PRAN: Programmable Radio Access Networks*

Wenfei Wu *◇ Li Erran Li* Aurojit Panda† Scott Shenker†
Bell Labs, Alcatel-Lucent* UW-Madison◇ UC Berkeley†
wenfeiwu@cs.wisc.edu, erranlli@research.bell-labs.com, apanda@cs.berkeley.edu, shenker@icsi.berkeley.edu

ABSTRACT

With the continued exponential growth of mobile traffic and the rise of diverse applications, the current LTE radio access network (RAN) architecture of cellular operators face mounting challenges. Current RAN suffers from insufficient radio resource coordination, inefficient infrastructure utilization, and inflexible data paths. We present the high level design of PRAN, which centralizes base stations' L1/L2 processing into a cluster of commodity servers. PRAN uses a flexible data path model to support new protocols; multiple base stations' L1/L2 processing tasks are scheduled on servers with performance guarantees; and a RAN scheduler coordinates the allocation of shared radio resources between operators and base stations. Our evaluation shows the feasibility of fast data path control and efficiency of resource pooling (a potential for a 30× reduction on resources).

Categories and Subject Descriptors

D.2.8 [**Computer-Communication Networks**]: Network Architecture and Design—*Wireless Communication*

General Terms

Design; Performance

1. INTRODUCTION

The radio access network (RAN) of the cellular network infrastructure provides wide-area wireless connectivity to mobile devices. Current LTE RANs consist of a collection of largely independent base stations. The data plane of base stations runs standard LTE PHY and MAC, and supports a small number of QoS classes. Base stations run distributed algorithms to loosely coordinate among each other for handover and interference management.

The current RAN architecture faces three big trends. To cope with exponential traffic growth, operators are increasingly deploying small cell base stations in a dense and unplanned fashion. To support the growing diversity of applications such as intermittent machine type-communication (MTC) [12] and automotive applications, new protocols are being standardized. To reduce the huge deployment and operational cost, LTE RAN infrastructure sharing mechanisms have been developed.

However, these band-aid solutions run into significant problems. First, distributed coordination with unplanned and dense deployment makes radio resource allocation very inefficient and makes managing interference harder. This calls for a logically centralized coordinator. Second, current RAN data plane at layer 1 and layer 2 (L1/L2) is not programmable and only offers a small number of QoS classes. Diverse applications and evolving protocols require a programmable data plane. For example, video traffic of gold subscribers should go through Unequal Error Protection (UEP) block, sensors with limited computation capability should use repetition coding instead of Turbo coding. Third, base stations are provisioned for peak demands and the excess capacity is idle at other points. Since the time of peak demand varies from base station to base station the network itself is heavily over-provisioned. Fourth, the need for RAN sharing among operators should not compromise the ability to control the data plane. Current LTE RAN sharing mechanisms do not provide operators direct control of the shared radio resources.

To overcome these problems, we present PRAN, a Programmable Radio Access Network. PRAN centralize the L1/L2 processing of base stations into a cluster of commodity servers. We make the following contributions in designing PRAN:

- PRAN coordinates radio resource (time, frequency,

*This work was performed when Wenfei was an intern at Bell Labs and was supported by NSF grant CNS 1218668.

HotNets-XIII, October 27–28, 2014, Los Angeles, CA, USA.

http://dx.doi.org/10.1145/2670518.2673865

location) usage among operators and enables each operator to flexibly control its radio resources.
- PRAN offers unprecedented programmability to configure different data paths for each operator or each application, so that protocols can be easily upgraded and replaced.
- PRAN enables dynamic computational resources sharing and scheduling among base stations' processing so as to increase the utilization of the infrastructure.
- Based on real traces of more than 3000 base stations from a national cellular operator, our initial evaluation shows PRAN reduces CPU requirements by a factor of 30.

2. PRAN OVERVIEW

We design the PRAN architecture to satisfy the new requirements facing current RANs, then we present PRAN's use cases. We finally address its implementation challenges.

