Virtual Base Station Pool: Towards A Wireless Network Cloud for Radio Access Networks

Zhenbo Zhu, Qing Wang, Yonghua Lin –IBM Research China, Beijing, China
Parul Gupta, Smruti Sarangi, Shivkumar Kalyanaraman – IBM Research India, Bangalore, India
Hubertus Franke – IBM Research US, Yorktown Heights, US

***Abstract*—The mobile Internet has seen tremendous progress due to the standardization efforts around WiMAX, LTE and beyond. There are also early trends towards adoption of software radio and a growing presence of general purpose platforms in wireless networking. Such platforms are programmer-friendly and with recent advances on multi-core and hybrid architectures, allow signal processing, network processor class packet processing, wire-speed computation and server-class virtualization capabilities for software radio realizations of 3G and 4G wireless stacks. Software radio over IT platforms will enable the virtualization of base stations and consolidation of virtual base stations into central pools (a local "cloud site") with fiber connectivity to towers, which we call a Wireless Network Cloud (WNC). A Virtual base station (BS) pool supporting multiple BS software instances over a general OS and IT platform is an important step towards the realization of the larger WNC concept. This paper introduces the first TDD WiMAX SDR BS implemented on a commodity server, in conjunction with a novel design of a remote radio head (RRH). We also present the first working prototype of a virtual BS (VBS) pool, exploring the systems challenges in supporting a VBS pool on multi-core IT platforms. The results from our VBS pool prototype for WiMAX verify that these solutions can meet system requirements including synchronization, latency and jitter.**

I. Introduction

Wireless network infrastructures (e.g. base stations, gateways) and telecom IT infrastructures, like data centers, billing servers, service delivery platform middlewares, have traditionally not intersected. However, in other networking contexts, IT platforms have recently been considered to replace custom platforms that are based on FPGAs, DSPs etc. For instance, soft-switches in telephony and control-plane processing in data networking (eg: routing and signaling) have been consolidated in blade server computational platforms [1][2][3], while data-plane packet processing is performed on network processors. Similar trends are emerging in 4G wireless where the control plane elements like the Mobility Management Entity (MME), Policy and Charging Rules Function (PCRF) etc will be supported on general purpose servers, traditionally used for IT applications.

Software Radio offers the opportunity for the MAC and PHY layers in wireless network stacks to be consolidated on multi-core servers that offer rich programming environments and a large community of developers. Specifically, large-volume, low-unit cost hybrid IT systems will offer multi-cores, massive multi-threading, sufficient I/O throughput and accelerators for signal processing to meet the requirements of such mixed-mode workloads. We expect these key capabilities and accelerators to become integrated a single System-on-Chip (SoC) processor. Equally important, with IT systems increasingly providing capabilities for virtualization, we can realize concepts such as virtual base stations and allow the support of multiple virtual operators on common infrastructures with robust isolation support. However, realizing software radio on these platforms involves shifts in programming styles: using massive parallelism while ensuring real-time performance, tackling issues of OS and non-OS jitter, and leveraging accelerators.

Though backhaul from base stations to the core is expected to remain a constraint in several geographies, the long-term solution to backhaul capacity demand is to lay fiber or provide metro-ethernet capacity to towers. The combination of fiber-to-towers and virtualization via software radio allows a more radical deployment approach: physically unbundle the base stations into virtual base stations (VBS) implemented on IT platforms, pooling them at local "cloud" sites and connect them to shared Remote Radio Heads (RRH) attached to the towers.

This new network architecture for next-generation wireless access networks, called the *wireless network cloud (WNC)* was proposed in [4], and is also illustrated in Figure 1. As described in [4], WNC supports (a) multiple wireless standards over a low cost platform, (b) mobile virtual network operator (MVNO) network-sharing models, and (c) integration of both the service plane along with the wireless data- and control-planes for workload balancing in different districts. Ultimately this increases system resource utilization and decreases capital investment requirements. The leading operator in China, China Mobile, also has articulated a similar C-RAN vision from an operator perspective [5], noting that a "cloud" infrastructure to support the RAN could reduce costs for operators.

We envision a three-step migration from the traditional view of software radio to the concept of a wireless network cloud:

Step 1: Single BS implemented on an IT platform.

Step 2: A Virtual base station pool supporting multiple software BS instances over an IT platform without additional hypervisor and VM support.

Step 3: Wireless network cloud built with a large scale VBS pool (with hypervisor and VM support), and with capabilities to deliver elasticity of service provisioning and pricing to users and operators.

The first two steps are the focus of this paper where we

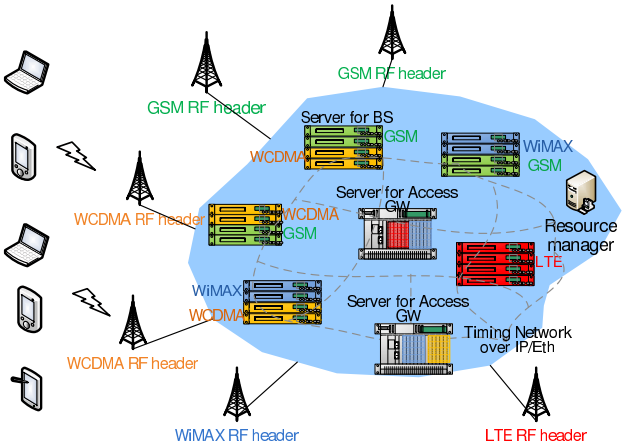


Fig. 1. Wireless Network Cloud

enable single and multiple SDR VBS instances over an IT platform to meet strict real-time and latency requirements necessary for the wireless system. We present the first experimental proof point of the VBS pool on an IT platform. This paper's contributions are in the following areas:

- A novel Time Division Duplex (TDD) remote radio head with an Ethernet interface (ERRH) that enables the SDR BS concept over an IT platform in a more convenient way.
- The first realization of virtual base stations that can be mapped flexibly onto pools of underlying IT platforms while meeting real-time constraints.
- Analysis of the performance constraints of PHY and MAC layers respectively and optimization of VBS performance to meet the stringent real-time requirements of jitter, latency etc.
- The first demonstration of an ERRH based WiMAX BS prototype with MAC and PHY stacks running on a commodity server, running test workloads comprised of simultaneous web-browsing, video and VoIP sessions, and support for timing on both the uplink (UL) and downlink (DL), as required in a TDD WiMAX BS.

