice migration.

In the vNFL layer, just like the vanilla vPDP, one or more P4 programs can run simultaneously, but the difference is that they bind virtual interfaces in the vLL layer to communicate with the outside world instead of physical interfaces. With this separation design, when moving a slice across physical nodes, both vLL and vNFL programs are moved, and there is no configuration change required on both layers.

Thus, with vLL configurations and vNFL programs, a physical network is "sliced" into multiple virtual networks with diverse programmable network functions.

b) Isolation & Resource Allocation: FestNet ensures that slices are isolated and can only run with assigned resources. A slice consists of one vNFL program and multiple vLL virtual interfaces it binds. In vLL, each virtual interface can only communicate with its peer(s) in other physical nodes, which makes sure that virtual links do not interfere with each other. For vNFL programs, several isolation methods can be used, including running them as processes, in VMs, in containers, or even in special hardware.

The most important resources of a slice to differentiate SLAs include bandwidth, CPU power, and memory. Bandwidth isolation can be conveniently achieved in vNFL by using P4's stateful object `meter`. VMs, containers, or `cgroups` configurations can be adopted to allocate CPU power and memory for a slice.

c) Live Slice Migration: FestNet allows stateful NFs with static states, dynamic states, or both to run as slices. Static states are realized via P4 runtime commands and can be loaded into the newly created P4 program on the target node during live slice migration. Dynamic states are realized by using P4's stateful object `counter` and `register`. Thanks to `register`, most if not all popular stateful NFs can be programmed in P4. With the help of APIs that interact with a specific P4 target, fast read and write operations on the `counter` and `register` arrays can be achieved, which enables state copying when moving a stateful slice with dynamic states.

In FestNet, when moving a slice, the following are migrated: vLL configurations, the vNFL program with its runtime commands, and vNFL states if the program is a stateful NF. Figure 3 shows the procedures of moving the second slice on the physical node S2 (source) to S3 (target):

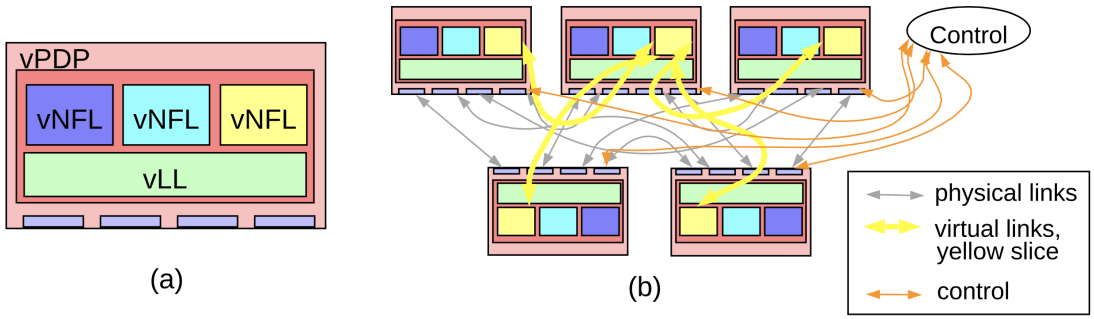

Fig. 1: FestNet: (a) Node architecture (b) Network architecture

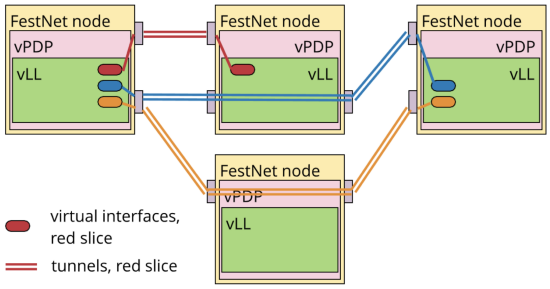

Fig. 2: vLL tunneling mechanism

Step 1: On S3, establish the same vLL configurations as on S2. I.e., create vLL interfaces Vx_a′ and Vx_b′ with the same MAC and IP addresses as Vx_a and Vx_b, respectively. *Step* 2: On S3, establish the same vNFL configurations as on S2. I.e., start vNFL2′ on S3 with the same P4 program, runtime commands, and computing resources (CPU power and memory) as vNFL2 on S2, binding vLL interfaces Vx_a′ and Vx_b′. *Step* 3: On S1 (upstream), if the moving slice is stateless: redirect traffic from going through Vx_a on S2 to going through Vx_a′ on S3; if the moving slice is stateful: stop traffic and buffer packets at Vx_u on S1, copy states from vNFL2 on S2 to vNFL2′ on S3, then release the buffer and redirect traffic from going through Vx_a on S2 to going through Vx_a′ on S3. *Step* 4: On S2, remove vLL interfaces Vx_a and Vx_b, terminate the vNFL2 program, and release all resources assigned. On S4 (Downstream), update the permanent vLL fdb entries to point to Vx_b′ on S3 (for future use after the learned dynamic entries expire). Now the live slice migration process is complete.