2.1 PRAN Architecture

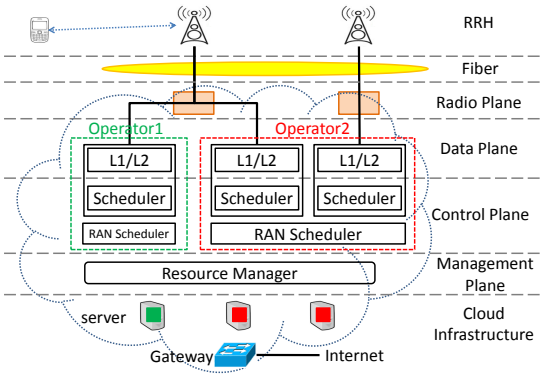

Figure 1: PRAN Architecture

PRAN's hardware infrastructure consists of remote radio heads (RRHs) distributed in a region, centralized commodity servers with DSP accelerators and high-speed interconnects between RRHs and servers. A RRH has antennas and analog components that convert received radio signals into digital samples (I and Q) in the uplink and turn received digital samples into analog signals and transmits them over the air in the downlink. The high-speed interconnects can be fibers or microwaves connecting RRHs with the cluster of commodity severs. The commodity servers perform L1/L2 and other data processing in the RAN.

PRAN has 4 planes: **a radio plane**, **a data plane**, **a control plane** and **a management plane**. The radio plane isolates radio resources among different operators. The data plane includes data paths of UEs' data streams (a stream of bits processed by L1/L2 such as transport block in MAC, code block in PHY, I and Q samples after modulation), and there can be multiple data paths in the data plane (e.g. different modulation schemes, MIMO or not). The control plane configures the data plane and directs each data stream through one of the available data paths according to the environment (e.g. channel condition) or an operator's policy (e.g. middleboxes, application types). Each base station's data plane and control plane are packed as a task; the management plane assigns computational resources (e.g. CPU cores) to tasks.

2.2 Use Cases

We present a few use cases for how one could use PRAN to solve problems in current cellular network deployments.

New PHY/MAC. PRAN's modular design makes it easy to introduce new PHY and MAC protocols in the data plane or control plane. For example, for the data plane, an operator can reconfigure the symbol length, frame structure and physical channels. For the control plane, an operator can reconfigure the scheduling algorithm to support cross-carrier scheduling. Cross-carrier scheduling enables one carrier to schedule transmissions of another carrier. Operators can also introduce new interference management algorithms.

Flexible Data Paths. PRAN offers per-flow customization of data paths. For example, video traffic uses Unequal Error Protection in the physical layer and subject to video transcoding if there is congestion. Voice traffic is scheduled by the semi-persistent scheduler instead of the dynamic scheduler. Voice traffic may go through the echo cancellation function depending on the measured channel quality. Voice traffic header must be compressed.

Flexible RAN Sharing. Current LTE RAN sharing is controlled by a single entity. In PRAN, each operator can have a RAN scheduler to directly control the radio resources of its own slice so as to schedule transmissions and manage interference.

2.3 Challenges and Solution Overview

To realize a programmable RAN, we face several challenges. We leverage properties of L1/L2 functions to design effective solutions.

Challenge 1: Satisfy Time Constraint. In a RAN, many data operations in L1/L2 have very strict time constraints. For example, a transport block (a sequence of bits from MAC layer to PHY layer) of a subframe must be processed within the subframe's duration (1ms in LTE), otherwise data will be backlogged in the MAC layer; a received transport block must be acknowledged within a certain time interval (4ms in LTE), otherwise the sender would regard that block as lost and retransmit it. It is challenging to meet deadline requirements if multiple UEs are processed in parallel and many L1/L2 functions are performed in software.

Solution: While predicting the time needed to

compute a function is hard in general systems, most processing operations in L1/L2 take time proportional to length of the input and are independent of the exact input (e.g. turbo coding, FFT). Therefore we can easily predict the processing time needed to process particular input. In a RAN, downlink and uplink data transmissions are scheduled ahead of time and the resource requirements are predictable, we can allocate computation resources to the processing operations to satisfy a given deadline. As a further optimization, servers can have DSP chips installed, so some data processing (e.g. turbo coding, FFT) can be offloaded to hardware reducing processing time.

