Physical Wireless Resource Virtualization for Software-Defined Whole-Stack Slicing

Matthias Sander-Frigau, Tianyi Zhang, Hongwei Zhang, Ahmed E. Kamal, Arun K. Somani
Department of Electrical and Computer Engineering, Iowa State University
{msfrigau,tianyiz,hongwei,kamal,arun}@iastate.edu

***Abstract*—Radio access network (RAN) virtualization is gaining more and more ground and expected to re-architect the next-generation cellular networks. Existing RAN virtualization studies and solutions have mostly focused on sharing communication capacity and tend to require the use of the same PHY and MAC layers across network slices. This approach has not considered the scenarios where different slices require different PHY and MAC layers, for instance, for radically different services and for whole-stack research in wireless living labs where novel PHY and MAC layers need to be deployed concurrently with existing ones on the same physical infrastructure. To enable whole-stack slicing where different PHY and MAC layers may be deployed in different slices, we develop *PV-RAN*, the first open-source virtual RAN platform that enables the sharing of the same SDR physical resources across multiple slices. Through API Remoting, PV-RAN enables running paravirtualized instances of OpenAirInterface (OAI) at different slices without requiring modifying OAI source code. PV-RAN effectively leverages the inter-domain communication mechanisms of Xen to transport time-sensitive I/Q samples via shared memory, making the virtualization overhead in communication almost negligible. We conduct detailed performance benchmarking of PV-RAN and demonstrate its low overhead and high efficiency. We also integrate PV-RAN with the CyNet wireless living lab for smart agriculture and transportation.**

I. Introduction

Recent advances in 5G virtualization [1], through the use of Network Function Virtualization (NFV) and Software Defined Networking (SDN), have led to the development of the concept of Network Slicing [2]. Although there is no consensus on the strict definition of a Network Slice, it generally refers to an end-to-end logical portion of the network resources. IETF defines Network Slicing as the collection of a set of technologies to create specialized and dedicated logical networks as a service (NaaS) in support of network service differentiation and meeting the diversified requirements from vertical industries [3]. Thus, network slicing has the capability for the network to provide a multitude of services that demand very specific requirements in terms of latency, reliability, and bandwidth. According to telecommunications Standard Developing Organizations (SDOs), network slicing is a key enabler for accommodating diverse 5G services such as ultra-reliable and low-latency communications (URLLC), extreme mobile broadband (xMBB), and massive machine-type communication (mMTC).

This work is supported in part by the NSF awards 1827211 and 1821962.

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

Critical to network virtualization and slicing is the virtualization and slicing of radio access networks (RANs) in addition to core networks and edge/cloud resources. Despite much progress in RAN virtualization and slicing, existing work has mostly focused on sharing communication capacity, and the different slices tend to use the same PHY and MAC layers [4], [5], [6]. Nonetheless, the heterogeneous requirements of 5G and beyond wireless services may well demand different PHY and MAC layers. For instance, due to the radically different requirements that URLLC and mMTC pose to communication timeliness, reliability, throughput, and energy efficiency, optimal URLLC and mMTC solutions may well differ in their PHY and MAC layers, examples including the use of scalable OFDM numerology and compressive-sensing-based signal detection for URLLC and mMTC respectively. Therefore, there exists the unmet need for RAN virtualization solutions that enable different slices to run different PHY and MAC layers. For convenience, we call this type of RAN virtualization *whole-stack slicing*. Whole-stack slicing is not only important for production 5G-and-beyond systems, it is also important for enabling whole-stack research and innovation in wireless living labs where novel solutions in the PHY and MAC layers can be experimented together with existing solutions [7], [8].

To enable the use of different PHY and MAC layers at different RAN slices, we need to enable the sharing of the physical wireless radios, a.k.a. remote radio heads (RRHs), across slices. In this study, we use USRP software-defined-radios (SDRs) [9] as the physical radios, and we propose the *PV-RAN* platform that virtualizes the RAN SDRs. PV-RAN uses the Xen hypervisor [10] as the base virtualization platform for the virtual RANs, and its current implementation assumes that the open-source wireless software platform OpenAirInterface (OAI) [11] is used as the protocol implementation platform. As shown in Figure 1, PV-RAN spans two main components of each RAN:

- *PV-Back* in the Dom0 of the base station computer: it interacts with the SDR and implements the logic necessary to virtualize and multiplex SDR radio resources.
- *PV-Front* in each DomU of the base station computer: it runs the full network protocol stack (e.g., from PHY to PDCP in LTE) with a modified version of the USRP low-level interface of OAI that accounts for SDR radio resource virtualization.

Compared to the C-RAN architecture that makes use of at

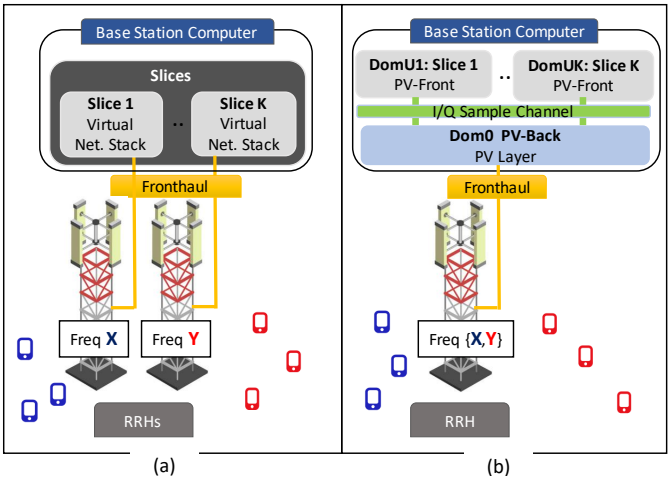

Fig. 1: (a) Traditional C-RAN where the fronthaul connects slices (with virtualized network stacks) to their dedicated remote-radio-heads (RRHs). (b) PV-RAN where the fronthaul connects a shared RRH to multiple slices.

least one radio per slice, the PV-RAN platform can support multiple slices using the same radio.

