A software WiMAX medium access control layer using massively multithreaded processors

M. Chetlur
U. Devi
P. Dutta
P. Gupta
L. Chen
Z. Zhu
S. Kalyanaraman
Y. Lin

This paper presents a multithreaded software implementation of the Worldwide Interoperability for Microwave Access (WiMAX™) medium access control (MAC) layer and its performance results on massively multithreaded (MMT) systems. The primary design goals of the implementation are to support the high WiMAX data rates, seamlessly scale with the number of available hardware threads, and provide per-flow guaranteed services. Our experimental results demonstrate that multithreading can be exploited to meet the high data rates of WiMAX and thus validate the IBM wire-speed processor MMT chip as a suitable platform for building WiMAX network appliances. The implementation consists of separate threads in the data and control planes, and thread coordination through concurrent data structures to enable multithreading in both the uplink and downlink data paths.

Introduction

Next-generation general-purpose processors are emerging as a potential replacement for protocol-specific network equipment. Next-generation systems, such as the IBM wire-speed processor (WSP) massively multithreaded (MMT) chip, offer both a multicore and a multithreaded programming model, and special-purpose hardware accelerators. Thus, WSP-based systems can potentially replace hybrid network solutions containing Intel x86 processors, application-specific integrated circuits, field-programmable gate arrays, digital signal processors, and network processors.

By enabling high-speed broadband Internet access using mobile wireless platforms, fourth-generation wireless technologies, such as Worldwide Interoperability for Microwave Access (WiMAX**), present a major technological and business model shift in cellular telephony. Structurally speaking, the vertically integrated wireless network stack is being standardized into horizontal layers, such as radio-interface layers based on orthogonal frequency-division multiplexing (OFDM) and *multiple-input–multiple-output* antenna technology, IP-based network layer and transport layer, and service layers based on the Session Initiation Protocol, the Service Delivery Platform (SDP), and the IP Multimedia Subsystem.

Network appliances using the IBM WSP MMT chip can be efficient alternatives to hybrid solutions in next-generation networks. WSP-based network appliances are envisioned to involve fully integrated stacks that are delivered as fully configured hardware with integrated software. In WiMAX networks (**Figure 1**), WSP-based appliances should address the evolution of base stations (BSs) using software radios and virtualization. For example, the massive multithreading feature of WSP will be suitable for virtualization and consolidation of multiple BSs in a wireless network cloud (WNC) [1]. In addition, WSP-based appliances will play a significant role in the extension of the IBM SDP and Service Provider Delivery Environment to the WiMAX core network, to ensure end-to-end quality of service. These WSP-based network appliances would require efficient software implementation of the network stack consisting of the physical (PHY) layer, medium access control (MAC) layer, and network layer. Therefore, the implementations must exploit the multicore and multithreaded features of WSP and its accelerators.

The IBM WSP MMT chip consists of 16 64-bit PowerPC* cores [2]. Each PowerPC core consists of four concurrent hardware threads. The 16 PowerPC cores are organized into four groups of four cores each. Each group is provided with an

Digital Object Identifier: 10.1147/JRD.2009.2037681

0018-8646/10/$5.00 © 2010 IBM

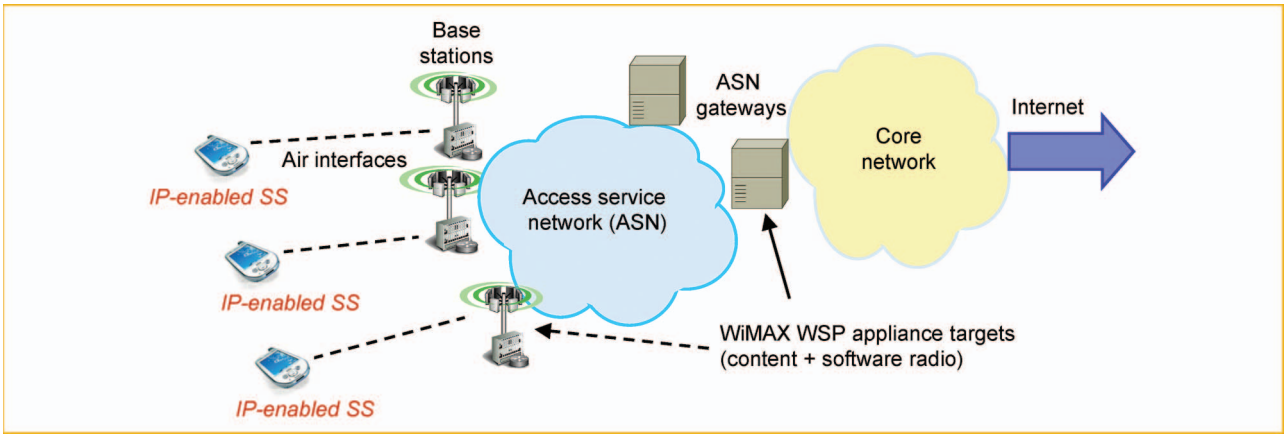

Figure 1

WiMAX architecture diagram and WSP-appliance opportunity spaces (SS: subscriber station).

on-chip 2-MB L2 cache that is shared among its cores for a total of an 8-MB L2 cache for 16 cores. The 8-MB L2 cache, along with 16 cores, is interconnected using an on-chip high-speed system bus. WSP also has cache-coherent accelerators with their own special instructions for cryptography, compression and decompression, pattern matching, and eXtensible Markup Language (XML) processing.

In this paper, we focus on the software implementation of the WiMAX MAC layer on MMT processors, such as WSP. In order to fully utilize the WSP MMT capabilities, a software implementation of the WiMAX MAC layer should scale with the increasing number of cores and threads. This presents various design options and challenges. In this context, this paper presents the design and implementation of a software multithreaded WiMAX MAC and studies its performance on multicore processors.

At the time of writing this paper, the WSP chip was not available. Hence, we conducted our experiments on one existing multicore processor system: an Intel x86 processor system. We also study the performance of our multithreaded MAC implementation on a Mambo WSP simulator. These results help in understanding the performance of a WiMAX MAC workload on multithreaded systems and, hence, on WSP. Although our experiments do not include the speedup due to accelerators, we discuss the impact and opportunity to use accelerators within the MAC implementation.

Contribution

As previously mentioned, in this paper, we present a multithreaded software implementation of the WiMAX MAC layer. In order to realize an efficient MAC layer, our design employs the following: 1) efficient concurrent data structures to improve thread scalability; 2) a minimum memory copy policy within the MAC layer to reduce memory overhead; and 3) thread pooling to reduce thread creation and destruction overhead. Our implementation resides in the user space of the Linux** OS and uses POSIX** application programming interfaces and standard scheduling mechanisms to allow easy porting of our implementation across a wide range of processors.

Our work demonstrates the capability of next-generation multicore processors in replacing the customized network processors and network elements in WiMAX. The WiMAX MAC layer implementation and its performance analysis are part of the efforts to characterize WiMAX workloads for WSP. Finally, to the best of our knowledge, our work is the first software implementation of WiMAX MAC for a general-purpose multicore processor. Therefore, it may provide insights into the design choices available for future WiMAX MAC implementations.

The remainder of this paper is organized as follows: The next section provides a detailed architecture and design of software MMT MAC and a discussion of various design considerations. In the following sections, we present detailed experimental results and related work. This paper concludes with a discussion of some directions for future work.

Software MMT WiMAX MAC

IEEE 802.16 standard and WiMAX MAC

We give a brief introduction to WiMAX MAC before we describe our design and implementation. WiMAX is a wireless communication technology based on the IEEE 802.16** standard. The IEEE 802.16 standard [3] defines the PHY- and MAC-layer specifications for "last-mile" connections in wireless metropolitan area networks (WMANs). Here, the term *last mile* refers to the final segment

of delivering connectivity from a communications provider to a customer. The WiMAX MAC layer is point to multipoint (PMP) with optional mesh support. The MAC layer can support multiple PHY specifications such as WMAN-Single Carrier (SC), WMAN-SCair (SCa), WMAN-OFDM, and WMAN-Orthogonal Frequency-Division Multiplexing Access (OFDMA). In this paper, we focus on PMP MAC for WMAN-OFDMA. In WMAN-OFDMA, a BS allocates the PHY transmission resources available in time and frequency to different subscriber stations (SSs) based on their connection requirements. The BS does this allocation on both the uplink (UL) and the downlink (DL).