Challenge 2: Handle Bursty Traffic. Computational resources (servers) are shared by base stations, and downlink and uplink data transmissions are scheduled at the subframe granularity (1 ms in LTE). When bursty traffic comes, it is hard to reallocate computational resources to busy base stations at fine granularity [10]. This is because the transport blocks and the state needed have to be sent to other servers. This overhead is not negligible [15, 5, 17].

Solution: Adapting to traffic patterns, we try to make the optimal tradeoff between the amount of dedicated resources and shared resources to minimize the total computational resources needed. Different UE's L1/L2 data block processing can be performed independently. Offloading at the granularity of per UE per data block makes it easier to use dynamic resources to meet deadline requirements.

Challenge 3: Provide Programmability. RANs should support the need for rapid change of L1/L2 functions and per operator customization of data paths.

Solution: PRAN modularizes L1/L2 processing into blocks which are used to implement basic processing. PRAN provides mechanisms for operators to program and implement their own processing blocks. Operators are required to provide both the code and timing constraints for these blocks, the timing constraints can then be used by a scheduler to drive processing. Further, we envision providing a compiler that is capable of targeting multiple backends (i.e., x86, DSPs, FPGAs, etc.) and can use this compiler to allow the scheduler to place processing blocks on a variety of hardware[6, 11, 19, 18].

3. PRAN PLANES

In PRAN, the radio plane enables the radio resource sharing among operators; To achieve flexible control, the control plane of L1/L2 processing tasks is decoupled from the data plane; and the management plane assigns computational resources to the base station processing so as to provide performance guarantee.

3.1 Radio Plane

Our radio plane ensures isolation of radio resources among different operators applying the technique in RadioVisor [9]. Our radio plane enables each operator to flexibly processing its data plane. In particular, each operator controls its own match-action table at the radio slicing tier. The radio slicing tier performs iFFT and demaps the I and Q samples of resource blocks to their relevant operators. An operator can match on RRH, one or more resource blocks and direct the I and Q samples to a specific server for further processing.

3.2 Data Plane

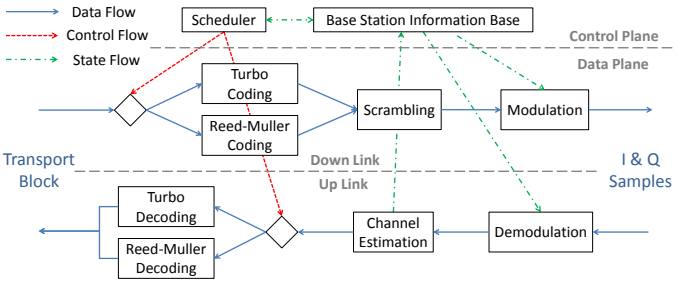

Figure 2: Data Plane and Control Plane (PHY)

LTE link layer has three sublayers: packet data convergence protocol (PDCP), radio link control (RLC) and media access control (MAC). When a data stream traverses L1/L2, it is transformed multiple times. The data streams are segmented into fixed sized data blocks. These data blocks are called transport blocks in MAC, code blocks in PHY. In the downlink, an IP packet experiences header compression in PDCP, segmentation in RLC, coding, scrambling and modulation in PHY; in the uplink, the transformation is reversed. PRAN attaches meta data to each block. In meta data, some properties of a block are attached (e.g. UE ID, subframe number). These meta data fields are used to select specific data path.

A base station's data plane is abstracted as a directed graph composed of **decision blocks** and **processing blocks**. Note that, unlike OpenRadio [3], the graph can be cyclic. For example, if successive interference cancellation is used, there will be a cycle in the graph. Suppose there are two transmissions in the received I and Q samples. Once one stream is decoded. Its effect is subtracted from the original I and Q samples, and the resulting I and Q samples (together with the decoded bits) are routed to the start of the decoding pipeline. The cycle traversal will terminate when all the component data streams are processed.

Processing blocks: A processing block is usually single-in-single-out with data streams traversing it, and it implements a simple functionality in L1/L2. A processing block can also attach properties to the data blocks it processes. A processing block may need configurations from the control plane; and they can

also write to base station information base (BIB) in the control plane (more on BIB in control plane section).