The contributions of this work are as follows:

- We present the first open-source design and implementation of the virtualization of physical SDR resources to enable whole-stack slicing. Besides supporting whole-stack slicing in production systems, PV-RAN allows researchers to prototype new cellular network PHY and MAC layers using exclusively open-source software: PV-Back and PV-Front use Xen paravirtualization and its inter-domain communication mechanism to transport I/Q samples, and the virtualized slices use OAI as the wireless network protocol stack.
- PV-RAN enables running paravirtualized instances of OAI without requiring modifying OAI source code. This is accomplished through a novel *API Remoting* method that intercepts UHD library function calls from paravirtualized OAIs in DomUs and redirect them to the privileged domain Dom0 where the library call is executed and forwarded to the SDRs.
- PV-RAN ensures slice isolation in shared SDR access, and it features novel and effective techniques such as lightweight, in-band timestamping synchronization between PV-Back and PV-Front.
- We perform detailed performance benchmarking of PV-RAN, and demonstrate that the inter-domain communication channel of Xen is a fast and efficient interface for the transport of time-sensitive I/Q samples. In comparison to OAI running in bare-metal computers without virtualization, the overhead due to virtualization is insignificant.
- We have integrated PV-RAN with the software-defined cyberinfrastructure CyNet [7], the first field-deployment of open-source, virtualized cellular platforms for smart agriculture and transportation.

The rest of this paper is organized as follows. We discuss related work in Section II. in Section III, we present the system model and introduce the Xen hypervisor and CyNet wireless living lab. We present the design and implementation of PV-RAN in Section IV. We evaluate the performance of PV-RAN in Section V. We share concluding remarks in Section VI.

II. Related Work

Virtualized RAN (vRAN) centralizes and virtualizes a baseband-unit pool (vBBU) at a strategic location (e.g., central office), where the vBBU can be deployed on commercial-off-the-shelf (COTS) servers rather than proprietary hardware. vRAN allows operators to rapidly expand the capacity and coverage of the network, thus minimizing capital expenditure (CAPEX) and operating expenditure (OPEX) [12]. Garcia et al. [13] proposed an analytical framework for vRAN called FluidRAN that optimizes the placement of vRAN functions jointly with the routing policy according to network and computing resources. Nikaein et al. [14] have considered potential vRAN architectures that could meet real-time deadlines and intensive computational and I/O requirements.

With the availability of SDR platforms and OAI and the growing interest toward virtualization, researchers can prototype low-cost cellular networks and evaluate the performance of different virtualized RAN architectures [15]. In [16], Tran et al. evaluated a C-RAN testbed based on the OAI RRH/BBU split and virtualized the OAI BBU-pool in VMWare virtual machines. Their study provides interesting insight into the computational requirements of vBBU under different radio resource configurations. The H2020 5G-PPP SliceNet project [17] introduced a slice-friendly virtualized 5G RAN where the Central Unit (CU) and Distributed Unit (DU) run in LXC containers. In [18], Trindade et al. containerized OAI in Docker and used Kubernetes to orchestrate the RRH and BBU containers. Nikaein et al. [19] investigated the performance of virtualized BBUs under containers (Docker, LXC) and KVM virtual machines.

Esmaeily et al. [5] recently introduced a testbed 5GIIK capable of performing E2E network slicing. They rely on the ETSI NFV MANO framework to manage and orchestrate the resources required for creating, managing, and delivering services through different slices. The FlexRAN [4] platform allows the programmable control of the underlying RAN infrastructure via virtual control functions, and it adds a virtualization layer over the RAN infrastructure to enable communication capacity allocation across slices.

The aforementioned studies have touched on various aspects of RAN virtualization, but they have not considered virtualizing the physical wireless resources for whole-stack slicing. The work closest to PV-RAN is that of the Wireless Spectrum Hypervisor [20]. It considered multiplexing several OFDM signal streams through a shared SDR. It cannot support slices running non-OFDM PHY layers; the spectrum hypervisor requires moving the OFDM modulation component from the user slices to the hypervisor layer, thus not being transparent to the slice users; it also uses ZeroMQ (a.k.a. ZMQ) bus for communication between user slices and the hypervisor, which

introduces much higher overhead than the lightweight inter-domain communication mechanism adopted by PV-RAN, as we show in Section V.

III. Preliminaries

System model. As shown in Figure 2, the system consists

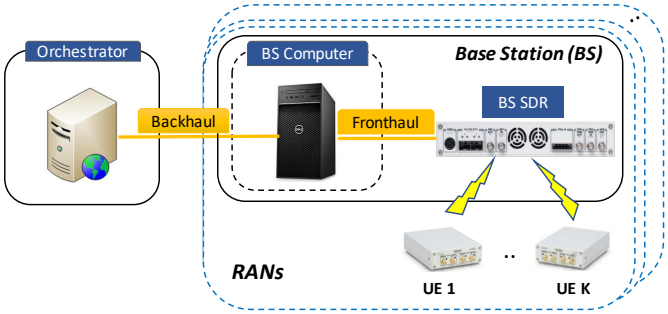

Fig. 2: System model

of an orchestrator and a set of RANs. Each RAN has a base station (BS) and a set of User Equipment (UEs), with each BS consisting of a SDR and a computer. The SDR serves as the remote radio head (RRH), and the computer runs the cellular network stacks realized as software. A network slice consists of a RAN slice at one or more RANs, and each RAN slice runs a cellular network stack implemented using OpenAirInterface (OAI) [11] with potentially different PHY and MAC layers. We consider Frequency-Division-Multiplexing (FDM) in RAN slicing such that different slices of a RAN operate in non-overlapping frequency bands. The specific frequency division strategies at different RANs could be different or the same, and this is coordinated through the central orchestrator and defined by software. Within this context, we study the systems issues of how to virtualize the BSes so that a BS SDR can be shared among multiple slices of the associated RAN to support whole-stack slicing (i.e., with each slice running potentially different PHY and MAC layers).