The 802.16 MAC consists of a service-specific convergence sublayer (CS) and the common part sublayer (CPS). In the DL, the CS layer performs packet classification and associates the packets with appropriate service flows and connections. Service flows and connections are identified by service flow identifiers and connection identifiers (CIDs), respectively. Note that, in a BS, each service flow is mapped to a distinct connection. In this paper, we interchangeably refer to service flows and connections.

Optionally, packet header suppression (PHS) is performed before passing the packets as MAC service data units (SDUs) to the CPS layer. The CPS performs resource allocation and scheduling for quality of service (QoS), and creates the structured data sequence for frame construction at the PHY. Each frame in WiMAX occupies a fixed-size contiguous region, in both time and frequency, and has distinct subregions (called subframes) for DL and UL. A slot is the minimum time-frequency resource in the frame that can be allocated for transmission or reception. The BS scheduler allocates slots in a frame to each SS (in both UL and DL) based on the requirements of the SS.

The latest WiMAX standard supports the following five QoS scheduling types: 1) unsolicited grant service (UGS) for the constant bit rate (CBR) service; 2) real-time polling service (rtPS) for the variable bit rate (VBR) service; 3) non-real-time polling service (nrtPS) for non-real-time VBR; 4) extended real-time polling service (ertPS) for guaranteed and VBR applications such as voice-over-IP with silence suppression; and 5) best-effort (BE) service for service with no rate or delay requirements. It is the left to the scheduler and undefined in the standard to determine MAC SDUs for transmission in order to achieve the predefined service quality parameters of connections with different QoS classes. The SDUs scheduled for transmission are further fragmented and/or packed into protocol data units (PDUs) with appropriate headers. The standard allows an optional automatic repeat request (ARQ) mechanism for the transmitted PDUs to improve the transmission reliability.

As part of the UL, SSs request bandwidth grants for their rtPS, nrtPS, and BE connections. The BS scheduler performs resource allocation for UL, based on the grant requests and the QoS requirements of SSs, and broadcasts this information in the UL-MAP. Based on the received UL-MAP, the SSs transmit their SDUs in the UL slots allotted to them. In addition to the preceding data plane functions in DL and UL, the WiMAX MAC layer performs management functions to facilitate connection establishment and maintenance as part of the control plane.

Architecture

We begin with a high-level overview of the architecture of our implementation for the entire WiMAX MAC layer of a BS.

High-level overview

Our WiMAX MAC layer implementation consists of a fast *data path* performing per-frame processing activities and a slower *control path* dealing with relatively infrequent management messages (MMs). The control path consists of requests such as service flow creation and teardown (i.e., disconnect) and channel quality measurements. Currently, our implementation focuses on the fast data plane within the MAC layer of a BS with stringent timing constraints for every frame.

In both the DL and UL, the fast path can be subdivided into the following five major components: 1) CS; 2) *scheduler* (SCH); 3) *packing and unpacking* and *fragmentation and reassembly* module (P&F); 4) *ARQ* engine (ARQ); and 5) *security* module. The three modules in the middle of the list (i.e., 2–4) comprise the CPS. The security module exchanges encryption keys in the control path and performs data encryption and decryption in the data path.

Downlink

The CS, SCH, and P&F are implemented in the DL. In our implementation, the SDUs are enqueued using a CBR generator to simulate the CS layer. In every frame, the scheduler determines the allocation to each service flow based on its QoS class and its backlog in the SDU queue. This resource allocation is conveyed to the P&F module in the form of *bursts* and *frame maps*. The frame maps contain information about the payload to be sent in a frame, where the payload is obtained from the SDU queues and the waiting-for-retransmission (ReTx) queues of the ARQ. Using these maps, the P&F module dequeues from the appropriate queues to construct the MAC PDUs. A MAC PDU is an encapsulation of multiple MAC SDUs or their fragments, and is the basic unit of payload generated in the MAC layer. Multiple MAC PDUs are concatenated into bursts. These bursts form the input to the PHY layer. Finally, the PHY layer constructs physical frames for transmission over the air. As mentioned earlier, the ARQ engine is optional and responsible for improving reliability by retransmitting the lost MAC PDUs. The ARQ module receives its input partly from the scheduler on the blocks scheduled for transmission. The ARQ module is also responsible for updating the status of the transmitted blocks and starting

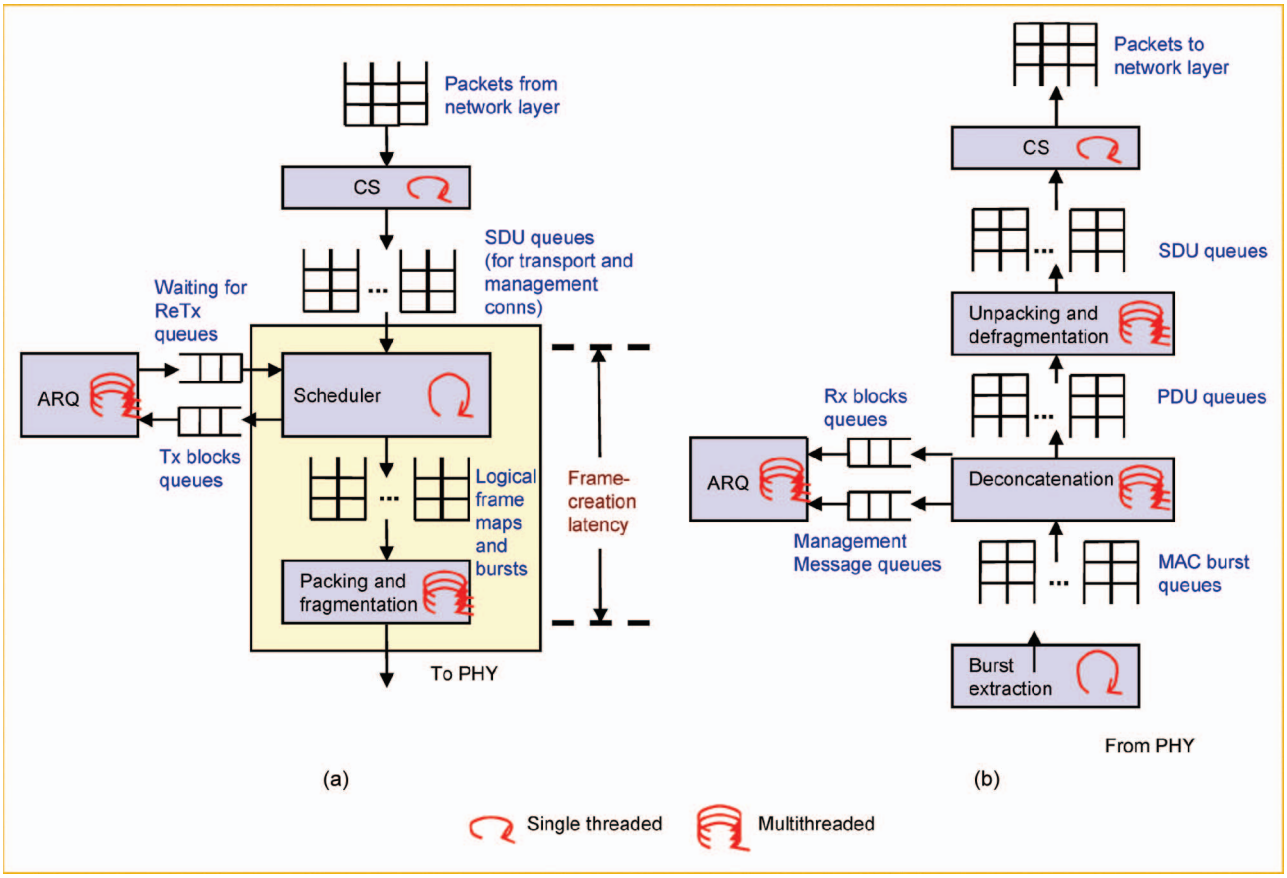

Figure 2

Pipelined architecture for the (a) DL and (b) UL (CS: convergence sublayer, ARQ: automatic repeat request, Tx: transmission, Rx: reception, ReTx: retransmission).

their timers before acknowledgements start arriving for them. A block diagram of this architecture is shown in **Figure 2(a)**. Each module is described in detail in the detailed component design section.