Decision blocks: A decision block usually has one or more inputs and multiple outputs to different processing blocks. A decision block has a table of match-action rules. The match field of a rule is properties of the input data blocks, and the action field is one or more of the output ports. Some ports connect to upper or lower layer. Some ports connect to processing blocks. Some ports are NULL ports (data blocks will be dropped, e.g. missed deadline). The match field can use the meta data attached to data blocks. The decision block has a pipeline of reading meta data, matching meta data, and taking action. Decision block is dumb and only accept configurations from the scheduler in the control plane.

State caching and invalidation: Processing and decision blocks cache the state information read from the control plane. The state information is tagged with its subframe number. When a processing or decision block processes a data block with the next subframe number, the old state will be invalidated and new state will be requested or configured by the control plane. This simple mechanism ensures consistent state.

Figure 2 is an example of control plane and PHY part of data plane. Turbo/Reed-muller coding/decoding are processing blocks which require no configuration; modulation/demodulation needs the configuration of current modulation and coding scheme (MCS) (i.e. the modulation and coding rate) from the control plane; channel estimation block writes the result to the control plane. A transport block's meta data contains the UE ID. The control plane decides the MCS that a UE should use, and writes the rules (UE ID as match and output port as action), so that the transport block is switched to its current coding/decoding processing block.

Reconfiguring Data Plane: The operator can also reconfigure the directed graph of the data plane. For example, it can dynamically insert a processing block, add the associated rules in the decision block and activate specific ports of the connected blocks.

3.3 Control Plane

The control plane of a base station consists of a scheduler and a base station information base (BIB).

Information base: BIB is used to share information between the scheduler and the processing blocks in the data plane. BIB has three types of information: (1) cell (each base station has multiple cells) specific information such as cell ID, cell specific scrambling sequence, number of antennas, (2) UE specific information which contains static information such as UE capability, UE ID and dynamic information such as flow ID, MCS, application type, transmission mode (MIMO or not), (3) network-wide configuration information such as frame length, size of control channel, etc. The union of BIBs are the RAN information base (RIB).

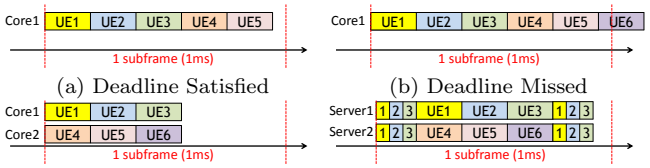

Figure 3: Offload Processing

Lock free shared access of BIB: There are two writers: the scheduler and the channel estimation block. The scheduler writes to the BIB that are read by multiple processing or decision blocks. The scheduler will not initiate a subframe's processing pipeline before it finishes writing the BIB. Therefore, there is no need for locks for the shared access of BIB. The channel estimation block writes to the BIB. The scheduler reads the BIB only after it was notified that new channel estimation information is ready.

Each base station scheduler or the cooperative RAN scheduler has the logic to determine a UE flow's processing pipeline and configures the blocks in the data plane. As Shown in Figure 2, the scheduler decides and configures what modulation and coding rate to use per subframe (1 ms) based on channel state information reported by UEs.

In an LTE control plane, each base station scheduler basically implements the following logic:

- Setup per-flow data path.
- Receive channel condition estimation from UEs and determine their MCSs and transmission modes (e.g. MIMO or not).
- The RAN scheduler may make partial decisions on resource block (time and frequency) allocation in order to reduce interference.
- The base station scheduler assign resource blocks to UEs within the constraints made by the RAN scheduler.
- Schedule retransmission of failed data blocks (no acknowledgement or validation failure).

3.4 Management Plane

A base station's control plane and data plane are packed as one base-station task. The management plane is in charge of allocating computational resources to these tasks. PRAN dedicates computational resources to each base-station task adaptively and periodically, and it also reserves a share resource pool. When a task receives bursty traffic whose processing requirement exceeds the current resource allocation, the task offloads computation onto a shared-pool of currently idle resources.

Predicting resource needed per subframe: L1/L2

processing has the following properties which helps us to propose a precise resource allocation.

- The data processing in L1/L2 is CPU bound. The number of CPU cycles that a task gets is the critical factor that determines its processing time. So we only consider CPU cores allocation.
- The data processing has fixed computation steps (e.g. FFT, turbo coding), so the processing time to transfer a data block is fixed and can be profiled.
- Radio resource blocks are scheduled and assigned to UEs. Therefore, the scheduler knows the resource requirement of a subframe before the data plane processing of the subframe starts.