For simplicity of exposition, here we assume that each RAN only has one BS SDR, but the virtualization solution PV-RAN is readily extensible to the case where each RAN may have multiple SDRs. As we will explain shortly, a key task of PV-RAN is multiplexing/demultiplexing I/Q samples between the BS SDR and the network stacks of different RAN slices. Therefore, the architecture considered here resembles the RAN functional split Option 8 as defined by 3GPP [21]. The PV-RAN design, however, is readily extensible to other functional split options such as Options 7-1 and 7-2 where frequency-domain I/Q samples need to be multiplex-/demultiplexed between the BS SDR and different network stacks running on the BS computer. Similarly, the PV-RAN framework can be extended to network slicing strategies involving Time Division Multiplexing (TDM), and the network stacks could be based on other open-source cellular software platforms such as srsLTE. These are all interesting topics for future exploration, but their detailed studies are beyond the scope of this work.

Xen hypervisor. Considering the security benefits of virtual machines (VMs) as compared with containers [22], as well as the facts that VMs can be made lightweight [22] and that paravirtualization enables efficient I/O operations, we use the Xen hypervisor [10] as the base virtualization platform in our PV-RAN implementation. Other paravirtualization platforms such as KVM may also be used (e.g., thanks to its Nahanni shared memory interface), but detailed study is beyond the scope of this work.

In particular, we use Xen to manage the BS computer VMs used by individual slices of a RAN. Xen hypervisor runs directly on top of the hardware and provides abstraction and isolation required for the operation of multiple guest OSes hosted on the same physical computer. A Xen-based virtualization system consists of the hypervisor and the privileged domain Dom0 that has direct hardware access and can manage one or more unprivileged guest domains (DomUs).

In Xen, Dom0 has privileged access to hardware (e.g., SDR) while DomUs are typically not allowed to access hardware directly. However, Xen hypervisor supports an I/O virtualization mechanism, known as the *split-driver model*, allowing DomUs to access hardware through the use of virtual device drivers. In particular, a front-end device driver lies in a DomU and a back-end device driver resides in Dom0 and talks to the native device driver. The front-end and back-end drivers use I/O ring buffers to communicate requests/replies. I/O ring buffers rely on the *grant table* mechanism to share memory pages between domains, and they use *event channels*, an asynchronous notification mechanism, to notify a domain when there are waiting data in the I/O ring buffers. Thus, grant tables and event channels are convenient Xen facilities that allow to create I/O channels to establish bi-directional communication between domains. As we will explain in detail in Section IV, PV-RAN adopts the split-driver model for efficient I/Q sample streaming between Dom0 and DomUs.

CyNet wireless living lab. CyNet [7] is a field-deployed wireless living lab, consisting of two cellular RANs deployed at the Iowa State University Curtiss Research Farm and Research Park in the City of Ames, Iowa, USA. Each RAN has 2-3 USRP X310s as the BS SDRs and several USRP B210s as UEs. This work is motivated by the need for CyNet to partition its RANs into different slices that employ different PHY and MAC layers to support research and education in wireless, agriculture, and transportation respectively. PV-RAN has been tested and deployed with the CyNet infrastructure.

IV. PV-RAN Design and Implementation

In our implementation of PV-RAN, the BS SDR is the USRP X310, and the UE SDR is the USRP B210. For ease of discussion, we will refer to these specific SDRs and related software modules (e.g., the SDR driver UHD) in this section, but the design and implementation strategies of PV-RAN are readily applicable/extensible to other SDR platforms.

A. Architecture

Figure 3 shows the architecture of the PV-RAN. The software of the full OAI network stacks of user slices are deployed

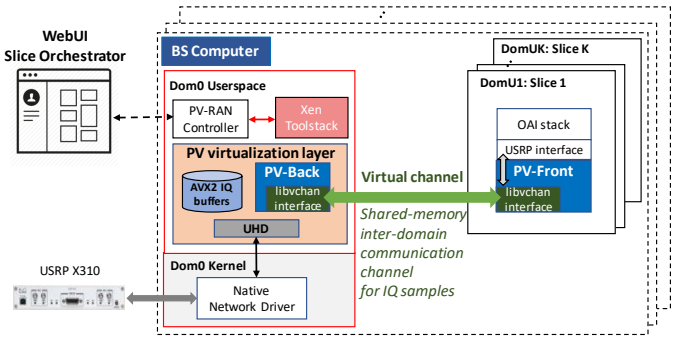

Fig. 3: Architecture of PV-RAN

in the DomUs of the Xen-virtualized BS computer, and they communicate with the physical SDR via the PV-Front in the individual DomUs; the virtual channel carries the I/Q samples through the Xen libvchan interface; PV-Back is deployed in Dom0 to perform all the baseband I/O operations from/to the USRP X310 via Dom0's kernel native network driver; USRP X310 integrates the antennas and acts as the RRH, and it communicates with Dom0 using the USRP Hardware Driver (UHD). The PV-RAN controller in Dom0 acts as a middleware layer between the WebUI Slice Orchestrator and the Xen toolstack `xl`, a Xen tool for the management of DomU lifecycle, by enforcing security and control. It enables the WebUI Slice Orchestrator, a Web portal for software-defined orchestration and management of DomUs, to send orchestration messages to the Xen toolstack that controls DomUs. When a specific action is invoked from the slice orchestrator (i.e.: requesting the state of a slice, creating a slice), an orchestration event is sent as a ZeroMQ request message to the PV-RAN controller. Upon reception of the request message, the PV-RAN controller deserializes it and translates the orchestration event as a DomU operation that can be executed by the Xen toolstack.

In order to run multiple heterogeneous paravirtualized OAI protocol stacks in different DomUs (i.e., user slices), each isolated OAI instance must appear to have dedicated access to the SDR. Since only one process can access the USRP Hardware Driver (UHD) at a time, we must decouple RF I/O (a.k.a. baseband I/O) from the PHY layer to allow multiple concurrent OAI instances to simultaneously access the USRP device. This is achieved by virtualizing the SDR hardware resource and delegating all RF I/O operations to a shim layer located in the Xen Dom0 and between the PHY layer of each paravirtualized OAI instance and the actual UHD. This shim layer is called the *PV virtualization layer*. The virtualization layer provides the following capabilities:

- **Slice isolation:** the virtualization layer provides a dedicated virtual channel to each paravirtualized OAI instance so that the OAI instances in different user slices are isolated from one another. The virtual channels are created using the *libvchan* interface, a Xen interface for inter-domain communication (IDC) that mainly relies on `gntalloc`, a userspace grant allocation driver that allocates shared pages, and `gntdev` that maps the granted pages into the address space of a userland process.
- **Virtualization transparency:** the OAI source code in user slices (i.e., DomUs) does not need to be changed in order to use PV-RAN. This is accomplished through *API Remoting*, a software-based paravirtualization technique, that virtualizes the SDR at the programming API level. In particular, the virtualization layer intercepts SDR function calls at run-time from the OAI instances running in DomUs, replaces them with API *stubs*, and forwards them to Dom0 which in turn interacts with the physical SDR to actually execute the function calls.