The procedures for stateless and stateful live slice migration differ in *Step 3*. If the moving slice is stateless, only traffic redirection is needed. In the case of moving a stateful slice, traffic has to be stopped, and packets need to be buffered on the upstream node to wait for the state copying process to complete. In FestNet, we devise data plane buffers on every physical node in the network to facilitate this process. One buffer can be associated with one vLL interface for buffering and releasing packets running through this interface. Only when all states have been copied to the target slice will the buffer of the vLL interface on the upstream node be released. In the designs of [24] and [23], it is the controller that buffers packets during the migration of NF states. On the other hand, in the FestNet design, we adopt data plane buffers, which is a more robust solution since the control plane should be focusing on handling control messages instead of intervening in the data plane's job.

How to enable immediate traffic redirection without packet loss in *Step 3* for both stateless and stateful slice migration? To address this challenge, we devise vLL to inherit the "flood and learn" mechanism from Ethernet switches and maintain an fdb for each virtual interface. Moreover, vLL extends the mechanism by storing more information in the fdb. In addition to the MAC address of a peer virtual interface and the switch port number through which this MAC address can be reached, the IP address of the remote vLL tunnel endpoint on the peer physical node is included in an fdb entry. The MAC address recorded in an fdb entry can be reached by tunneling Ethernet messages to the IP address recorded in the same entry. Another contributing feature is vLL's "soft local and hard remote" configuration when creating vLL virtual interfaces. With the help of these vLL features, now we examine in detail what occurs in the vLL layer during the traffic redirection illustrated in Figure 4. The process can be divided into two stages:

Stage 1: When the vLL virtual interface Vx_u is created, the remote tunnel endpoint of Vx_u is originally specified as the physical interface Ph2Int1 on S2. This configuration generates a permanent fdb entry containing the IP address of Ph2Int1 (192.168.12.2) and the MAC address of the peer vLL virtual interface Vx_a on S2 (00:aa:bb:00:02:01). vLL guarantees that Vx_u and Vx_a are in the same virtual Ethernet. When a packet arrives at Vx_u and needs to be forwarded to the vLL virtual interface with the MAC address of 00:aa:bb:00:02:01, its fdb is checked, and there is a match. The packet is then passed to the local tunnel endpoint Ph1Int1 and tunneled to 192.168.12.2 (the IP address of Ph2Int1). Finally, Ph2Int1 passes the packet to Vx_a. Now suppose we want to move a slice from S2 to S3. First, we need to establish vLL virtual interfaces and the vNFL program exactly the same as on S2. For instance, Vx_a′ has the same IP address and MAC address as those of Vx_a.

Stage 2: Then we can update the fdb entry of Vx_u from the original mapping of 00:aa:bb:00:02:01 - 192.168.12.2 to 00:aa:bb:00:02:01 - 192.168.13.3. After the change, when a packet arrives at Vx_u destined to the vLL virtual interface with the MAC address of 00:aa:bb:00:02:01, an fdb check

