A SELF-ORGANIZING WIRELESS SENSOR NETWORK AND DISTRIBUTED COMPUTING ENGINE FOR COMMODITY AND FUTURE PALMTOP COMPUTERS

by

Haifeng Xu

B.S., Zhejiang University, 2009

M.S., University of Pittsburgh, 2012

Submitted to the Graduate Faculty of

the Swanson School of Engineering in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2015

UNIVERSITY OF PITTSBURGH

SWANSON SCHOOL OF ENGINEERING

This dissertation was presented

by

Haifeng Xu

It was defended on

June 25th 2015

and approved by

Alex K. Jones, PhD, Associate Professor, Department of Electrical and Computer Engineering

Hai Li, PhD, Assistant Professor, Department of Electrical and Computer Engineering

Yiran Chen, PhD, Associate Professor, Department of Electrical and Computer Engineering

Ervin Sejdic, PhD, Assistant Professor, Department of Electrical and Computer Engineering

Melissa M. Bilec, PhD, Associate Professor, Department of Civil Engineering

Dissertation Director: Alex K. Jones, PhD, Associate Professor, Department of Electrical and Computer Engineering

A SELF-ORGANIZING WIRELESS SENSOR NETWORK AND DISTRIBUTED COMPUTING ENGINE FOR COMMODITY AND FUTURE PALMTOP COMPUTERS

Haifeng Xu, PhD

University of Pittsburgh, 2015

The embedded class processors found in commodity palmtop computers continue to become increasingly capable while retaining an energy-efficient footprint. Palmtop computers themselves, including smartphones and tablets, provide a small form factor system integrating wireless communication and non-volatile storage with these energy-efficient processors. Also, various wireless connectivity functions on mobile devices provide new opportunities in designing more flexible, smarter wireless sensor networks (WSNs), and utilizing the computation power in a way we could never imagine before. In this dissertation, I present a WSN concept for current and future generation tablet devices. My contributions include developments at the system level, architecture level, and collaborative design between different layers of the system. At the system level, I developed Ocelot, a grid-like computing environment for palmtop computers in place of traditional workstation or server class machines to compute highly parallel light to medium-weight tasks in an energy efficient manner. Additionally, I developed Lynx, a self-organizing wireless sensor network, which is a further step taken in exploiting the potential of palmtop computers. At the architecture level, to increase the storage capacity of future palmtop computers, I explore the use of a new storage class magnetic memory, Racetrack Memory (RM), throughout the memory hierarchy. Thus, I developed FusedCache, a naturally inclusive, dual-level private cache design for RM that provides fast uniform access at one level, and non-uniform access at the next, which allows RM to be effective as close to the processor as an L1 cache. For higher levels of the memory

hierarchy such as the last level cache, I propose a Multilane Racetrack Cache (MRC), an RM last level cache design utilizing lightweight compression combined with independent shifting. MRCs allow cache lines mapped to the same Racetrack structure to be accessed in parallel when compressed, mitigating potential shifting stalls in an RM cache. Finally, leveraging the lightweight compression from MRC and the need for efficient communication in Lynx, I present a cross-level design combining memory-level lightweight compression with network-level packet transfer, together with a technique called Source-Aware Layout Reorganization (SALR) to increase the compressibility of sensor data.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

PREFACE

Earning a PhD is tough. Mine would have been a mission-impossible without help from some really important people. I would first thank my advisor, Dr. Alex Jones. Without the precious opportunity he provided to pursue my graduate study here at the University of Pittsburgh, I could never have gone so far geographically. I would never have reached so far in the field of advanced study and research without his relentless support, insightful instructions, and valuable advice. His attitude towards research steers me in the right direction for my PhD study and future work.

I want to thank Dr. Zhi-Hong Mao and Dr. Rami Melhem for their co-advising on my Master's thesis and my research work as a PhD candidate, respectively. Their outstanding experience in research and positive attitude towards work have always been inspiring and encouraging to me. I also thank Professor Hai Li, Professor Yiran Chen, Professor Ervin Sejdic and Professor Melissa Bilec for serving as my committee members and giving me valuable advice on my PhD proposal, defense and dissertation. I further thank my fellow researcher and good friend, Yong Li, for his help in my study and life in Pittsburgh. I am also very grateful to Dr. Michelle Jones for her help and advice in revising this dissertation.

Finally, I thank my wife and my parents for their invaluable care and support.

1.0 INTRODUCTION

1.1 MOTIVATIONS

Over the last decade, there has been a dramatic shift in the computing market away from stationary computers to portable palmtop computers. Fig. 1 shows the global market share of personal computing devices by operating system from 1975 to 2012 [1]. Since the release of the first generation of iPhone in 2007 and Android operating system in 2008, the aggregate market share of these two major palmtop platforms has dramatically climbed, reaching 60% by the end of 2012, while the market share of desktop and laptop computers has dropped below 40%. One thing that remains constant is the continuous availability of computation resources that can be leveraged to solve problems. The popular distributed computing platform, BOINC [2], has gathered enormous processing from personal computers to help researchers in multiple areas, including astrophysics, physics, biology, and climatology. Utilizing the increasingly large processing power of palmtop computers to do lightweight distributed computing tasks is a promising, and even necessary approach. However, such a computing system on mobile computers has significantly different trade-offs, particularly related to energy/battery life.

Palmtop computers usually have smaller form factors and less processing power compared to traditional computing devices. However, they also consume less power thanks to energy-optimized components such as reduced instruction set computing (RISC) based low power embedded processors [3], Lower Power DDR (LPRRD) RAM [4], non-spinning eMMC (flash) storage, and Bluetooth Low-Energy [5], etc. Researchers have found in [6] that each core on an Atom chip (palmtop class) could handle web search queries at half the rate of a Xeon chip (workstation/server class) core but required just 20% of the energy per request. A

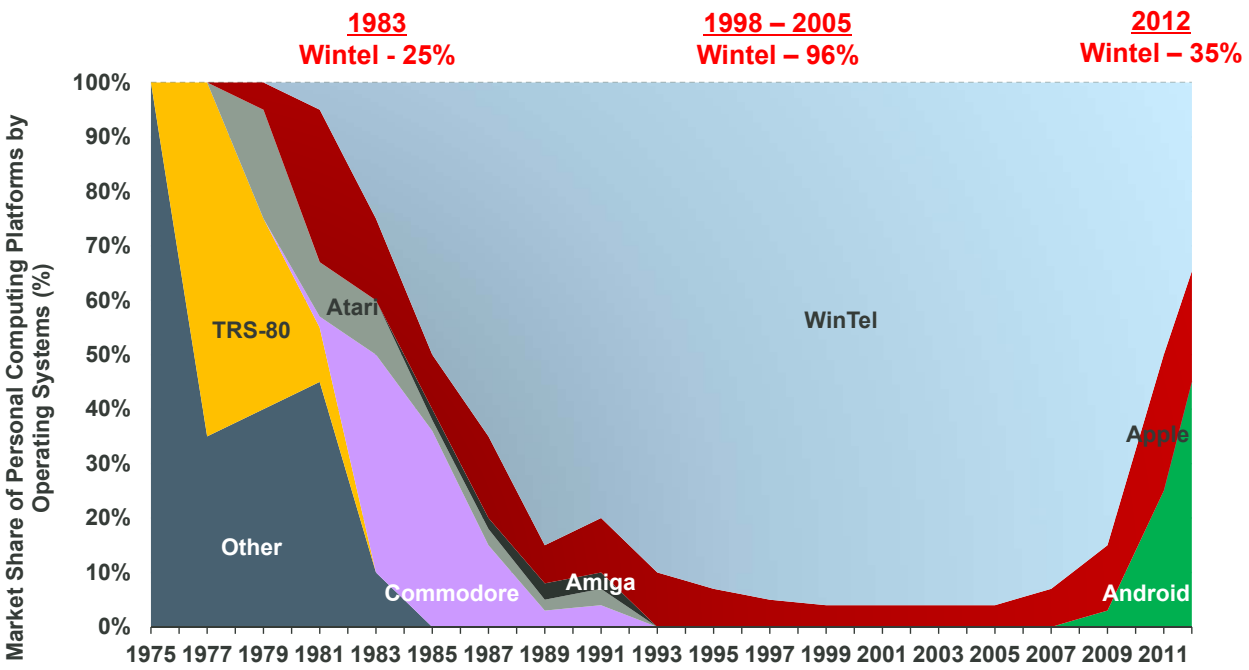


Figure 1: Global Market Share of Personal Computing Platforms by Operating System from 1975 to 2012.

better per-task energy efficiency is the main goal of this work, to be achieved by distributed computing on energy-efficient palmtop computing devices. However, the central server(s) providing services such as database storage, web query, and task scheduling, becomes the largest power component of the otherwise lightweight platform. Considerable benefit would be gained by conducting the distributed computing in a pure peer-to-peer (P2P) fashion without a server, which requires each computing node to be self-organizable in terms of data storage, data communication, and task management.

In addition to the processing capability of the palmtop devices, on-board storage is another limiter in processing and storing data. With the development of micro-electro-mechanical (MEMS) sensor technology, more on-board and external sensors can be integrated with the palmtop computers as sensor nodes to collect data at a finer granularity. However, issues such as higher power dissipation with CMOS technology scaling limit the growth of storage capacity of future palmtop computers. Emerging non-volatile memories (NVMs),

such as phase change memory (PCM), spin-transfer torque MRAM (STT-MRAM), and Racetrack Memory (RM), are promising in addressing these issues with their near-zero static power and much higher density than traditional static random-access memory (SRAM). Among those NVMs, RM has the highest density thanks to its unique physical structure. However, drawbacks such as high dynamic write power and potential domain shift operations upon each access make it less competitive both performance- and energy-wise. Lightweight hardware compression is one method shown to be effective in mitigating these drawbacks.

Aside from data storage, data transfer is another important topic in palmtop computing. Software compression is effective in reducing data transfer time but requires nonnegligible compression and decompression overhead. A lightweight mechanism to increase the efficiency (latency and energy) is needed to address the data transfer problem. Since hardware algorithms leverage low dynamic ranges existing in the consecutive data blocks, it may be possible to better organize the data structures to increase the compressibility, thus further reducing the transfer time.

1.2 PROBLEM STATEMENT

It is desirable to leverage the capabilities and ubiquity of commodity palmtop computing to build wireless sensor networks (WSNs) with on-board data processing capabilities. Fusion of a WSN node and a distributed computing client provides collaborative data collection and processing capability. Fusion of a distributed computing client and a server relaxes the restriction of a dedicated server for better energy efficiency and fits well in the context of P2P WSN. However, to accomplish these fusions effectively requires advancements at the system, memory architecture, and communication levels, as pressure on data storage as well as transmission becomes larger. The goal of this work is to create a WSN with decentralized distributed computing capability using palmtop devices, and explore the potential advancements in multiple levels of the palmtop system with emerging memory technologies and compression.

1.3 DISSERTATION OVERVIEW

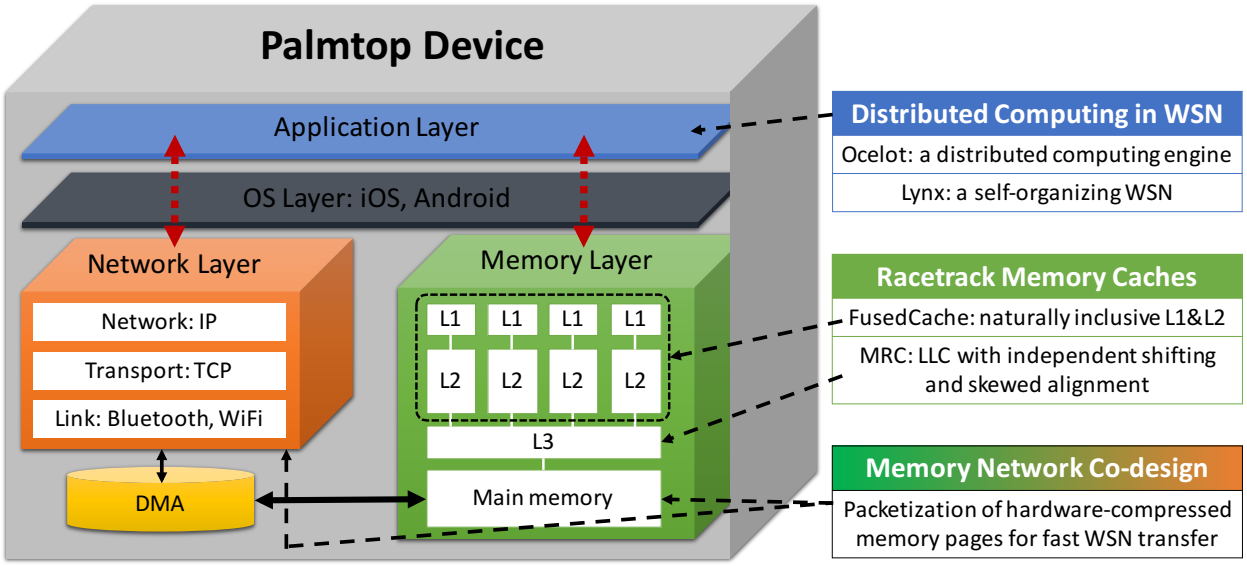


Figure 2: Overview of this dissertation work.

Fig. 2 illustrates different components of this dissertation, and relates these components to system components in existing and future palmtop computers. This work incorporates application level WSN platform implementations including Ocelot and Lynx for utilizing the processing power and rich connectivity of existing palmtop computers, architecture level cache designs including FusedCache and MRC for increased storage density with emerging Racetrack Memory, and cross-level memory network co-design for accelerated wireless transfers with seamless packetization of compressed memory pages. In the remainder of this section, I describe in detail how this work tackles the aforementioned challenges.

1.3.1 Application Level: Distributed Computing in a Wireless Sensor Network

First, the initial steps for developing a distributed mobile computing platform named *Ocelot*[1] are presented. Ocelot can be used in scenarios where highly parallel, lightweight computational tasks are required, saving the energy resources that would normally be required by

[1]An ocelot is a small wild cat related to leopards. The name was selected for this system because the relationship between an ocelot and a leopard is similar to that of an embedded processor in a tablet with workstation-class processors.

workstation or server machines. The Ocelot system working in such a scenario is shown in Fig. 3. Additionally, Ocelot can be used in an application specific WSN scenario in which it is difficult/impossible or undesirable to deploy a dedicated computational infrastructure. Conceptually, Ocelot is modeled after the Berkeley Open Infrastructure for Network Computing (BOINC) [2], an open source middle-ware system for volunteer and grid computing on PCs and servers. However, BOINC is complex, requiring a considerable amount of computational effort just to initiate and manage connections with the BOINC servers. On a workstation, this overhead is minimal, but on a mobile platform, this overhead becomes considerable. In contrast, Ocelot is a lightweight mechanism to deploy computation tasks on smartphones and tablets using the Android and iOS platforms. Though mobile devices come with embedded processing capabilities, their ubiquity can overcome this drawback through their flexibility in development and for lightweight tasks they can often provide much better power-efficiency.

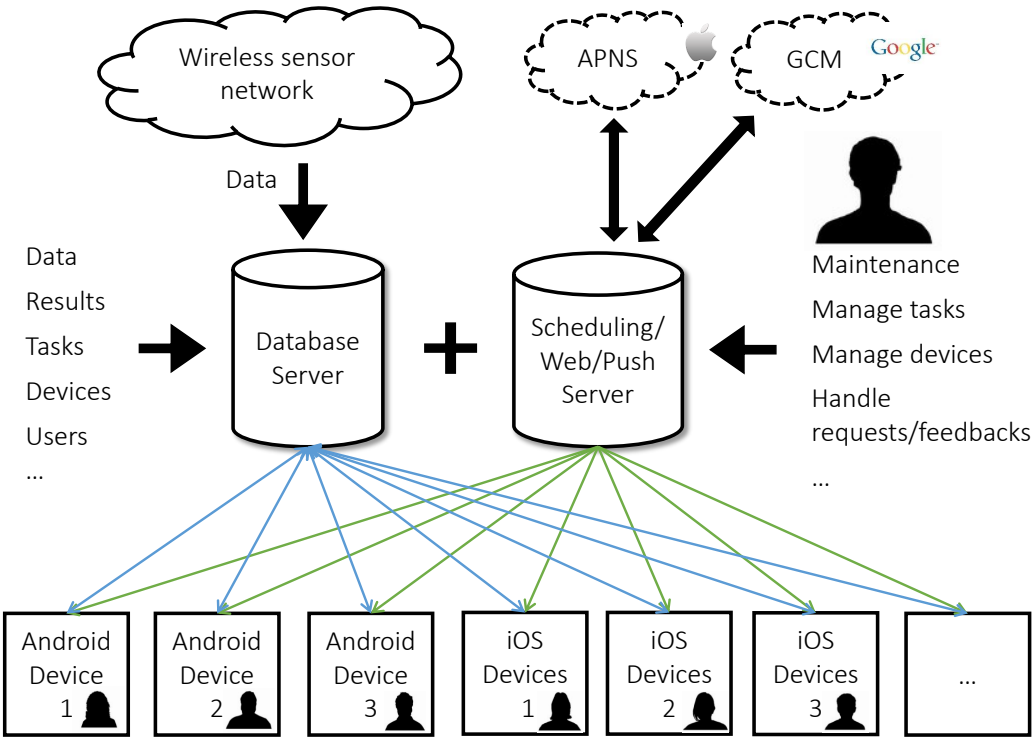


Figure 3: Overview of the Ocelot system.

Of course, security features built into a system provides a challenge to task distribution over an open network. Smartphone and tablet platforms such as Android and iOS restrict the operating system features required to implement a grid style computing platform (such as receiving external data and code to process for a workunit). These security mechanisms are important given the nearly global access capability to avoid an attacker from using these devices to spy on their users or to steal sensitive information. This work principally describes the mechanism and performance of such a system with only a simple method to handle secure operations (e.g., registering devices and physical security).

In particular, an Ocelot dashboard is used for interactive project management to evaluate usage workloads and to monitor activity. Using a sustainable building-based study, use cases for managing sensor data and computing life-cycle assessment algorithms are shown. Palmtop devices with Ocelot are compared to dedicated workstations to demonstrate the energy benefits of my approach. On average, computations on a palmtop device require 67% less energy than on a workstation.

I also introduce *Lynx*[2], a self-organizing wireless sensor network research environment based on commodity hardware/software systems. Given the proliferation of tablets, smartphones, and music players, there is a continually expanding platform of devices that can be used to form a WSN. The typical smart-phone or tablet contains multiple communication mechanisms such as Global System for Mobile communication (GSM) [7], Code Division Multiple Access (CDMA) [8], Near Field Communication (NFC) [9], Bluetooth [10] and WiFi [11]. Further, these devices often provide a significant number of on-board sensors including light sensing, temperature sensors, etc., as well as the capability to connect to various types of external sensors. These devices are battery operated, making them highly portable and mobile.

Fig. 4 provides an overview of how the Lynx system can construct a WSN from commodity tablet and palmtop computing platforms. A dedicated WSN node (fixed and preconfigured) consists of the palmtop computer running the Lynx client with one or more sensors monitoring certain physical or environmental conditions in an area. Sensing data can be stored locally and/or passed through wired (e.g. USB) or wireless connections (e.g.

[2]The name Lynx was also selected because of its similarities to ocelots and leopards.