In what follows, we elaborate on the PV-RAN virtualization layer.

B. PV virtualization layer

Figure 4 shows the internals of the PV virtualization layer.

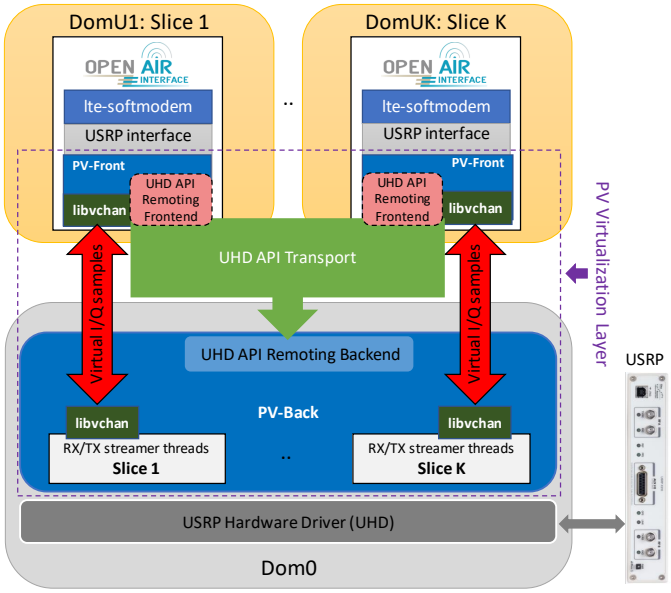

Fig. 4: PV-RAN virtualization layer

It consists of several cooperating modules located in DomU and Dom0. In what follows, we first present API Remoting for transparent virtualization, and then we present PV-Back for SDR virtualization and inter-domain streaming of I/Q samples. (Information on more detailed implementation strategies can be found in the technical report [23].)

1) Transparent virtualization via API Remoting: OAI provides low-level interfaces to different types of SDR platforms such as HackRF, BladeRF, and USRP. In this paper we focus the discussion on a commonly used base station SDR — USRP X310, but the design and implementation strategies can be applied to other SDRs. The USRP interface (usrp_lib.cpp) is a software module that sits between the PHY layer and the SDR RF front-end. It implements functions that rely on UHD API functions to configure the USRP X310 (channel, bandwidth, gains, etc.) and perform RF I/O operations (i.e., send and receive time-domain I/Q samples). Since Dom0

interacts directly with the USRP X310 via UHD, all the UHD API calls in DomU must be offloaded to Dom0. One naive approach to addressing this challenge is to directly modify the source code of OAI in each DomU. While OAI is an open-source software allowing for modifications, a software-based mechanism that enables the sharing of an SDR device among DomU guests without requiring OAI modification would be much more desirable. To this end, we develop the API Remoting method that comprises the following modules:

- *UHD API Remoting Frontend:* this module is a dynamic library that relies on library interposition. It intercepts at run-time each UHD library function call and substitutes it with a *stub* function or *wrapper* that will execute another piece of code that forwards the call through the UHD API Transport to the UHD API Remoting Backend. The dynamic library is loaded during execution into the memory space associated with OAI lte-softmodem process. This is achieved through the use of the `LD_PRELOAD` environment variable that instructs the loader to load the dynamic library. Each stub makes a call to `dlsym()` with `RTLD_NEXT` to locate the address of the related symbol in memory. Since OAI uses C++ bindings of the UHD API, we have to use mangled symbol names. Each UHD API C++ function has a mangled name that can be found in the dynamic symbol table.
- *UHD API Transport:* this transport component is a Xen event channel that allows the UHD API Remoting Frontend in PV-Front to send short control messages to the UHD API Remoting Backend in PV-Back. Each control message represents a UHD library function that must be executed by the UHD in Dom0. For instance, upon start-up, the OAI instance in DomU (in particular, *usrp_lib.cpp*) will first issue the `uhd::device::find()` call to find the USRP device connected. This call is then intercepted by the UHD API Remoting Frontend, and, upon interception, the stub calls another function that sends an `INIT` request to Dom0's UHD API Remoting Backend, which in turn 1) informs PV-Back in Dom0 to start streaming for this OAI slice and 2) returns to the UHD API Remoting Frontend in DomU the type of USRP device connected.
- *UHD API Remoting Backend:* this module handles the call requests received over the UHD API Transport and executes them on the SDR device using the actual UHD API. In its initial state, it waits for an incoming `INIT` request from a DomU's UHD API Remoting Frontend. This message is transmitted upon the execution of lte-softmodem in a DomU. The UHD API Remoting Backend processes the message and spawns RX/TX streamer threads which will be discussed shortly.

2) SDR virtualization through PV-Back in Dom0: Based on API remoting, PV-Back in Dom0 interacts with PV-Front in DomUs to perform RF I/O operations on behalf of paravirtualized OAI instances in different DomUs (i.e., user slices). It is designed to (i) initialize the USRP device, configure the RX/TX frequencies, gains, bandwidths, and rates of the communication channel according to OAI base station (e.g., eNodeB) configuration in DomU, and map UHD RX/TX streamers to a dedicated SDR radio channel; (ii) create bi-directional virtual channels and event channels for I/Q sample streaming and UHD API transport respectively; and (iii) run high-priority RX/TX streamer threads for the continuous I/Q sample streaming between the USRP device and paravirtualized OAI instances in DomUs through the stream-based libvchan communication interface (to be discussed shortly).