Uplink

The data flow of the modules and its associated functionalities are reversed in the UL, with the exception of the scheduler. The scheduler has no specific tasks to perform on the PDUs received in the UL direction. In this direction, first, the UL subframe is parsed using the UL-MAP and separated into UL bursts. Next, the bursts are deconcatenated by the P&F module to extract MAC PDUs, which are then unpacked and/or defragmented to obtain MAC SDUs. The newly obtained transport MAC SDUs are delivered to a higher layer through the CS module. Management SDUs, such as bandwidth grant request messages and ARQ feedback messages, are delivered to the appropriate module by P&F. The UL is depicted in **Figure 2(b)**.

Threading model

In the DL [refer to Figure 2(a)], the end-to-end frame-creation latency (FCL) is the delay from the time when the processing of a frame starts in the CPS to the time when the corresponding bursts and frame maps are given to the PHY for transmission. The following components contribute to this delay: 1) dequeue of SDUs in CS; 2) resource allocation in SCH; 3) MAC PDU and burst creation in P&F; and 4) determining ReTx blocks in the ARQ module. The FCL may be lowered by using multiple hardware threads if one or more of the stages exhibits parallelism. Each of these stages can be expedited by assigning a thread pool consisting of more than one thread. Furthermore, the CPS modules can be implemented as pipelined stages. Although pipelining would not lower FCL, it can help increase the frame-generation throughput and may be sufficient to guarantee frame generation within the required PHY frame duration. In principle, a similar design combining pipelining with multiple threads per pipeline stage is also feasible in the UL layer.

In our implementation, P&F is performed after SCH, i.e., the two modules are not pipelined. We are looking into alternative design choices that can facilitate concurrent execution of SCH and P&F. On the other hand, our P&F is multithreaded and capable of constructing MAC PDUs and bursts in parallel within a frame. Similarly, our ARQ engine is multithreaded and a natural candidate for seamlessly scaling with available threads. A noteworthy aspect of our overall implementation is its *minimum-copy* feature. Payloads of transport and management SDUs are transferred across stages and modules by reference only and copied only during burst (PHY SDU) creation, just before transmitting to the PHY.

The remainder of this section describes the internal features of each of the modules in detail and the opportunities that exist for parallelism.

Detailed component design

CS and SDU queues

The CS forms the interface between higher network layers and the remainder of the MAC. The SDU queues serve as the interface between CS and CPS, and are a collection of enhanced concurrent lists maintained for every active connection (both transport and management) in the MAC layer. These concurrent lists provide the basic functionalities to simultaneously peek, insert, and remove MAC data units by the threads from CS, SCH, and MAC management modules. The SDU queues establish a unified view for the transport and management data units transmitted during DL processing. This collection of disparate concurrent lists provides transparent and multithreaded interaction among different modules of the MAC layer and enables seamless scalability with increasing numbers of threads.

The connection of each SDU queue is synchronized using a distinct mutex lock; thus, accesses to queues of different connections are not serialized but are concurrent. The various modules of the MAC are "aware" of the connection-specific functionalities to be performed in this unified queue. For instance, scheduler functionality varies while accessing the SDU queues for MMs versus accessing transport data. The SDU queues have the necessary functionalities to treat data units based on their type (management or transport), QoS classes, ARQ support, fragmentation support, etc. The SDU queues also provide support functionalities to the scheduler to maintain statistics necessary for scheduling. The statistics maintained include the overall number of bytes for a connection, the overall number of SDUs for a connection, and the overall payload deficit necessary to maintain the transfer rate for a UGS connection.

Design considerations

The main motivation of the SDU queue design is to enable seamless thread scalability and transparent interaction among modules through SDU queue interfaces. The natural connection-level parallelism in the WiMAX specification is exploited with efficient concurrent lists for individual connections. Various management connections such as basic primary and secondary connections and transport connections, i.e., UGS, rtPS, ertPS, and BE connections, are maintained and managed in the same SDU queue structure, thereby providing transparent interaction among the scheduler, ARQ, and P&F modules.

Scheduler

The role of the BS scheduler is to schedule SDUs from different service flows while satisfying their promised QoS and to allocate PHY transmission resources among them. The scheduler performs resource allocation for both UL and DL and packages this information, along with other control information, in the UL-MAP and DL-MAP, respectively.

In the DL, the scheduler examines all the data units including MMs, transport SDUs, and ARQ blocks waiting for ReTx, and schedules them in order of priority and QoS. Basic MMs are given higher priority over primary MM, and both of these precede transport and ARQ data. Within a service class, ARQ blocks waiting for ReTx are scheduled before not-sent packets. Across service classes, flows are given priority based on their QoS requirements, e.g., UGS connections are scheduled before BE. For the experiments in this paper, only UGS and BE connections are considered. For this case, the priority order is defined as follows:
Basic MM > Primary MM > UGS ARQ ReTx > UGS not-sent > BE ARQ ReTx > BE not-sent.

The SDU queues of UGS CIDs are sequentially traversed as long as empty slots are available in the DL subframe. Sufficient resources are allocated to satisfy the minimum reserved traffic rate of a CID before considering the next CID. The DL scheduler does not perform the actual packing and fragmentation function. Instead, it conservatively estimates the total bytes (including overhead) needed based on the following connection attributes: packing enabled or disabled, fragmentation enabled or disabled, ARQ enabled or disabled, SDU size, PDU size, ARQ block size, etc. After scheduling resources for UGS connections, the remaining slots are equally divided among contending BE connections as "fair share" [4]. If the requirement for any BE connection is less than this "fair share," the balance slots are reclaimed into a pool, and this pool is used to service BE connections with requirements larger than the fair share. Subsequently, based on the number of slots in the DL subframe that are allocated to a burst, the scheduler assigns a region in the DL subframe to the burst, and constructs the DL-MAP, using the algorithm given in [5].

In the UL, the scheduler allocates slots for UGS connections according to their guaranteed transfer rate before examining any grant requests. If free slots are available in the UL subframe after servicing all UGS CIDs, a fair scheduling mechanism similar to that used in DL is employed for UL BE connections.

In the current implementation, the scheduler operates in an essentially sequential manner (e.g., allocate resources for one CID and only if more slots are available, and process the next CID) and in a single thread. In the future, we plan to explore scheduling heuristics with some parallelism.

Packing and fragmentation, concentration (P&F)

In the DL, the P&F module constructs PDUs and their bursts according to the specification in the logical burst map built by the scheduler. Since the different bursts are independent entities, their construction may proceed in parallel. In our design, this module is assigned a thread pool consisting of as many threads as the number of available processors. P&F is also amenable for concurrent execution in the UL.

Automatic repeat request

Overview

The IEEE 802.16 standard includes an optional ARQ engine for retransmitting packets erroneously received and, thereby, improving reliability at the MAC layer. The ARQ can be enabled on a per-connection basis. When enabled, MAC SDUs are logically partitioned into *blocks* of uniform length, with the possible exception of the last block of an SDU. Each block is assigned a *block sequence number (BSN)*, which wraps around after 2,048. With each SDU fragment in a PDU, the transmitter includes its starting BSN. The BSN is used by the receiver to send acknowledgements, i.e., positive for blocks that are correctly received and negative for blocks that are assumed to be lost. A sliding window is used to limit the number of unacknowledged blocks in flight. The size of this window, which can be up to 1,024 blocks, provides a tradeoff between the throughput of a connection and overhead in terms of the amount of state maintained for that connection. In the DL, the window slides when all blocks leading up to an unacknowledged block are either positively acknowledged or *discarded*, following a timeout waiting for acknowledgment. Before timing out, an unacknowledged block is retransmitted, possibly multiple times. In the UL, the sliding window moves when all blocks leading up to a future block are either received or purged and ignored, following a timeout. If either the transmitter or the receiver has its window unchanged for a sufficiently long time, then synchronization between the transmitter and the receiver is assumed to be lost for the connection, and the connection is reset.