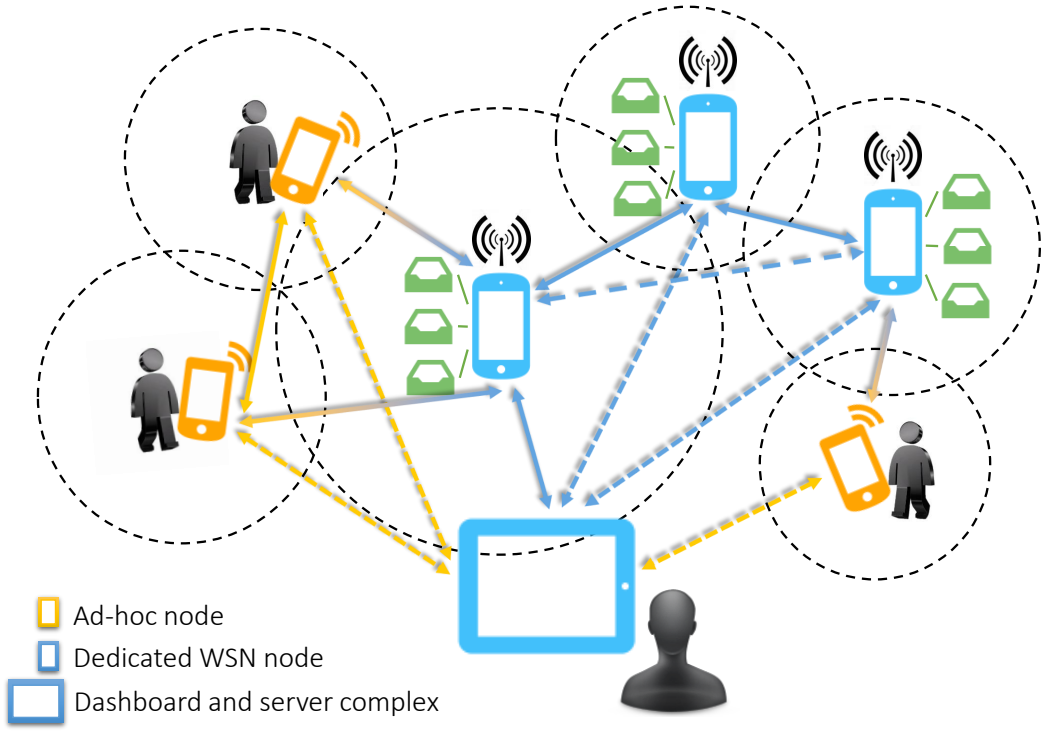


Figure 4: A Lynx wireless sensor network illustration.

WiFi or Bluetooth) on the Lynx node. Transient mobile devices (such as a smart-phone sitting in a pocket) can run the Lynx software and become an ad-hoc Lynx node. Though not required to gather sensor data, ad hoc nodes can serve as a bridge and help to relay data within their connection coverage and capacity. Ocelot and Lynx are then seamlessly integrated to provide the capability of distributed computing over self-organizing wireless sensor networks in a fully decentralized fashion.

1.3.2 Architecture Level: Cache Designs with Emerging Racetrack Memory

With the development of MEMS sensor technology, more on-board and external sensors can be integrated with palmtop computers as sensor hubs/nodes to collect data at a much finer granularity. Therefore, future palmtop computers may experience storage shortage at the current technology scaling speed. Emerging storage technologies must be leveraged to relax the storage restriction for future WSN applications. A modern consumer-grade hard disk

drive (HDD) can accommodate terabytes of data with a single magnetic read/write head. The mechanical movement latency, including seek and rotation, measures in milliseconds, making HDDs at the very bottom of the memory hierarchy of computer systems. Spintronic domain-wall "Racetrack" memory (RM), recently proposed and demonstrated by IBM [12], is a promising candidate to overcome density limitations while retaining the static energy benefits of STT-MRAM. RM integrates several magnetic "domains" into one nanowire or track. In other words, each track can store multiple bits of non-volatile data with one or more discrete access points similar to a magnetic tunnel junction (MTJ) for reads/writes to the aligned domain as shown in Fig. 5. By applying spin-polarized current pulses, data bits can be shifted along the track to be aligned with these access points. In this fashion, RM resembles traditional hard disks with disk rotations aligning magnetic data with read/write heads. However, unlike mechanical disks, shifting speed is typically similar or faster to data access, making RM attractive for its density, speed, and non-volatility at higher memory hierarchies such as main memory and on-chip caches. RM demonstrations of memory array structures [13] and content addressable memories (CAMs) [14] show fabrication feasibility, as well as great potential for density, performance, and power consumption, suggesting that we have reached the appropriate time for circuit- and architecture-level design exploration.

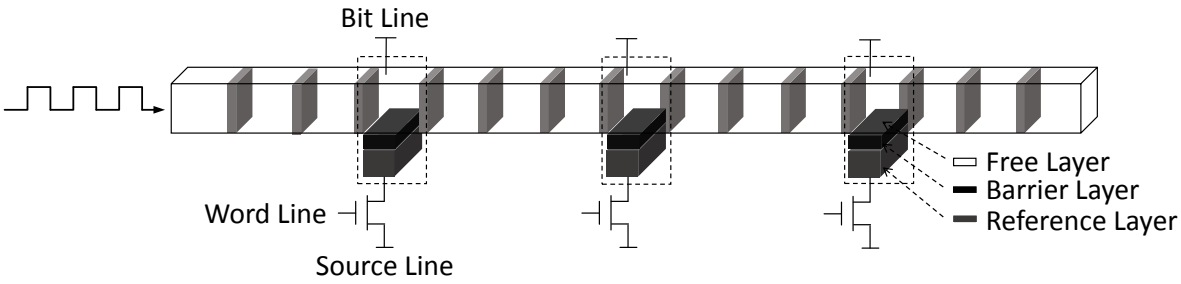


Figure 5: Planar Racetrack memory.

By leveraging shift-based writing [15], racetrack writes become competitive with SRAM, eliminating a drawback of STT-MRAM for caches near the core (e.g. L1), due to their relatively large incidence of unfiltered writes. However, for dedicated memory structures, such as private L1 caches where timing predictability is crucial to maintain pipeline integrity, non-uniform access behavior is problematic. Further, I note that accesses closer to the core should minimize access latency while father accesses can tolerate longer latency.

Therefore, I propose *FusedCache*, an RM cache that combines two logical levels of the cache hierarchy into a single physical storage array. Area-wise, an STT-MRAM cell can be considered equivalent to RM of length equal to one. By extending the length of the nanowire to hold more domains, the density of the storage is increased but the physical size of the array is not, as the CMOS access transistor still dominates a nanowire with ten or more domains. Moreover, the domain aligned to the access point has a deterministic access time (i.e., no shifting required). Thus, by replacing a STT-MRAM array with N-length RMs and enforcing the property that the domains aligned with the access point are the faster level cache (L1), the additional N-1 domains are treated as a next-level cache (L2) that tolerates longer non-uniform access latency due to shifting. Thus, I have *fused* together two levels of the memory hierarchy.

FusedCache has unique properties that make it valuable for use in the memory hierarchy as described. Beyond an obvious advantage that the cache retains a low-latency for L1 while removing the need for a dedicated separate L2 array thus saving area and power, the inclusion and eviction properties become naturally maintained in the structure. Consider a miss in L1. By shifting to access the data at an L2 location, the new data location becomes aligned with the access point and has automatically been promoted to L1, while the data formerly in L1 is automatically evicted to L2. Thus, the cache is *naturally* inclusive. Further, the tag array for both levels can be combined. In this L1/L2 case, the L2 tags are stored, but the L1 tags can be accessed by combining the L2 tag and the Racetrack shift position. To implement set-associativity in FusedCache, I present two designs. In the first design, the ways of each L1 set are matched to correspond to the same ways of the L2 set to which they belong through background swap operations. In the second design, I demonstrate a hybrid associativity organization where the L2 may have a larger number of ways than the L1.

Fig. 6 shows a 3D Racetrack schematic with a single read/write port. Additional bits add only to the height of the vertical structure. Thus, a much larger storage density can be achieved over a planar structure (Figure 5). Spin-polarized current pulses are applied at one end of the nanowire to shift data from destination domains to the one at the access port. Including more domains on a single nanowire increases density, but without additional access ports, inevitably resulting in longer average shift distance and correspondingly higher shift

latency and energy. Prior proposals for RM in caches [16, 17] focus on leveraging planar, multi-port-on-one-track (MP1T) Racetracks. However, planar structures considerably limit the density and static power advantages of RM over STT-MRAM. In contrast, a one-port-on-one-track (1P1T) orientation can leverage the 3D structure to dramatically increase the density per area of RM while reducing design-time and run-time complexities at the cost of a larger shifting overhead. In terms of energy, like most non-volatile memories (NVMs) such as PCM and STT-MRAM, RM is dominated by dynamic power with an asymmetric energy of writes being larger than reads. Unlike other NVMs, shifting energy tends to be a significant, even dominant, component of the dynamic power.

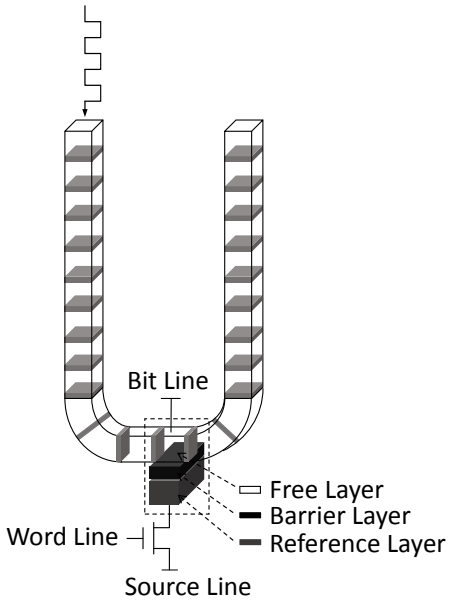


Figure 6: A schematic of Racetrack with three-dimensional structure.

To address the aforementioned issues, I propose *multilane Racetrack caches* (MRC), a simple, energy efficient, RM-based last level cache (LLC) design using *in-place compression* and *independent shifting* to reduce data access and shifting overheads. In MRC, lightweight compression is used to reduce the cache line size in order to reduce the number of bits stored in the cache block. Further, by allowing independent shifting at a finer granularity than a line (e.g., byte-level), fewer tracks may require shifting to access compressed cache lines. Finally, using *skewed alignment* or adjusting the starting location of compressed lines within their cache blocks, contention for port resources may be alleviated by treating independent

shifting as *multiple Racetrack lanes* that allow concurrent data access. Energy consumed by reads, writes, and shifts can be reduced through all these methods and the reduced contention can also lead to performance benefits. While MRC can be applied effectively to both MP1T and 1P1T structures, compared to an isocapacity baseline 1P1T RM cache, MRC achieves 5% performance gain and 19% energy reduction.

1.3.3 Cross-Level: A Memory Network Co-design with Compression

Compression turns out to be an effective and sometimes necessary approach to improve the usability and efficiency of both high level WSN applications and low level cache designs. I demonstrate in Lynx that software-based compression, although it requires extra time to compress and decompress, is able to significantly reduce data transfer time over the network. I also demonstrate that hardware-based lightweight compression is effective in saving dynamic write power and shifting energy of emerging Racetrack Memory, with minimal latency and energy overhead.

To leverage lightweight compression for reducing data communication latency, I propose a memory network co-design approach that compresses memory pages using low-latency hardware compression, and uses this compressed form to packetize payloads for communication over the wireless network links. While traditional software-based compression can outperform lightweight compression in terms of compression ratio, it requires a considerable runtime overhead to compress the transmitted data. And although traditional lightweight compression tends to work poorly on the traditional storage structure of sensor data across multiple sensors, I propose to better utilize the low dynamic range existing in most data tied to a single sensor. Thus, data storage layout is reorganized according to sensor source to improve the data compressibility, especially for lightweight compression but also benefiting traditional software compression approaches.

This approach is supported by the observation that sensor data from the same source tend to have a low dynamic range. Fig. 7 shows the five number summary[3] of NOAA 1981-

[3]A five number summary consists of the minimum, first quartile, median, third quartile, and maximum of the dataset.

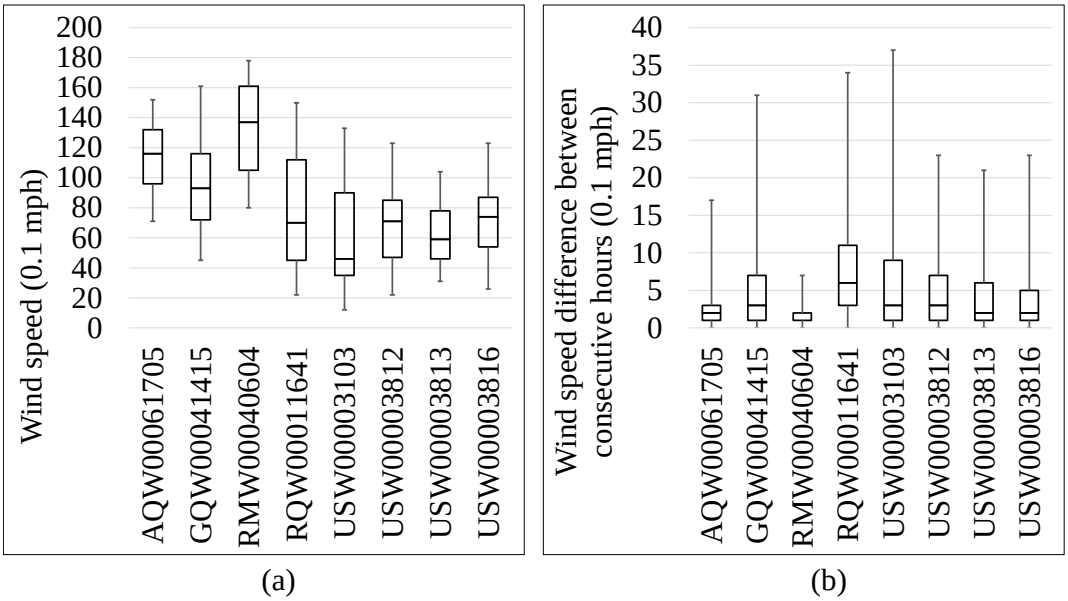


Figure 7: Five number summary of NOAA 1981-2010 wind speed normals of select eight U.S. stations: (a) hourly wind speed; (b) wind speed difference (absolute value) between consecutive hours.

2010 hourly wind speed normals[4] for eight arbitrarily selected U.S. stations [18]. The wind speeds at each station tend to vary within a reasonably small range, especially for consecutive timestamps. Similar patterns are observed in other types of data as well, including utility consumption, climatic, financial, and potentially many other datasets. This consistency provides an opportunity to optimize how this data is handled within WSNs with the goal of improving their performance and efficiency.

1.4 CONTRIBUTIONS

The adoption of software and hardware compression techniques provides an opportunity to make contributions to the areas of both high level WSN systems and low level cache/memory architecture.

[4] A "normal" of a particular variable is defined as the 30-year average.

To create a WSN system with commodity mobile hardware, I make the following contributions:

- **Ocelot:** I provide an energy-efficient distributed computing platform with commodity palmtop computers. Ocelot is the first attempt to utilize the growing processing power of mobile devices, and is a lightweight implementation of an energy-efficient distributed system, comprising clients, web/database servers, and assistive dashboards.
- **Lynx:** I simplify the creation of a WSN by leveraging existing infrastructure. Using Lynx, it is possible to quickly construct a sensor network from existing devices, many of which are already deployed by users for their more traditional purpose (e.g., smartphone, tablet function). Lynx can provide access to sensor data from on-board sensors and external sensors.
- **Integration:** I provide a smart, fully-distributed network for mobile computing. Lynx is designed as a platform for Ocelot. Integration of these two platforms creates a fully distributed computing engine without any central servers or infrastructure, as all the network communication utilizes peer-to-peer (P2P) connections. Moreover, by removing dedicated central servers in the Ocelot system, the overall energy consumption is further reduced.

To advance the storage capacity of palmtop systems, I make the following architecture contributions:

- **FusedCache:** I demonstrate how a pseudo-sequential access structure (e.g., a Racetrack magnetic nanowire) can naturally integrate multiple memory storage levels in a single, highly dense storage array using the idea of *storage hierarchy driven* shifting, and describe a particular instantiation of storage hierarchy driven caching for various set-associativity parameters ranging from direct-mapped to hybrid set-associativity with different numbers of ways at each level, and support the value of the fused L1/L2 cache through a detailed power and energy evaluation, including sensitivity studies of different cache sizes and associativities, while also considering both iso-capacity and iso-area SRAM cache replacements.

- **MRC:** I adopt lightweight in-place compression to reduce the number of bits stored, thus saving read and write energy of RM caches. Based on compression, I propose independent shifting to control Racetrack movements at a finer granularity than the cache line, to further save shift energy of compressed data in RM caches. Based on independent shifting, I propose skewed alignment to accelerate accesses to compressed cache lines in RM caches with independent shifting.

In the memory network co-design work, I make the following contributions:

- I propose a WSN system design that directly uses hardware/lightweight compression to compress memory pages in-place in order to natively compress data packets and avoid the need for software compression for improved efficiency in wireless communication.
- I propose the Source-Aware Layout Reorganization (SALR) approach to significantly increase compression ratios of sensor data for hardware/lightweight-based compression approaches, and integrate it with data communication using in-place hardware/lightweight compression and evaluate it using various types of data comparing the combined technique to using on-demand software-based compression for communication.

1.5 DISSERTATION STRUCTURE

The rest of this dissertation is organized as follows. Chapter 2 provides a comprehensive list of prior work in related areas such as distributed computing, wireless sensor networks, data compression in both memory and networks, as well as emerging Racetrack Memory. Chapter 3 describes the features and implementation details of the Ocelot project, and a dynamic life cycle assessment case study to evaluate its feasibility and effectiveness. The Lynx project is discussed in Chapter 4, and evaluated using several approaches, including basic data transfer, scalability tests, tests as a hardware emulation platform by swapping the routing algorithm modules, and integration tests with Ocelot. Chapter 5 discusses Fused-Cache, a lower level cache design to ensure uniform accesses to the latency-critical L1 cache given RM's non-uniform access pattern. In Chapter 6, I explain how I extend the bene-

fits of lightweight compression from saving energy to improving performance with emerging Racetrack Memory, which is a promising candidate as a high density memory technology on future palmtop computer. I explain in Chapter 7 a memory network co-design for improving the efficiency of wireless transfer with lightweight compression, as well as a data layout reorganization technique specially designed to improve the compressibility of sensor data from multiple sources. Finally, I conclude in Chapter 8.

2.0 PRIOR ART

2.1 BOINC AND DISTRIBUTED COMPUTING

The Berkeley Open Infrastructure for Network Computing (BOINC) is a middle-ware system connecting central servers and numerous distributed client computers for public-resource computing and storage. It was originally developed for the *SETI@home* [19] project before it became useful as a platform for other distributed applications in areas as diverse as mathematics, medicine, molecular biology, climatology, and astrophysics. BOINC makes it possible to accumulate the enormous processing power of the public volunteered personal computers (PCs) all around the world. For example, the pioneer project, *SETI@home*, performs digital signal processing of radio telescope data from the Arecibo radio observatory to help the *Search for Extraterrestrial Intelligence* (SETI). The *Climateprediction.net* [20] project tries to quantify and reduce the uncertainties in long-term climate prediction, based on large numbers of computer simulations with varying forcing scenarios and internal model parameters. PC owners participate in one or more of the BOINC projects as resource volunteers, and they can specify how their resources are allocated among these projects.