We can predict the resource utilization of a base station in a subframe in terms of CPU cores. Assume a subframe lasts time T, if a data path takes t to process a data block in 1 subframe, that data path's resource requirement can be profiled as $c = \frac{t}{T}$ CPU cores. In 1 subframe, if UE i of a base station requires c_i cores, then that base-station task requires at least $C = \sum_{i=1}^{n} c_i$ cores. To obtain the actual number of cores needed, we use a simple greedy bin packing algorithm that leaves sufficient slack times at each core (to absorb variations of processing time). For example, in Figure 3(a), the total processing time of 5 UEs' data blocks is less than 1 subframe's length, this base-station task can be dedicated to 1 core; if a 6th UE joins (Figure 3(b)) and the total processing time exceeds the subframe length, there must be a UE suffering from missed deadline, so the base-station task needs at least 2 cores.

Although the resource requirement of each base-station task is predictable, it is not feasible to dynamically allocate resources for every subframe. The reason is that: when a base station task is reallocated to another server, the state migration consumes a significant amount of its subframe duration, and the remaining time is little for data processing. We propose to use historical data to reallocate resources periodically. For example, a base-station task should be given more resources during its daily peak hours; if in the past few periods, a base-station task always exceeds its dedicated resources, it should be assigned more dedicated resources in the next period.

Dynamic resource pooling: In the period between 2 resource allocation adjustments, a base-station task has a fixed amount of dedicated resources, so it is possible that traffic bursts happen and miss deadlines. We propose to reserve a shared resource pool for bursty traffic. The resource pool can be an idle core in each server, or a few idle servers. When a scheduler predicts that it would miss deadline based on current dedicated resources, it would offload a part of its workload to the shared resource pool. For example, the scheduler can start a new thread to use the idle core in its server (Figure 3(c)); or it can move data to an idle server and make that server process it (Figure 3(d)). In the latter case, the data movement overhead should also be considered. Based on traffic pattern and data movement overhead, we optimally allocate the amount of dedicated resources and reserve the shared resource pool so that the total resources required are minimized. The detailed algorithm is omitted due to space limitation.

3.5 Language and Interfaces

We envision a set of programming tools and interfaces designed to help operators utilize our design [13, 2]. These tools are designed to help address various concerns which we list below:

Compiling processing blocks: For programming processing blocks, we envision a compiler which can (a) target multiple backends including processors, DSP chips, etc. and (b) can create a performance profile for the processing block. The performance profile is then used by the scheduler when deciding how to schedule pieces of the data path.

Data path Linking: Given a set of processing blocks and decision blocks we provide a linker that combines the code and estimates resources needed to run the entire data path.

Common Functions: Many of the processing blocks are shared across multiple kinds of data paths, e.g., encoding and decoding functionality, etc, which are provided by a shared library and are optimized for a variety of hardware platforms.

Domain specific languages (DSL): Finally to aid operators we plan on providing a DSL to simplify the creation of processing and decision blocks, control plane and management interfaces to controllers. Developers write code that derives from processing and decision blocks. The compiler automatically generates relevant code for control and configuration. For example, developers can write $Viterbi.insert(beforeBlockID, afterBlockID)$ to insert the Viterbi processing block without worrying about the details of how the blocks are inserted.

4. EVALUATION

We modify OpenAirInterface [1] LTE implementation to restructure the downlink processing into our data plane blocks. We measure base station traces from a national cellular operator and estimate the resource requirements in sharing and dedicating modes. We also show the feasibility of data plane configuration and dynamic resource allocation. Our testbed consists of Dell T5500 servers connected via Ethernet, each with 8 cores and 16GB memory.