Design

To realize the ARQ specification, in both DL and UL, the state of each block in the sliding window needs to be maintained for each ARQ-enabled connection. In our design, in the DL, the ARQ engine consists of a *transmitted block processing* (TxPROC) module and a *timer management* (TIMER) module. TxPROC receives its input in per-connection *transmitted block queues* from the scheduler on the blocks scheduled for transmission. The sliding window expands (advances at the right end) when a fresh block is transmitted (subject to the length of the window remaining within limits). Each transmitted block queue is *single-enqueuer single-dequeuer* and hence can be accessed without locks. Furthermore, since the queues are perfectly parallel at the level of connections, it is easy to control the number of threads assigned to dequeue the queues and process them based on the load and the number of available hardware threads. A queue is said to be single-enqueuer (or single-dequeuer) if exactly one software thread can enqueue into (or dequeue from) it. Transmitted block queues can be designed to be single-enqueuer single-dequeuer by partitioning each logical queue into multiple physical queues by CIDs and ensuring that the threads for processing the queues in each direction are also similarly partitioned by CIDs.

The TIMER module is responsible for scheduling blocks for ReTxs and retiring them in a timely manner. For this purpose, two timers, i.e., a *retry timeout timer* and a *block lifetime timer*, are associated with each block. The timers of each connection and their processing are independent of one another. Therefore, similar to TxPROC, the TIMER module is also easily parallelizable. In our implementation, timers of all connections are maintained in order of their expiration times using a priority queue (implemented as a pointer-based binary heap). If concurrent processing is desired, the queue may be partitioned into multiple sets with each set for a group of connections.

When the retry timeout timer of a block expires, it should be scheduled for ReTx by the scheduler. The details of all such blocks are communicated to the scheduler using *waiting-for-ReTx blocks queues*, again with one per connection. These queues may require insertions at arbitrary points and hence are implemented as linked lists. These queues are accessed by at least two threads and therefore use locks for synchronization. In the UL, the ARQ engine is informed of blocks received by P&F using single-enqueuer–single-dequeuer *received blocks* queues, and of MMs such as ARQ-Feedback, ARQ-Discard, etc., using *MM* queues, which are again single-enqueuer single-dequeuer. Processing associated with these queues is also amenable to flexible hardware thread allocation.

Open issues

In this section, we discuss some open issues with respect to implementing a scalable MAC and providing real-time guarantees.

Scalability issues

Unlike many packet-processing applications, WiMAX MAC is not inherently parallel at the level of packets or even connections. While most modules of the MAC exhibit natural connection-level parallelism, the scheduler is sequential in that the allocations of a connection or the regions of a connection in the DL and UL subframes cannot independently

be determined. Additionally, it is not obvious whether the scheduler's subtasks can be pipelined. As mentioned earlier, this limitation can partially be overcome by pipelining the various MAC modules. However, in order to fully realize the benefit of pipelining, design choices should carefully be considered. In this respect, we discuss two issues here.

First, communication among modules should involve mechanisms that incur low overhead. In our implementation, accesses to the concurrent queues used for the purpose (with the exception of those that are single-enqueuer single-dequeuer) are currently coordinated using locks. Some of the queues, e.g., the waiting-for-ReTx queues of an ARQ, are actually linked lists, to which insertions and deletions can be at arbitrary points and hence are somewhat time consuming to coordinate. Furthermore, since synchronization based on locks can lead to priority inversions, special care should be taken to ensure they are avoided. One alternative to lock-based synchronization is the lock-free approach. Lock-free synchronization is guaranteed to not lead to deadlocks or priority inversions and can be implemented with very little or no kernel support (and, hence, low overhead) but has the drawback that low-priority threads could "starve" (i.e., the threads may be denied execution for an inordinately long time and are unable to progress in their computation); thus, lock-free synchronization is generally recommended only for small objects. We plan on exploring the viability of this approach to our system in future work.

Second, if the number of software threads exceeds the number of cores or hardware threads, then threads should be prioritized such that they execute at the right time, and later stages are not starved. A simple static-priority scheme may not suffice since, under such a scheme, a lower-priority thread may starve. In our current implementation, threads are scheduled under the default time-shared round-robin algorithm. Although this algorithm cannot lead to starvation (assuming all threads are of equal priority), it is not capable of executing threads based on their urgency (or dynamic priority), which can lead to undue delays in frame creation. Sophisticated dynamic-priority schemes will be considered in future work.

Real-time issues

The IEEE 802.16 standard requires that a PHY SDU (logical frame) be delivered to the PHY layer once for every frame duration (5 ms in our implementation). Missing these deadlines due to the possible loss of synchronization between the BS and SSs can be quite expensive in terms of the time wasted for resynchronization, as well as possible revenue loss due to a decrease in quality of service. Commercial BS deployments have very low tolerance for deadline misses since a connection may not be served at its accepted QoS level if frames are not transmitted in time. Hence, it is mandatory to determine bounds on the percentage of deadlines that may be missed and keep them within limits. Such bounds can be arrived at by profiling the code to determine the execution costs of various modules and understanding the extent of external interferences, such as interrupts, the behavior of the OS, and the scheduling algorithms used. Providing strict guarantees when the underlying OS is of nonreal time is nearly impossible. Tuning the system to provide real-time guarantees, including porting to an OS with adequate real-time support and using real-time schedulers will be part of future work.

Accelerators

Next-generation processors have built-in accelerators to efficiently perform specialized operations. For instance, the WSP has accelerators for cryptography, pattern matching, and XML operations. PHS in the CS of WiMAX MAC involves pattern-matching operations on the packet header. In addition, the encryption and decryption of payload in the security module can make use of cryptography engines. The use of WSP accelerators to expedite MAC-layer functionalities will be part of future work.

Experimental analysis of MMT MAC

In this section, we present our experimental setup and a detailed analysis of experimental results. The architecture described in the previous section is implemented in C using POSIX threads. The experiments were performed on an Intel-based multicore platform consisting of two Intel Xeon** Dual Core 5160 processors for a total of four cores with a 32-KB L1 cache per core, 4-MB L2 cache per processor (which is shared by the two cores on it), and 7 GB of random-access memory running Linux (Red Hat** Enterprise Linux 5, kernel version 2.6.18).

Our experiments have two main goals. The first goal is to validate that our MMT MAC implementation can support high throughputs comparable to those estimated from the WiMAX standard. The exact throughput values and the computational load depend on the SS, CID, and traffic configuration chosen, and we study two such configurations. A second goal is to understand the effect that increasing the number of cores has on the performance of the MAC layer. More cores can speed up execution through parallelization, but there is a tradeoff due to thread switching and synchronization overheads. Through these goals, we want to study how the multicore multithreaded nature of WSP can help support high throughputs in WiMAX networks, possibly for multiple sectors of a BS or even multiple BSs.