Beyond the concept of VBS pools, the wireless network cloud allows elastic capacity provisioning, faster mobility management, novel interference management and cooperative MIMO techniques to be used. Some of these techniques are practically infeasible with the standard basestations-on-towers architecture (and the proprietary DSP/FPGA designs) due to the latencies and overheads involved in coordination[6]. In future work, we will support these features that become possible due to the proximity of virtual base stations in the wireless network cloud architecture.

The rest of the paper is organized as follows. Section II presents the proposed structure of VBS pool and highlights the system design challenges. Section III presents our VBS pool prototype design along with the ERRH design. Section IV dives deeper into the individual components and describes the chosen design for each. Section V presents performance evaluation of the VBS pool prototype at the component level and the End to End system level and Section VI describes a DEMO application system over our VBS pool prototype running a mixed workload. Section VII presents related work and Section VIII concludes the paper.

II. Virtual Base Station Pool Structure and Challenges

This section describes the virtual base station pool structure and important system challenges in implementing this concept on IT platforms. Subsequent sections describe solutions to these issues.

A. Virtual Base Station Pool Structure

Figure 2 shows the structure of a virtual base station pool. It consists of three parts: remote radio head (RRH), the physical processing nodes and networks, and virtual base station (VBS) instances.

In our proposed architecture, the novel ERRH is used. RRHs deployed in the market today transmit the digitized waveforms to the base station through an interface such as Common Public Radio Interface (CPRI) [7] or Open Base Station Architecture Initiative (OBSAI) [8]. These are essentially customized layer 2 protocols running over a time-division multiplexing (TDM) link. A general-purpose IT platform will not have TDM interfaces and hence will require additional hardware design to interconnect with current RRHs. Further, these protocol designs only support point-to-point, point-to-multipoint and chain topologies[7], whereas our new ERRH can leverage the rich switched metro Ethernet topologies available on the market. The ERRH devices deployed on towers are connected to the virtual base station pool via 1 Gigabit Ethernet (GbE) or 10 GbE links over optical fibers. More generally, we propose to have a group of RRHs connected to a group of VBSes (in a pool) via a switched metro Ethernet network for higher availability, and for dynamic workload allocation. Compared with frequency division duplex (FDD) mode, in order to work with TDD-mode selected in this paper, RRHs will have to overcome higher challenges on timing control and synchronization. The high level architecture of VBS pool could also support the traditional TDM RRHs, but it needs extra hardware component (e.g. PCIe adaptor card) to convert the CPRI/OBSAI interface to PCIe data, which is not the focus of this paper.

The VBS pool operates over a set of IT servers with multi-core general purpose processors (GPP). A hybrid processing pool can be made available to match the workload of base station stack components. For example, Intel x86 and IBM POWER processors are well matched to the computation - intensive portions such as the physical (PHY) layer. Processors like IBM PowerEN and Sun Niagara, which are heavily multi-threaded (and PowerEN has several accelerators), are well suited for network processing, e.g. MAC and transport layers.

The internal network between processing nodes can be a 1 GbE, 10GbE or Infiniband, optimized for low-latency and

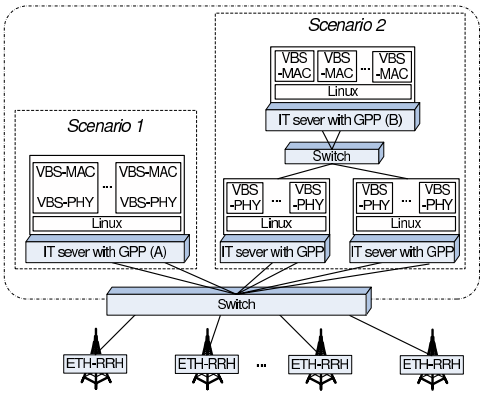


Fig. 2. Structure of VBS pool

jitter. This network will be used to carry all the data, control and timing signals.

The functions of a base station (PHY, MAC, and transport layer, etc.) can be implemented in software using one or more software instances, referred to as a virtual base station instance. In a VBS pool, we envision two deployment scenarios for virtual base stations over physical resources(Figure 2):

Scenario 1: The physical GPP node can be flexibly shared by multiple virtual base station instances. Any VBS instance will be deployed on the same GPP node.

Scenario 2: The processing resources from two or more GPP nodes can be abstracted and combined together to support a single base station instance. For example, the PHY layer of a VBS (VBS-PHY) and the MAC layer of a VBS (VBS-MAC) can be run on two different GPP nodes. In a hybrid resource pool, such mapping will provide better processing efficiency.

B. Challenges

This section describes the challenges in realizing the VBS pool structure shown in Figure 2. The main challenges to address are VBS - ERRH synchronization and realtime control.

Synchronization among base stations is a very strict requirement especially in wireless TDD systems [9]. In wireless TDD system, data transmission and reception are time multiplexed on the air. All base stations have to start their data transmission and reception phases exactly at the same time, and correspondingly, the mobile stations receive and transmit data at those same timespots. A lack of such tight synchronization results in radio interference and the communication quality will significantly degrade even to the point where communication channels fail. Usually, an accurate synchronized clock (e.g. GPS signal) is used at each RRH, which gives each base station a global reference timing. On the base station, all the data transmitted through the downlink (DL) and uplink (UL) needs to be synchronized with the RRH clock, the VBS - ERRH synchronization in our case.

In 4G wireless standards like Wimax and LTE, to ensure strict real time guarantees to the application layer, the standards usually define a very strict round-trip latency in wireless Layer 1 and Layer 2. Thus, the wireless frame can not be processed much earlier than the transmission timespot. To avoid timing synchronization failure, it is also typically required that at least 99.99% of frames be processed within the specified deadline. Hence frames should arrive at the RRH right on the timespot they have to be transmitted to the air (at best a little bit earlier) .

In the VBS pool structure, there are two kinds of latency. The first kind is the computational latency for a single software instance, e.g. the UL or DL of the PHY layer. The PHY layer is computationally intensive yet requiring high I/O throughput. We need to ensure that the processing completes within the frame deadline despite this computational load.

The second kind of latency is the communication latency between VBS instances. In scenario 2 of Figure 2, the frame of one VBS instance will move from the VBS-PHY on one processing node to the VBS-MAC mapped onto another processing node. When the VBS instance is thus split across multiple nodes, the overhead associated with data queueing and movement can be close to a millisecond and can have impact on meeting realtime requirements.