The RX/TX streamer threads are in charge of streaming I/Q samples. They are created on-demand upon the reception of an `INIT` request triggered when lte-softmodem is started in a DomU. Each paravirtualized OAI instance triggers the creation of two streamer objects: one RX streamer object that is the Dom0 interface to receive I/Q samples, and one TX streamer object that is the Dom0 interface to transmit I/Q samples. PV-Back maps both streamers to one SDR radio channel per DomU and spawns two threads for the reception/transmission of I/Q samples. Besides continuously receiving and transmitting I/Q samples, these two threads enable *lightweight, in-band timestamping synchronization* between PV-Back in Dom0 and and PV-Front in DomU as follows:

- *RX streamer thread:* it continuously reads `nsamples` (e.g., 7680 I/Q samples for 25-PRB channels) from the USRP device and then writes those samples to the virtual channel. During the first run, this thread generates a timestamp `rx_timestamp` and transmits it to PV-Front. This timestamp is then used by OAI's `trx_usrp_read` function as the time at which the first sample was received. (This timestamp is also used to compute the `tx_timestamp` as we discuss shortly.) After the first run, each time the thread performs an iteration (i.e.: at each subframe), it uses the previous `rx_timestamp` value and increases it by a fixed number depending on channel bandwidth (e.g., 7680 for 25-PRB channels); so does that OAI instance (in particular, the `trx_usrp_read` function) in the corresponding DomU. This way, the OAI instance in DomU and the TX streamer thread in Dom0 are synchronized in `rx_timestamp` without any explicit coordination/messaging after the first run.
- *TX streamer thread:* it continuously reads `nbytes` (e.g., 30720 bytes for the 7680 I/Q samples for 25-PRB channels) from the virtual channel and then transmits those samples to the USRP device. During the very first run, this thread is locked and waits for the RX streamer thread to compute the `rx_timestamp`. Once computed, the thread execution is resumed and it computes the appropriate `tx_timestamp`, which is used by PV-Back to inform the UHD `send()` function at what time the first sample must be sent. The `tx_timestamp` is computed considering the channel bandwidth. For instance, for 25-PRB channels, `tx_timestamp` is computed as `rx_timestamp + 30640`. The OAI instance

in the corresponding DomU (in particular, functions in lte-ru.c) also calculates, in the same way as the TX streamer thread in PV-Back, the `tx_timestamp` based on the `rx_timestamp` it receives from the RX streamer thread, and it uses it in the OAI code execution in DomU. This way, the TX streamer thread in Dom0 and the OAI instance in DomU are synchronized in `tx_timestamp` without any explicit coordination/messaging. Then, for each iteration of the virtual channel read, the thread calculates the `tx_timestamp` again by using the previous `tx_timestamp` value and increasing it by a fixed number (e.g., 7680 for 25-PRB channels).

3) Inter-domain streaming of I/Q samples: PV-Back in Dom0 and PV-Front in DomUs create shared memory pages to communicate with each other. This functionality is available through the use of the grant table mechanism that allows kernel memory pages to be shared between domains. Each domain possesses a grant table which is a set of pages shared between itself and the Xen hypervisor. The domains involved in this mechanism are referred to as the client and the server, where the server offers the memory used for communication and transmits its credentials (grant reference, event channel ID) to the client in a Xen directory service known as XenStore. The XenStore is a centralized database that stores key-value pairs and relies on the Linux kernel interface XenBus for messaging. To get access to the server's shared memory, the client needs the server's domain ID and the XenStore path in which the server offered its credentials. This phase is known as the rendezvous procedure.

Now, let's observe the mechanism in more detail by considering a DomU A, the client, wishing to communicate with another DomU B, the server:

- DomU B creates a page entry in its grant table and then advertises the index of this entry (grant reference) to DomU A through a dedicated Xen event channel. The kernel driver *gntalloc* is the one in charge of creating grant references.
- Upon reception of the grant reference, DomU A validates the grant and maps the page via the *gntdev* device into the address space of the application running in DomU A.

Once the rendezvous procedure has been completed, Xen makes use of shared data structures called *ring buffers* for bulk data transfer. Historically those ring buffers have been used by split drivers to communicate I/O requests-responses.

In this paper we use Xen virtual channel library *libvchan*, a library that implements a datagram-based (packet-based and stream-based) interface on top the standard Xen ring buffers. Libvchan allows to specify the size of the rings and whether or not to perform blocking I/O operations on the virtual channel. As OAI deals with streams of I/Q samples, we use the stream-based communication interface that is composed of *libxenvchan_read* and *libxenvchan_write* functions to read I/Q samples from a buffer and write I/Q samples to a buffer respectively. The blocking I/O flag is set on the bi-directional virtual channel to ensure that I/O operations are performed sequentially.

V. Measurement Evaluation

We have implemented the PV-RAN design [23] and integrated it with the CyNet wireless living lab [7]. A recorded demo of the operational PV-RAN can be found at [24]. Here we conduct detailed measurement benchmarking to characterize the performance of the PV-RAN platform.

A. PV-RAN testbed

Fig. 5: PV-RAN testbed at ISU Research Park

We set up a testbed in the basement of the Iowa State University (ISU) Research Park to conduct our measurement evaluation (see Figure 5), with the basement having significantly less interference from surrounding wireless equipment. The testbed is composed of:

- One PV-RAN platform running two slices: the base station computer hosting PV-RAN under the Xen hypervisor is a Dell Precision 3630 with 64GB of RAM and 8 CPU cores operating at 3.1GHz. Each DomU slice has been set up as a paravirtualized guest under Ubuntu 16.04 and is configured with 4 vCPUs and 6GB of RAM. Each DomU slice runs OpenAirInterface as the LTE protocol stack. Bi-directional virtual channels transport I/Q samples between each DomU PV-Front and Dom0 PV-Back. Dom0 runs under Ubuntu 18.04 and has direct I/O access to the USRP X310 which is connected to the base station computer through OS2 single-mode fiber.
- Two UEs: each UE runs on an Intel NUC7i7BNB under Ubuntu 16.04 with a low-latency kernel. Each NUC is connected via USB 3.0 to a USRP B210 that has been equipped with two VERT900 antennas that operate in TV White Space (TVWS) frequencies.