BOINC is designed to support applications that have either large computation requirements or storage requirements, or both. The main equirement of the application is that it be divisible into a large number (thousands or millions) of jobs that can be done independently. Thus, it is essentially a simple form of distributed computing platform without communications among the individual computers.

Ocelot uses principles similar to BOINC, except that it focuses on moving lightweight tasks from less power efficient workstations onto ubiquitous palmtop computers. The number of mobile phones per capita has reached nearly one per inhabitant worldwide [21], and

smartphone penetration has reached over 50% in developed regions like EU5 [22] and U.S [23]. This resource can be utilized to reduce our need for workstation or server computers and reduce energy per computation.

2.2 WIRELESS SENSOR NETWORKS AND MOBILE CONNECTIVITY

A WSN consists of spatially distributed nodes with capabilities such as sensing, wireless communication, or even lightweight data processing. Usage scenarios range from real-time tracking, to monitoring of environmental conditions, to ubiquitous computing environments, to monitoring of the health of structures or equipment. Nodes within the network can often self-organize, cooperatively passing their data through predefined protocols and providing paths to a center collector/storage. Often they also provide a distributed computational resource and local storage for sensor information. As such, WSNs have become an area of significant interest and research activity including a wide variety of academic and commercial research groups, product lines, etc.

The purpose of traditional WSN is to collect various kinds of sensor data, including temperature, pressure, and sound data at different locations (e.g., different labs in an engineering school building) through the interconnected network. Additionally, the processing power of modern mobile electronics has become powerful enough to take on sizable tasks. With large numbers of these devices connected, computationally complex algorithms such as signal and image processing tasks on sensor data can be efficiently completed by partitioning tasks onto independent mobile processors assigned from the mobile device pool.

A particular area of recent interest for WSNs is the tracking of environmental information and resource utilization by sensing local temperature or light levels or sensing energy usage. This interest has been accelerated by the increasing availability of palmtop computing, in particular, smartphone and tablet devices that are highly connected using various wireless protocols such as WiFi, Bluetooth, NFC, and optical communications such as infrared (IR). With the rapid development of *System-on-Chip* (SoC) technologies, these embedded systems continue to become more powerful computational engines while still remaining energy effi-

cient. As a result, in-network data processing and low-to-medium effort computation has become possible in a WSN created with commodity devices as nodes. Just as in WSNs, performance is tempered with energy resources (e.g., battery life) as metrics to be considered. However, by deploying work wisely in such a system it may be possible to avoid the need to utilize heavy iron servers in many environments to process the sampled data in the system. Additionally, the global availability of these devices may create a network of computational resources that can be used much like grid computing has done in the past with idle personal computers or workstations.

2.2.1 Physical Layer Communication

With the increasing popularity of mobile electronic devices including smartphones and tablet computers, connectivity technologies and computation capability are rapidly evolving. Commodity wireless protocols such as WiFi and Bluetooth are becoming more competitive with protocols typically designed to work with WSNs such as Zigbee [24]. For example, the development of WiFi 802.11n, with its multiple-input and multiple-output (MIMO) antennas, has led to support for new protocols such as *WiFi-Direct* [25] in tablets and smartphones. WiFi-Direct allows a device to find nearby WiFi-Direct capable devices and form a group to communicate over a peer-to-peer (P2P) link without wireless access points (base stations) in the infrastructure mode. This allows real world implementations of ad-hoc routing to be feasible and robust with commodity devices [26]. Bluetooth Low Energy (BLE) also shows good capability for short range data transmission. It represents a trade-off between energy consumption, latency, piconet size, and throughput [27].

2.2.2 System-level Management and Optimization

Most hardware oriented studies in the WSN field focus on infrastructure [28, 29] or routing improvements [30, 31, 32, 33] in an attempt to improve network coverage area [34, 35] or reduce energy consumption for communication [36, 30, 37, 38, 39, 40, 41]. There is also a significant amount of theoretical work on improving routing algorithms and developing more efficient low-level communication protocols. Further, some efforts explore methods for

distributing data across the network to maximize data availability when link connectivity is transient and unreliable, while minimizing other metrics such as system storage and energy overheads. The goal of such research include maximizing network lifespan under a fixed aggregate system energy storage capacity, maximizing system robustness, and to determine best practices for network self-organization and configuration [42, 43, 39, 44, 45].

2.2.3 Self-Organizing Network

The Self-Organizing Network (SON) was originally proposed for the Long Term Evolution (LTE) mobile technology [46]. SON configures and optimizes the increasingly large and complex mobile networks automatically, so that manual interactions can be reduced and the efficiency of the network can be increased. A SON generally features functionalities such as self-configuration, self-optimization, and self-healing [47]. Adding or reducing network nodes should be configured and managed with the least possible human intervention, and communication workloads should also be dynamically balanced for better efficiency. When failure happens in any of the network nodes, the network structure should be reconfigured to reduce negative impacts.

The initial focus of SON development was to reduce operating costs in mobile radio networks [47], but the concept is now increasingly recognized as a method for any network to intelligently organize and manage its nodes at scale. The challenges of organizing a peer-to-peer network on small wireless sensors such as limited power, limited radio range, and potentially high geographical volatility, and all point to an adaptive and efficient self-organizing solution. For example, implementing the *Automatic Neighbor Relation* (ANR) detection feature in a SON helps to distributively establish the network map of a WSN. Nodes within the radio range of each other can communicate directly; nodes out of range are still able to communicate with the help of one or more "bridge" nodes, which sit in between.

2.3 ARCHITECTING WITH RACETRACK MEMORY

As a descendant of STT-MRAM, RM comprises an array of magnetic nanowires arranged vertically or horizontally on a silicon chip. A nanowire consists of many magnetic domains separated by *domain walls* (DWs). Each domain has its own magnetization direction. Similar to STT-MRAM, binary values can be represented by the magnetization direction of each domain. For a horizontally-arranged planar strip (Figure 5), several domains share one access point for read and write operations [48]. The DW motion is controlled by applying a short current pulse on the head or tail of the strip in order to align different domains with the access point. Since the storage elements and access devices in RM do not have a one-to-one correspondence, a random access requires two steps to complete: *Step 1–shift* the target magnetic domain and *align* it to an access transistor; *Step 2*–apply an appropriate voltage/current to *read* or *write* the target bit. Intrinsically, the operations in step 2 are the same as that used for accessing STT-MRAM. The DW motion, however, requires extra domains at both sides of a magnetic nanowire to prevent data loss.

RM, the apparent *third-generation* magnetic technology, is being widely investigated. Significant progress has been made in device physics and process development of Racetrack memory [49, 50, 51]: For example, demonstrations in single devices and array structures were recently presented by IBM [13], NEC [52, 53], and CNRS [54], giving evidence of its process feasibility and great potential. Recent device-level research focuses on improving shifting speed and the introduction of efficient pinning mechanisms [55, 56]. For example, the calibration of torques adjusts parameters of spin-orbit torques to improve the efficiency and controllability of DW shifting [57, 58, 59, 60]. Various forms of storage applications based on RM have been demonstrated, such as array integration [13], lower level cache [61], content addressable memory (CAM) design and fabrication [14, 62], reconfigurable computing memory [63], and a shift register realized by perpendicular magnetic anisotropy (PMA) technology [48, 64]. Iyengar et al. [65] analyzed the influence of process variation and Joule heating on DW dynamic performance for embedded memory. These efforts demonstrate fabrication feasibility; show great potential for density, performance, and power consumption; and suggest the appropriate time for circuit- and architecture-level design exploration.

The extreme high density and power efficiency of RM inspires a new approach for building on-chip memory components across a diverse spectrum of computational platforms. On-chip memory components, e.g., caches, network buffers and scratchpads, usually consume a large area/power budget of the whole computing unit [66, 67, 68]. Replacing traditional memory technologies with RM can save considerable area and power while promising further scalability for energy-efficient computing [61, 69]. However, on-chip memory components built with RM may sacrifice performance due to a prolonged access delay [70]. For example, minimizing the number of access ports per nanowire can maximize the density and minimize the leakage power; unfortunately, since each RM cell has to be shifted to an access port prior to a read/write, fewer access ports means longer shift distances, resulting in longer access latency.

Most existing research focuses on architectural optimizations to address the negative impact of shifting on system performance with planar, MP1T Racetracks. It is demonstrated that the access latency of RM caches can be impacted by shifting policies and their relationship to access locality [61]. In the CPU-domain, RM was proposed for on-chip caches [61, 69, 15, 71, 72]. Venkatesan et al. [16] explore different head selection and update policies with both read/write ports and read-only ports in an RM cache design. Sun et al. [17] swap heavily accessed cache blocks closer to the read/write ports, which distribute across the tracks at a fixed gap. In the GPU-domain, Mao at el. [70] reschedule GPU warp issue order based on the distances between the current data location and the access ports to reduce shifting. A Racetrack-based cache hierarchy was also considered for GPUs, in which different cache layers employ different RM cell designs [15], and cell pre-shifting and data migration were used to reduce the shift overhead [73]. Adding more access ports per track (MP1T) can reduce average shift distance but also compromises the potential storage density, which is a significant advantage and motivation of using RM compared to the similar STT-MRAM.

2.4 EFFICIENT DATA TRANSFER IN WSN

2.4.1 Data Compression

The popularity and demand for lightweight, battery powered computing continues to increase. This is particularly true in the context of portable, palmtop computing devices, which continue to see an ever-increasing demand for digital content from streaming music and video to various types of data accessed from the cloud. This demand continues to push the storage capacity and network bandwidth of these devices. As such, integrated data compression has become an enabling factor of the success of this class of devices. Moreover, these advancements can be applied in the context of WSNs.

Mathematically, compression and decompression are processes of data encoding and decoding, respectively. These coding specifications can be categorized as lossy or lossless, depending on whether information is lost or not during the process. General-purpose, lossless compression algorithms such as GZIP [74] can be applied on any file. Lossy coding specifications usually target special-purpose data in which a certain degree of information loss is tolerable, such as JPEG standards [75] for image compression, MPEG2 layer III (mp3) for audio compression [76], and the H.264 standard [77] for video compression. The level of compression overhead ranges in these examples from moderate to extreme. Software-based compression approaches usually achieve good compression ratios but are often domain specific and their overhead may not be tolerable in a mobile application.

In contrast, several low-overhead approaches have been recently proposed to take advantage of partial compression with the benefit of a significantly reduced overhead. Lightweight compression takes advantage of data containing a high frequency of certain simple patterns, which occurs in many applications [78, 79]. Pekhimenko et al. also observed that among sub-blocks within a cache line or a memory page, values have low dynamic ranges. Thus, they proposed base-delta-immediate (BDI) compression, which leverages this low dynamic range to compress data [80]. The BDI approach is described in more detail in Section 7.1.1. The same group developed an approach called linear compressed pages (LCP), which uses BDI compression to increase capacity in the memory system [81]. BDI/LCP has the potential for

performance or energy efficiency improvement as fewer accesses are redirected to the next level of the memory hierarchy, which avoids cases of much higher access latency (e.g., cache misses, page faults, etc.). However, block concatenation, addressing, and realignment caused by value changes can lead to considerable memory management overheads in techniques like LCP. Instead of saving space to reduce page fault rate, Memzip [82] and SOCO [83] compress memory pages *in place* using lightweight compression algorithms. These approaches attempt to address other memory system features such as memory bus contention, improved memory reliability, and reduced energy consumption from memory accesses.

2.4.2 Compression in WSN

WSN nodes usually have smaller form factors and less processing power compared to traditional computing devices. However, with the development of embedded processor technologies and wireless connectivities, the Ocelot platform makes it feasible to deploy mobile devices as nodes or sensor hubs in WSNs [84], given their rich on-board P2P connectivities such as Bluetooth and WiFi-Direct. Due to the limited bandwidth of wireless P2P connections, it is desirable to compress sensor data before sending it to the network, and due to the limited processing power of WSN nodes, low-complexity compression algorithms are generally used.

A significant element of the effort in minimizing data transfer in WSNs considers both data availability when nodes and links can be unreliable as well as minimization of data redundancy in the system. One approach is "distributed" compression [85], which uses complicated *coset* codes to explore spatial correlation in dense networks, where Hamming distances of data from different sources tend to be small. In-network compression, or data aggregation in multi-hop routing scenarios, such as [86], uses a technique derived from signal processing, named *compressed sensing*, to exploit the sparsity of sensory data for compression, and tries to reduce unnecessary traffic near the sink node[1]. Based on MapReduce [87], a prevailing programming model in the distributed computing field, WSN researchers implement a practical service platform preprocessing sensor data by partitioning and interleaving data sets, as transfer costs grow significantly when the network scale grows [88].

[1] A sink node has only incoming traffic, and is responsible for data collection, aggregation, and storage.

In my work, instead of focusing on WSN optimization based on the organization of the network and data duplication in the network to ensure availability, I collaboratively improve the system by leveraging properties of the sensor data itself. As has been observed by [89], data recorded from a single physical phenomenon tends to demonstrate a temporal correlation between each consecutive observation of a sensor node. In Section 7.1 I describe my proposal to leverage this to effectively utilize lightweight compression in the WSN.

3.0 OCELOT: A WIRELESS SENSOR NETWORK AND COMPUTING ENGINE WITH COMMODITY PALMTOP COMPUTERS

3.1 IMPLEMENTATION OVERVIEW

By developing Ocelot, I extend the idea of BOINC's distributed computing onto commodity palmtop devices. There are some clear similarities between a mobile device and a computer. Above all, they both have network capability and can easily access the Internet if available. Both have the capabilities of storing, processing and calculating data. However, palmtop devices are equipped with less powerful processors that require significantly fewer transistors than processors typically found in workstation or server class computers. The benefits of leveraging these embedded processors are reduced cost, less heat, and lower power usage. These are all traits desirable for use in light, portable, battery-powered devices. Ocelot explores and utilizes processing power within those palmtop devices while balancing the battery usage in the aggregate according to fixed rules. I show in Section 3.7 that by partitioning tasks and assigning them to devices in parallel, significant speedups can be obtained while maintaining a lower energy consumption than traditional computers. As these palmtop devices continue to grow in popularity, this class of compute engine will become an increasingly large resource that can be utilized through technology advancements and sheer number of nodes.

3.2 OCELOT BASICS

Ocelot can function in an infrastructure mode assuming access to an external network, or it can form a *self-organizing* local network infrastructure. Using WiFi-Direct and Bluetooth, Ocelot employs a basic mechanism that provides the facility to discover peers, locally maintains peer lists, discovers routes through recursive queries to neighbors, and sends messages or files. Dynamic and decentralized routing algorithms such as *Distance Vector* can be implemented on top of the basic Ocelot network infrastructure providing facilities for managing link costs, connectivity, and local storage. Simple mechanisms for load balancing and heterogeneous processing capacities are currently implemented and more sophisticated ones [90] can also be easily explored.

Similar to BOINC, each project is associated with a *master URL*, which also works as the home page address of the project website. Thus a script or file address under the website directory containing the master URL string as its prefix. A data set is referred to as a *workunit*, and a compute function an *application*. A workunit is the smallest unit of input/result data into which the computational task can be divided. *Extensible Markup Language* (XML) was used as the data wrapper format in Ocelot. XML's generality in representing arbitrary data structures helps the server, the client, and a display/dashboard to generate, communicate and identify data information in a well-organized way.

Input and result workunit examples are shown in Listing 3.1 and 3.2, respectively. An input/result XML file may contain more than one workunit and the size is decided according to the computation effort when generating the *task*. A task is the smallest unit of computation work which consists of a unique task id, creation time, project, application, application version, input file download address, result file upload address, current status, and last status update time. The upload process not only stores the result file on the server, but also parses its contents and records them in the database table. After a task is finished by one Ocelot client, the result file address is also added to that information entry together with the account name of the contributor, usually the client device's owner. Ocelot provides a simple version of credits for completing a number of tasks.

An Ocelot project can consist of one or more applications, which abstractly are independent unique computational tasks required within the project. These applications consist of segments of executable code that can be executed by a client on a dataset from the server that requires processing by that application. The code is usually stored in different forms for different platforms, and can either be integrated in the client program or later downloaded to the client. Details are explained in Section 3.4. Tailoring the application code is the most important part of efficiently using Ocelot. Ocelot nodes in many cases can, execute interpreted (e.g., Java) or compiled (e.g., C) codes, and can often utilize multi-threading, and potentially graphics-processing acceleration, but require tight memory management for efficient operation.

```
<work_unit>
  <electricity_usage>
    2012-02-03 00:00:00,8.4,5.5,8.4,4.9,1.2,1.0,
    5.3,12.7,4.7,0.0,0.0,0.0,52.0,0
  </electricity_usage>
  <grid_mix>
     0.79,0.00,0.01,0.22,0.00,0.00
  </grid_mix>
</work_unit>
```

Listing 3.1: An input workunit example

```
<work_unit>
<timestamp>2012-02-03 00:00:00</timestamp>
  <EI_MCSI_LDP1>
    7.287619 3.297980 0.005776 4.585421 0.012276 0.000918 0.000000 0.096997 0.021904
  </EI_MCSI_LDP1>
  <EI_MCSI_MLEE1>
    4.771655 2.159392 0.003782 3.002359 0.008038 0.000601 0.000000 0.063510 0.014342
  </EI_MCSI_MLEE1>
                          .
                          .
                          .
  <EI_MCSI_TOTAL>
    45.113823 20.416071 0.035758 28.385937 0.075994 0.005680 0.000000 0.600457 0.135594
  </EI_MCSI_TOTAL>
</work_unit>
```

Listing 3.2: A result workunit example

By separating applications with data sets, project owners gain more flexibility in adjusting task loads and assigning different applications at the same time. Potential optimization

or additional functions can also be added to the application by simply releasing a new version instead of modifying the whole task.

3.3 OCELOT SERVERS

As shown in Fig. 3, Ocelot servers store data, schedule computation tasks, communicate with Ocelot clients, and provide interactive functions to both end users and system administrators. Currently, the Ocelot server runs on a computing workstation. The Ocelot Dashboard provides a tablet-based interface to the Ocelot server. A physical server can host more than one Ocelot project as long as it can be associated with multiple master URLs and has enough bandwidth and processing power.

3.3.1 Database Server

The database server is a core component of an Ocelot project. All project related information, including raw data and results, is stored in the database. Additional scheduling information, including task descriptions, accounts, and registered devices, is also stored in the database. Ocelot server application programming interfaces (APIs) for Ocelot Client and Dashboard have limited access privileges in order to ensure data safety.

3.3.2 Web Server

The web server provides a simple mechanism to access the data and manage the project. It provides additional facilities for presenting project data externally, a test gateway for developers, and even conducting surveys. File download and upload services are also handled by the web server. A dedicated script is used to parse uploaded result data and record it in the database. Therefore, the result is kept in both the form of database entries and physical files, satisfying various possible needs of post production. To limit the resource usage, the web server component is removed from Ocelot on the Lynx network.

3.3.3 Scheduling Server

The scheduling server handles task requests from clients and dashboards, generates new tasks from unattended data, and distributes tasks according to the device status. Details about the device status are explained in Section 3.4.