Moreover, in our VBS structure with ERRHs, we use Ethernet to transmit data between ERRHs and general-purpose servers supporting the VBS instances. A significant challenge is to overcome jitter inside the operating system and between the VBS instance and ERRH introduced by Ethernet.

Unlike a custom design using DSPs or FPGAs, the VBS workload on a general purpose processor (GPP) is multi-threaded and runs over a general-purpose operating system, which may not have real-time features. Consequently, interrupt handling and thread-level scheduling at the OS-level causes jitter, especially when the CPU utilization is high. Controlling this OS jitter while maintaining real-time performance at high utilization levels is a key technical challenge.

The rest of the paper addresses these challenges for the VBS pool prototype together with achieved performance results.

III. Virtual Base Station Pool Prototype Design

In this section, we describe the design of Virtual Base Station pool prototype.

The VBS pool prototype is designed to be modular and open. It consists of RRH interface layer, wireless stack layer, data communication layer and some other supporting libraries. By selecting different interfaces and different modules, it can support different VBS pool scenarios, multiple wireless communication protocols, multiple physical GPP nodes and multiple radio-headers with different connection interfaces. The software stack of the VBS pool can be implemented on a single IT server (eg: x86 or POWER), or a cluster thereof interconnected by Gigabit Ethernet(GbE).

To connect the Virtual Base Station pool prototype to general IT interfaces, we designed and implemented a new Ethernet-based time-division duplex (TDD) Remote Radio Header [1] as shown in Figure 3. The ERRH consists of a radio front-end and a new ERRH adapter board.

[1]The details of the design and implementation of this Ethernet-Based Remote Radio Header will be presented in a separated paper

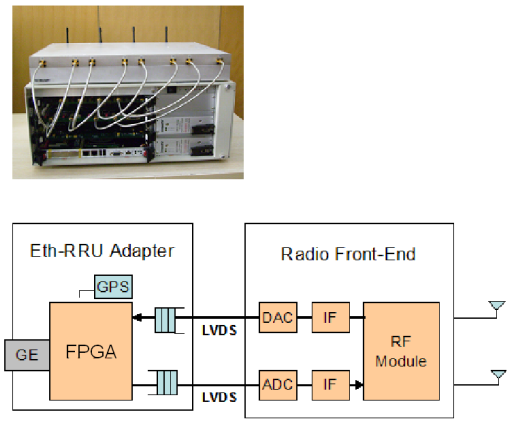


Fig. 3. Ethernet based Remote Radio Header prototype

The radio front-end contains components such as RF, IF and base band A/D converter (ADC, 14-bit, 92.16MSPS). The radio front-end transmits and receives the radio signals through antennas in the TDD mode. It supports up to 10MHz signal bandwidth.

The ERRH adapter board uses high performance FPGAs and provides:

Interface translation: Low Voltage Differential Signal (LVDS) interface is used to connect ERRH adapter and radio front-end. Gigabit Ethernet interfaces are provided for the base band data transmission to the host. Using this ERRH adapter, the RRH appear as a Ethernet Node and flexibly connect to the VBS Pool via a switched Ethernet network.

TDD timing control: The ERRH adapter board fully controls the Tx/Rx data channels on the adapter board and the radio front-end, and ensures the ERRH to operate accurately in TDD mode.

Timing synchronization with VBS through Ethernet: The ERRH adapter board provides a mechanism to synchronize the virtual BS in TDD mode. It also tolerates the jitter introduced by Ethernet and adjusts the synchronization in runtime.

Hardware configuration: The ERRH adapter board provides the control interface of the ERRH. The host application can configure the ERRH via Ethernet.

The physical interface between the ERRH and the VBS in our prototype system involves a GbE link to setup a bi-directional data logical link to transmit the I/Q data between VBS and RRH, and a bi-directional control logical link to configure the VBS and control the RRH. Both the data and control signaling are transmitted in a packet format over Ethernet. Our protocol includes identifiers and timing information along with the data and commands.

IV. Virtual Base Station Pool Prototype Implementation

This section describes the Virtual Base Station Pool Prototype implementation. Figure 4 shows our VBS pool prototype implementation in the first stage. Two IBM X3650 servers with 2 way Intel X5355 processors are used as the hardware platform of the VBS pool prototype that supports multiple

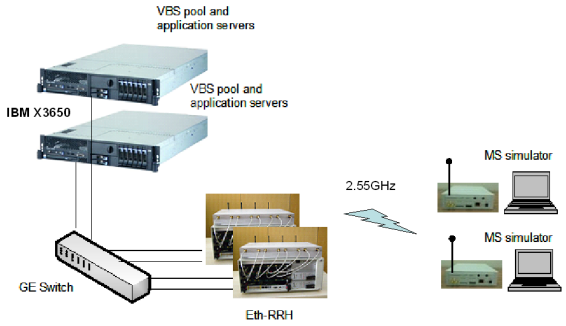


Fig. 4. Hardware configuration

Parameters		Values
System Parameters		
System Frequency Band		2.55GHz
System Channel Bandwidth (BW in MHz)		10
Sampling Frequency (Fs in MHz)		11.2 (n=28/25)
OFDMA Symbol Parameters		
FFT Size (NFFT)		1024
Ratio of CP		1/8
OFDMA Symbol Duration (Ts)		102.9 μs
DL PUSC	One Symbol Slot	1 subchannel, 2 OFDMA symbols
	Preamble	Yes
	Subchannels	30
UL_PUSC	One Symbol Slot	1 subchannel, 3 OFDMA symbols
	Ranging	Initial and Periodical
	User Subchannels	23
	Ranging Subchannel	12
Link Parameters		
Link Direction		Downlink & Uplink
Duplex Mode		TDD
DL:UL Ratio		35:12
Frame duration (T_frame)		5 ms
Number of Frames (per second)		200
Number of OFDMA Symbols in one frame		48
Channel Coding		
FEC		CC; 1/2
Modulation		16 QAM, QPSK
Sub-Channels		Full
MIMO Parameters		
Tx number		2, STC Matrix A and MatrixB
Rx number		2, STC Matrix A and Matrix B

Fig. 5. System profile

VBSes (including different scenarios of VBS realizations).The Linux OS is selected for the VBS pool. GbE links are used to connect the server with the RRHs and each other. Laptops are used as the mobile station which runs a full function of mobile station software stack.