B. Network performance tuning

The USRP X310 is connected via single-mode fiber to one of the SFP+ network interface of the base station computer which provides a 10Gbps network speed. Table I summarizes the parameters that must be fine-tuned to allow the best network performance possible. We used ifconfig and ethtool to configure the network interface. It is recommended to use jumbo frames (for larger-packet transmission support), to increase the size of RX/TX kernel buffers and Ethernet buffer rings, to disable Ethernet frame pauses (RX/TX autoneg off), and to minimize the raise of interrupts to the CPU during packet reception to improve CPU utilization.

Network interface parameters	
MTU size	8000 bytes
Queue length	1000 bytes
RX/TX autoneg	off
Ethernet buffer rings	4096 bytes
Interrupt moderation	3 μsecs

TABLE I: Hardware and software setup

C. Throughput and Latency

To assess the smooth operation of PV-RAN, we compare the throughput of one PV-RAN slice running OAI against a non-virtualized instance of OAI running in Xen Domain-0 on the BS computer. Throughout the rest of this section we refer to the latter as Bare-metal OAI. Both the PV-RAN OAI and Bare-metal OAI use the same LTE bandwidth of 25 PRBs (5 MHz) and operate at the TVWS downlink frequency and uplink frequency of 592MHz and 496MHz respectively. The main eNB parameters are shown in Table II along with UE parameters.

eNB configuration	
LTE bandwidth	25 PRBs
Downlink frequency	592 MHz
Uplink frequency	496 MHz
PDSCH reference signal power	-27 dB
UE configuration	
RX gain	130 dB
TX gain	0 dB
Max UE power	0 dB

TABLE II: eNB and UE configuration

We attach the UE to the eNB and measure the UDP throughput for 180 seconds using the iperf tool. Results from Table III show that both PV-RAN and Bare-metal OAI exhibit

OAI version	Mean Throughput (Mbps)
Bare-metal	8.29 $\pm$ 0.00593 [8.28407, 8.29593]
PV-RAN	8.29 $\pm$ 0.00733 [8.28178, 8.29822]

TABLE III: Mean throughput and 95% confidence interval of PV-RAN and bare-metal OAI with 25 PRBs

the same mean throughput of 8.29Mbps. Besides, PV-RAN and bare-metal OAI throughput are statistically equivalent at the 95% confidence level. Figure 6 shows the detailed temporal behavior of PV-RAN and bare-metal, confirming the statistical equivalence again.

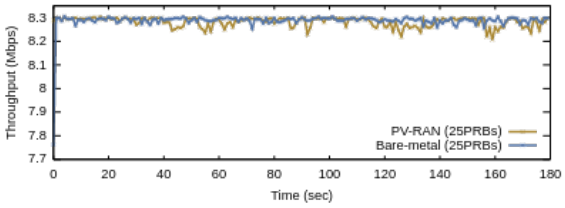

Fig. 6: Throughput for 25 PRBs

Next, we evaluate the performance when two OAI slices are running on PV-RAN. In this experiment, we measure the RTT and throughput between the OAI eNB and UE in each slice. Each OAI eNB uses a channel bandwidth of 25 PRBs. Both slices operate in TVWS bands: slice 1 runs at 592MHz (downlink) and 496MHz (uplink), and slice 2 at 580MHz (downlink) and 484MHz (uplink). Each slice occupies a different radio channel on the USRP X310. We connect one UE to each slice and measure the RTT with the ping command and the UDP throughput with the iperf command. The ICMP echo requests have been sent from each UE to their respective eNB.

The top graph in Figure 7 illustrates the RTT variation over time for each UE-eNB pair. The average RTT is of 25ms for slice 1 and 22ms for slice 2. We witness similar trends with OAI running in Xen Domain-0 (bare-metal OAI). The throughput of each OAI slice is depicted in Figure 7. It can be

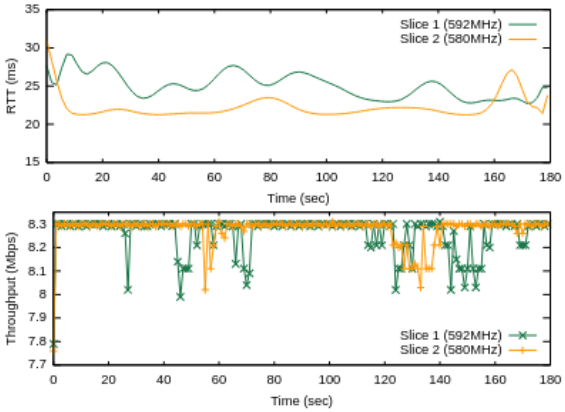

Fig. 7: RTT and Throughput for each PV-RAN OAI slices

seen that slice 1 achieves a stable throughput of ~8.25Mbps on average while slice 2 achieves ~8.27Mbps on average, which are also comparable to throughput achieved with bare-metal OAI. The reason why slice 2 has slightly lower RTT and higher throughput than slice 1 is because the UE in slice 2 is closer to its eNB and thus have better channel quality. (The slice throughput is less stable at the beginning phase when UEs and eNBs dynamically tune their communication parameters such as transmission power and MCS schemes.)

D. CPU and memory utilization

It is of paramount importance to understand the CPU consumption of our PV-RAN platform. For this study, all CPU cores ran at their maximum frequency in Dom0. Each slice spawns two threads in PV-Back, namely *stream-rx* that receives I/Q samples from the USRP device and writes them to the virtual channel, and *stream-tx* that reads I/Q samples from the virtual channel and sends them to the USRP device. We pinned each thread onto a single CPU core using `pthread_setaffinity_np` to have a better understanding of CPU usage at a thread level. In this experiment, the useful Linux task monitoring tools `top` and `htop` have been used to collect CPU consumption of individual threads.