Necessity of resource pooling: We collect real-world base station bearer traces including MCS, UE ID, base station ID in the granularity of 1ms. There are more than 3300 base stations and 20 million records

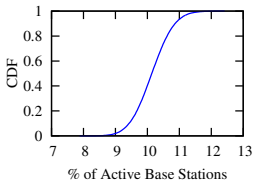

Figure 4: Active Base Station(%) CDF

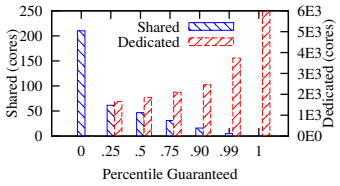

Figure 5: Resource Sharing v.s. Dedicating

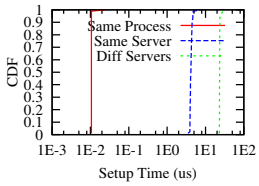

Figure 6: Scheduler Control

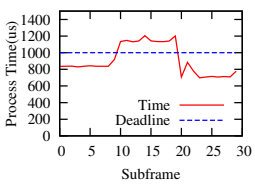

Figure 7: Resource Allocation

per minute. A base station is active in a subframe if it transmits downlink data to UEs or receives uplink data from UEs. Figure 4 is the distribution of active base stations every 1ms. We profile the resource utilization of a data path in terms of CPU cores. In the experiment, we guarantee $P\%$ of each base station's subframes can be processed by the dedicated resource, and the remaining subframes processed by the resource pool. We find the max requirement on resource pool in the whole duration as the shared resource requirement. We vary P and draw the shared and dedicated resource utilization in Figure 5. When $P = 0$, the shared resource is 210 cores, which means all the base stations processing can actually be handled by 210 cores. When $P = 1$, the dedicated resource is 5972 cores, which indicate if each base station is provided resources for its peak-time workload, they need 5972 cores in total. In the cases in between, the more resources are dedicated, the less shared resources are needed; but dedicating resources are much less efficient than sharing one. There is a potential of 30× reduction on resources compared with peak allocation. As shown in Figure 4, in our traces, only 12% (400) base stations have data to transmit for more than 99% of subframes. Therefore, $P = 0$ is optimal in terms of total resources used.

Control speed: We run a down-link processing pipeline, which includes turbo encoding, scrambling and OFDM modulation. The scheduler dynamically changes the encoding scheme (e.g. QPSK, 64QAM). We measure the speed of this state change. We put scheduler at different location and show the results in Figure 6. If the scheduler and the processing pipeline are run in different threads of the same process, the state change overhead is negligible, which is about 0.01us. If the scheduler and the processing pipe line are in the same server but different processes, they do inter-process communication which takes about 5us. If the scheduler and the processing pipeline are in different servers, network delay is introduced, which is about 23us.

Dynamic resource allocation validation: We still run the down-link processing of a base station (Figure 7). Initially, the base station is sending data to 4 UEs, in each subframe (1ms) the total processing time of all 4 UEs' data is about 800us, so the deadline is satisfied. After 1 frame (10ms), another 2 UEs join the RAN, the total processing time of 6 UE's data in 1 subframe is about 1200us, which misses the deadline. The scheduler start to use another core and 3 UEs' processing is dedicated to 1 core. The processing is in parallel, so the total completion time is reduced to about 700 us.

5. RELATED WORK

Sora [14] and BigStation [16] demonstrates the feasibility of real time communication using commodity servers to implement WiFi PHY and MAC; MAC Processor [15] presents an API to control WiFi MAC layer using the abstraction of events, actions and conditions; SoftRAN [8] addressed the requirements for logically centralized control and management. However, they do not provide any mechanisms to dynamically reconfigure existing MAC/PHY data paths.

OpenRadio [3] discussed how to build programmable base stations, but these were limited by the hardware capabilities of particular DSPs. Furthermore, OpenRadio does not provide any mechanisms to satisfy timing constraints of software data paths.

RadioVisor [9] provides the algorithm to isolate PHY radio resources. It does not provide the software architecture for actual virtualization of software data paths.

CloudRAN [7, 20] statically associates each base station to a base band unit (BBU), is not programmable and does not provide centralized control. CloudIQ [4] showed that pooling resources can reduce the overall cost of a network, however it relied on statically allocating resources (rather than dynamically reallocating resources according to requirements).

6. CONCLUSION AND FUTURE WORK

With the rapid evolution towards an untethered and connected mobile world, radio access networks can no longer remain monolithic and inflexible. PRAN, a programmable radio access network architecture, can meet the changing requirements of RANs. We plan to fully implement our design and deploy in our PRAN testbed with RRHs and servers with DSP accelerators.