The VBS pool can in principle support different wireless communication protocols. We have implemented the IEEE 802.16e (Mobile Wimax) standard in the VBS for our current prototype. The system profile complies with 802.16e standard [12] and WiMAX forum suggestions[13]. The system profile details are provided in Figure 5.

A. VBS Pool Framework

The VBS pool framework is designed to support multiple VBS instances on a single server node. And for each VBS instance, the framework is also designed to initialize, connect, and invoke different software stack layers to operate as an integrated VBS. It also controls the synchronization and TDD timing of each VBS.

VBSes operation

Because one VBS will connect with a dedicated RRH, multiple VBSes on a single server node can be treated as separated processing channels linking to their dedicated data interfaces

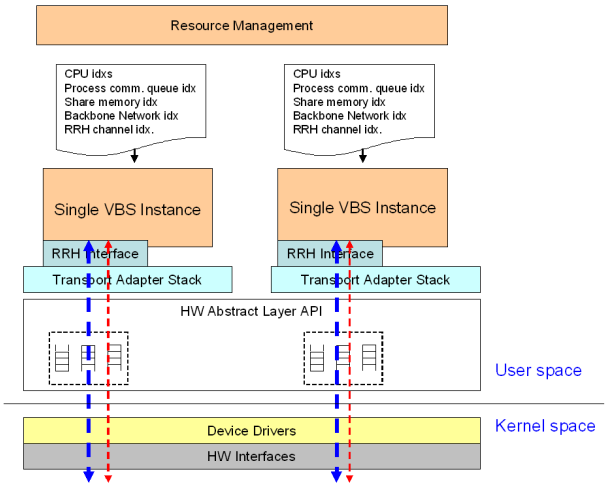


Fig. 6. Multiple VBS on single physical node

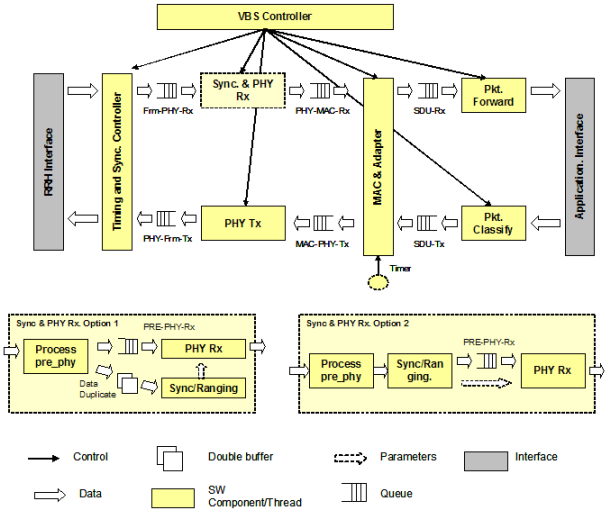


Fig. 7. An example of WiMAX VBS framework

on that RRH. Thus, to support multiple VBS instances, as shown in Figure 6, the VBS Pool framework provides separate RRH interfaces to each VBS instance via a Hardware Abstract Layer. Each VBS instance will run as a process within the Linux system independently. To avoid resource conflicts on the same server node, the resource management component of the VBS Pool framework within each server node provides the physical resource allocation to each VBS in the pool.

The VBS pool framework is also introduced to handle a single VBS instance. To maximize flexibility, the framework only links each software (SW) component and controls the initialization and startup phase of each VBS. The framework provides the data interface - the blocking / non-blocking FIFO queues that can be used between software threads or processes on the same server node. It also provides the network data link via Ethernet to support hybrid processing among different server nodes (scenario 2 of Figure 2). Each SW component has its own threading model; it is self-managed and driven by the data traffic it actually sees. With the VBS pool framework, the SW components can be integrated into one or more VBS instances on a single node (scenario 1) or spread out across nodes (scenario 2). The VBS pool framework does not maintain an accurate timing clock itself. The accurate timing clock information is embedded into the data stream from the ERRH interface. The framework helps maintain the VBS synchronization with the timing clock from the ERRH.

Figure 7 shows an example of the VBS framework for a single VBS instance. The VBS controller is used to initialize every components. The Timing and Synchronization Controller is a component to read/write data from/to the ERRH interface and is also used to keep the TDD timing in the VBS and synchronization with ERRH. The *Process_pre_phy* component prepares the data for the ranging and PHY block and it can be implemented to handle the ranging and PHY components in parallel or in sequence. The adapter layer provides the library for the MAC layer to convert the data format to/from the MAC layer.

VBS - ERRH Synchronization There are two different types of synchronization needed in our VBS pool, one is the synchronization between the Base Station and Mobile Station, the other is the synchronization between the VBS and the ERRH. The first kind of synchronization is obtained by physical layer techniques like ranging in UL and preamble synchronization in DL. We will discuss the second kind of synchronization challenges and our proposed mechanism in this part.

Since the ERRH owns an accurate reference timing from GPS, it provides the TDD control in the RRH. The VBS needs to synchronize with the ERRH and has to run under the same TDD referene timing.

However, since VBS is a software implementation on Linux, there can be unpredictable jitter in VBS processing and data transmission due to OS or Ethernet network interference. Compared to FPGA/DSP based BS system with accurate reference clock and hard real-time processing guarantee, the synchronization is a much bigger challenge for wireless communication systems in our software solution, especially for the TDD mode used in our system.

We propose a new mechanism in Figure 8 that adjusts the timing synchronization dynamically.

From the view of the VBS, in the UL Rx side, no accurate timing synchronization is required. Since the ERRH controls the TDD timing, it will transmit the UL data to the VBS at the right time. The VBS UL data path is set to the Ethernet data-driven mode. Once Ethernet data packets are received by the VBS, the VBS will start the UL processing with no delay.

In the DL Tx side, accurate timing synchronization is required between the VBS and ERRH. According to the TDD timing model, the first data block from the VBS is expected to arrive at the RRH just at the start time of the DL timing slot. In case of arrival delays, there will be data loss for the DL frame transmission. In contrast, if it arrives too early, there will be delays further up in the DL stack.