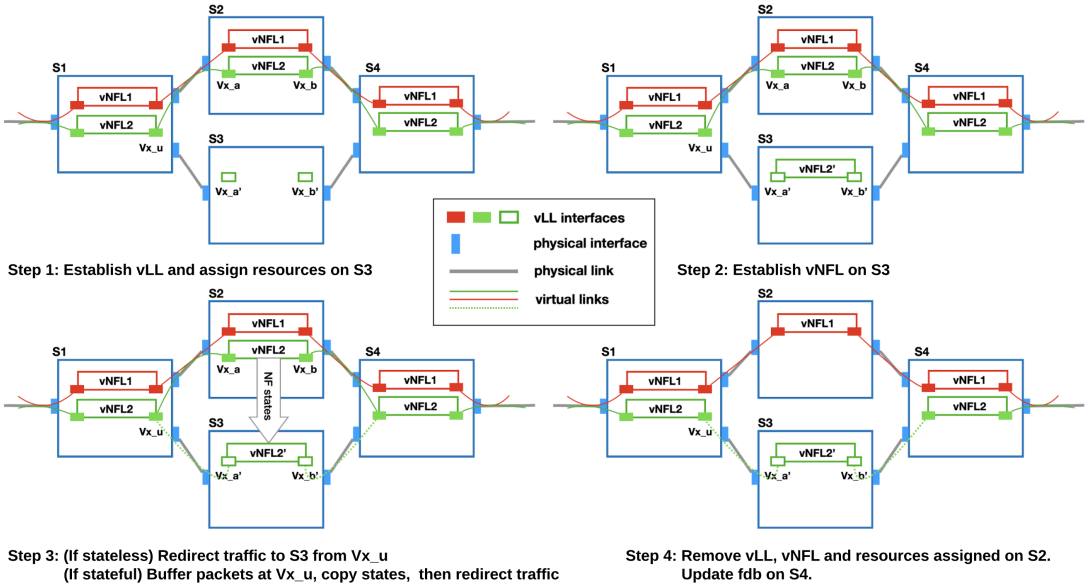

Fig. 3: Slice moving procedures

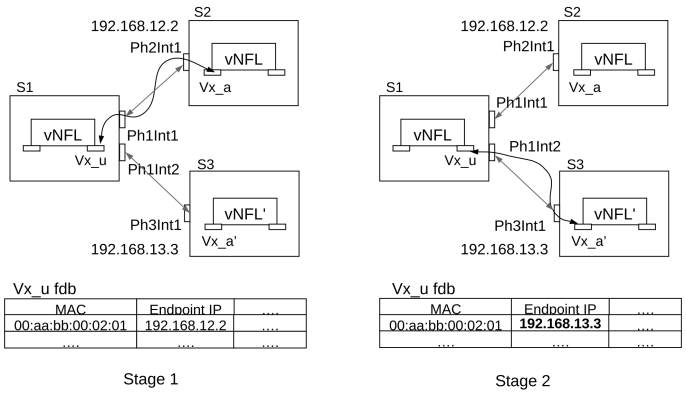

Fig. 4: Redirecting traffic by manipulating fdb

returns the IP address of a new remote tunnel endpoint Ph3Int1. Thus the packet is tunneled to 192.168.13.3 through the local tunnel endpoint Ph1Int2. Ph3Int1 then passes the packet to Vx_a′ for further processing. Now the traffic has been redirected.

d) vNFL Agnostic: During the live slice migration process, not only is there no disruption on user traffic, but also there is no need to change any vNFL configurations, including both the P4 program and its runtime commands on all other physical nodes. The reason is that traffic redirection is solely achieved by manipulating fdb on the vLL layer. If we replace vNFL with other routing methods, e.g., routing and forwarding in the Linux kernel, the traffic redirection still works with no packet loss. The live slice mobility mechanism is not constrained by upper-layer network functions. With the vNFL agnostic feature, after a slice migration, not only does the logical topology of the data plane stay unchanged, but also the functionality of this data plane requires no modification or additional re-configurations, which significantly simplifies network management tasks.

e) FestNet Controller: A central controller that connects to all physical switches in the network is included in the FestNet architecture. The controller is responsible for performing slice operations such as creation, deletion, modification, and migration of slices. The controller also keeps a record of real-time global topology information as it creates and deletes slices. The knowledge of real-time global topology is crucial for designing algorithms that help automate slice operations. OpenFlow controllers have to discover the network topology through the topology discovery mechanism (OFDP), which decreases control plane performance and poses security threats [25], [26]. The FestNet controller design does not have these disadvantages.

IV. IMPLEMENTATION

This section describes our proof-of-concept implementation of the FestNet prototype. With reference to Figure 1(a), the FestNet implementation uses P4 behavioral model (BMv2) [27] with appropriate P4 programs to realize the vNFL layer and VXLAN to realize the vLL layer. The controller application is implemented using Python. The code consists of the controller part running on the controller node and the switch part running on each switch node. The runtime CLI APIs developed to interact with BMv2 are adopted and integrated into the controller part to enable fast read and write of NF states. Multiple physically connected Linux boxes in the Emulab [28] testbed are used as the platform of the FestNet prototype. Figure 5 illustrates the topology of the prototype, which is a typical snippet of a physical transport network that could support three logical subnets. A slice can be moved among S2, S3, and S3′.

a) vLL Configurations: VXLAN is used to create virtual interfaces and establish virtual links across physical nodes. VXLAN interfaces are able to tunnel MAC layer messages over the IP layer. Just like physical switches bind physical

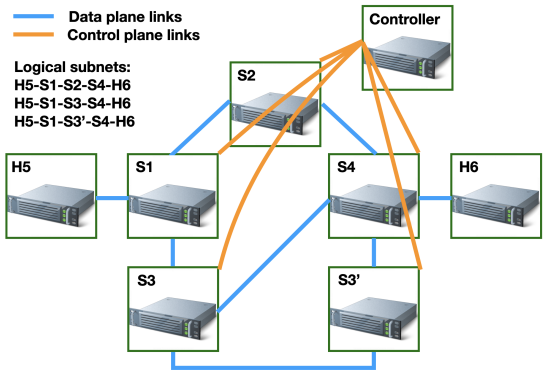

Fig. 5: FestNet prototype topology

interfaces, BMv2 softswitches can bind VXLAN interfaces. The connected virtual links and softswitches form a slice in the network. Inside a physical node, actions on the flow of packets running in and out of VXLAN interfaces are defined by P4 programs and their runtime commands.