3.3.4 Push Server

The push server utilizes push notification APIs - APNS [91] from Apple and GCM [92] from Google - to communicate with iOS and Android devices, respectively. The push service in Ocelot makes bidirectional communication possible and efficient by avoiding the need for the clients to poll the server for work at particular time intervals. Instead, task assignment or other remote operations on clients can happen immediately by pushing pre-defined messages, reducing the consumption of battery life in clients. A private push module using P2P links instead of commercial servers is developed for the integration of Ocelot with Lynx.

3.3.5 Additional Servers

In some cases, additional server functionality can be integrated with Ocelot. For example, a private mail server is used in the case study as a mechanism to extract data from some external commercial sensors who send data through emails on a daily basis. Thus, in this manner, the mailed data is still available to be processed with mobile sensor nodes on the Ocelot system. Additional servers are not implemented in Ocelot with Lynx.

I provide a brief overview of the main functions in Ocelot:

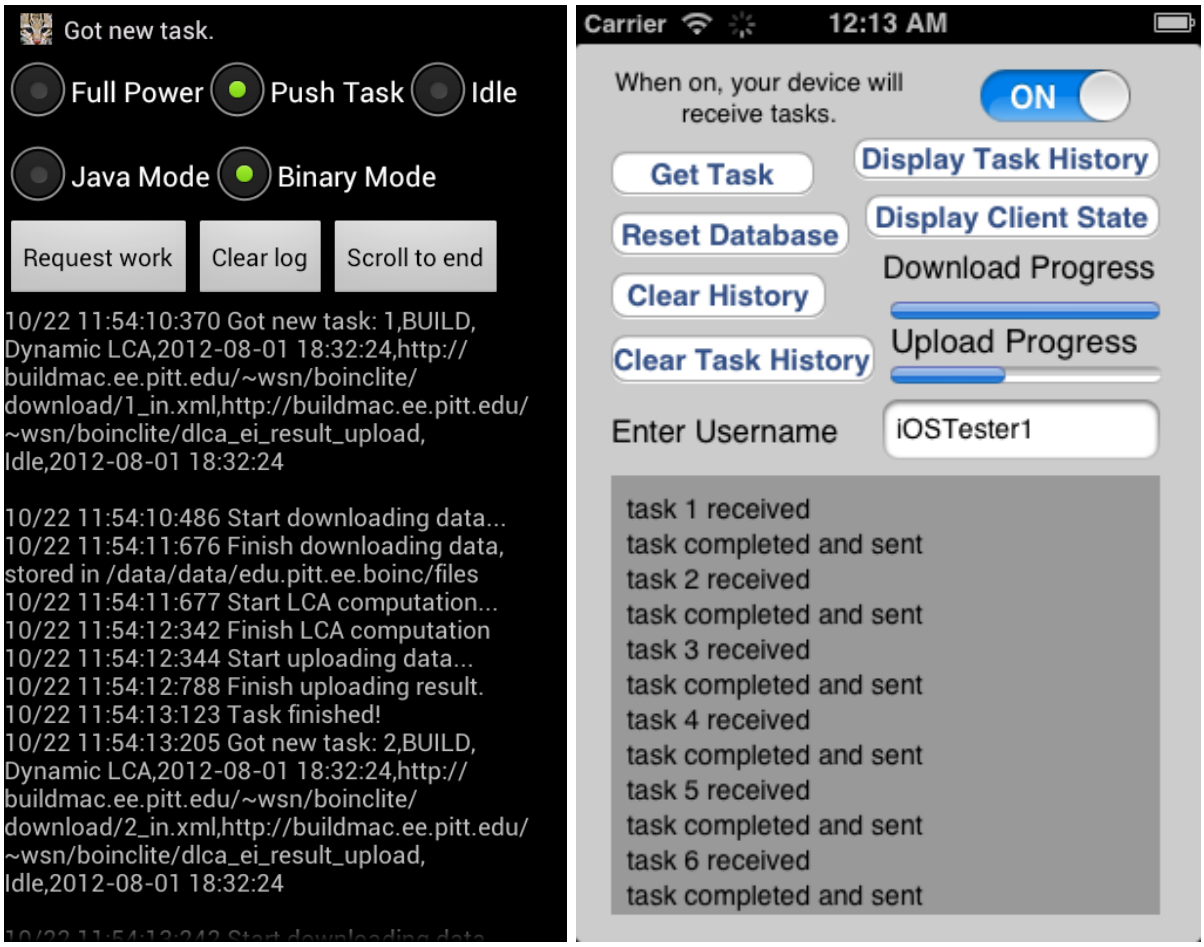
- `register` registers a mobile client node as available for use in the system based on the client policy selected by the user. Even in the "idle" mode, this registered device can be used to communicate traffic between nodes for greater system connectivity.
- `unregister` removes the client from operation in the system. Systems that become idle for long periods of time (due to going out of range, or exhausted batteries) time out and the system automatically unregisters them, requiring them to re-register when they become available.

- `battStat` is a function that reports the remaining battery life in the client device, and whether it is charging.
- `dataLookup` is used to look up raw data, pre-processed input data, and computed results according to specified criteria such as date, whether or when it is computed, etc.
- `deviceLookup` lists information about devices previously registered with the Ocelot server (even when currently not registered). Each entry includes: a unique device id, the platform, owner's account, current registration status, power connection status, remaining battery, and total battery capacity. Complete and regularly updated device specs (battery capacities in particular) are necessary in order to calculate aggregate battery and computing power available.
- `taskLookup` lists information of both active and finished tasks. One active task entry includes: a unique task id, associated project and application, creation time, input download URL, result submission URL, last update time, status (whether it is computed or being computed). Each finished task entry includes additional information such as computation start and finish time, contributing device id and owner's account, result file location, etc.
- `fileDownload` downloads application code or input data files from the server.
- `resultUpload` submits computed results to the server. Successful uploads activate an XML parsing script on the server.
- `taskFeedback` is called every time when a task is finished, so that the server can migrate the active task entry to the finished task table, meanwhile recording relevant information.
- `taskCredit` records information including the number of tasks each device finishes and number of tasks each user contributes.
- `sendPush` utilizes third party APIs to send notifications to registered devices. The Ocelot server (or dashboard) uses this function to notify clients that there are tasks available for computation.
- `requestWork` allows clients to spontaneously ask for tasks to compute, and is activated in testing or client's "Full Power" mode, which will be explained in Section 3.4.

3.4 OCELOT CLIENT

Ocelot originates from BOINC's idea of public-resource computing but focuses more on building a middle-ware system with energy-efficiency and distribution-flexibility. Ocelot is conceptually possible on most mobile platforms, and I chose the most popular of the mobile operating systems as Ocelot's computation platform. My current implementation supports both Android and iOS. Fig. 8 shows the legacy Ocelot client apps running tasks on both operating systems. Client apps on different platforms have different user interfaces (UIs) but share similar APIs for communicating with the servers and typically execute the same application core algorithm. In the combined system of Ocelot and Lynx, since all nodes share the same roles like peers in a P2P network, a client can also become a server or even a dashboard, which is described in Section 3.5. Thus, while in the legacy Ocelot system, clients, servers, and dashboards are physically separate components, they are integrated on all the participant devices as virtually different function modules.

Ocelot clients provide distributed computation power. However, donating a device to operate as an Ocelot client results in Ocelot applications consuming some portion of the device's battery power. Thus, the limited battery capacities of mobile devices greatly restrict the amount of work we are able to compute between charging. To provide the user some control over this, the Ocelot client app is able to switch between different work modes according to both power settings defined by the user and client's power status. "Full Power" mode continuously requests tasks from the server. This mode is only available when set manually by the device's owner or when the device connects to an external power supply. "Push Task" is the most commonly used mode in which the client receives push notification from the server and reacts according to the message content. The scheduling server notifies the clients once there are some available tasks to complete. For a client to receive push notifications requesting work, the client must be registered with the push server even if it has already installed the Ocelot client. The database server keeps a record of the power status of all clients (e.g. remaining battery life, charging status), by both sending a status fetch push to the client pool periodically and letting the client report its power status after finishing each task. The scheduling server looks up in the registered client pool and always



(a) Client on Android

(b) Client on iOS

Figure 8: Ocelot client apps on both Android and iOS platforms.

chooses clients either connected to the power supply or with ample remaining battery life (above 50% in current setting). If no clients begin accepting tasks after a preset period (five minutes in current setting), devices with less power are notified. Those with very limited remaining power (below 20% in current setting) will not be pushed tasks. "Idle" mode is usually set by the device owner to avoid receiving pushed tasks from the Ocelot server. This mode allows the device to communicate on the network and potentially forward information from node to node, but conserves power when charging is not available for a long period of time.

```
runCommand( "chmod 755 "+fileDir+appName );
runCommand( "."+fileDir+appName+" "+fileDir+inFileName+" "+fileDir+outFileName );
                          ...
private void runCommand(String cmd) {
  try {
    Process process =
      Runtime.getRuntime().exec(cmd);
    process.waitFor();
  } catch (Exception e) {
    throw new RuntimeException(e);
  }
}
```

Listing 3.3: Binary execution on Android

A common Ocelot computation procedure starts with a task request from either a client or a dashboard. The client receives a task immediately after the request if there are unfinished ones. If the task request is initiated by a dashboard instead of a specific client, the scheduling server starts to work and clients in non-idle mode successively receive pushed tasks according to their power status. Once a client receives a task, it examines whether the application code required is available locally on the device. This is not the case if the client has not received any task from that application before or if the application has been updated on the server. In this case, it must download the application executable binary from the web server to execute the application and stores it locally for future use. Input file downloading starts next, and the computation process begins, which might take a couple of seconds or even several minutes according to the task size. A code excerpt of binary execution on Android is shown in Listing 3.3. The Java method "*Runtime.getRuntime().exec()*" is used to run a Unix shell command at runtime on Android. Access permission to the application code must be modified before the program can execute it as shown in the code. Smaller task sizes are encouraged so that more parallelism can be exploited to accelerate the computation and individual batteries are not depleted more than necessary. The resulting file is uploaded to the server once computation completes and the client's latest power status is reported as well.

3.5 OCELOT DASHBOARD

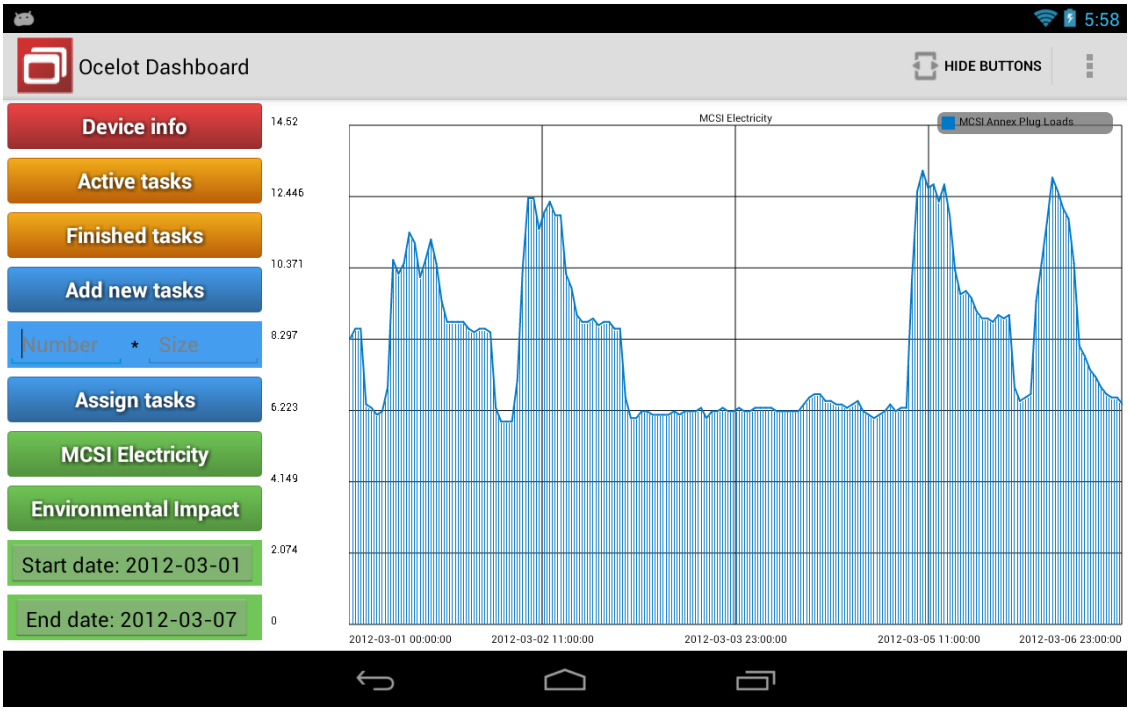


Figure 9: Ocelot dashboard.

I leverage some of the powerful features of the tablet touch interface to provide an dashboard in the Ocelot system. The Ocelot dashboard has functions such as basic information lookup, manual task generation, and assignment. The dashboard also provides on-demand an interactive charting engine to display result data. Fig. 9 shows an interactive chart of one week of result data. If result data for an application is requested to be displayed in a chart, but that data has not yet been computed, the dashboard will start the application to generate tasks to compute the requested display data automatically. The dashboard user will be notified that computation is underway and once the data is ready, the chart is displayed. Direct SQL query is supported on the Ocelot dashboard for advanced project administrators to easily make changes to databases, and potentially destructive actions like deleting or truncating tables are password protected.

3.6 SECURITY AND LIMITATIONS

Security on mobile operating systems (OS) has become a particular concern, as tablets and especially smartphones often store personal information, and emerging new features of mobile devices give opportunity to new threats. Both Android and iOS operating systems shield system files and resources from the user's apps, and restrict them from accessing files stored by other apps or from making changes to the device using a technique named "Sandboxing"[93]. Sandboxing prevents the Ocelot Client app from malicious actions to the system or other apps and/or from being infected by other malicious apps. Since the App Sandbox is in the kernel, this security model extends to executing externally obtained native code and calling OS level applications. Moreover, other special mechanisms are enforced such as *app permission requesting* on Android and *app code signing* on iOS. All of these steps taken by mobile OS developers ensure the healthy and long-lasting operation of the systems. Meanwhile, the safety mechanisms also create challenges in creating the Ocelot Client and in extending distributed computing to mobile devices. A direct effect is that binary code execution on iOS is prohibited. Rather than jailbreaking test devices, I combined the application code into the Client app instead of downloading it from the server like the Android Client. Thus, with new application codes, a new Client app must be updated in iOS.

A record of the information of all participant devices is also maintained by enforcing registration with Ocelot server before they can accept tasks. Owner's verified email addresses are associated with his/her device as the unique id, as an identifier in case of suspicious behavior. On Android devices, the owner's Google Play Store id or primary email account is used. However, iOS does not allow third-party apps to collect user information, so the user provides an id in the Client app. Similar limitations are subject to change in the future depending on the APIs released or updated by Google and Apple.

3.7 CASE STUDY: DYNAMIC LIFE CYCLE ASSESSMENT OF A LEED SILVER BUILDING

Life cycle assessment (LCA) is a technique for providing a comprehensive and quantitative analysis of the environmental impacts by a process or product throughout the entire life cycle. It is method that can be adopted to understand how a building and its occupants use electricity, water, and other energy resources from its construction to destruction, and how the building operation impacts on the environment. Established guidelines for performing detailed LCAs are well documented by the Environmental Protection Agency (EPA), Society for Environmental Toxicologists and Chemists (SETAC), the International Organization of Standardization (ISO), and the American National Standards Institute (ANSI). And some practitioners' guidance on methods to conduct LCA on buildings is provided by [94]. Still, LCA is not commonly used in building industry practice due to several possible factors, including the perceived complexity of LCA application on buildings, or the exclusion of internal building effects important to researchers, such as indoor environmental quality (IEQ) and other dynamics of the building's operating phase [95]. The latter appears to be more of a critical challenge since researchers have demonstrated concerns that environmental impacts of building design are dominated by the impact of operating phase of the building (e.g. energy usage, emissions, etc.) rather than impacts from construction, or destruction [96, 97].

A dynamic LCA (DLCA) approach has been developed, which essentially performs traditional static LCA continuously using real time data collected from a building. The University of Pittsburgh's Mascaro Center for Sustainable Innovation (MCSI) building, part of the Swanson School of Engineering (SSOE), is used as the first test bed for this DLCA approach, though the method will subsequently be extended to other buildings. A wireless sensor network is deployed to monitor MCSI's internal building metrics such as indoor environmental conditions, from indoor air contaminant concentrations to temperature and relative humidity. Sensor data is communicated via WiFi to the Ocelot data server from multiply locations of the MCSI building.

In the DLCA computation used as a case study of my Ocelot system, I mainly focus on the evaluation of electricity usage. As we know, electrical power is produced by power plants

Table 1: Major specs of devices used in the experiments

Device List	CPU	Memory	OS	Power /w
Toshiba Thrive	1G 2-core Cortex-A9	1GB	Android 3.2.2	1.4
Nexus S	1G 1-core Cortex-A8	512MB	Android 4.0.4	1.5
Nexus 7	1.2G 4-core Cortex-A9	1GB	Android 4.1.1	2.2
iPhone 4s	0.8G 2-core Cortex-A9	512MB	iOS 5.1.1	1.6
iPad 1	1G 1-core Cortex-A8	256MB DDR	iOS 5.1.1	4.0
iPad 3	1G 2-core Cortex-A9	1GB DDR2	iOS 5.1.1	4.3
Mac mini (circa 2010)	C2D P8600	8GB	Mac OS 10.8	25.6
Mac mini (circa 2011)	i5-2410M	8GB	Mac OS 10.8	23.9
Custom build desktop 1	Pentium G620	8GB	Ubuntu 12.04.1	57.2
Custom build desktop 2	Xeon 1230 v2	16GB	Windows 7	127.0
Thinkpad W520	i7-2760QM	16GB	Ubuntu 12.04.1	32.1

using various resources like coal, natural gas, nuclear fuel, etc. Thus, the production process itself consumes electricity as well. My algorithm calculates the total electrical power needed to generate the total electricity consumed in the building, as well as the environmental impacts it produces (e.g. all waterborne and airborne emissions). Additional historical data collected by some commercial meters is also used to provide enough workloads to Ocelot clients. Within the LCA and buildings arena, electricity use is a significant component of the building's life cycle. Thus. I am focusing my initial efforts on better understanding of electricity usage.