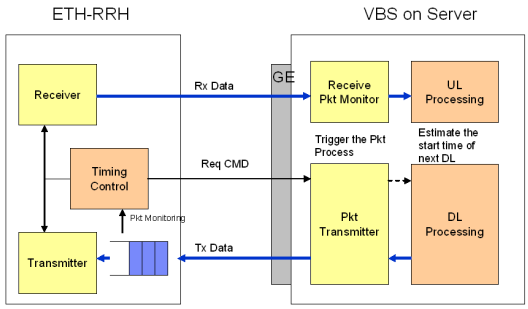


Fig. 8. Proposed mechanism for VBS and ERRH synchronization

In our system, since the ERRH has the reference timing of the system, it is used to trigger the VBS DL processing before every DL timing slot by an Ethernet packet. The timing trigger is adjusted in the runtime. At the beginning, an initial value will be set in the ERRH to indicate at what time the Ethernet packet trigger should be transmitted. The VBS DL waits for the Ethernet packet trigger, and once it is received, it starts the DL processing and transmits the DL frame to the ERRH. The RRH monitors the arrival of the DL frame, measures the time difference between the arriving time and the TDD DL timing slot, and adjusts the next timing trigger. The adjustment will consider the different penalties for the DL transmission being too late or too early. The default timing values for the ERRH and VBS are set based upon the Ethernet topology between the VBS and its peer Ethernet RRH.

Real-time Control

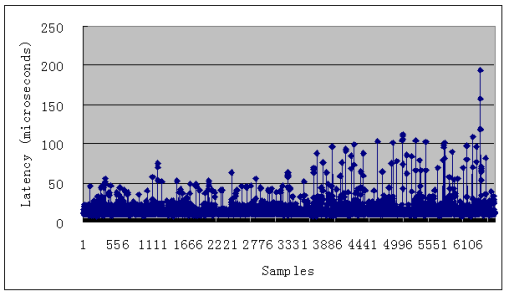
There are some particular designs and optimizations in the VBS pool framework to control the overall jitter and latency, and improve the precision of the synchronization between the ERRH and VBS.

The VBS pool framework provides the interface between SW components. It defines the operations of the SW components to the data interface. Hence, the SW component works in a data-driven mode and the latency caused by data transmission between the data interfaces is minimized.

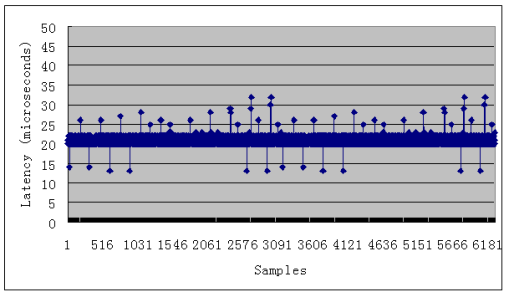
When the VBS framework initializes and starts up each SW component, it specifies the dedicated hardware resources to the SW components considering workload and jitter sensitivity of the component. Because most components tend to have a steady workload given the parameter configuration, the hardware resource sharing of the SW components and the system jitter control can be more balanced.

In Linux, hardware interrupts and softIRQs interrupt running applications, which introduces jitter and latency for data processing and data transmission. To mitigate these real-time OS challenges, we use the following methods in Linux:

Pre-emptable real-time kernel patch: generally available, in this kernel patch, all softIRQs are handled by kernel threads or system processes. They are scheduled by the Linux scheduler and can be preempted by other kernel threads or system processes.



(a) Latency trace without optimization



(b) Latency trace after Linux OS optimization

Fig. 9. Jitter tracing for UL frame receiving through Ethernet

Optimize the Linux scheduler and application priority: Based on the experiment in [14], we adjusted the strategy of the scheduler and reset the priority of each thread within the VBS pool system. It provides the threads in user space with the best real-time performance.

Core isolation: All device interrupts and visible daemon processes are moved to dedicated physical CPU cores. The key threads of VBS pool applications are bound to other physical CPU cores.

Linux uncertainty reduction: We removed all the unnecessary daemons and service in the Linux System to reduce the uncertain impact from other jobs in Linux.

Figure 9 shows the latency traces we measured in our system before and after the real-time optimization. The latency between the HW interrupt rise (packet arrives) and the packet received by the Receive Packet Monitor in VBS (shown in Figure 8) is logged. We define 20% above the average latency is the threshold of acceptable jitter. Before the optimization, on average, every 7.45 packets will have one packet exceed the acceptable threshold. And the maximum latency is about 1278.6% (12x) over the average latency. After the optimization, on average, every 172 packets will have one packet exceed the acceptable threshold and the maximum latency can be reduced to 54.6% over the average latency. Note that 1/172 is not related to 99.99% we discussed about the VBS challenges. 99.99% is the required successful rate of frame synchronization. Here we only discuss the efficiency of jitter control on the assumption that 20% above the average latency is the threshold of acceptable jitter.

TABLE I
ALGORITHM FOR KEY PHY COMPONENTS

Module	Algorithm
Channel coding	CC, tail biting and zero padding
Modulation	16QAM, 64QAM, QPSK
Ranging	Auto correlation
Channel estimation	Linear interpolation
STC decoder (MatrixA)	Zero Forcing
STC decoder (MatrixB)	Zero Forcing and MMSE
Demodulation	Hard, Soft, CSI (channel state information) mode
Channel decoding	Viterbi, soft decision (4 bit)
Synchronization	Van de Beek algorithm [15]

B. PHY

A standard compatible WiMAX PHY transmitter is implemented according to the system parameters defined in Figure 5. Most of the modules of WiMAX PHY transmitter, such as the randomizer, subcarrier allocation, interleaver, modulation etc, have been defined in the IEEE 802.16e standard whereas the design of the receiver algorithms is open. We selected algorithms considering the tradeoff between system performance and computation complexity. The selected algorithms are listed in Table I.

For the PHY layer, the processing latency is a vital factor for system performance. When the algorithms of each module are selected, we should optimize each module for the computation workload reduction to satisfy the processing time and throughput requirements. The techniques of Single Instruction Multiple Data (SIMD), look up table, data and structure alignment and loop unrolling are used to optimize the PHY layer processing.

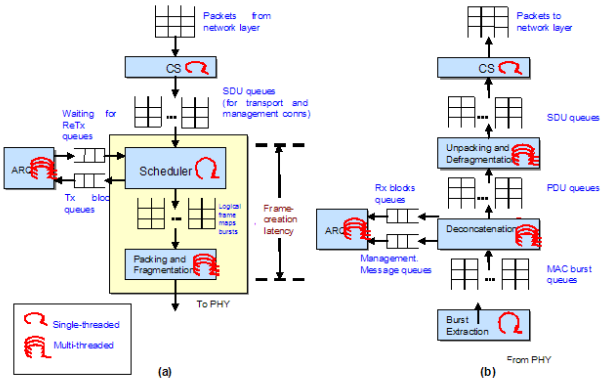


Fig. 10. Multithreaded architecture of Wimax MAC (a)Downlink (b)Uplink

C. MAC

In our prototype, we implemented the Point to Multipoint (PMP) WiMAX MAC layer for Mobile WiMAX. We present a multi-threaded software implementation of WiMAX MAC layer. In order to realize an efficient MAC layer, our design is multithreaded to parallelize the workload, efficient concurrent data structures to improve thread scalability, minimum memory copy policy within the MAC layer to reduce memory overhead and thread pooling to reduce thread creation and destruction overhead. Figure 10 shows the multithreaded design of UL and DL of the Wimax MAC layer.