Test framework

The test framework is shown in **Figure 3**. The application traffic is generated by a CBR generator. The generated packets are enqueued into the transport CID queues by the CS layer. To simulate the MM load, an MM generator enqueues primary and basic MMs in their respective CID queues. As our test framework does not include a separate implementation of SS functionality, we simulate the

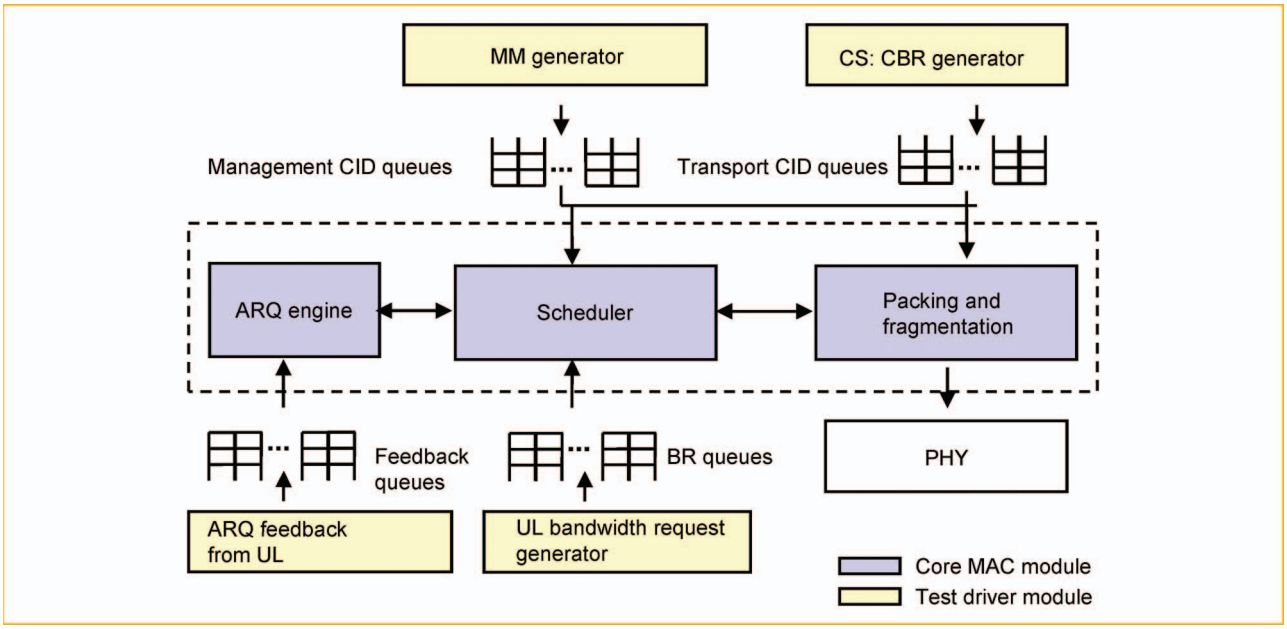

Figure 3

Drivers in the test framework in DL.

following two inputs from the UL messages that are required for the DL computations: The first input comes from the ARQ feedback generator to generate the UL ARQ-Feedback MMs for the DL ARQ module. The second input comes from the UL bandwidth request (BR) generator to generate the BR headers for the UL-MAP calculations by the scheduler. Each generator communicates with the core MAC modules with its dedicated set of concurrent queues.

In our experimental setup, the CID space is partitioned based on the CID type: Basic, Broadcast, Primary Management, and Transport CIDs. Transport CIDs are of two QoS scheduling types: UGS and BE. (There are no secondary management connections in our experimental setup.)

Experimental results

Service traffic characteristics

For all of our experiments, we use synthetic CBR service traffic based on the ITU-T G.711 codec. For each transport CID, the application traffic rate is either at the rate of the G.711 codec or twice the rate of G.711. We create synthetic G.711 traffic by generating an empty packet of size 200 bytes (where 160 bytes is the G.711 packet size and 40 bytes is the header overhead for Real-Time Transport Protocol (RTP), User Datagram Protocol (UDP), and IP) every 20 ms. Synthetic traffic at twice the rate of G.711 is created by generating one 200-byte packet every 10 ms.

Base system parameters

The following base system parameters are kept fixed in all our experiments. The channel bandwidth is 20 MHz, and the frame duration is 5 ms. The number of OFDM symbols in a frame is 48 (which is a number that depends on the channel bandwidth and cyclic prefix of OFDM); the ratio between the DL and UL subframe durations is 2 : 1; and the subchannel permutation scheme is Partial Usage of SubChannels (PUSC). The most robust modulation and coding scheme (MCS) allowed by the IEEE 802.16 standard is assumed. This scheme is referred to as *64-QAM-3/4*, which stands for 64-quadrature amplitude modulation and 3/4 coding rate.

Test parameters

The following test parameters are kept fixed in all our experiments. First, the duration for each experiment is set to 300 s, thus generating 60,000 frames. Second, P&F is always enabled. Third, ARQ is always enabled, the ARQ block size is set to 100, the window size is set to 1,024, and the percentage of blocks for which feedback is lost is set to 1%.

Results

The primary goal of our experiments is to show that our implementation can sustain throughputs close to theoretically estimated values. To determine whether a given throughput can be sustained, we measure the FCL for that throughput, i.e., the difference between the time CPS starts the computation for a frame to the time when the frame is delivered to the PHY [as

Table 1 Calculations for the estimation of the MAC layer payload throughput (excluding overheads) that a DL subframe can carry. Here, N_{FCH}, which refers to the number of slots needed for the frame control header, is equal to 4 (N_{SS}: number of subscriber stations, FFT: fast Fourier transform, TTG: transmit/receive transition gap, RTG: receive/transmit transition gap, QAM: quadrature amplitude modulation).

Serial number	*Parameter*	N_{SS} = 15 SS	N_{SS} = 30 SS
1	Bandwidth	20 MHz	20 MHz
2	Number of FFT subcarriers	2,048	2,048
3	Frame duration (FD)	5 ms	5 ms
4	Number of OFDM symbols	48	48
5	DL:UL ratio	2:1	2:1
6	Number of usable DL symbols for data	30	30
7	Number of DL subchannels	60	60
8	Number of slots in DL (N_{DL})	900 slots	900 slots
9	Estimated DL-MAP overhead ($N_{DL\text{-}MAP}$)	36 slots	70 slots
10	Estimated UL-MAP overhead ($N_{UL\text{-}MAP}$)	12 slots	22 slots
11	Slot fragmentation overhead ($N_F = N_{SS}$)	15 slots	30 slots
12	Useful data slots ($N_U = N_{DL} - N_{DL\text{-}MAP} - N_{UL\text{-}MAP} - N_F - N_{FCH}$)	833	774
13	Bytes per slot for 64-QAM ¾ code rate (N_{bps})	27	27
14	Available data bytes per frame ($N_D = N_U \times N_{bps}$)	22,491	20,898
15	Payload bytes per frame (N_{PL})	21,420	19,900
16	Estimated payload throughput ($T = 8 \times N_{PL}/FD$)	34.272 Mb/s	31.84 Mb/s

shown in Figure 2(a)]. If the FCL is less than the frame duration (5 ms) for most of the frames sent during the experiment, we conclude that the corresponding throughput can be sustained by our implementation.

Before presenting our results, in **Table 1**, we describe the calculations for the expected MAC DL payload throughput for two example scenarios in which there are 15 and 30 SSs in the system. MAC payload is the data bytes received by the MAC layer from the upper layers in the form of MAC SDUs but does not include the overhead bytes due to the frame headers that are included in a MAC PDU. The throughput values are estimated based on the channel bandwidth, MCS, frame duration, DL:UL subframe duration ratio, and the overheads due to headers and control information. The exact achieved throughput can vary a little around this value, depending on the connection attributes (e.g., P&F-enabled or -disabled SDU size), specific load conditions, and the scheduler implementation.

In Table 1, the rows from *Bandwidth* to *DL:UL ratio* give the various PHY layer parameter values that are assumed in our calculation [3]. For this choice of parameter values, each frame contains 32 DL symbols and 16 UL symbols. Accounting for one preamble symbol for PHY synchronization and assuming a Receive/Transmit Transition Gap of 1 OFDM symbol duration [3] leaves us with 30 DL symbols that can be used for sending data. The permutation scheme considered is PUSC [3], which leads to 60 DL subchannels and, hence, 900 DL slots. The chosen modulation is 64-QAM, and the forward error correction code rate is 3/4, which leads to 27 data bytes per slot [3].

So far, the parameters previously discussed are fixed based on the system configuration. Next, we estimate the variable overheads (e.g., the sizes of DL-MAP and UL-MAP) that depend on the connection attributes, with the actual data being transmitted, and also the MAC implementation. We also account for the possible slot fragmentation overhead, i.e., the space wastage that can potentially occur because the smallest unit of allocation is a slot. Finally, we estimate the header overhead, assuming one G.711 SDU per PDU (i.e., 10 bytes of header is needed for 200 bytes of payload), and use it to calculate the payload bytes per frame. Thus, we obtain estimated throughputs of 31.84 and 34.272 Mb/s when the number of SSs is 30 and 15, respectively.