7. REFERENCES

[1] Openairinterface system. In *www.openairinterface.org*.

[2] E. Axelsson, K. Claessen, G. Devai, Z. Horvath, K. Keijzer, B. Lyckegard, A. Persson, M. Sheeran, J. Svenningsson, and A. Vajda. Feldspar: A domain specific language for digital signal processing algorithms. In *IEEE/ACM MEMOCODE*, 2010.

[3] M. Bansal, J. Mehlman, S. Katti, and P. Levis. OpenRadio: A programmable wireless dataplane. In *ACM HotSDN*, 2012.

[4] S. Bhaumik, S. P. Chandrabose, M. K. Jataprolu, G. Kumar, A. Muralidhar, P. Polakos, V. Srinivasan, and T. Woo. CloudIQ: a framework for processing base stations in a data center. ACM Mobicom, 2012.

[5] G. Bianchi, P. Gallo, D. Garlisi, F. Giuliano, F. Gringoli, and I. Tinnirello. Maclets: active MAC protocols over hard-coded devices. In *ACM CoNEXT*, 2012.

[6] P. Bosshart, D. Daly, M. Izzard, N. McKeown, J. Rexford, D. Talayco, A. Vahdat, G. Varghese, and D. Walker. Programming protocol-independent packet processors. *SIGCOMM CCR*, 2014.

[7] ChinaMobile. C-RAN: The road towards green RAN. white paper, 2011.

[8] A. Gudipati, D. Perry, L. E. Li, and S. Katti. SoftRAN: Software defined radio access network. In *ACM HotSDN*, 2013.

[9] S. Katti and L. E. Li. Radiovisor: A slicing plane for radio access networks. In *ONS*, 2014.

[10] J. Kerttula, N. Malm, K. Ruttik, R. Jäntti, and O. Tirkkonen. Implementing TD-LTE as software defined radio in general purpose processor. In *ACM SRIF*, 2014.

[11] J. Neel, P. Robert, and J. Reed. A formal methodology for estimating the feasible processor solution space for a software radio. In *SDR*, 2005.

[12] M. Z. Shafiq, L. Ji, A. X. Liu, J. Pang, and J. Wang. A first look at cellular machine-to-machine traffic: Large scale measurement and characterization. In *SIGMETRICS*, 2012.

[13] G. Stewart, M. Gowda, G. Mainland, B. Radunovic, and D. Vytiniotis. Ziria: Wireless programming for hardware dummies. (MSR-TR-2013-135), 2013.

[14] K. Tan, J. Zhang, J. Fang, H. Liu, Y. Ye, S. Wang, Y. Zhang, H. Wu, W. Wang, and G. M. Voelker. Sora: High performance software radio using general purpose multi-core processors. 2009.

[15] I. Tinnirello, G. Bianchi, P. Gallo, D. Garlisi, F. Giuliano, and F. Gringoli. Wireless mac processors: Programming mac protocols on commodity hardware. In *IEEE INFOCOM*, 2012.

[16] Q. Yang, X. Li, H. Yao, J. Fang, K. Tan, W. Hu, J. Zhang, and Y. Zhang. Bigstation: Enabling scalable real-time signal processingin large mu-mimo systems. In *ACM SIGCOMM*, 2013.

[17] X. Zhang, J. Ansari, and P. Mähönen. Demo: runtime MAC reconfiguration using a meta-compiler assisted toolchain. In *ACM SIGCOMM Demo*, 2012.

[18] X. Zhang, J. Ansari, L. M. A. Martinez, N. A. Linio, and P. Mahonen. Enabling rapid prototyping of reconfigurable mac protocols for wireless sensor networks. In *IEEE WCNC*, 2013.

[19] X. Zhang, J. Ansari, G. Yang, and P. Mahonen. Trump: Supporting efficient realization of protocols for cognitive radio networks. In *IEEE DySPAN*, 2011.

[20] Z. Zhu, P. Gupta, Q. Wang, S. Kalyanaraman, Y. Lin, H. Franke, and S. Sarangi. Virtual base station pool: towards a wireless network cloud for radio access networks. In *ACM CF*, 2011.