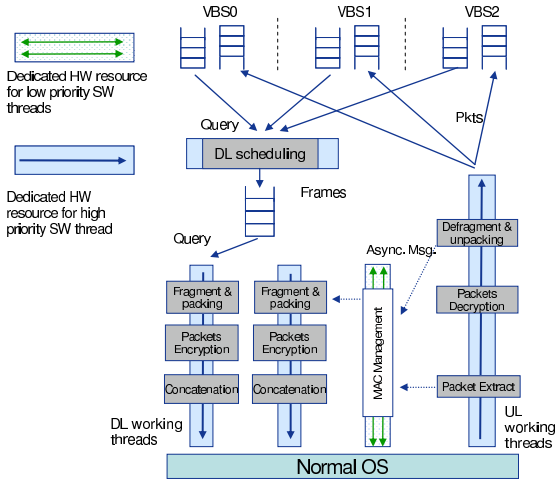


Fig. 11. New model for handling multiple MAC layer

To improve the real-time performance of the MAC layer, we optimized the MAC processing flow and reduced the average processing latency. Moreover, we used multi-threading to accelerate the MAC processing in parallel, and work stealing [17] is used in the multi-thread implementation.

However, for the VBS Pool scenario 2 (Figure 2), usually multiple MAC instances need to be supported on one processing node. Considering the multi-threaded software implementation of our MAC, if a dedicated system process is used for each logical MAC instance, massive software threads will be host on the node. In our study, nearly all the tasks have a relatively light computation workload, but some of them are very sensitive to jitter and latency. This places pressure on the OS scheduler. Therefore, we also proposed a new model that efficiently supports multiple MAC instances on single processing node.

Shown in Figure 11, one MAC process can handle multiple logical MAC instances in the new model, where common working threads are used. Each common working thread can handle the same tasks belonging to different MAC instances. The processing is driven by the input data. We divided the MAC processing tasks into two groups based on their real-time requirements. The set of tasks that needs to observe strict deadlines, like DL scheduling, packing, fragmentation and encryption, will be allocated more working threads, higher priority and dedicated hardware resource. The set of tasks with more relaxed deadlines of a tens of milliseconds or more, like ARQ and MAC management, will have limited working threads and hardware resources.

V. PERFORMANCE EVALUATION

The performance of our VBS pool prototype is evaluated in two dimensions: *component performance* and *end-to-end system performance*. Recall that we use the notion of a VBS framework that links software components together and

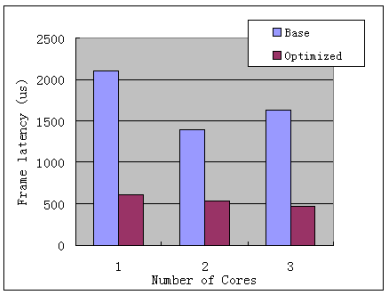


Fig. 12. Average frame processing latency using 64QAM modulation.

controls the timing and start up of the VBSs. Each software component has its own implementation model. Therefore, we evaluate the performance of each component first and then measure the overall system performance. The metrics of interest are maximum throughput supported, processing latency and the scalability of multiple VBS instances as the number of general purpose processor (GPP) cores is increased. The server used for our experiments is an IBM x3650 server with two Intel Clovertown processors @ 2.66GHz (8 physical cores in total).

A. MAC and PHY Component Evaluation

In this section, the performance of PHY and MAC are evaluated separately.

Two experiments are designed for MAC stack performance evaluation. Single MAC instance experiments measure the throughput and latency for one MAC instance in a VBS, and the multiple MAC instances experiment measures how the number of MAC instances supported in a VBS pool scale with increase in the number of cores.

Single MAC Instance: To better illustrate the effectiveness of optimization methods described in previous sections, we measure the single MAC performance for 20MHz bandwidth and 64QAM modulation. This configuration supports a payload throughput of over 30 Mbps and hence has a bigger computational load. Figure 12 shows the average frame processing latency for the **base** configuration without any optimizations and the **optimized** configuration with all the software stack and Linux OS optimizations enabled described in earlier sections. We observe that frame latency in the **base** configuration does not decrease with increase in the number of cores as one would expect, pointing to high thread synchronization overhead and OS jitter. Though, the optimizations successfully reduced the average frame latency to about a third of the base values.

Multiple MAC Instances: In this experiment, we host multiple MAC instances, including UL and DL processing, on a single server. The system profile we defined in Figure5 is configured as the experiment parameters. The system requirement is that at least 99.99% of frames must finish their processing within the 5 millisecond frame duration deadline. The results reveal that each CPU core can support up to 4 MAC instances while meeting this requirement, and that it scales ideally with six CPU cores supporting a total of 24 MAC instances.

The PHY layer performance is also evaluated under the profile defined in Figure 5. In this experiment, the goal is to find the maximum PHY layer throughput supported on a multi-core GPP platform. We enabled from one up to eight UL(DL) instances of PHY layer, each of which was bound to a dedicated CPU core.

The results for both UL and DL reveal that one core of an Intel Clovertown processor can support PHY layer UL processing at 5.06Mbps, or DL processing at 51.27Mbps. The scale-up ratio for PHY layer processing is close to linear at 7.98 for 8 cores.

B. Virtual BS Pool System Performance Evaluation

The end-to-end performance of VBS instances in the VBS pools is evaluated for the two topologies (scenarios) previously described in Figure 2.

The first scenario evaluates the case of one physical server node being shared by several base station instances. The MAC layer and PHY layer are processed on the same server in sequence. The second scenario evaluates the case of heterogeneous processing resources combined together to support multiple base station instances.