We consider two sets of application traffic load for our experiments, as shown in **Table 2**. The configuration for *Set 1* is 30 SSs, with eight UGS and three BE connections each, leading to a total of 330 CIDs. The SDUs for each CID are generated at the rate of a G.711 connection, i.e., 200-byte packets every 20 ms. *Set 2* has 15 SSs, with eight UGS and four BE connections each, leading to a total of 180 CIDs. The SDUs for each of these CIDs are generated at twice the rate of a G.711 connection, i.e., 200-byte packets every 10 ms. The goals for choosing these configurations were the following: 1) to have a realistic traffic pattern, which is captured in G.711 connections; 2) to have a large number of CIDs for simulating heavy computational load; and 3) to simulate the scenario of multiple SSs while keeping their number reasonable, because the overheads increase with the number of SSs, leading to reduced payload throughput. We test both these traffic loads for one, two, and three cores. As shown in Table 2, the sustained DL payload throughputs of Set 1 and Set 2 (26.66 and 29.08 Mb/s, respectively) are close to but less than the estimated throughputs in Table 1. This can be improved with advanced scheduler algorithms and will be explored in future work.

Table 2 Parameters and computation of supported DL payload throughput for two test configurations (SS: subscriber station, UGS: unsolicited grant service, CIDs: connection identifiers, BE: best-effort service, ARQ: automatic repeat request).

Parameters	*Set 1*	*Set 2*
Number of SS (N_{SS})	30	15
Number of UGS CIDs per SS (N_{UGS})	8	8
Number of UGS CIDs per SS (N_{BE})	3	4
Total number of CIDs ($N_T = (N_{UGS} + N_{BE}) \times N_{SS}$)	330	180
Data rate per CID (R)	80 Kb/s	160 Kb/s
ARQ retransmission rate (R_{ARQ})	0.01	0.01
Supported DL payload throughput ($= N_T \times R \times (1 + R_{ARQ})$)	26.664 Mb/s	29.088 Mb/s
Estimated theoretical throughput (from Table 1)	31.84 Mb/s	34.272 Mb/s

Figure 4 presents both DL and UL average FCL values for Set 1 and Set 2. As shown in Figure 4, for all four cases, the average latencies are well within the desired limit of 5 ms. In our experiments, we also observed that the latencies are within 5 ms for at least 99.99% of the frames.

In Figure 4, comparing the average latencies for DL frame processing with varying numbers of cores, we note that the DL latency decreases by 38% for Set 1 and 45% for Set 2 when the number of cores is increased from 1 to 2. The reduction is less than 50% (of that which is ideally possible when computation can equally be divided between the two cores), because a significant portion of the computation in DL (i.e., the scheduler) is sequential. Furthermore, in the DL, the scheduler sequentially processes the allocation for each SS and CID; therefore, its computation linearly increases with the number of CIDs. In our experiments, Set 1 has a higher number of CIDs than Set 2. Hence, when going from one core to two cores, the improvement seen in Set 1 is smaller than that seen in Set 2. However, for all four cases in Figure 4, the latency does not decrease while going from two to three cores. We surmise that this is because, in the current design, the concurrent fraction and computational load for a single MAC instance is not high enough to subsume the overhead (e.g., synchronization) imposed by an additional software thread running on the third core. In addition, in the DL, we note that, despite the higher throughput of Set 2 compared with that of Set 1, the latency is smaller for Set 2, as shown in Figures 4(a) and 4(b). Similar to the case previously discussed, this trend arises because the computation required in the sequential portion of frame creation (i.e., the scheduler) is higher in Set 1 compared to Set 2.

Figure 5 presents the average latency and the fraction of frames with latencies less than 5 ms when three MAC instances are simultaneously run. This experiment simulates the case when multiple sectors of a BS are hosted on the same machine. The results presented in Figure 5 show a clear advantage of increasing the number of cores in the multisector case. A single core cannot support three sectors: The average frame latency is much larger than 5 ms, and the percentage of frames meeting the 5 ms deadline is less than 50%. Increasing the number of cores to two and three gives steady improvement in the percentage of frames sent within 5 ms and in the average latency. We can expect even greater advantage in a WNC [1], in which many instances of MAC are expected to run on a processor cloud. Thus, in a WNC, a multithreaded MAC implementation such as ours would enable very efficient utilization of the computational resources.

As we previously mentioned, in all our experiments for a single instance of MAC (Figure 4), the FCL is maintained within the desired limit of 5 ms for at least 99.99% of the frames. Given that our implementation is not making use of a real-time OS, this is clearly a good indicator of the stability and predictability of our system. However, as the commercial BS deployments have very low tolerance to deadline misses, in future work, we would like to investigate how to meet the 5-ms guarantee for even higher fraction of frames.

Experiments using the WSP Mambo simulator

We also ported our code to a WSP simulator modeled using Mambo, which is a full system simulator for PowerPC* architecture platforms, and conducted experiments using it. A simulator for the complete WSP chip is not available at the time of writing this paper. In addition, the current version of Mambo does not support the cryptoaccelerator. Thus, assuming that the encryption and decryption computations can be offloaded to the accelerator unit at negligible cost, we excluded such computations from our code. In the cycle-accurate mode, with a single enabled A2 core (a 64-bit PowerPC core), the simulation time is roughly 10,000 times the actual execution time, and this time almost linearly increases with the number of A2 cores. Hence, in our experiments, we enabled only a single A2 core (running at the frequency of 2.3 GHz) with four hardware threads (whereas a WSP version-1 chip has 16 A2 cores). To compare the performance with the x86-based implementation, we conducted the experimentation using Mambo for 300 s. We performed experiments by varying the number of hardware threads from one to four. We found the simulated A2 core to be capable of sustaining throughputs as sustained by the x86-based system described earlier while meeting the 5-ms limit on latency on more than 99% of the