All devices used in my experiments are listed in Table 1 together with their major specifications. All operating systems were updated to their newest versions. The client app on mobile devices was ported to computers only for comparison in the experiments. And to eliminate the impacts of different network environments, all devices were connected to the same wireless router during all tests. I also developed a method to greatly reduce the download/upload time using data compression. Both the server and the client are able to compress and decompress data files. Compression is identified automatically by file format

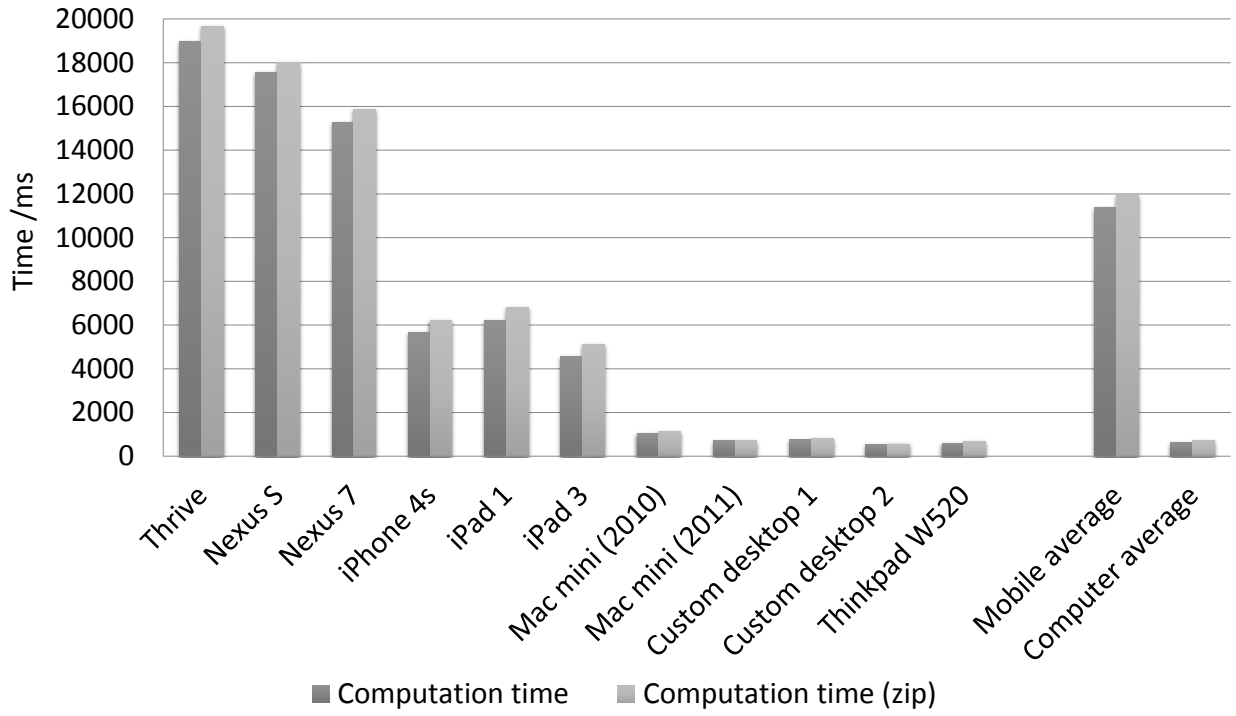


Figure 10: Average computation time per task

so no additional message needs to be passed. Each test includes 20 tasks with the same work load in both normal XML format and compressed Zip format, and the results are presented in average time per task. I demonstrated that by migrating distributed computation platforms from traditional computers to palmtop computers, energy cost per task can be significantly reduced. And with several devices working in parallel, good speedups can be achieved as well.

3.7.1 Energy saving

As shown in Figure 10, computation time is strictly relevant to the processor speed and memory capacity. iOS devices have the advantage of executing native compiled code rather than having to run code on top of Dalvik—a process virtual machine—as in Android. Thus, iOS outperformed Android devices with similar hardware specifications. The iOS equipped iPad 3 took only one-third of the computation time that the Android equipped Nexus 7

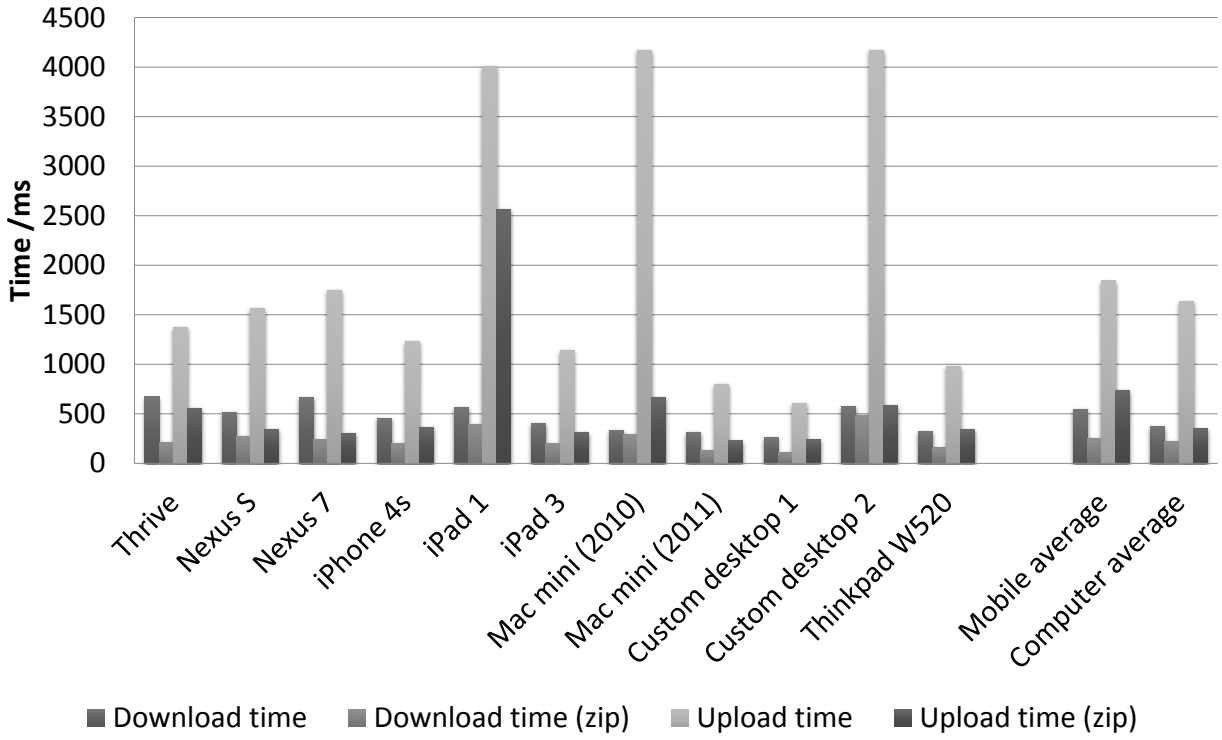


Figure 11: Average download/upload time per task

needed to calculate one task. Both represented the best available devices in their categories on the market when I conducted the experiments. And quite expectedly, traditional computers completely outperformed mobile devices, averaging 5 times faster. I counted data compression/decompression time into the computation time so on average it took about 5% longer to compute with Zip data files. But, as discussed below, data compression saves a huge amount of communication time and thus reduces the total task time.

Network performance is related to several factors such as protocols, network adapters, processing speeds, signal strength and other environmental impacts. As I mentioned in previous section, my experiment setup ensured all devices were in similar network environment so the communication results were caused only by the difference of devices themselves. Figure 11 shows average download/upload time per task. It should be noted that Zip file transfer, on average saved about 75% of the time needed to transfer a regular data file, at the maximum compression rate of about 80%. The advantage of transferring compressed

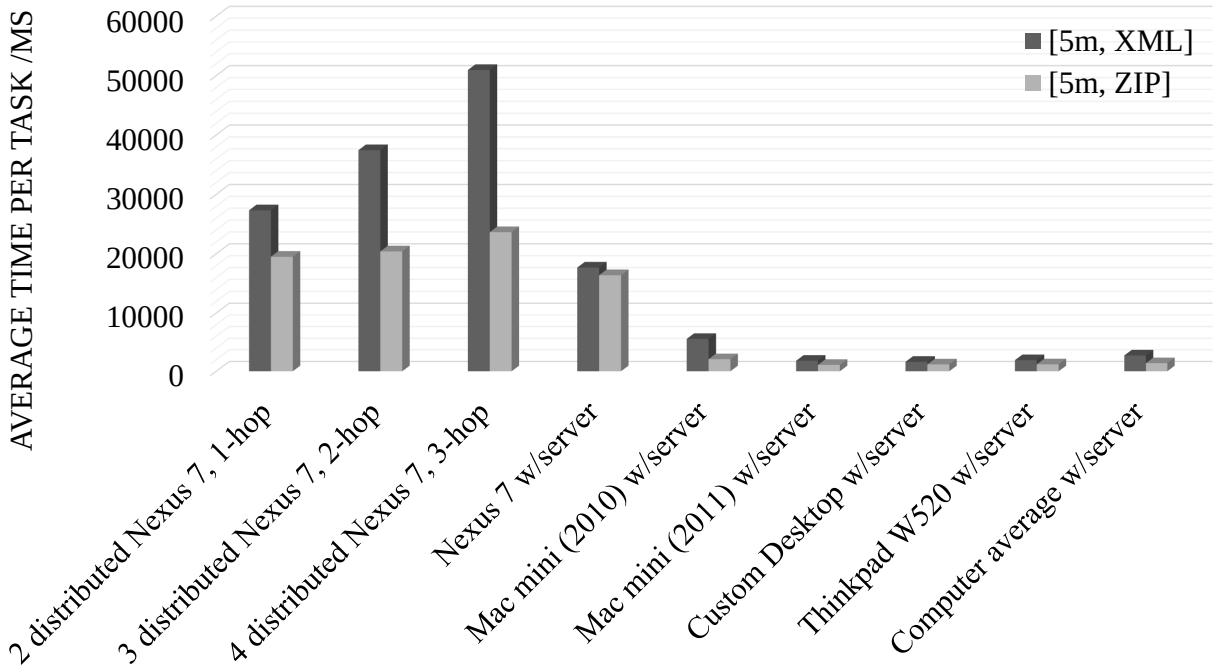


Figure 12: Average total time per task

files was weakened, but still existed in the average total time per task shown in Figure 12, considering the communication time takes only a small portion of the total time. And with lower computation time, fast-computing computers gained better improvement over data compression than slow-computing palmtop devices. Thus, I projected that utilizing data compression in Ocelot projects would gain better results with increasingly fast devices.

This result might appear discouraging, though it should not be unexpected that mobile devices require more computation time than workstations to complete one task, on average (between 5-10 times depending on use of compression). However, workstations are optimized for performance while sacrificing some energy efficiency. Table 1 lists all measured AC power consumption of each device while running the LCA test. By multiplying power by time per task, I determined energy consumption per task for each target device shown in Figure 13. On average, mobile devices consumed 29.8J for uncompressed communication and 27.8J with compression, while workstations consumed an average of 217.0J and 84.4J, correspondingly. Consequently, mobile devices saved energy by 86% in contrast to a workstation using un-

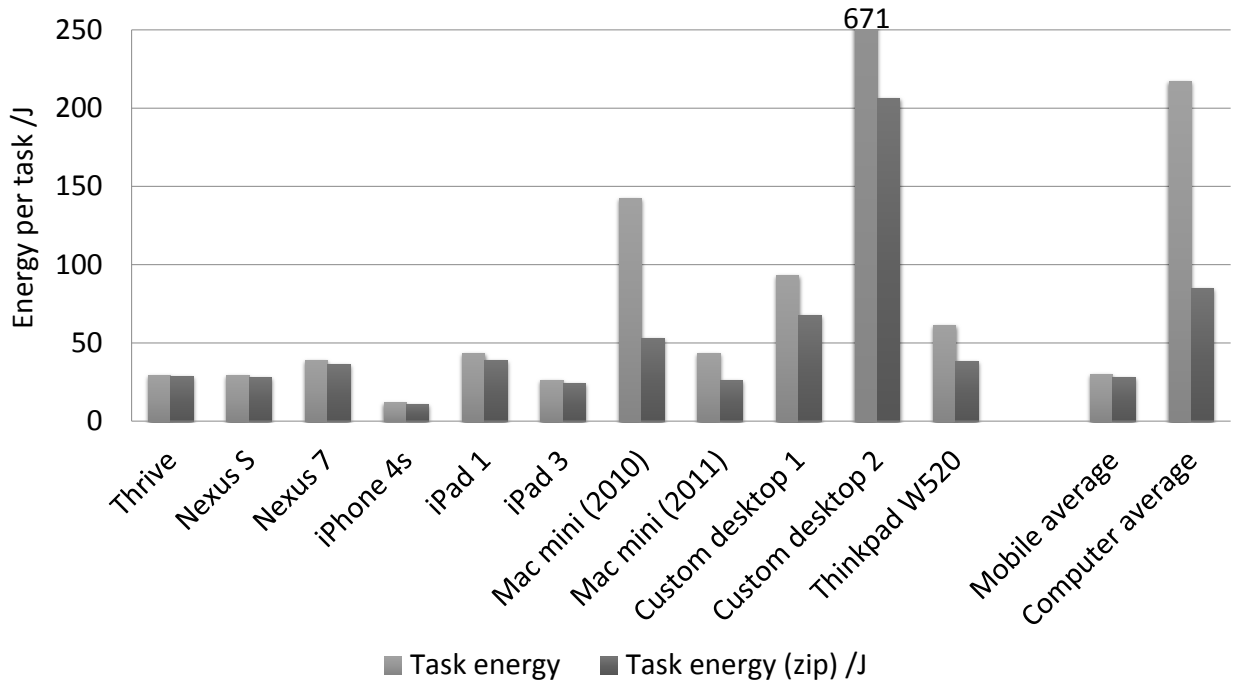


Figure 13: Energy consumption per task

compressed files and 67% using compressed files. This is to say nothing of the idle time spent between jobs in which the workstations require considerably higher idle power than the palmtop devices.

3.7.2 Intra-device parallelism

Instead of increasing clock speeds, which allows applications to run faster without special tuning, processor manufacturers have been increasing the number of cores within a single chip. Thus, I also explored the possibility of better utilizing the power of multi-core processors within those palmtop devices. The major computation in the DLCA algorithm is large matrix multiplication, which can be modified into a threaded version by partitioning the columns and rows. For example, one 16x16 matrix can be partitioned into four 8x8 matrices, or sixteen 4x4 matrices. So I slightly modified the code and added my fine-grained multithreading implementation.

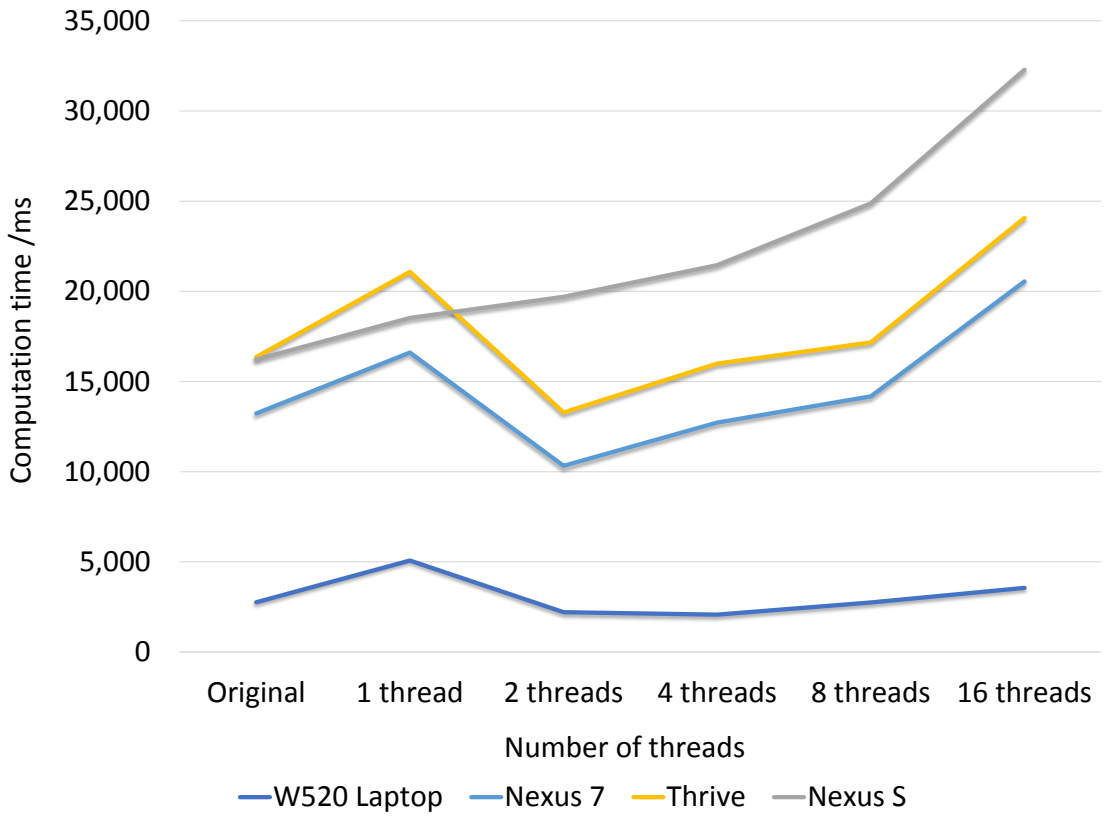


Figure 14: Task computation time of intra-device multithreaded execution

Figure 14 shows computation time per task with different numbers of threads on three Android devices and a quad-core laptop running Ubuntu. The threaded code adds overheads like creating threads and passing values between threads, thus the "1 thread" (multithread-ready but not multithreaded) code ran slower than the original code for all testing devices. Only Nexus S smartphone gained no improvement on multithreaded code due to its single-core processor. Other than that, both Thrive and Nexus 7 reached their best performance on the "2 threads" code, saving 19% and 22% of their computation time, respectively, compared to the original code. Adding additional threads showed that the overheads increase when number of threads increase and the additional parallelism gain is outweighed by the threading overhead. The laptop reached its best on "4 threads" code thanks in part to its better access to data in the memory system. This effort to improve device performance using multithreading showed positive results, and could be applied to other Ocelot projects.

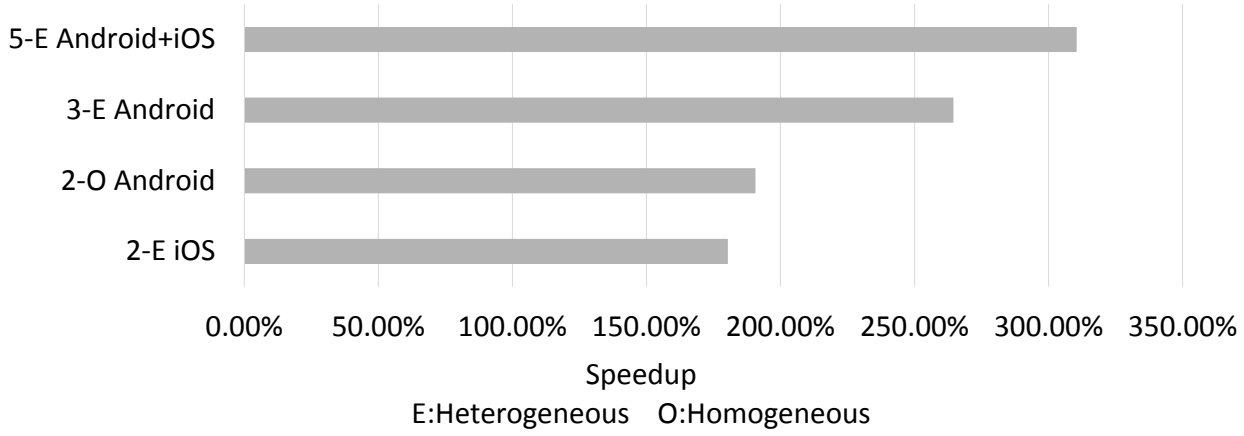


Figure 15: Task completion speedups of inter-device parallel execution

3.7.3 Inter-device parallelism

One of the most important advantages of distributed computing is the capability to have many clients to compute in parallel. I conducted multiple tests using different number of clients, and defined the *speedup* as the ratio of computation time of the fastest device in single client test to the computation time of the slowest device in parallel test. I conducted experiments with two heterogeneous devices, two homogeneous devices, three heterogeneous devices, and five heterogeneous devices, respectively. Results are presented in Figure 15.