Considering the different workload characteristics of MAC and PHY layers, the hybrid architecture (scenario 2) can improve resource utilization by leveraging different architectures best suited for different workloads. For example, some GPP nodes can focus on the processing of multiple MAC instances and other GPP nodes can focus on the processing of multiple PHY instances. The important issue in this scenario is the communication latency between two server nodes. We use two IBM X3650 servers to simulate two different server nodes in hybrid architecture. In this topology, data from MAC to PHY (and vice versa) has to be communicated from one set of nodes to another, adding latency to the system. Our focus of testing this topology is to identify the latency overhead of such additional data traffic, and whether it is within acceptable limits.

The experiment results are shown in Figure 13. For both scenarios, the hardware resources are allocated fairly. One VBS will use two physical CPU cores of the multi-core system. We tested the cases of one and two VBS instances on the multi-core system. They have very similar average latency.

We also tested the WiMAX stack configured with 2ms frame duration, where the total number of OFDMA symbols is 19, and the DL:UL ratio is selected as 13:6. The VBS with 2ms frame duration configuration has a much shorter average latency than the VBS with 5ms frame duration configuration. This is because both the PHY stack and MAC stack have fewer packets or symbols to be processed in each frame, and the frame to be buffered between the layers and interfaces also have a shorter time duration. Thus, the VBS processing latency for each frame has an obvious relationship with the frame duration.

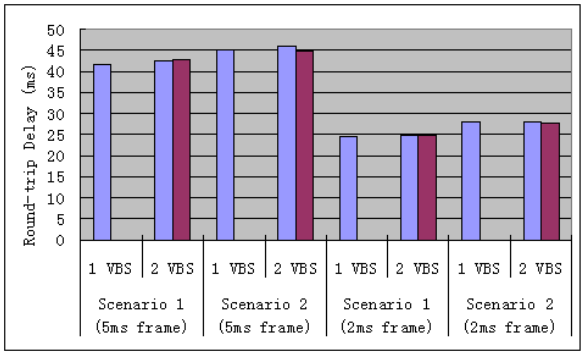


Fig. 13. VBS pool performance evaluation

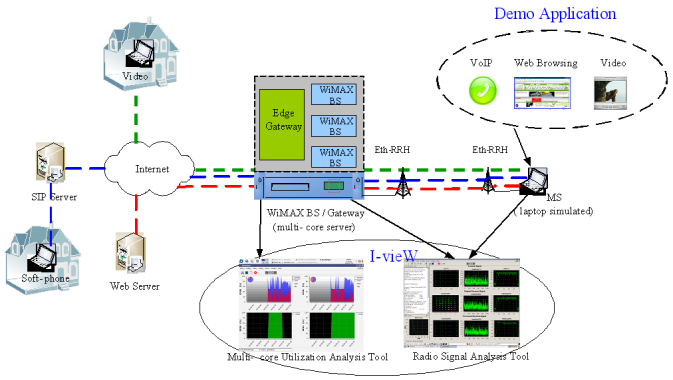


Fig. 14. Application DEMO system

Comparing the first and second scenario results, about 3ms delay is added to the round-trip latency by a physical switch between two platforms (i.e. between scenario 1 and scenario 2 in the figure). Note that the increase in end-to-end round trip latency of 3 ms to a base end-to-end latency of up to 25-40 ms is acceptable. Thus, the end-to-end performance results show that the VBS pool prototype can meet the basic requirements of Radio Access Networks. With low-latency switches and interface cards, this latency could be further reduced.

VI. Application Demo System Based on VBS Pool Prototype

Based on our VBS pool prototype, an application demo system is designed and implemented to demonstrate the VBS pool prototype. Three applications, the bidirectional VoIP application, Web Browsing application and bidirectional Video Phone application are setup on the prototype system.

In this application system, multiple VBSs are setup on the IBM X3650 server. The software based mobile station is connected to the VBS via a 2.55GHz radio link. It can connect with the SIP phone and video phone in the Internet for the VoIP call and bidirectional video call. It can also access the web through the radio link. For the VoIP application, the SIP signaling is used to establish the VoIP call which is bidirectional G.711 encoded streams over IP. For the video phone application, a bidirectional H.264 encoded 352*288 format video @ 15fps is used. The round-trip delays and quality expectations of the VoIP and video are satisfactory. The system is monitored with a tool that we have custom developed called I-view to analyze the system utilization and radio signal in runtime.

VII. Related Work

General purpose processors for Software radio have been discussed for several years. With the emergence of more powerful multi-core architectures and new instruction sets, GPPs are becoming more capable of handling wireless base band processing. Compared with traditional BS which are based on complex multi-architecture platforms comprising a mix of ASICs, network processors, DSPs or FPGAs, GPP based BSs are more scalable and flexible. This has led to new SDR platforms being developed by various groups. [18] compares several popular architectures and proposes a new multi-core architecture called SODA, which meets the requirements for W-CDMA and 802.11a systems. [19] presents the multi-core GPP based Sora software radio platform for 802.11a/g systems and solves many design challenges in meeting the processing and latency requirements of these systems. [19] is closest to our work; however, it targets a single terminal or access point in Wifi based LANs whereas we target TDD BS system and BS pooling with RRHs for 4G wireless broadband in wide area networks. This is an intermediate step in the realization of the wireless network cloud and the design challenges are quite different because of the need to support TDD timing, TDD RRHs and multiple base station instances on a shared platform.

Besides work by research communities, there also exist commercial SDR solutions based on GPP platforms, like Vanu's multiran solution [20]. This is a cost-effective option, since the expense of antennas, electronics, and backhaul can all be shared among operators. Such commercial deployments reflect the industry acceptance for GPPs used in BS design. However, Vanu's solution is for 2G and FDD systems , where the system requirements are far more relaxed, no matter for throughput or timing control.

Focusing now on prior work on component level implementations and optimizations. For the MAC layer, the typical workload involves multi-task control and massive packet processing. The papers [21][22] consider network processors, based on Intel IXP^{TM} 2xxx series of processors, and general-purpose processors based on the generic Intel Architecture. Design issues and inherent challenges are discussed for both platforms, while implementation details and performance evaluation are presented only for the IXP platform. However, the latency and real-time issues are critical here and are not discussed with results in [21]. In addition, our experiments with multiple BS instances show that the MAC layer throughput supported by multi-core servers scales much higher than with network processors. Network processors are notorious for being hard to program and do not offer the same ease, flexibility and availability of programming skills as for development targeted towards a general purpose processors.