The CPU core usage per thread is reported in Figure 8 for a single OAI slice running on PV-RAN. The *main* thread listens for incoming PV-Front requests and is responsible for the creation of USRP device object and virtual channel data

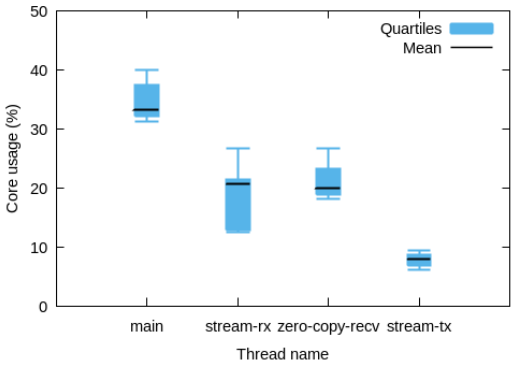

Fig. 8: Core usage per thread for Dom0 PV-Back

structures. It has a mean CPU core consumption of 33.3%. It can be observed that an additional thread named *zero-copy-recv* is spawned by the call to the UHD *recv()* function in the *stream_rx* thread. This thread is one of the most influential thread in terms of CPU consumption for OAI eNBs. It helps to minimize unnecessary buffer copy operations between the NIC and userspace by bypassing buffer copy operations from the NIC to the kernel space and from the kernel space to the userspace. *stream-rx* and *zero-copy-recv* consume 20.7% and 20% respectively. These tasks play a major role in CPU consumption, while in contrast the thread *stream-tx* has a mean consumption of only 8% of its CPU core.

We also performe a comparative analysis of the total CPU consumption (over all vCPUs for PV-Front and over all CPU cores for PV-Back) for different LTE bandwidths of OAI running as a PV-RAN slice. The CPU usage over all cores is depicted in Figure 9. It can be seen that, for a LTE bandwidth

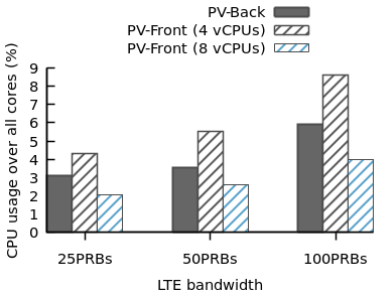

Fig. 9: Mean CPU usage of PV-Back and PV-Front

of 25 PRBs, PV-Back consumes on average 3.09% of all CPU cores. When the LTE channel bandwidth is doubled (i.e., with 50 PRBs), the CPU usage of PV-Back increases only by 0.43%. PV-Front on the other hand consumes 5.5% of all vCPUs when DomU uses 4 vCPUs for 50 PRBs. Doubling the number of vCPUs leads to a reduction of ∼47% of the total vCPU usage of PV-Front for 25, 50 and 100 PRBs. Therefore, the CPU usage overhead due to virtualization in PV-RAN is small, and tends to be negligible as channel bandwidth increases.

The current implementation of PV-Back supports LTE bandwidths of 25, 50 and 100 PRBs in FDD mode. We plan to incrementally add support for other features such as TDD mode. The PV-Back code is composed of ∼1,500 lines of code and its memory consumption (Resident Set Size, measured with the pmap tool) does not increase significantly with the channel bandwidth as depicted in Figure 10. The PV-RAN implementation will be made available upon publication.

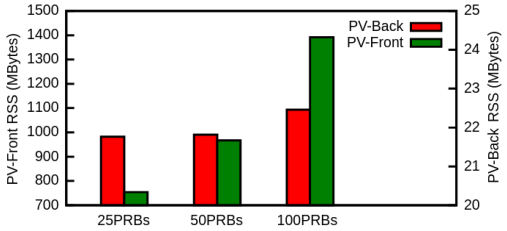

Fig. 10: Resident Set Size (RSS) of Dom0 PV-Back and DomU PV-Front for different channel bandwidths

E. Latency overhead

To complete our measurement study, we focus on latency overhead which, if not controlled to be low, can be detrimental to I/Q sample streaming and cellular communication performance. Since we add an extra layer between OAI USRP interface and UHD I/O functions to transfer I/Q samples, we need to evaluate the latency overhead introduced by the PV virtualization layer. For this case study, in particular, we add timestamps in PV-Front and PV-Back to measure the latency overhead caused by the virtual channels. First, we present the mean I/Q streaming latency (i.e.: transmission time) for different LTE bandwidths, relying on the buffer data format that is native to OAI. OAI relies on SIMD (single instruction multiple data), more precisely AVX2 intrinsics ($\_m256i$ data type) for x86-64 architectures to store 16-bit precision I/Q samples. Figure 11 shows the I/Q streaming

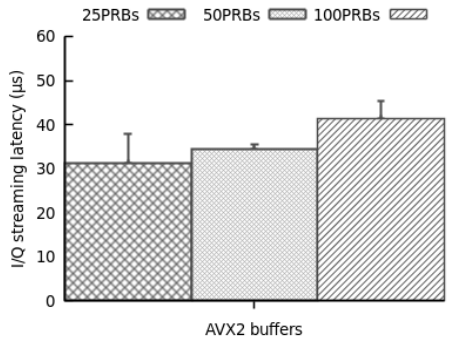

Fig. 11: Latency of inter-domain I/Q sample streaming

latency in PV-RAN. Surprisingly, for a channel bandwidth of 25 PRBs, the mean streaming latency introduced by the virtual channel is small and only 31.3μs on average. For a channel bandwidth of 50 PRBs which doubles the ring buffer size, the mean streaming latency is also only 34.3μ. This low latency overhead and high efficiency of PV-RAN explain the good throughput performance of PV-RAN discussed earlier.

In addition to the streaming latency induced by the transmission of I/Q samples, we investigated the latency overhead caused by the execution of I/O operations on those channels (ie.: `read` and `write`). We measured the overhead latency induced by libvchan I/O operations when DomU PV-Front writes I/Q samples (referred to as the *writer*) and PV-Back Dom0 reads them (referred to as the *reader*). In Figure 12 we can observe that the writer latency is relatively small. Up to 100 PRBs, the writer latency does not seem to be affected by the increase in ring size (the writer latency goes from 19.3μs to 24.4μs for 25 PRBs and 100 PRBs respectively). On the other hand, the reader latency increases proportionally to the increase in LTE bandwidth. We noticed a mean reader latency of 52.5μs and 81.1μs for 25 PRBs and 50 PRBs respectively, and both exhibit a standard deviation of $\sim$30% of their mean value. This higher latency paves the way for possible future improvements and minimizing the overhead is left as future work.