None of these four sets achieved an ideal speedup. Homogeneity does help to give a theoretically good speedup, as in heterogeneous systems, the slower devices cannot keep up with the faster ones. It can easily be concluded from the result comparisons that the more diverse the device composition becomes, the less speedup would be gained. However, by adding new members to the client pool, I saw significant savings in total task completion time.

4.0 LYNX: A SELF-ORGANIZING WSN PLATFORM LEVERAGING COMMODITY HARDWARE

4.1 LYNX BASICS

There are three elements in the Lynx system: *Lynx node*, *Lynx packet*, and *Lynx task*. A Lynx node integrates all useful information related to the local node such as peer map, link map, task queue, etc., as well as relevant methods handling the information. A Lynx packet is a data wrapper of messages or files, including the packet data type, a route that packet should traverse, and the content. A Lynx task is a subclass of Lynx packet with additional fields like task type, next hop address, and packet payload. Lynx tasks are generated and pushed into a task queue in the Lynx node, which is checked and executed periodically. Fig. 16 demonstrates how Lynx node modules self-organize and their relationship.

4.2 CONNECTIVITY CHOICE

The choice of network protocol is independent of the Lynx implementation. In my implementation, I have demonstrated the self organizing concept with both WiFi-Direct and Bluetooth. WiFi-Direct as the connection approach provides advantages due to its wider coverage and better speed. Unfortunately, due to limits in the network stack, it was only possible for a maximum of two devices to be connected in pairs simultaneously. Reconnecting required unacceptably high additional recovery time (e.g. seconds), which made multi-hop routing unstable and inefficient. Given that WiFi-Direct is relatively new, the qualities of driver and APIs still have room to improve. Fortunately, connectivity protocols can be

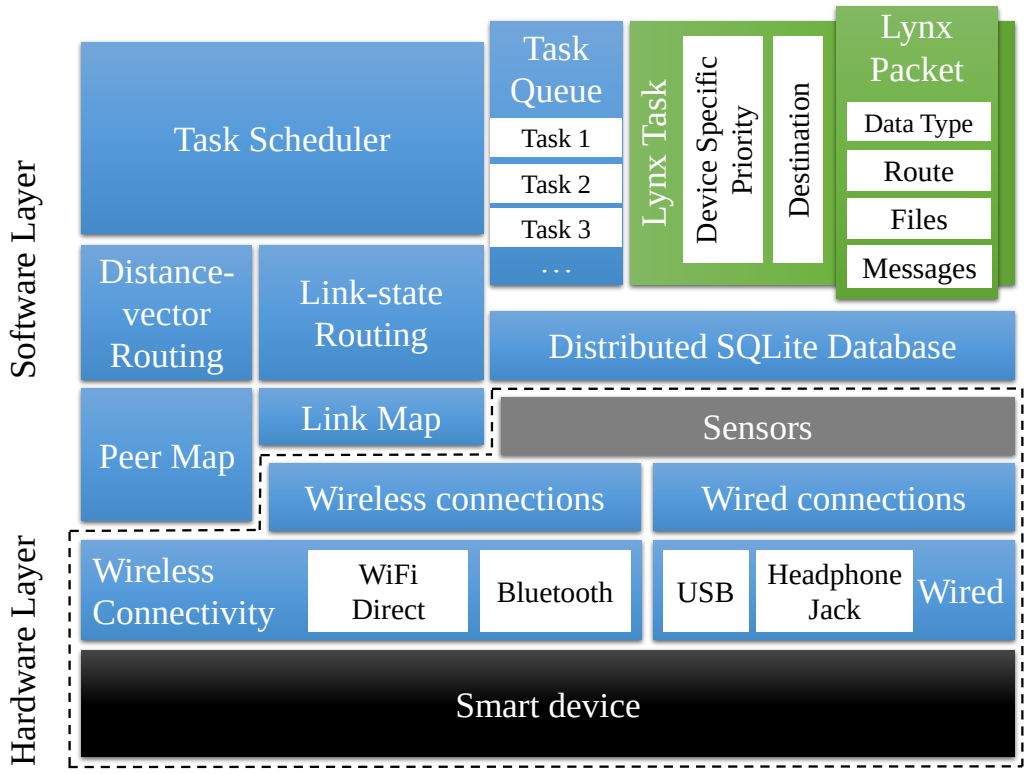


Figure 16: The structure of a Lynx node

abstracted and all functional modules can be developed upon that abstraction layer [98]. I was able to separate high level design with low level implementation in Lynx. Thus, I demonstrated the system using Bluetooth connectivity for the current version of Lynx app on Android. As other better link protocols are developed and become mature, Lynx will be able to leverage them even in a hybrid multi-protocol fashion.

4.3 SENSOR INTEGRATION

Reading, recording, and sending sensor data are the most important functionalities of a WSN node. Several prevailing commodity mobile platforms, such as iOS and Android, provide hardware capable of doing those tasks. Wireless protocols such as WiFi, Bluetooth,

or NFC can be used to connect mobile devices to sensors. It is also possible to send analog sensor data through the 3.5 mm headphone jack by modulating and demodulating the analog signals [99]. Sensed data can be stored either in files or as database entries on the device. By querying the node connected with the sensor, other nodes in the network can retrieve the sensor data in real time over the established P2P network.

While exploiting the possibility of sending data through the analog link in related projects, I made use of a commercial product named Sensordrone as the testing sensor. It senses nice different environmental conditions, and provides Bluetooth connection APIs to access the data with Android devices. Although it is a point-to-point connection, making it impossible to simultaneously connect multiple devices to a sensor, the sensor data could be read remotely by querying the connected node. This link could either be a direct link or a multi-hop one. Details are discussed in Section 4.5.

4.4 MAINTENANCE OF PEER AND LINK STATES

Wireless application programming interfaces (APIs) typically provide functions such as discovering nearby devices, remembering paired devices, connecting to/disconnecting from a remote device, listening to a server socket for incoming connections, etc. A Bluetooth MAC address is a unique string that identifies a device, and can be used to initiate connections to that device. There is a 128-bit universally unique identifier (UUID) that serves to identify radio frequency communication (RFCOMM) channels during the connection establishment. After two devices are connected via Bluetooth, additional sockets can be created to transfer data.

Conceptually, a similar process works for other relevant wireless protocols, including WiFi-Direct. To achieve feasible and efficient routing, each in-network node must have a real-time, global knowledge of the current network structure. MAC addresses are used to uniquely identify devices. Since each device knows its nearby devices, they can tell others their peer lists (a list of MAC addresses), and, at the same time, receive and record others' peer lists. Peer lists form a peer map as illustrated in Fig. 17. Similarly, a link list and a

Figure 17: A Lynx peer map illustration. MAC addresses are used in Lynx network to uniquely identify devices.

link map are also maintained, indicating the connection status of local device and the whole network, respectively. To maintain the two lists and two maps, Lynx provides the following functions:

- `Peer list scan`: periodically scans nearby nodes and updates the peer list.
- `Peer connect`: periodically connects to Lynx nodes in its peer list but not in its link list, aborts if number of failure exceeds a threshold.
- `Peer list report`: after each scan, broadcasts its peer list changes (if any) to connected neighbors.
- `Link state report`: broadcasts its connection status changes (if any) to connected neighbors.
- `Peer/link map report`: periodically broadcasts its peer/link map to connected neighbors.
- `Task checker`: periodically check and execute if there is any available Lynx Task in the local task queue.

- `In-network sensor connection status`: broadcasts sensor connect/disconnect events and connection information to all nodes in the same network (using multi-hop routing if needed). Information includes the sensor name and MAC address, the node's name and address to/from whom the sensor connects/disconnects.

4.5 MULTI-HOP ROUTING

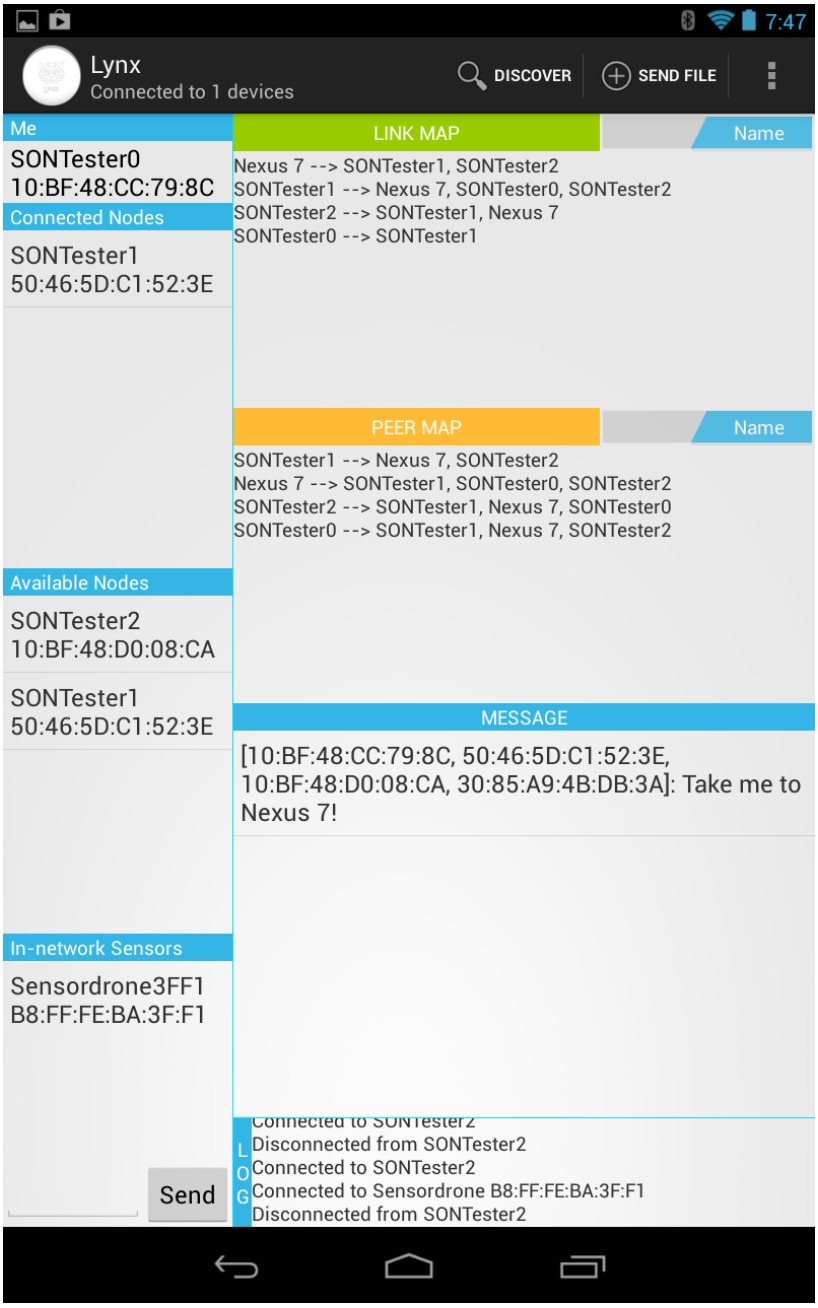
When two nodes are not directly linked but need to communicate with each other, it is necessary to find an indirect route with some other nodes in between to make the connection as shown in Fig. 4. Since I maintain in each node a global link map, I can easily find the optimal route by modifying some classical link-state routing algorithms, in this case, a modified Dijkstra's algorithm. Algorithm 1 provides detailed pseudo-code. Assuming the distance of each connected node pair is one, the shortest route found represents the route with least number of hops.

A 4-hop message routing test is shown in Fig. 18. Four Nexus 7 tablets were connected only to adjacent devices as shown in each tablet's "Connected Nodes" list. "SONTester0" (hop 0) was tasked to send a message to "Nexus 7" (hop 3). Lynx calculated the best route (in this case, the only feasible route, SONTester0 → SONTester1 → SONTester2 → Nexus 7) and sent the message to the destination via two "bridges", as shown in the "MESSAGE" record of the app window. Files can be sent in a similar fashion. The specific routing algorithm is implemented only to show the feasibility of the Lynx WSN and to conduct experiments. The Lynx, as a platform, has the flexibility to incorporate any routing algorithms, either Link-state or Distance-vector, as illustrated in Fig. 16, including those optimized for other criteria such as minimizing energy, avoiding nodes with low battery, etc.

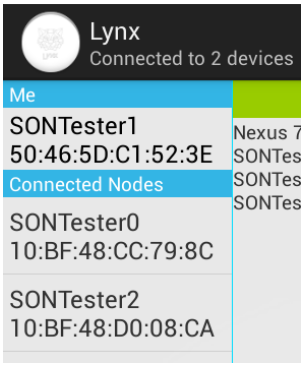
Algorithm 1 Find the shortest route: Modified Dijkstra's algorithm

Require: M is the link map, N is the total number of nodes in M, $dist[s, v]$ and $route[s, v]$ record the shortest distance and route from node s to node v, respectively

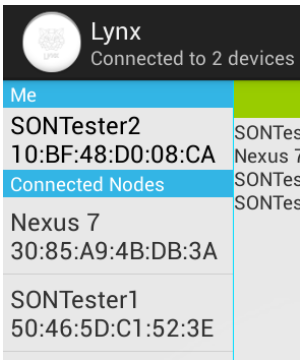
```
1: procedure ShortestRoute(s, M)
2:     for all node v in M do
3:         if v connects to s then
4:             dist[s, v] = 1
5:             route[s, v] = {s, v}
6:         else
7:             dist[s, v] = N + 1
8:         end if
9:     end for
10:    M' = {s}
11:    while M != M' do
12:        pick a node u in M but not in M'
13:        M'=M' ∪ {u};
14:        for all node w not in M' do
15:            if w connects to u then
16:                if dist[s, w] > dist[s, u] + 1 then
17:                    dist[s, w] = dist[s, u] + 1
18:                    route[s, w] = route[s, u] plus w
19:                end if
20:            end if
21:        end for
22:    end while
23: end procedure
```



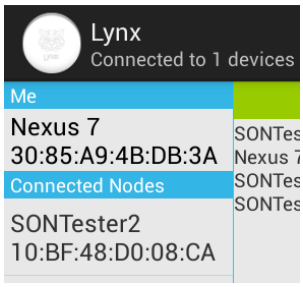
The message initiator (hop 0)



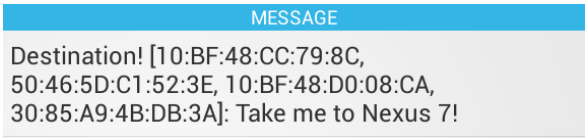
Status of hop 1



Status of hop 2



Status of hop 3 (destination)



The message arrives at the destination through 4 hops

Figure 18: Sending a multi-hop message

4.6 SCALABILITY AND LIMITATIONS

Theoretically, there is no limit to the scale of the network, but the performance is likely to degrade as size increases. As the size of the link map grows, finding the shortest route will take more time, as the run time complexity of my modified Dijkstra algorithm is $O(E + V * logV)$, where E is the number of connected edges and V is the number of nodes of the network. However, the computational time of finding a route is negligible compared to the data transfer time. When the density of the network grows, due to the throughput limit of a single node, transmission rate of each individual route traveling through the same node will reduce. However, in a real world setup, the network density is likely to remain in a reasonable range considering the physical size of devices and the limited coverage of wireless signals. Thus, in conclusion, the Lynx network is capable of scaling, while maintaining reasonable performance in real world applications.

4.7 EVALUATION OF LYNX

To examine the capabilities of Lynx, I created a testbed of Android palmtop computers containing multiple Nexus 7 tablets. All were equipped with the Bluetooth 3.0 module and running the Android 4.2.2 Operating System. Several heterogeneous setups of additional android devices were possible. The results reported use homogeneous devices in order to provide a more precise, apples-to-apples comparison of the results of the multi-hop routing tests.

In order to ensure one-hop/multi-hop transfer, I manually configured connect daemon on all devices to set up the network into 1-hop, 2-hop and 3-hop structures, with 4 different distances from each other, ranging from 1 meter to 15 meters and separated by a wall. Due to the limited range of Bluetooth signal coverage, 15 meters was the greatest working distance (free of dropping connections) that could be achieved. The tested file was a common Ocelot task input file in XML format, with a size of 189,006 bytes. A compressed ZIP file of that particular XML file with a size of 22,610 bytes was also tested, since data compression was

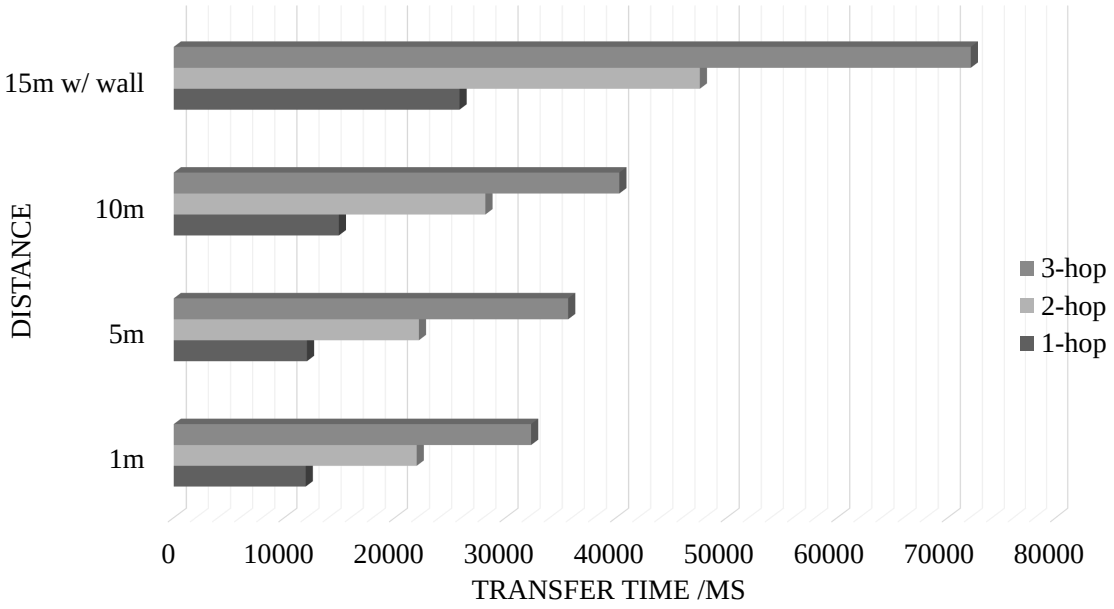


Figure 19: Average XML file transfer time

utilized to save communication time in Ocelot. A significant data compression ratio of 8.36/1 was achieved because of the orderly content in the number-based XML file.

Each final value reported is an average of the results of six tests. The transfer time is calculated by the initiator device. The timer begins when the file starts to send, and ends when a returned ACK signal is received. The averaged ACKing time was 348 milliseconds, which is negligible compared to the transfer time.