When moving stateful slices, a buffer is required to stop traffic and buffer packets during the states copying process. We implemented the buffer using the plug Qdisc, which can be attached to a VXLAN interface and perform packet buffering and releasing.

Before releasing the buffer of a VXLAN interface, a htb Qdisc needs to be attached to the local endpoint to smooth the traffic burst for the flow running through this VXLAN interface by specifying the `burst` and `cburst` parameters, with a filter ensuring that only the specified flow is smoothed. Otherwise, the flood of packets running out of the buffer will overwhelm the BMv2 softswitches along the traffic path and cause them to drop packets. BMv2 itself is a research prototype that focuses on data plane programmability. It is never designed to provide comparable performance as industrial switches. If P4 hardware adopts our FestNet design, performance will not be an issue, and there will be no need to add the htb Qdisc.

b) vNFL Programs: We adopted an approach of instantiating multiple BMv2 processes in the Linux environment as one vPDP instance. On a single physical node, multiple BMv2 processes can be started with different P4 programs running in each one of them.

We implemented five illustrative slices for the FestNet prototype: (1) *IPv4_router*: A stateless router for bandwidth-sensitive applications such as HD movie streaming, HD video conference, web page surfing, and other legacy applications. (2) *End-to-End Segment Routing over IPv6 (E2E_SRv6)*: A stateless SRv6 router for reliability- and latency-sensitive applications such as autonomous vehicles, drones, and smart factories. (3) *GPRS Tunneling Protocol (GTP) Segment Routing over IPv6 (GTP_SRv6)*: A stateless GTP based SRv6 router for the same collection of applications as *E2E_SRv6* but provides transition and adaptation for legacy 4G LTE networks. (4) *Stateful_firewall*: A firewall with dynamic states (connection status and flow information), representing various stateful NFs that can be integrated into FestNet. (5) *Stateful_IPv4*: A stateful IPv4 router with dynamic states recording the number of packets and IP bytes (IP header + IP payload) passing through its interfaces. We use this slice to illustrate stateful live slice migration.

We chose SRv6 to illustrate FestNet due to its prominence recognized by both academic and industrial applications. It is proposed by 3GPP release 16 as one of the candidate user plane protocols for the 5G transport network to accommodate upcoming heterogeneous requirements of new use cases [29]. SRv6 significantly improves the network's controllability, reliability, efficiency, and flexibility, by simplifying both the control plane and data plane.

All the five vNFL programs were written in P4_16, the most up-to-date version of P4. We implemented them by modifying open-source code and adapting it to the FestNet platform. The original Stateful_firewall only supports TCP, and we added the code that supports UDP with "pseudo" connection status and flow information. We implemented Stateful_IPv4 by adding counting functionalities to IPv4_router. There are other open-source P4 programs that are suitable to provide services for future networks, including Inband Network Telemetry (INT) [30] and Named-Data Networking (NDN) [31]. They can also be loaded as slices in FestNet with some modification and adaptation.

c) Slice Isolation & Resource Assignment: In FestNet, each slice consists of vLL and vNFL configurations. In our prototype, VXLAN provides vLL isolation [32], and vNFL isolation is guaranteed by Linux since BMv2 softswitches exist as processes. Bandwidth resources can be assigned by specifying `meter` parameters in the runtime commands of P4 programs. CPU power and memory resources are allocated to slices via `cgroups`.

d) Slice Operations and Management: We integrated slice operations and management into the controller application, from which the following commands can be executed: (i) *create_slice*: create the specified slice on the specified physical node; (ii) *delete_slice*: delete the specified slice on the specified physical node; (iii) *runtime_cmd*: enter the runtime CLI for the specified slice on the specified physical node to change runtime configurations of the slice such as adding/deleting table entries and setting `meter` parameters; (iv) *move_stateless_slice*: move the specified stateless slice from the specified source physical node to the specified target one; (v) *move_stateful_slice*: move the specified stateful slice from the specified source physical node to the specified target node.

Upon executing *create_slice*, computing resources are allocated, vLL configurations are established, and the vNFL program is instantiated with its runtime commands loaded. *move_stateless_slice* first creates a new instance of the specified slice on the target node, then redirects traffic on the upstream node, updates fdb entries on the downstream node, and finally deletes the old slice and assigned resources on the source node. *move_stateful_slice* is more complicated and does more work under the hood. Before traffic redirection, it

signals the buffer attached to the VXLAN interface on the upstream node to stop traffic and start buffering packets. Then the controller reads states from the moving slice on the source node and writes them to the newly created slice on the target node. Upon completion of copying states, it redirects traffic on the upstream node, releases the buffer, and updates fdb entries on the downstream node.