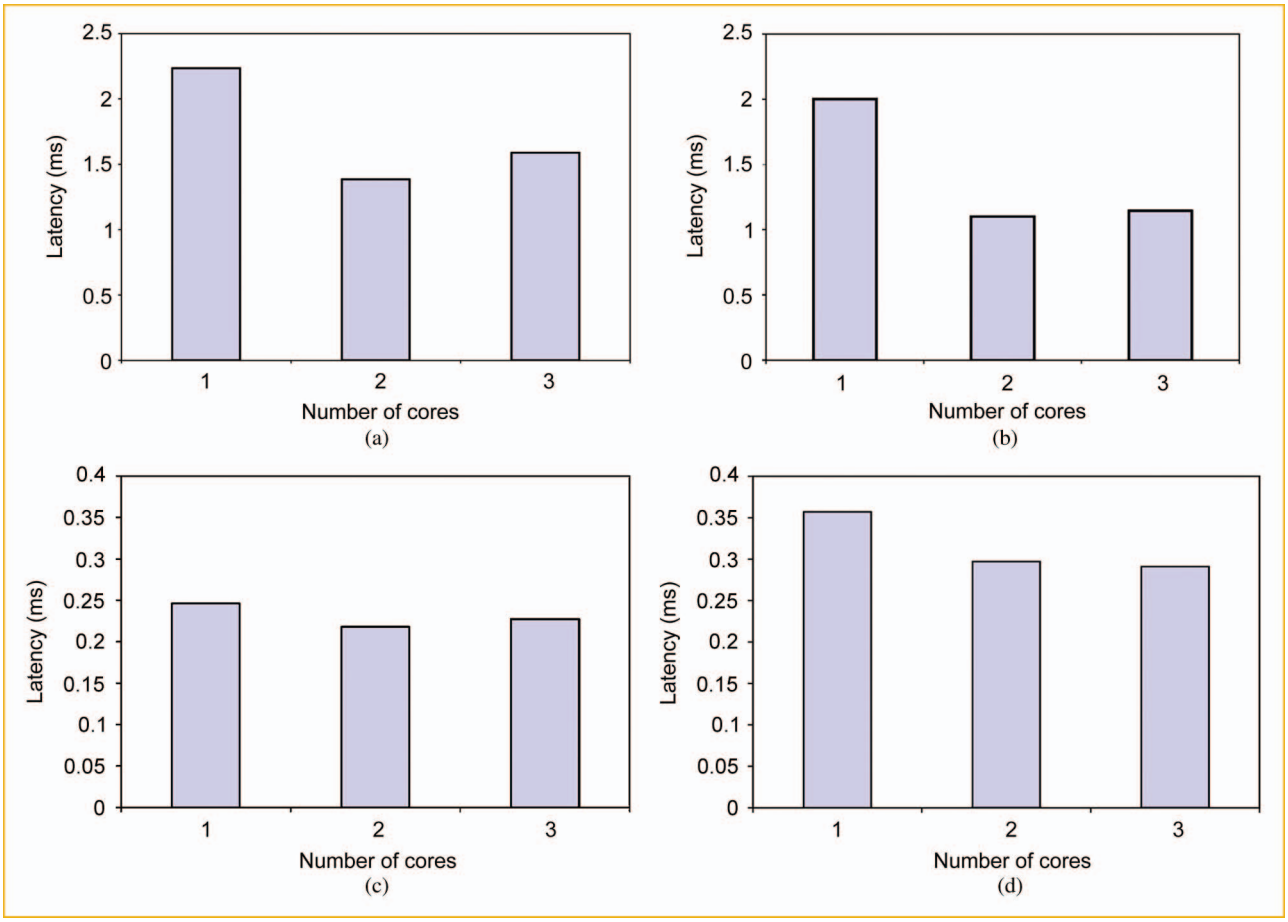

Figure 4

Average frame latency values for four cases. (a) DL with Set 1. (b) DL with Set 2. (c) UL with Set 1. (d) UL with Set 2.

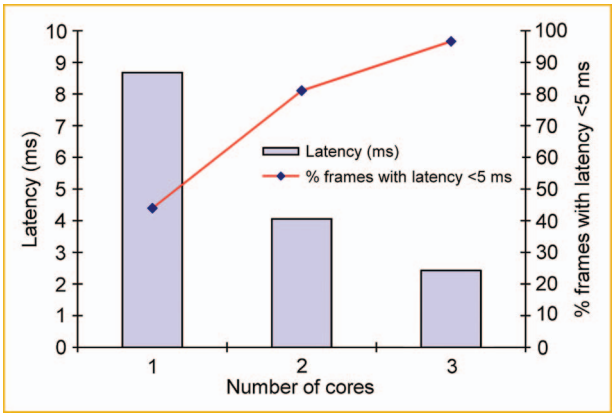

Figure 5

Average frame latency and percentage of frames with latency less than 5 ms when three MAC instances are simultaneously run.

frames, regardless of the number of hardware threads enabled. Since all the hardware threads share common functional units, enabling more hardware threads can be expected to only sublinearly improve latency, which is confirmed by the results observed. These preliminary numbers for WSP are promising but have to be validated on a more sophisticated simulator and, ultimately, directly on the WSP chip. Finally, WSP, with its 16 cores, also offers a very good scope for scalability in a multisector BS setup and in a WNC [1], by its ability to host multiple MAC instances corresponding to multiple sectors of the same BS or MAC instances of different BSs.

Related work

There have been some prior efforts, both research [6, 7] and commercial, in implementing the MAC layer of WiMAX BSs on programmable processors. Reference [6] considers both network processors, based on the Intel IXP** 2xxx series of processors and general-purpose processors based on the generic Intel Architecture. Design issues and inherent

challenges are discussed for both platforms, whereas implementation details and performance evaluation are presented for the IXP platform. In the IXP implementation, different modules of the fast data plane are bound to separate threads on the microengines, whereas the slow control plane runs on the XScale** control processor. According to their performance studies, the IXP 2350 and 2850 network processors can each easily support four independent 10-MHz channels with a combined throughput of 148.608 Mb/s and can guarantee scalability. In [7], attention is limited to IXP 2350. The design details and implementation issues presented are similar to those in [6], whereas the performance results are more elaborate, with the conclusion that the IXP 2350-based design is both flexible and scalable. Both of these papers do not discuss how the performance scales with the number of microengines and threads.

There is a significant body of work in optimizing individual MAC modules, such as the ARQ, scheduler, and P&F. With reference to the ARQ, Sayenko et al. studied the impact of ARQ parameters, such as window size, block size, scheduling of block ReTxs, and feedback type, on the application performance [8]. Chatterjee et al. presented performance improvement using an ARQ-enabled protocol for streaming applications [9]. Hempel et al. presented the selective request ARQ mechanism and its performance evaluation for ns-2 simulators [10]. However, there is no prior work on a multithreaded design of an ARQ targeting multicore processors.

WiMAX scheduling is a well-researched area, and the related work pointed out in this section presents a few of the recent works in WiMAX scheduling. Li et al. presented a survey on mobile WiMAX, with specific emphasis on QoS provisioning and mobile WiMAX specification [11]. Cicconetti et al. presented the QoS support in an 802.16 specification [12]. Sayenko et al. presented an efficient scheduling solution that is capable of allocation based on QoS requirements, BRs, and WiMAX network parameters [13]. Huang et al. evaluated the performance of DL QoS scheduling with different radio resource management for streaming applications [14]. Belghith and Nuaymi presented a comparative study of WiMAX scheduling algorithms and enhancements to scheduling for an rtPS QoS class [15]. Jain presented scheduling schemes as part of a WiMAX system evaluation methodology [16]. Bacioiccola et al. presented a simple data region allocation algorithm and its evaluation [5]; our implementation adopts this algorithm for data region allocation.

Conclusion and future work

In this paper, we have presented the design and implementation of a software WiMAX MAC that exploits the massive multithreading in next-generation processors, such as WSP from IBM. Our experiments have demonstrated that the implementation can sustain close to the theoretically estimated peak throughput. Improving performance in this regard using alternative designs for the scheduler and P&F modules will be part of future work. Two other main directions for future work are given as follows: First, although our x86-based implementation and Mambo WSP implementation satisfy the real-time requirement on frame duration for 99.99% and 99% of the frames, respectively, commercial deployments cannot tolerate any frame-duration deadline misses. We are currently investigating techniques and system requirements that can provide better real-time guarantees. Second, during our implementation, we have identified some components of the MAC that can be expedited using the accelerators in WSP. We plan to include such optimization in the next version of the implementation.

References

1. Y. Lin, L. Shao, Z. Zhu, Q. Wang, and R. K. Sabhikhi, "Wireless network cloud: architecture and system requirements," *IBM J. Res. & Dev.*, vol. 54, no. 1, Paper 4:1–12, 2010, this issue.
2. H. Franke, J. Xenidis, C. Basso, B. M. Bass, S. S. Woodward, J. D. Brown, and C. L. Johnson, "Introduction to the wire-speed processor and architecture," *IBM J. Res. & Dev.*, vol. 54, no. 1, Paper 3:1–11, 2010, this issue.
3. IEEE 802.16 Working Group, *IEEE Standard for Local and Metropolitan Area Networks–Part 16: Air Interface for Fixed Broadband Wireless Access Systems*, IEEE Std. 802.16-2004, Oct. 2004.
4. C. So-In, R. Jain, and A. A. Tamimi, "Scheduling in WiMAX: Baseline multi-class simulations," in *WiMAX Forum Application Working Group Meeting*, Washington, DC, Nov. 19–20, 2007. [Online]. Available: http://www.cse.wustl.edu/~jain/wimax/schd711.htm
5. A. Bacioccola, C. Cicconetti, A. Erta, L. Lenzini, and E. Mingozzi, "A downlink data region allocation algorithm for IEEE 802.16e OFDMA," in *Proc. 6th ICICS*, Singapore, 2007, pp. 1–5.
6. G. Nair, J. Chou, T. Madejski, K. Perycz, D. Putzolu, and J. Sydir, "IEEE 802.16 medium access control and provisioning," *Intel Technol. J.*, vol. 8, no. 3, pp. 213–228, 2004.
7. M. Wu, F. Wu, and C. Xie, "The design and implementation of WiMAX base station MAC based on Intel network processor," in *Proc. Int. Conf. Embedded Softw. Syst.*, 2008, pp. 350–354.
8. A. Sayenko, V. Tykhomyrov, H. Martikainen, and O. Alanen, "Performance analysis of the IEEE 802.16 ARQ mechanism," in *Proc. 10th ACM Symp. Model., Anal., Simul. Wireless Mobile Syst.*, 2007, pp. 314–322.
9. M. Chatterjee, S. Sengupta, and S. Ganguly, "Feedback-based real-time streaming over WiMax," *IEEE Trans. Wireless Commun.*, vol. 14, no. 3, pp. 64–71, Feb. 2007.
10. M. Hempel, W. Wang, H. Sharif, T. Zhou, and P. Mahasukhon, "Implementation and performance evaluation of selective repeat ARQ for WiMAX NS-2 model," in *Proc. 33rd IEEE Conf. Local Comput. Netw.*, 2008, pp. 230–235.
11. B. Li, Y. Qin, C. P. Low, and C. L. Gwee, "A survey on mobile WiMAX," *IEEE Commun. Mag.*, vol. 45, no. 12, pp. 70–75, Dec. 2007.
12. C. Cicconetti, L. Lenzini, E. Mingozzi, and C. Eklund, "Quality of service support in IEEE 802.16 networks," *IEEE Netw.*, vol. 20, no. 2, pp. 50–55, Mar./Apr. 2006.