4.7.1 File Transfer Time

Fig. 19 and Fig. 20 show transfer times for the XML file and ZIP file, respectively. Not surprisingly, it took longer to transmit longer distances, especially when increased from 5 m to 10 m, and from 10 m to 15 m with an additional barrier in between. In addition, increasing the number of hops in the route nearly proportionally increases the transfer time in most cases. The [2-hop, 5m, XML] (a total distance of 10 m) scenario took nearly 50% longer than [1-hop, 10m, XML] the [3-hop, 5m, XML] (a total distance of 15 m) scenario took nearly 40% more time than [1-hop, 5m, XML], even with a wall in between. Routes

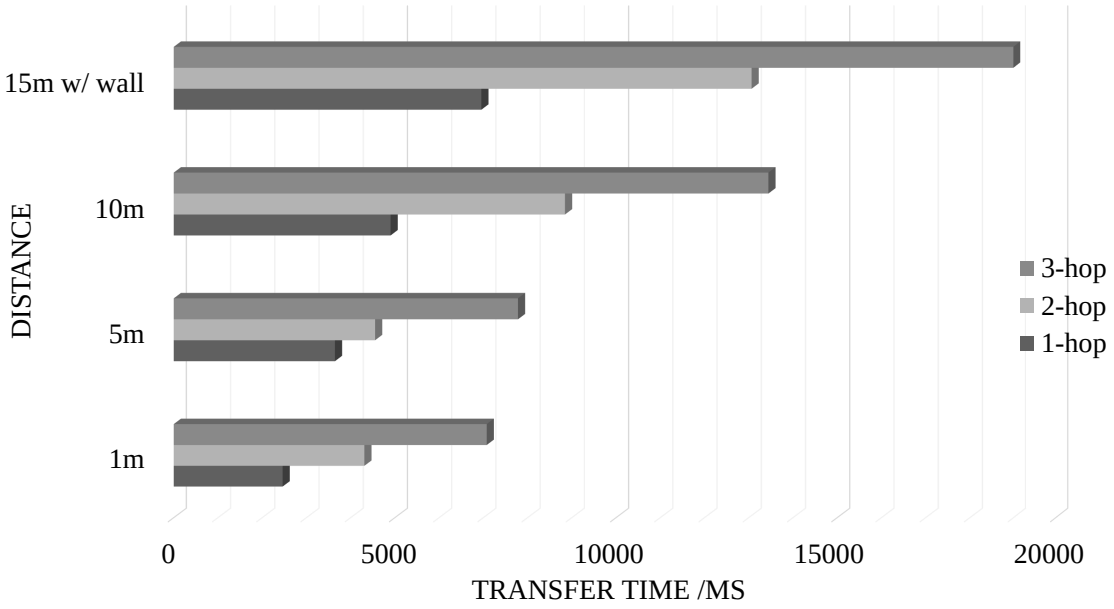
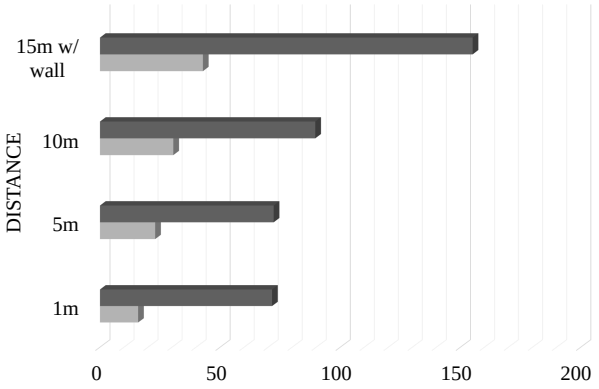


Figure 20: Average Compressed ZIP file transfer time

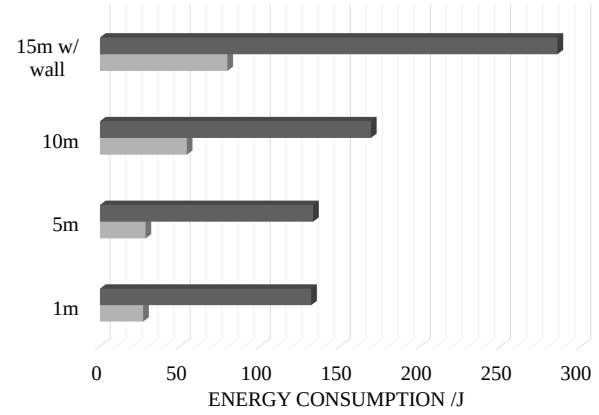
with fewer hops and slower transmit rates usually outperform those with more hops and faster rates in most scenarios. However, there is still a chance that a route with more but faster links will outperform one with fewer but slower ones, as seen in the [2-hop, 5m, ZIP] scenario. Data compression helped to save up to 80% of the transfer time.

4.7.2 File Transfer Energy Consumption

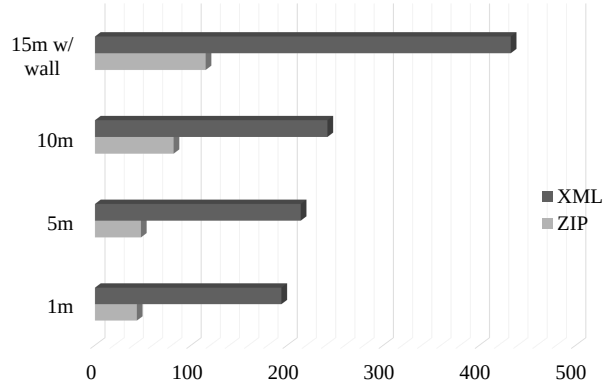
When conducting the file transfer tests, the electrical source energy consumed by a device sending and receiving files was measured using a commercial metering system called Plugwise, which basically consists of a socket measuring transient power consumption and a paired USB stick connected to the computer recording and aggregating those values. Average sending power P_s was 3.8 W and average receiving power P_r was 2.2 W, while simply running the app consumes P_o of 1.9 W. I also let the device calculate the total time T_z needed for compressing the XML file and decompressing the ZIP file, which was 579 milliseconds on average. Fig. 21 compares the energy consumption of transferring the XML file and the ZIP file in different scenarios side by side. It is obvious that the longer transmitting range requires



(a) 1-hop



(b) 2-hop



(c) 3-hop

Figure 21: Average energy consumption of transferring XML and ZIP files

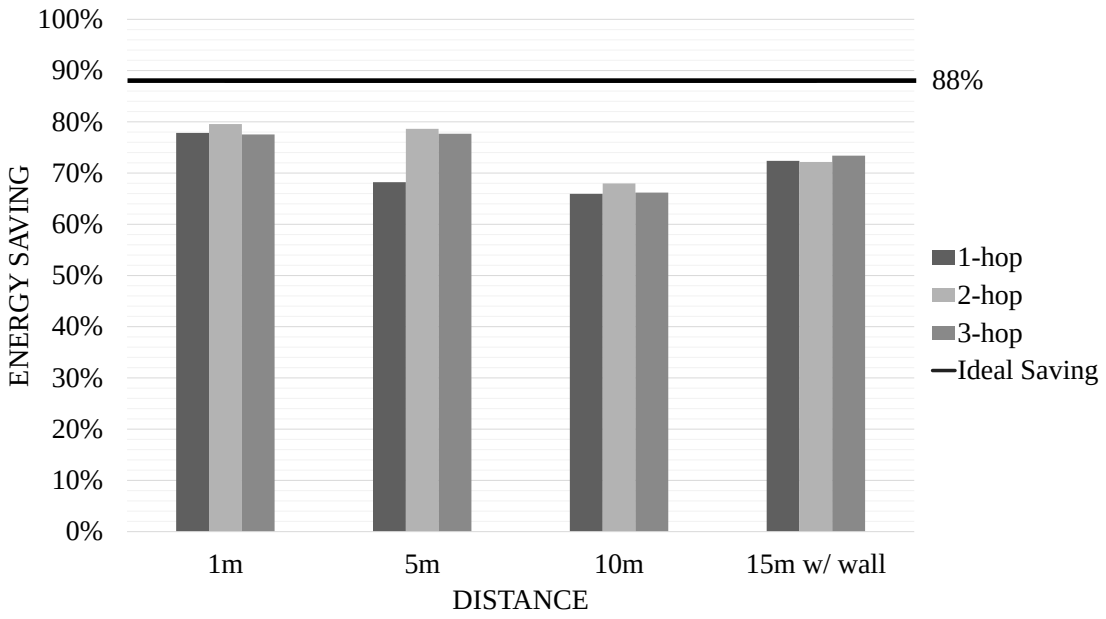


Figure 22: Energy savings by data compression in different scenarios

more energy due to a slower rate, resulting in increased transfer time. And similar to the transfer time discussed, energy consumption increases with the number of hops in a route in most scenarios. Therefore, the route with the fewest hops is typically more efficient than the route with shorter node-to-node distances. These results confirm that Lynx represents a system that can realistically demonstrate trade-offs for different system configurations.

$$E_{xml} = (P_s + P_r) * T_t \tag{4.1}$$

$$E_{zip} = (P_s + P_r) * T_t + P_o * T_z \tag{4.2}$$

An example of this can be shown in a comparison of compression overhead versus transmission overhead. Fig. 22 compares energy savings in 12 different scenarios with the ideal saving of 88% using data compression with a ratios of 8.26/1. The results vary between 65% and 80%, demonstrating that data compression in WSN communication greatly improves not only the network performance but the network life by saving valuable energy in distributed sensor nodes.

Table 2: Energy Consumption of Miscellaneous Processes

Daemon	Peer discovery	Link establishment	Peer/link state report (per hop)	Route calculation
Time /ms	215	167	480	86
Energy /J	0.41	0.50	1.44	0.16

4.7.3 Miscellaneous Energy Components

Besides the file transfer, as explained in Section 4.4, maintaining necessary peer and link states requires periodically running processes which also consume energy. I measured the average runtime of four processes including peer discovery, link establishment, and peer/link state report, and calculated the energy consumption based on the power measurement listed in Section 4.7.2. Peer state report has a packet size similar to link state report so the two energy components were combined. Table 2 shows the results, all of which are negligible compared to the file transfer energy.

4.7.4 Network Scalability

To examine how the Lynx network performs with a larger number of nodes and the impact of transient nodes, I increased the network size from 4 nodes to 8 and 32 nodes. Figure 23 shows the average time required to transfer a typical Ocelot input file of size 189,006 bytes. The average transfer time of each scenario is calculated based on results of 30 tests, through 1-, 2-, and 3-hop routes. I manually disconnected some of the ad-hoc nodes during the test so that the shortest paths of some routes became invalid. There is a slight increase in the average transfer time when the network becomes larger, probably due to longer time of calculating the shortest path with larger link map. Rerouting also increases the average transfer time as the optimal routes are invalidated. However, the overall performance degradation remains reasonable as the network scales.

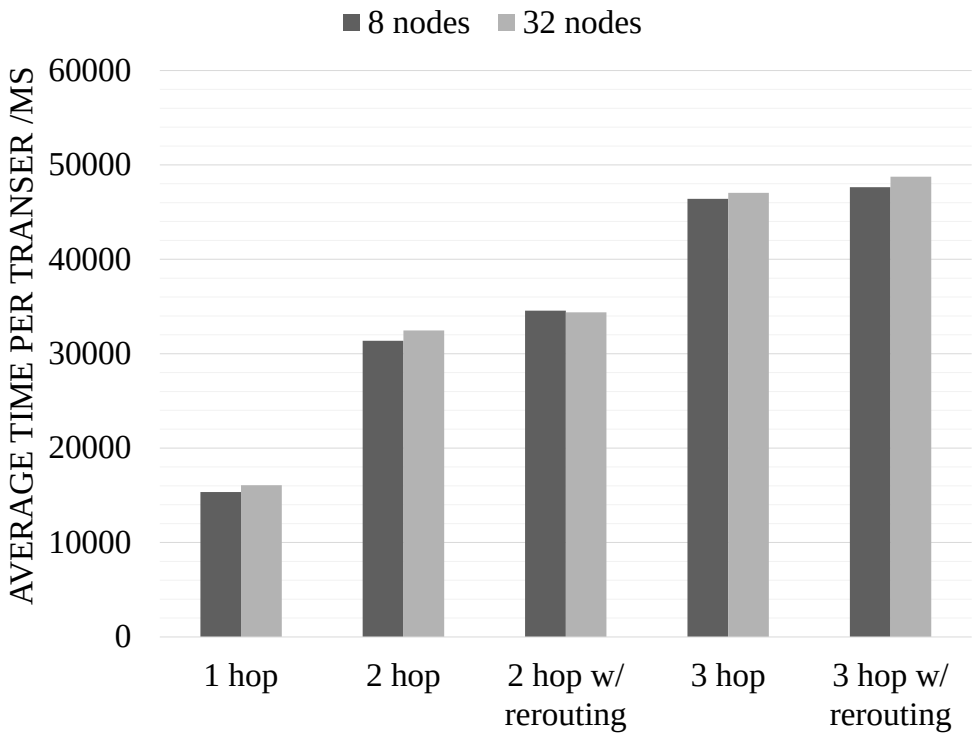


Figure 23: Average time of transferring a 189,006 byte XML input file, with different numbers of hops per route, w/ and w/o rerouting.

4.7.5 Lynx as a testbed

As introduced in Chapter 1, Lynx can serve as a natural platform for the study of various WSN routing and resource management algorithms, as the algorithm module is decoupled from the Lynx core, and is thus replaceable. The default algorithm used for multi-hop routing in Lynx is a modified version of Dijkastra's algorithm as explained in Section 4.5. When the network is large enough such that there is more than one shortest path, the default algorithm arbitrarily picks one path. However, a simple alternative is to choose the shortest path with the largest transmission rate to the next hop ("Greedy" algorithm). The detailed steps are described by Algorithm 2.

In this section, I demonstrate the capability of Lynx to serve as a testing platform of different routing algorithms in WSNs. I implement the greedy routing considering QoS

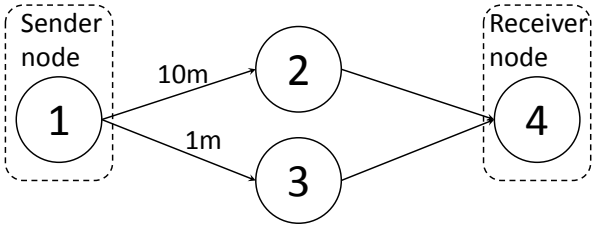
Algorithm 2 Find the shortest route considering QoS

Require: M is the link map, N is the total number of nodes in M, Q is the transmission rate map, *bestRoute* becomes the optimal route after the execution of the procedure

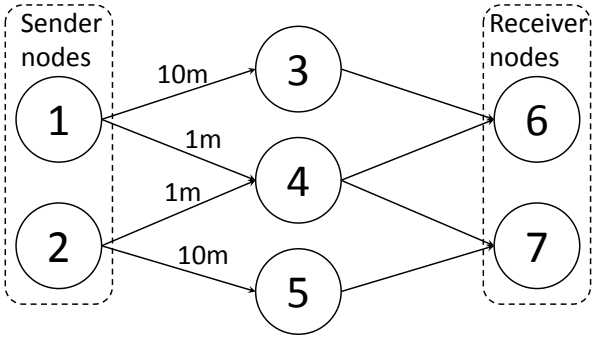
1: **procedure** SHORTESTROUTEQOS(s, M, Q)
2: candidateRouteList = ShortestRoute(s,M); ▷ Call modified procedure defined in Algorithm 1 which returns a list of shortest path
3: **for all** next hop h of route r in *candidateRouteList* **do**
4: $transRate = Q.getTransRate(h)$
5: **if** Q does not contains h or $bestTransRate < transRate$ **then**
6: $bestTransRate = transRate$
7: $bestRoute = r$
8: **end if**
9: **end for**
10: **end procedure**

(quality of service, in this case, transmission rate), and compare its performance with the default algorithm's. A Lynx network of four nodes was established as shown in Fig. 24(a). Node 1 sends data to node 4 through two-hop routes with either 2 or 3 as the bridge node. Node 3 was placed closer to the sender (∼1m) than node 2 (∼10m) to achieve a different (higher) transmission rate. A 189,006 byte XML file was sent every 60 seconds and the transfer time of both routing algorithms was recorded and shown in Fig. 25. The QoS algorithm always uses the faster route after having obtained the historical transmission rates of both routes, while the default Dijkstra routing arbitrarily chooses one.

Another experiment was carried out to calculate the performance difference in a larger network (Fig. 24(b)) with heavier traffic. The same XML file was sent from two sender nodes (1 and 2), to two receiver nodes (6 and 7), every 20 seconds instead of 60 seconds, and the average transfer time of the two concurrent transfers was recorded and shown in Fig. 26. During the 20 second interval, the data transfer completes the first hop of each route, but the second hop transfer is still in progress (given that the average 2-hop transfer of the same file takes about 30 seconds). Thus, the newly initiated transfer is likely to interfere with



(a) 4-node network



(b) 7-node network

Figure 24: Two network topologies used for testing the routing algorithm considering QoS. Edges indicate established connections; arrows indicate data transfer directions

the incomplete one and contend for bandwidth if two routes go through the same bridge node. The improved Dijkstra routing with QoS effectively avoids sustained performance degradation experienced with the original routing algorithm. In this case study, Lynx, serving as a testbed similar to software-based network simulators, effectively demonstrates the performance difference of different routing algorithms.

4.8 INTEGRATION OF OCELOT WITH LYNX

Figure 4 shows how the Ocelot system can be applied into a Lynx WSN. A dedicated WSN node (fixed and pre-configured) consists of an Ocelot client attached with one or more sensors monitoring certain physical or environmental conditions in an area. Sensing data is

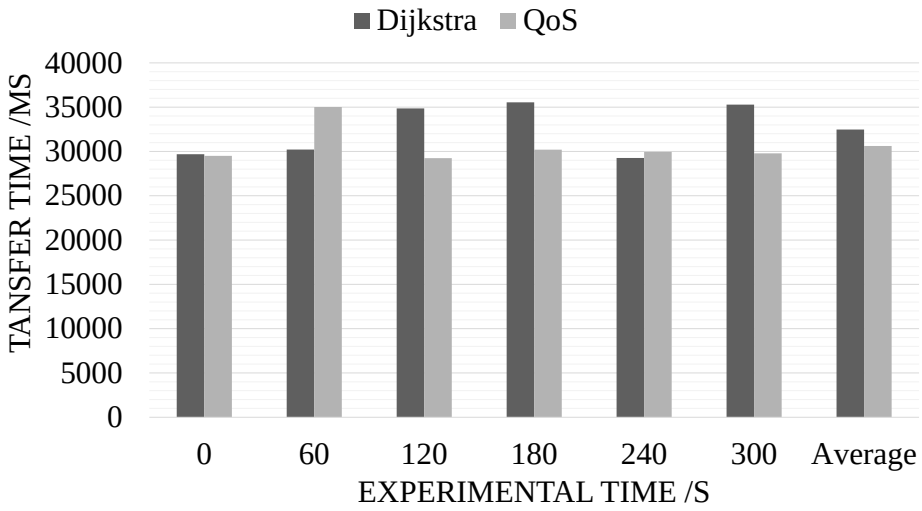


Figure 25: Transfer time comparison of two routing algorithms in the 4-node network

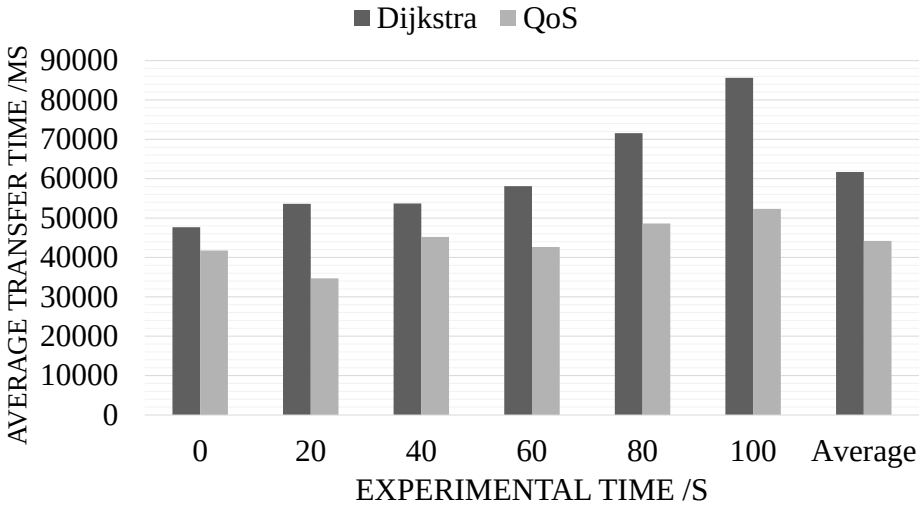


Figure 26: Transfer time comparison of two routing algorithms in the 7-node network

passed through wireless connections (e.g. WiFi, Bluetooth) on the Ocelot client instead of connecting particular communication device with sensors. Random mobile devices registered with the Ocelot server become an ad-hoc node and Ocelot client. Though not able to gather sensor data, they can take tasks and contribute their computation power. In addition, all these nodes can serve as a bridge and help to relay data within their connection coverage and capacity.