V. EVALUATION

By evaluating the FestNet prototype, we aim to answer the following questions: How much overhead does FestNet introduce? Does FestNet work as expected, with no packet and state loss after live slice migration? What is the impact of live slice migration on stateless and stateful slices? How fast does FestNet perform slice operations, compared with implementations in related work?

a) Environmental Settings: Figure 5 shows the prototype network topology for evaluation. All physical nodes we used are Dell PowerEdge R430 (d430) servers with 32 CPU cores and 64 GB of memory in the Emulab testbed. The controller and nodes S1 - S4 have Ubuntu 16.04 with the kernel version 4.4 as the operating system. The two hosts H5 and H6, which act as the sender and receiver of data packets, are installed with Ubuntu 18.04 with the kernel version 4.15, which supports establishing GTP endpoints in the kernel, essential for receiving and sending packets through the GTP_SRv6 slice.

b) Overhead: Table 1 shows the maximum data rates of all five slices running separately with no packet loss when there is no live slice migration in our experiment setting. Exceeding these maximum data rates for each slice will cause BMv2 softswitches to overwhelm and drop packets. When we run all five slices simultaneously with each slice at the same data rate, the maximum value for each slice is 16 Mbps. If running all five slices simultaneously with periodic live slice migration, the maximum data rate with no packet loss for each slice is 15 Mbps. So the overhead of FestNet live slice mobility compared with the baseline performance is $(16 - 15)/16 = 6.25\%$.

The baseline performance is not high because BMv2 is a research prototype that aims to provide data plane programmability, not high performance. We expect a hardware-based FestNet implementation to perform significantly better.

TABLE I: Baseline performance

Slice name	*Max data rate (Mbps)*	*CPU usage (%)*
IPv4_router	61	49 - 51
E2E_SRv6	56	53 - 55
GTP_SRv6	52	54 - 56
Stateful_firewall	59	49.5 - 51
Stateful_IPv4	59	48.5 - 50.5

c) Functional Evaluation: `iperf` UDP mode was used to verify that there is no packet loss and no state loss after live slice migration and to identify the impact of live slice migration on throughput. We did the following experiments: First, we created all five slices (Stateful_firewall was compiled to hold 100 flows of states) on S1, S2, and S4, with each slice allocated with five dedicated CPU cores (although BMv2 cannot use multiple CPU cores, we assigned the resources anyway to illustrate FestNet's function) and 10 GB of memory. Then we ran five processes of `iperf` in UDP mode on H5 and H6 at data rates ranging from 10 Mbps to 20 Mbps with a step of 2.5 Mbps for each slice. The sum of these data rates is 75, which is the same as that of the maximum data rates from our baseline/overhead evaluation. We chose these specific data rates in the experiments to make the results more visible on figures. Then we performed live slice migration for each slice at 10-second intervals in the following two patterns to illustrate the flexibility of FestNet: *Same-direction loop*: Move each slice from S2 to S3. After all the slices are on S3, move each slice from S3 to S3′. After all the slices are on S3′, move each slice from S3′ to S2. I.e., each slice has been moved following the path S2-S3-S3′-S2. *Different-direction loop*: Move each slice one by one, as in the pattern described above. But move IPv4_router and GTP_SRv6 by the path S2-S3-S3′-S2, and move the other three slices by the path S2-S3′-S3-S2. The second migration pattern corresponds to the use case when we need to spread slices to multiple physical nodes during the day and converge them back to a small subset of the network at night.

In these experiments, `iperf` outputs on H5 and H6 for each slice explicitly showed that there was no packet loss after each live slice migration. To verify that there was also no state loss, we read the `counter` array and `register` array in the Stateful_IPv4 slice and compared them with `iperf` outputs after each migration. The results showed that the states were consistent after the slice had been moved. The same verification applies to Stateful_firewall.

Figure 6 shows the impact on throughput for each slice with periodic live slice migrations in both patterns. The left part shows the throughput of each slice. For stateless slices, there is no impact on throughput. As can be expected, for stateful slices, we will experience fluctuations. I.e., during stateful slice migration, the buffer stops the traffic and then releases a burst of packets after the migration has been completed resulting in throughput changes. The right part of Figure 6 shows the throughput measured at the physical links of S2-S4, S3-S4, and S3′-S4. The results show that in FestNet, slices can be moved to adjacent and non-adjacent physical nodes with no packet and state loss.

d) Scaling of States: How will throughput be impacted when the number of states increases in stateful slices? To answer this question, we compiled Stateful_firewall with 100, 200, 300, 400, and 500 flows of states. Then we ran each variation at the data rates of 10, 15, and 20 Mbps for 40 seconds, with live slice migration occurring every 10 seconds. The results of all five variations were concatenated and shown in Figure 7.