13. A. Sayenko, O. Alanen, J. Karhula, and T. Hämäläinen, "Ensuring the QoS requirements in 802.16 scheduling," in *Proc. 9th ACM Int. Symp. Model. Anal. Simul. Wireless Mobile Syst.*, Terromolinos, Spain, 2006, pp. 108–117.
14. C. Huang, H. Juan, M. Lin, and C. Chang, "Radio resource management of heterogeneous services in mobile WiMAX systems," *Wireless Commun.*, vol. 14, no. 1, pp. 20–26, Feb. 2007.
15. A. Belghith and L. Nuaymi, "Comparison of WiMAX scheduling algorithms and proposals for the rtPS QoS class," in *Proc. 14th Eur. Wireless Conf.*, 2008, pp. 1–6.
16. R. Jain, "WiMAX system evaluation methodology version 2.1," in *WiMAX Forum*, 2008. [Online]. Available: http://www.wimaxforum.org/sites/wimaxforum.org/files/documentation/2009/wimax_system_evaluation_methodology_v2_1.pdf

Received December 22, 2008; accepted for publication January 17, 2009

Malolan Chetlur *IBM India Research Laboratory, Bangalore 560071, India (mchetlur@in.ibm.com).* Dr. Chetlur received the M.S. and Ph.D. degrees in computer engineering from the University of Cincinnati, Cincinnati, OH, in 1999 and 2007, respectively. He is currently working on next-generation telecom services and technologies, specifically the intersection of 4G telecom services and IT services, with IBM India Research Laboratory, Bangalore, India. Prior to joining IBM Research, he was a Principal Technical Staff Member with AT&T, developing system automation solutions and enterprise solutions. His research interests include parallel and distributed simulation, services research, and experimental computing. Dr. Chetlur is a member of the Association for Computing Machinery.

Umamaheswari Devi *IBM India Research Laboratory, Bangalore 560071, India (umamadev@in.ibm.com).* Dr. Devi received the M.S. and Ph.D. degrees in computer science from the University of North Carolina, Chapel Hill, in 2003 and 2006, respectively. She is a Research Staff Member with the Next-Generation Telecom Research Group, IBM India Research Laboratory, Bangalore, India. She is currently working on fourth-generation wireless technologies, with building telecommunication appliances on emerging and next-generation processing platforms as a special focus. Her research interests include real-time scheduling theory and operating systems, resource-allocation algorithms, and distributed systems.

Partha Dutta *IBM India Research Laboratory, Bangalore 560071, India (parthdut@in.ibm.com).* Dr. Dutta received the B.Tech. degree in electrical engineering from the Indian Institute of Technology, Kanpur, India, in 1999 and the Ph.D. degree in computer science from Swiss Federal Institute of Technology, Lausanne, Switzerland, in 2005, respectively. He is a Research Staff Member with the Next-Generation Telecom Research Group, IBM India Research Laboratory, Bangalore, India. His research interests are distributed systems and wireless networks, and he is currently working on problem management in large distributed systems and software architectures for fourth-generation telecom appliances. Dr. Dutta is a member of the Association for Computing Machinery.

Parul Gupta *IBM India Research Laboratory, Bangalore 560071, India (parulgupta@in.ibm.com).* Ms. Gupta received the B.Tech. degree in electrical engineering from the Indian Institute of Technology, Bombay, India, and the M.S. degree in electrical engineering from the University of California, Los Angeles, in 2002 and 2003, respectively. She is currently a Technical Staff Member with the Next-Generation Telecom Research Group, IBM India Research Laboratory, Bangalore, India. Her research interests include algorithm development for wireless communication systems, spanning concepts in physical-layer design, including multiple-input–multiple-output systems, medium access, and cross-layer routing protocols.

Lin Chen *IBM Research Division, China Research Laboratory, Beijing 100193, China (clchenl@cn.ibm.com).* Dr. Chen received the B.S. and M.S. degrees in computer science from Chongqing University of Posts and Telecommunications, Chongqing, China, in 1999 and 2002, respectively, and the Ph.D. degree in computer science from Shanghai Jiao Tong University, Shanghai, China, in 2007. She then joined the IBM Research Division, China Research Laboratory, Beijing, China, where she is currently a Staff Researcher with the System Software Group. Her research interests include general multicore IT infrastructure for wireless networks and future networks, network protocol and optimization in wireless ad hoc networks, and wireless sensor networks.

Zhenbo Zhu *IBM Research Division, China Research Laboratory, Beijing 100193, China (zhuzb@cn.ibm.com).* Mr. Zhu received the B.S. and M.S. degrees in control theory from Tsinghua University, Shenzhen, China, in 2003 and 2006, respectively. He then joined the IBM Research Division, China Research Laboratory, Beijing, China, where he has worked on the workload study and optimization of signal-processing algorithms and networks, and is currently a Researcher with the System Software and Networking Department. He is an author or coauthor of six technical papers. He is the holder of four patents.

Shivkumar Kalyanaraman *IBM India Research Laboratory, Bangalore 560071, India (shivkumar-k@in.ibm.com).* Dr. Kalyanaraman received the B.Tech. degree in computer science from the Indian Institute of Technology, Madras, India, in 1993, the M.S. and Ph.D. degrees from Ohio State University, Columbus, in 1994 and 1997, respectively, and the Executive M.B.A. (EMBA) degree from Rensselaer Polytechnic Institute, Troy, NY, in 2005. He is a Manager of the Next-Generation Telecom Research Group and a Research Staff Member with the IBM India Research Laboratory, Bangalore, India. Previously, he was a Professor with the Department of Electrical, Computer and Systems Engineering, Rensselaer Polytechnic Institute. His current research in IBM is on the intersection of emerging wireless technologies and IBM middleware and systems technologies. He was selected by the MIT *Technology Review* magazine in 1999 as one of the top 100 young innovators for the new millennium. He served as the TPC Cochair of IEEE INFOCOM 2008. He serves on the Editorial Board of the *IEEE/ACM Transactions on Networking*. He is a Senior Member of the Association for Computing Machinery.

Yonghua Lin *IBM Research Division, China Research Laboratory, Beijing 100193, China (linyh@cn.ibm.com).* Ms. Lin received the B.S. and M.S. degrees in information and communication from Xi'an Jiaotong University, Xi'an, China, in 2000 and 2003, respectively. She then joined the IBM Research Division, China Research Laboratory, Beijing, China, where she has worked on multiple projects related to multicore processors (general multicore processors and network processors) and networking, including topics such as high-end routers, Internet Protocol television media gateways, and mobile base stations, and is currently a Research Staff Member with the System Software and Networking Department, leading an IBM Research group for appliance and infrastructure for next-generation wireless access networks. She is author or coauthor of seven technical papers in related areas. She is the holder of 17 patents. Ms. Lin was the Chair of the Technical Program Committee of the International Software Radio Technology Workshop in 2008. She is a member of the Association for Computing Machinery.