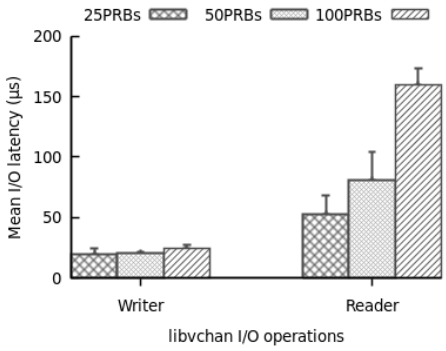

Fig. 12: I/O latency overhead of PV-RAN

VI. Concluding Remarks

As the first open-source platform for physical wireless resource virtualization, PV-RAN enables whole-stack slicing where different PHY and MAC layers may be adopted for diverse communication services and wireless living lab innovations. Using efficient techniques such as API Remoting, shared-memory I/Q sample streaming, and light-weight in-band timestamping synchronization, PV-RAN enables transparent, lightweight virtualization of SDRs, and it serves a solid foundation for further development. For instance, the PV-RAN platform can be extended to support SDR virtualization using TDM and hybrid-TDM-FDM, and the PV-RAN implementation can be ported to other open-source platforms such as KVM (using the shared-memory interface Nahanni) and srsLTE to expand community access to PV-RAN capabilities. PV-RAN can also be integrated with end-to-end network slice management systems such as those based on the O-RAN architecture and Open Networking Foundation implementation frameworks.

References

[1] F. Alvarez, D. Breitgand *et al.*, "An edge-to-cloud virtualized multimedia service platform for 5G networks," *IEEE Transactions on Broadcasting*, vol. 65, no. 2, pp. 369–380, 2019.

[2] 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.

[3] S. Peng, R. Chen, and G. Mirsky, "Packet network slicing using segment routing," Tech. Rep. Draft-penglsr-network-slicing-00. IETF, Tech. Rep., 2019.

[4] "flexran: A flexible and programmable platform for software-defined radio access networks."

[5] A. Esmaeily, K. Kralevska, and D. Gligoroski, "A Cloud-based SDN/NFV Testbed for End-to-End Network Slicing in 4G/5G," *arXiv:2004.10455*, 2020.

[6] M. Yang, Y. Li, D. Jin, L. Zeng, X. Wu, and A. V. Vasilakos, "Software-Defined and Virtualized Future Mobile and Wireless Networks: A Survey," *Mobile Networks and Applications*, vol. 20, 2015.

[7] "CyNet: End-to-End Software-Defined Cyberinfrastruture for Smart Agriculture and Transportation," https://www.ece.iastate.edu/~hongwei/group/projects/CyNet.html.

[8] Y. Xie, H. Zhang, and P. Ren, "Unified scheduling for predictable communication reliability in industrial cellular networks," in *IEEE ICII*, 2018.

[9] E. Research, "USRP Software-Defined Ratios," https://www.ettus.com/product.

[10] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the Art of Virtualization," in *ACM SOSP*, 2003.

[11] N. Nikaein, M. K. Marina, S. Manickam, A. Dawson, R. Knopp, and C. Bonnet, "OpenAirInterface: A Flexible Platform for 5G Research," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 5, pp. 33–38, 2014.

[12] V. Suryaprakash, P. Rost, and G. Fettweis, "Are heterogeneous cloud-based radio access networks cost effective?" *IEEE Journal on Selected Areas in Communications*, vol. 33, no. 10, pp. 2239–2251, 2015.

[13] A. Garcia-Saavedra, X. Costa-Perez, D. J. Leith, and G. Iosifidis, "FluidRAN: Optimized vran/mec orchestration," in *IEEE INFOCOM*, 2018.

[14] N. Nikaein, E. Schiller, R. Favraud, R. Knopp, I. Alyafawi, and T. Braun, "Towards a cloud-native radio access network," in *Advances in mobile cloud computing and big data in the 5G era*. Springer, 2017, pp. 171–202.

[15] F. Kaltenberger, A. P. Silva, A. Gosain, L. Wang, and T.-T. Nguyen, "Openairinterface: Democratizing innovation in the 5g era," *Computer Networks*, p. 107284, 2020.

[16] T. X. Tran, A. Younis, and D. Pompili, "Understanding the computational requirements of virtualized baseband units using a programmable cloud radio access network testbed," in *IEEE ICAC*, 2017.

[17] I. Sanchez-Navarro, A. S. Mamolar, Q. Wang, and J. M. A. Calero, "5gtoponet: Real-time topology discovery and management on 5g multi-tenant networks," *Future Generation Computer Systems*, vol. 114, pp. 435–447, 2020.

[18] I. Trindade, C. Nahum, C. Novaes, D. Cederholm, G. Patra, and A. Klautau, "C-ran virtualization with openairinterface," *arXiv preprint arXiv:1908.07503*, 2019.

[19] N. Nikaein, "Processing radio access network functions in the cloud: Critical issues and modeling," in *ACM MCS*, 2015.

[20] F. A. de Figueiredo, R. Mennes, I. Jabandžić, X. Jiao, and I. Moerman, "A baseband wireless spectrum hypervisor for multiplexing concurrent ofdm signals," *Sensors*, vol. 20, no. 4, p. 1101, 2020.

[21] "Study on new radio access technology: Radio access architecture and interfaces," *3GPP TR38.801 Release 14*.

[22] F. Manco, J. Mendes, K. Yasukata, C. Lupu, S. Kuenzer, C. Raiciu, F. Schmidt, S. Sati, and F. Huici, "My VM is Lighter (and Safer) than your Container," in *ACM SOSP*, 2017.

[23] M. Sander-Frigau, T. Zhang, H. Zhang, A. E. Kamal, and A. K. Somani, "Physical wireless resource virtualization for software-defined whole-stack slicing," Iowa State University, Tech. Rep. ISU-DNC-TR-20-02 (https://www.ece.iastate.edu/~hongwei/group/publications/PV-RAN-TR.pdf), 2020.

[24] "PV-RAN Demo," https://www.ece.iastate.edu/~hongwei/group/projects/CyNet/PV-RAN-CyNet-Demo.mp4.