Let us take the green line (variations running at 20 Mbps) as an example to see what happened during the experiments. For the first 40-second interval, during which time the variation with 100 flows of states was running, there was no packet loss, but fluctuations appeared. Starting from the second 40-second interval, we can see pits in the figure when live slice

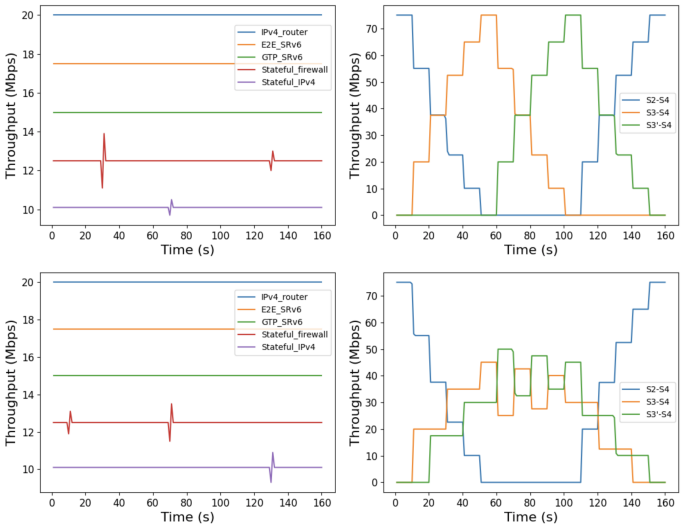

Fig. 6: Impact on throughput in two migration patterns

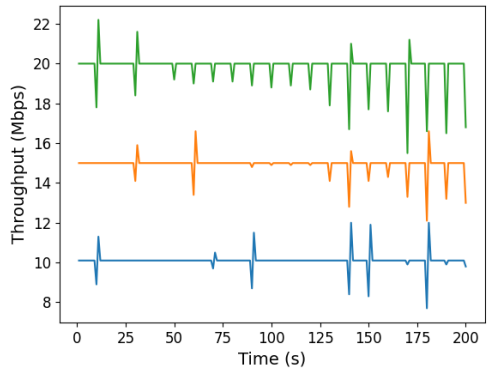

Fig. 7: Impact on throughput when states scale

migration occurred. These pits indicate packet loss during the migration processes since the throughput went down and did not go back up in the next second. Going further along the x-axis, when running variations with 400 and 500 flows of states, we experienced packet loss combined with fluctuations. The value of the throughput decrease is larger than that of the throughput increase in the next second. Looking at the green line alone, we can see that the more states the slice had, the more packets were dropped (the deeper the pits were). Compared with other data rates, the orange line experienced packet loss starting from the variation with 300 flows of states, and the blue line did not suffer from packet loss until the variation with 500 flows of states. Thus, as can be expected, the trend is that the higher the data rate is, the fewer flows of states the slice can carry to ensure non-disruptive live slice migration. These results suggest that for stateful NFs, there is a "feasibility region" where stateful live slice migration in FestNet operates without packet loss. In our BMv2 prototype, this region is relatively small. With a performant P4 hardware implementation, we expect the feasibility region to be much larger.

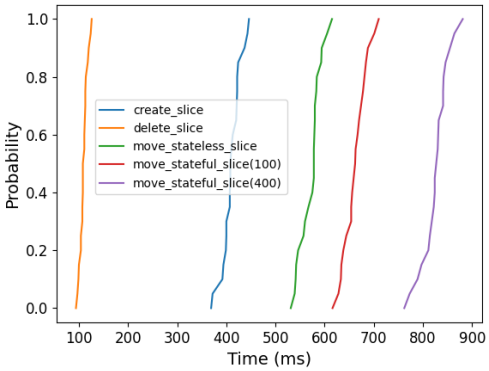

Fig. 8: Slice operation performance

e) Performance of Slice Operations: We used the `timeit` package in Python to measure the time cost by different slice operations. The measurements start from the very beginning of executions to the point when they are completely finished. We measured the following operations: create_slice, delete_slice, move_stateless_slice, and move_stateful_slice. For the move_stateful_slice operation, we included results of Stateful_firewall with 100 and 400 flows of states. Figure 8 shows the time cost by each type of slice operation. Live slice migration operations take longer because they need to create a slice on the target node and then delete the old one on the source after traffic redirection. Moving stateful slices takes the longest since it also involves copying states from the old slice to the new one. From Figure 8, we can see that creating slices in FestNet takes only about 400 ms, much faster than the provisioning times of network slicing implementations in [33]–[35], which all cost hundreds of seconds. The gap is because their work targets end-to-end network slicing and is implemented using VM and containers, which causes slow operations but could provide better slice isolation.