Finally, ideas similar to the WNC concept, i.e. of having

distributed antennas with centralized computation for the wireless access network, were introduced earlier in [23]. But this concepte was not developed into a system or an implementation. In VBS pool or WNC system, the long-distance link between the RRH and the baseband processing unit (BBU) is a fundamental requirement for centralized processing. As discussed, traditionally industry standards like CPRI and OBSAI are used for the optical fiber connection between RRH and BBU. To enable such a connection, we would need to develop extra hardware modules in IT servers to handle the conversion between CPRI/OBSAI and IT platform interfaces (e.g. PCIe or Ethernet), which is not economical. Considering Ethernet ports are commonly available in commodity servers, we use those to carry the data between RRH and BBU server. With the throughput of Ethernet increasing to 100Gb/s, the bandwidth of Ethernet is not an issue. In this paper, we demonstrate a working prototype of TDD Eth-RRH with Ethernet interface, discuss the system design challenges and propose, implement and evaluate corresponding solutions.

VIII. Conclusion & Future Work

In this paper we have proposed the structure of a Virtual Base Station (VBS) pool as a step towards realizing the broader notion of a wireless network cloud (WNC). The broader WNC concepts applies IT computational platforms, virtualization, pooling and cloud principles to a radio access networks (RAN) that is subject to stringent real-time constraints. Specifically, in this paper we have shown the feasibility of virtual base station stacks (MAC and PHY) using software radio, and for the first time demonstrated the idea of virtual base stations run over pooled multi-core IT platforms connected to remote radio heads. A prototype system has been designed to demonstrate end-to-end applications running over such a virtual base station pool. The real-time nature of base station operation and synchronization between virtual base station instances and the TDD Eth-RRH are examples of challenges addressed in this paper.

Applying cloud and virtualization principles more comprehensively to the radio access networks involves many more issues beyond the real-time computational and pooling issues of wireless stacks explored in this paper. The basic cloud model needs to be extended to more robustly solve large-scale synchronization, jitter and system latencies when a large number of virtual instances are pooled (eg: a city with a thousand towers is served by a small handful of pooled sites). Further, the promise of the cloud is to leverage statistical multiplexing gains, while allowing real-time elastic allocation and fine-grained pay-as-you-go pricing for users (i.e. virtual RAN operators). This implies the need to provision on-demand capacity to hotspots rapidly and provide pricing based upon dynamically provisioned capacity etc. Beyond the usual promise of virtualization, the wireless network cloud involves base station instances being physically close to each other. This proximity and real-time data/information sharing allows powerful next-generation interference management techniques (eg: cooperative MIMO, cooperative base-station scheduling, cooperative spectrum management including cognitive radio) and faster, seamless handoff for mobile users. We are in the process to build and demonstrate systems based on these advanced concepts.

References

[1] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, J. Zhan H. Yan, and H. Zhang. A clean slate 4d approach to network control and management. *ACM SIGCOMM Computer Communication Review*, 35(5), 2005.

[2] H. Yan, D. A. Maltz, T. S. E. Ng, H. Gogineni, H. Zhang, and Z. Cai. Tesseract: A 4d network control plane. In *NSDI*, 2007.

[3] T. V. Lakshman, T. Nandagopal, R. Ramjee, K. Sabnani, , and T. Woo. The softrouter architecture. In *ACM SIGCOMM Workshop on Hot Topics in Networking*, 2004.

[4] Y. Lin, L. Shao, Z. Zhu, Q. Wang, and R. Sabhikhi. Wireless network cloud: Architecture and system requirements. *IBM Journal of Research and Development*, 54(1), Feb 2010.

[5] China Mobile Research Institute. C-ran: The road towards green ran. White Paper. Ver 1.0.0. April, 2010.

[6] P. Gupta, A. Vishwanath, S. Kalyanaraman, and Y. H. Lin. Unlocking wireless performance with cooperation in co-located base station pools. In *Comsnets*, 2010.

[7] CPRI. Cpri specification v4.0 (2008-6-30). [online]. available: http://www.cpri.info/downloads/CPRI_v_4_0_2008-06-30.pdf.

[8] http://www.obsai.org/.

[9] Timing and synchronization in wimax networks. [online]. available: http://www.chronos.co.uk/pdfs/tel/symmetricom/Timing_and_Sync_in_WiMAX_Networks.pdf.

[10] http://www-03.ibm.com/systems/x/hardware/rack/x3650/index.html.

[11] http://ark.intel.com/Product.aspx?spec=sl9ym.

[12] IEEE 802.16 Working Group. Draft standard for local and metropolitan area networks– part 16: Air interface for broadband wireless access systems, p802.16rev2/d2. Dec. 2007.

[13] WiMAX Forum. Mobile system profile 3- release 1.0 approved specification (revision 1.7.1: 2008-11-07) http://www.wimaxforum.org.

[14] P. De, V. Mann, and U. Mittal. Handling os jitter on multicore multithreaded systems. In *IPDPS*, 2009.

[15] M. Sandell, J. van de Beek, and P. Borjesson. Ml estimation of time and frequency offset in ofdm systems. *IEEE Trans. Signal Processing*, 45:1800–1805, July 1997.

[16] IEEE. Ieee std 802.16 - 2004, part 16: Air interface for fixed broadband wireless access systems, 2004.

[17] M. M. Michael, M. T. Vechev, and V. A. Saraswat. Idempotent work stealing. In *PPoPP*, 2009.

[18] Y. Lin, Hyunseok Lee, Y. Harel M. Woh, S. Mahlke, and T. Mudge. Soda: A low-power architecture for software radio. In *Proceedings of International Symposium on Computer Architecture (ISCA)*, pages 89–101, Boston, Massachusetts, June 17-21, 2006 2006.

[19] 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. In *NSDI*, 2009.

[20] http://www.vanu.com/solutions/multiran.html.

[21] G. Nair, J. Chou, T. Madejski, K. Perycz, D. Putzolu, and J. Sydir. Ieee 802.16 medium access control and provisioning. *Intel Technology Journal*, 8(3):213–228, 2004.

[22] M. Wu, F. Wu, and C. Xie. The design and implementation of wimax base station mac based on intel network processor. *Proceedings of International Conference on Embedded Software and Systems*, pages 350–354, 2008.

[23] S. Zhou, M. Zhao, X. Xu, J. Wang, and Y. Yao. Distributed wireless communication system: A new architecture for future public wireless access. *IEEE Communications Magazine*, 41(3):108– 113, Mar 2003.