VI. CONCLUSION AND FUTURE WORK

The transport network will play an even more important role with the advent of network slicing. In this paper, we have shown that live slice mobility without disruption of current services will be a crucial feature to better support network slicing in the transport network. We have proposed FestNet, which is a flexible and efficient sliced transport network model, and presented its design and prototype implementation. Our evaluation shows that slices in FestNet can be moved to adjacent and non-adjacent physical nodes with no packet loss and no state loss for stateful NFs and that slice operations in FestNet are many times faster than implementations in related work.

In our current FestNet implementation, performing slice operations involves human interaction, either via typing commands or loading them from scripts on the controller. In order to allow autonomous slice operations, a machine-learning-based algorithm can be designed and implemented in the controller so that in certain scenarios, slices can be automatically created, migrated, or deleted according to pre-configured

requirements. The controller also needs to constantly monitor each physical node's traffic status and provide this information to the algorithm to make slice operation decisions.

ACKNOWLEDGMENT

This material is based upon work supported by the National Science Foundation under Grant Number 1647264.

REFERENCES

[1] J. Ordonez-Lucena, P. Ameigeiras, D. Lopez, J. J. Ramos-Munoz, J. Lorca, and J. Folgueira, "Network slicing for 5g with sdn/nfv: Concepts, architectures, and challenges," *IEEE Communications Magazine*, vol. 55, no. 5, pp. 80–87, 2017.

[2] F. Z. Yousaf, M. Bredel, S. Schaller, and F. Schneider, "Nfv and sdn—key technology enablers for 5g networks," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 11, pp. 2468–2478, 2017.

[3] X. Foukas, G. Patounas, A. Elmokashfi, and M. K. Marina, "Network slicing in 5g: Survey and challenges," *IEEE Communications Magazine*, vol. 55, no. 5, pp. 94–100, 2017.

[4] A. Nakao, P. Du, Y. Kiriha, F. Granelli, A. A. Gebremariam, T. Taleb, and M. Bagaa, "End-to-end network slicing for 5g mobile networks," *Journal of Information Processing*, vol. 25, pp. 153–163, 2017.

[5] M. R. Sama, X. An, Q. Wei, and S. Beker, "Reshaping the mobile core network via function decomposition and network slicing for the 5g era," in *2016 IEEE Wireless Communications and Networking Conference*. IEEE, 2016, pp. 1–7.

[6] I. da Silva, G. Mildh, A. Kaloxylos, P. Spapis, E. Buracchini, A. Trogolo, G. Zimmermann, and N. Bayer, "Impact of network slicing on 5g radio access networks," in *2016 European conference on networks and communications (EuCNC)*. IEEE, 2016, pp. 153–157.

[7] R. A. Addad, T. Taleb, H. Flinck, M. Bagaa, and D. Dutra, "Network Slice Mobility in Next Generation Mobile Systems: Challenges and Potential Solutions," http://www.mosaic-lab.org/uploads/papers/cec5563d-056f-441e-a9da-03833f3701a8.pdf, 2019, to appear.

[8] M. Kablan, A. Alsudais, E. Keller, and F. Le, "Stateless network functions: Breaking the tight coupling of state and processing," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, 2017, pp. 97–112.

[9] M. Olsson, C. Cavdar, P. Frenger, S. Tombaz, D. Sabella, and R. Jantti, "5green: Towards green 5g mobile networks," in *2013 IEEE 9th international conference on wireless and mobile computing, networking and communications (WiMob)*. IEEE, 2013, pp. 212–216.

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

[11] D. Hancock and J. Van der Merwe, "Hyper4: Using p4 to virtualize the programmable data plane," in *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*. ACM, 2016, pp. 35–49.

[12] C. Zhang, J. Bi, Y. Zhou, A. B. Dogar, and J. Wu, "Hyperv: A high performance hypervisor for virtualization of the programmable data plane," in *2017 26th International Conference on Computer Communication and Networks (ICCCN)*, July 2017, pp. 1–9.

[13] P. Zheng, T. Benson, and C. Hu, "P4visor: Lightweight virtualization and composition primitives for building and testing modular programs," in *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '18. New York, NY, USA: ACM, 2018, pp. 98–111.

[14] N. M. K. Chowdhury and R. Boutaba, "A survey of network virtualization," *Comput. Netw.*, vol. 54, no. 5, pp. 862–876, Apr. 2010.

[15] I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini, and H. Flinck, "Network slicing and softwarization: A survey on principles, enabling technologies, and solutions," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 3, pp. 2429–2453, 2018.

[16] H. Zhang, N. Liu, X. Chu, K. Long, A.-H. Aghvami, and V. C. Leung, "Network slicing based 5g and future mobile networks: mobility, resource management, and challenges," *IEEE communications magazine*, vol. 55, no. 8, pp. 138–145, 2017.

[17] X. Li, R. Casellas, G. Landi, A. de la Oliva, X. Costa-Perez, A. Garcia-Saavedra, T. Deiss, L. Cominardi, and R. Vilalta, "5g-crosshaul network slicing: Enabling multi-tenancy in mobile transport networks," *IEEE Communications Magazine*, vol. 55, no. 8, pp. 128–137, 2017.

[18] X. Costa-Perez, A. Garcia-Saavedra, X. Li, T. Deiss, A. De La Oliva, A. Di Giglio, P. Iovanna, and A. Moored, "5g-crosshaul: An sdn/nfv integrated fronthaul/backhaul transport network architecture," *IEEE Wireless Communications*, vol. 24, no. 1, pp. 38–45, 2017.

[19] M. Fiorani, P. Monti, B. Skubic, J. Mårtensson, L. Valcarenghi, P. Castoldi, and L. Wosinska, "Challenges for 5g transport networks," in *2014 IEEE international conference on advanced networks and telecommuncations systems (ANTS)*. IEEE, 2014, pp. 1–6.

[20] T. Taleb, I. Afolabi, and M. Bagaa, "Orchestrating 5g network slices to support industrial internet and to shape next-generation smart factories," *IEEE Network*, vol. 33, no. 4, pp. 146–154, July 2019.

[21] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2*, ser. NSDI'05. Berkeley, CA, USA: USENIX Association, 2005, pp. 273–286.

[22] Y. Wang, E. Keller, B. Biskeborn, J. Van Der Merwe, and J. Rexford, "Virtual routers on the move: live router migration as a network-management primitive," in *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4. ACM, 2008, pp. 231–242.

[23] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella, "Opennf: Enabling innovation in network function control," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 163–174, 2014.

[24] S. Rajagopalan, D. Williams, H. Jamjoom, and A. Warfield, "Split/merge: System support for elastic execution in virtual middleboxes," in *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, 2013, pp. 227–240.

[25] F. Pakzad, M. Portmann, W. L. Tan, and J. Indulska, "Efficient topology discovery in software defined networks," in *2014 8th International Conference on Signal Processing and Communication Systems (ICSPCS)*. IEEE, 2014, pp. 1–8.

[26] A. Azzouni, R. Boutaba, N. T. M. Trang, and G. Pujolle, "softdp: Secure and efficient topology discovery protocol for sdn," *arXiv preprint arXiv:1705.04527*, 2017.

[27] A. Bas. p4lang/behavioral-model. [Online]. Available: https://github.com/p4lang/behavioral-model

[28] M. Hibler, R. Ricci, L. Stoller, J. Duerig, S. Guruprasad, T. Stack, K. Webb, and J. Lepreau, "Large-scale virtualization in the emulab network testbed," in *USENIX 2008 Annual Technical Conference*, ser. ATC'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 113–128.

[29] 3GPP, "Study on user plane protocol in 5gc," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 29.892, 2019.

[30] N. Van Tu, J. Hyun, and J. W.-K. Hong, "Towards onos-based sdn monitoring using in-band network telemetry," in *2017 19th Asia-Pacific Network Operations and Management Symposium (APNOMS)*. IEEE, 2017, pp. 76–81.

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

[32] T. Sridhar, L. Kreeger, D. Dutt, C. Wright, M. Bursell, M. Mahalingam, P. Agarwal, and K. Duda, "Virtual extensible local area network (vxlan): A framework for overlaying virtualized layer 2 networks over layer 3 networks," RFC, 2014.

[33] D. Camps-Mur, F. Canellas, A. Machwe, J. Paracuellos, K. Choumas, D. Giatsios, T. Korakis, and H. R. Kouchaksaraei, "5gos: Demonstrating multi-domain orchestration of end-to-end virtual ran services," in *2020 6th IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2020, pp. 131–139.

[34] V. Theodorou and M.-E. Xezonaki, "Network slicing for multi-tenant edge processing over shared iot infrastructure," in *2020 6th IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2020, pp. 8–14.

[35] T. Lin, S. Marinova, and A. Leon-Garcia, "Towards an end-to-end network slicing framework in multi-region infrastructures," in *2020 6th IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2020, pp. 413–421.