Combining Lynx with Ocelot requires a distributed database and distributed task/resource management. For this approach, the SQLite database was employed for local node storage. Task and resource management is demonstrated through connection with a dashboard display. The dashboard node (e.g. a tablet) becomes both a Lynx and master Ocelot node. This node determines the data required to display a result for the user and automatically launches tasks for remote data collection and processing prior to aggregating the results for display. I conducted Lynx and Ocelot system experiments of task performance on the combined system. Similar to my previous tests, the network structures were manually configured into different scenario combinations. The network locations of task input data were manually configured to study the exact behaviors of communication in a controlled environment.

Several mobile devices as well as stationary computers were used in the experiments as listed in Table 1. As mentioned in Chapter 4, the current version of the Lynx app uses Bluetooth radio for communication. However, I have included the result of an experiment using WiFi-Direct as the input file transfer method to show the potential of the latest P2P connectivity. Multi-hop routing over WiFi-Direct is too inefficient due to hardware and API limitations, so only the 1-hop scenario was included here. For simplicity, only the 5 meter hop-to-hop distance was tested. Performances in scenarios with other distances can be projected provided that the computation time is stable and the communication time was presented in Section 4.7.1.

As a case study of the integrated system, I applied the same DLCA, as introduced in Section 3.7, on the electricity usage of the University of Pittsburgh's Mascaro Center for Sustainable Innovation (MCSI) building [84], and focused on the evaluation of electricity usage.

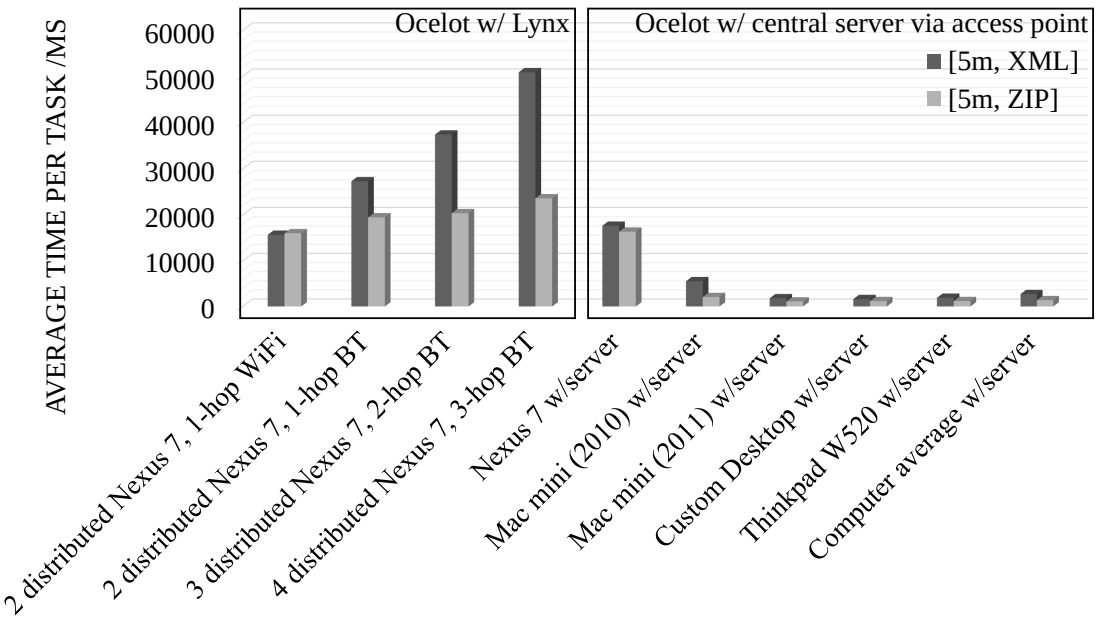


Figure 27: Average execution time per task

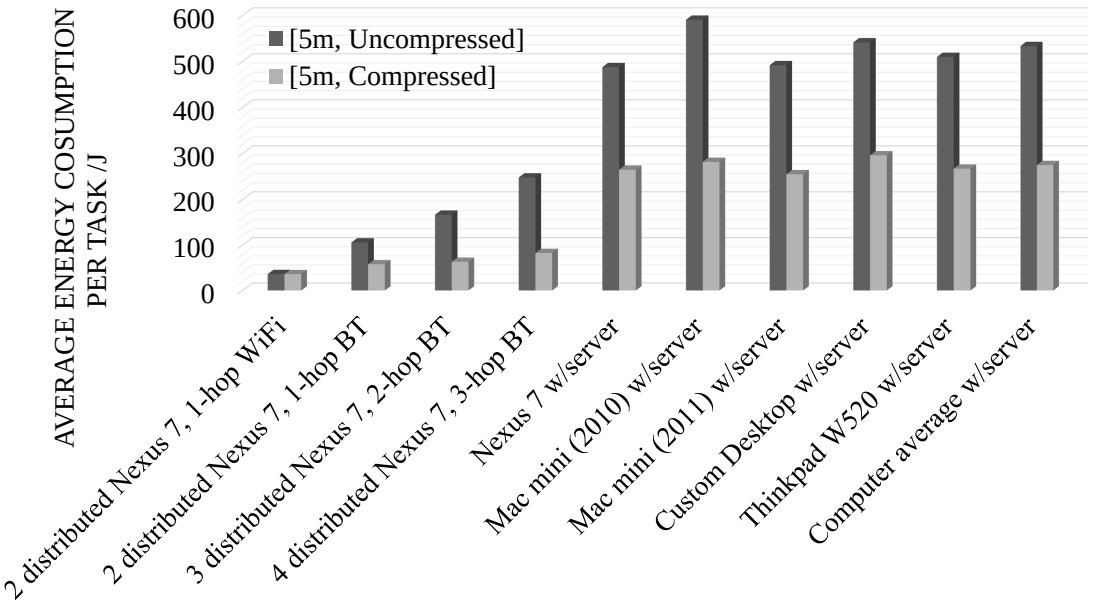


Figure 28: Average energy consumption per task

Fig. 27 shows the average execution time per Ocelot task over the Lynx network. Each task includes 1,000 data entries, each of which is similar to the structure of Listing 3.1. A total of 20 tasks were evaluated in each independent test. The distributed setups utilized the Lynx P2P network while the Ocelot-only systems [84] used an additional central server to obtain input data and send results back to it. The time for completing a task in the Lynx network was much longer due to the slower Bluetooth protocol, especially when the number of hops in a route increases. Data compression helped to mitigate the file transfer time and as such, the total time. However, in the distributed WiFi-Direct test, the faster WiFi-Direct link also made the advantage of transferring smaller files less evident. As a result, data compression failed to save total time because of the overhead to compress and decompress files. When considering Ocelot alone (i.e. computing nodes connected to a central server with legacy WiFi connection), traditional computers with more powerful processors outperform mobile devices. Lynx widened the gap due to slower transmission speeds. However, mobile devices consume less energy doing the same amount of work as shown in Fig. 28. Without any powerful, yet power-hungry, central server, the fully distributed Ocelot system saved energy by 70% even in the 3-hop BT scenario, using data compression compared to the average computer consumption. Combining Ocelot with Lynx saved energy by 88% with data compression and 78% without compression, according to the comparison of the [2 distributed Nexus7, 1-hop BT] scenario with the Ocelot-only [Nexus 7 w/ server] scenario. Moreover, full distribution of Ocelot using Lynx provides the best flexibility in constructing WSNs while also providing a computation facility, which is not typical of traditional WSNs or traditional distributed computing systems like BOINC.

5.0 FUSEDCACHE: A NATURALLY INCLUSIVE, RACETRACK MEMORY, DUAL-LEVEL PRIVATE CACHE

5.1 FUSEDCACHE

5.1.1 Mapping and Addressing

An overview of the FusedCache concept is shown in Figure 29. FusedCache assumes a hybrid structure of an SRAM tag array [Figure 29 (a)], RM data array [Figure 29 (b)], and head position array [Figure 29 (c)], consistent with previously proposed RM cache designs that utilize "stay-in-place" head policies [16, 72]. Within the RM data array, I define the basic cache element as a "track bundle," which is a collection of B simultaneously controlled tracks [Figure 29 (d)] where B is the width of a cache line, typically 512-bits or 64-bytes. Thus, if the Racetrack length is R, then a bundle can hold R cache lines[1]. Next, the FusedCache is divided into "track groups" where, each group contains A bundles, where A is the associativity of the cache. Logically, each track group contains a single L1 set and R L2 sets.

The addressing of a FusedCache is shown in Figure 30(a). Excluding the cache line "offset" bits, the lower bits of the address provide a "group id", G, field to address each track group. A group consists of A track bundles that form R L2 sets. The L2 set within the group is identified by the "set id", S, field of the address. Figure 30(b) and (c) show the addressing of regular L1 and L2 caches, respectively. The T field of the FusedCache address is equivalent to the regular L2 cache tag field. Essentially, the FusedCache concept embeds

[1] I assume each racetrack has a single access port and all track lengths refer to the lengths of valid domains ignoring the additional cells at both ends of tracks required to prevent shifting overflow. Thus, R length racetracks actually contain $2R-1$ domains.

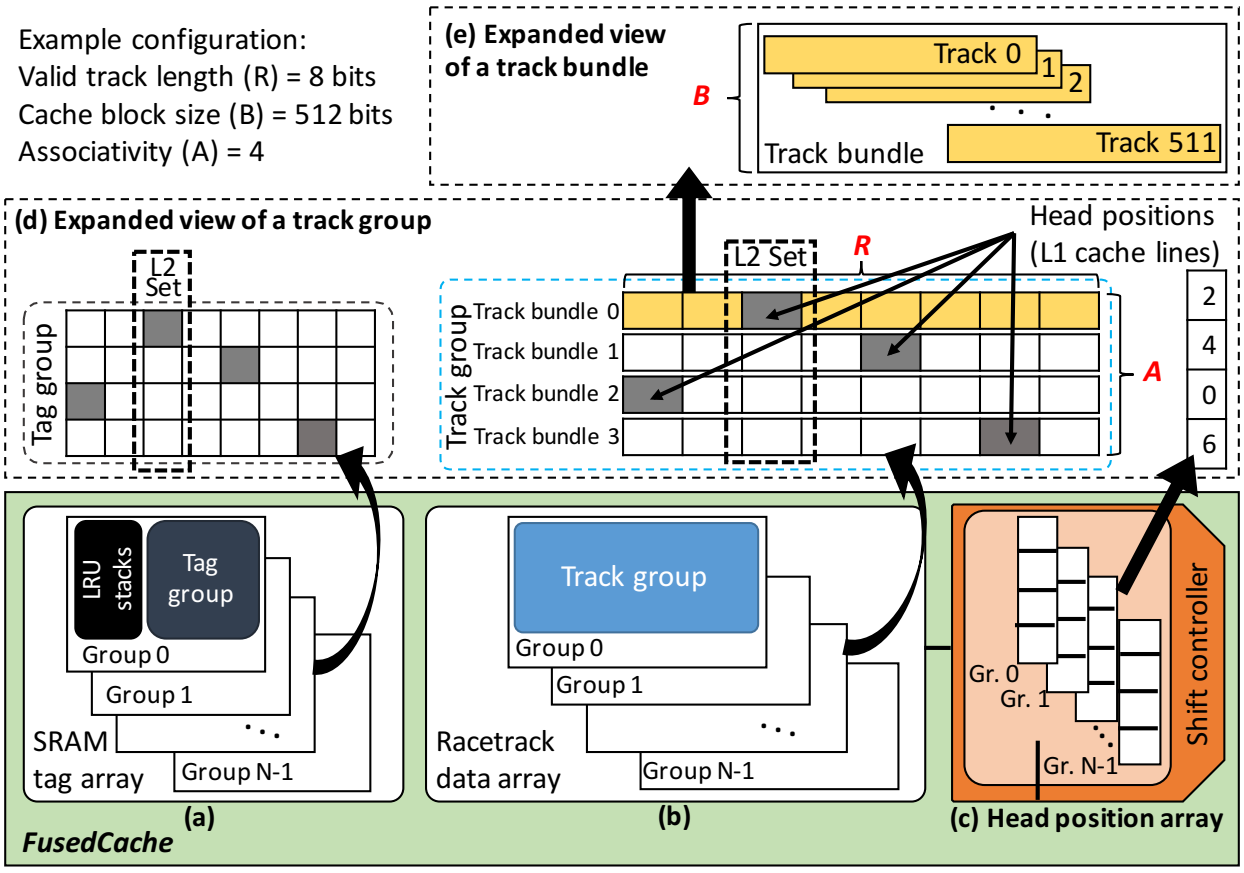


Figure 29: Overview of the FusedCache structure including an example group with parameters R=8, A=4. B=512.

into the L2 cache an L1 cache which is $1/R$ the size of the L2 cache. In other words, A of the $A*R$ cache lines that are in a track group form an L1 cache set as illustrated in Figure 29(d). These A L1 cache lines are the ones that are positioned under the access heads of the A bundles in the group. Thus, a FusedCache can serve an L1 or L2 lookup as follows:

- **L1 lookup:** T concatenated with S is equivalent to an L1 tag where G is equivalent to the regular L1 set index. This follows from the property that the data aligned with the access points within each track group functions as an L1 set in FusedCache.
- **L2 lookup:** T functions as an L2 tag and S concatenated with G is equivalent to the regular L2 set index as each track group contains R L2 sets.

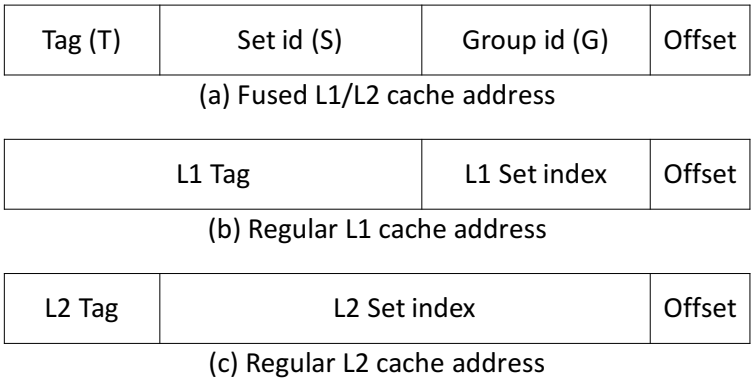


Figure 30: FusedCache addressing and indexing.

A more detailed illustration of a single track group is shown in Figure 29 (d) for a 4-way set associative cache, where each track group contains eight sets. Ways of the same set are mapped to different track bundles to ensure parallel access capability. In the example, four cache lines are identified as residing in L1 indicated in gray in both the tag and data arrays. These lines determine the alignment of the track bundle with the access point, which I refer to as the "head position". Thus, the L1 positions within the bundle are available in the corresponding head position storage. The four cache lines of each L2 set are located at the same index of each bundle. The set index $S = 2$ is highlighted for illustration in the figure. Any line within the set may be stored on any bundle. Thus, elements of the same set may be arbitrarily swapped. This is important to ensure that there are no L1 set conflicts when storing the L2 data as I described in the following section.

In this organization, access to an L1 line happens immediately and uniformly (without shifting), while access to an L2 cache line requires aligning (shifting) the access head to the L2 line's position on the bundle. Additionally, each group stores only the L2 tag array. The L2 tag match in FusedCache is the same as a regular cache tag match. L1 tags are constructed by concatenating the L2 tag with the head position indicator (set id) of that bundle as shown in Fig. 30. Hence, an L1 tag match is divided into two parts: matching T with the L2 tag stored, and matching S with the current head position. To enable the highest possible performance, parallel cache tag and data array accesses are assumed. Thus,

the read process starts in parallel with the tag matching process, while write operations wait until the tag is matched.

Given that each track group contains R L2 sets, I need R LRU stacks per group for determining which L2 cache line to evict upon a cache miss. Besides, an additional LRU stack is needed to determine the least recently used L1 cache line in the group. I denote the track LRU by LRU_{L1} for the following discussion. The details of LRU_{L1} maintenance are explained along with scenarios of cache hits and misses in the following section.

5.1.2 Handling cache hits/misses

To illustrate the process of set-associative cache accesses I use an example shown in Figure 31, which shows the structure of a track group G. The tag array for group G is represented by $\tau_{i,S}$ (the tag corresponding to the cache line stored at set S, way i), where $0 \leq i < A$ and $0 \leq S < R$, and the positions of the A heads in group G are denoted by $P_0, ..., P_{A-1}$, respectively[2]. If the line in bundle i at L2 set index S of group G is denoted by line (i, S), then lines (i, P_i), $(i = 0, ..., A-1)$ of group G form an associative set in L1. For example, in Figure 31, P_0=1, P_1=4, P_2=0 and P_3=6, which means that lines (0,1), (1,4), (2,0), and (3,6), denoted by `H`, `E`, `C`, and `Z` are in L1, while the remaining lines are in L2 but not in L1. The address of the cache line that is being accessed is represented by $< T, S, G >$ (ignoring the offset). Using this notation, the conditions of L1/L2 hits and misses can be expressed as follows:

- **L1 hits:** $T = \tau_{i,S}$ for some i $(0 \leq i < A)$ and $P_i = S$.
- **L1 misses, L2 hits:** $T = \tau_{i,S}$ for some i $(0 \leq i < A)$ and $P_i \neq S$.
- **L2 misses:** $T \neq \tau_{i,S}$ for any i.

In the direct-mapped configuration, accessing a cache line in FusedCache is the same as in TapeCache [16]. In this section, FusedCache's handling of accesses in the set-associative configuration is explained starting from a simple scenario where L1 and L2 have the same associativity.

[2]The following discussions are in the context of group G. For simplicity, G is removed from the notations when it is implicit in the context.

5.1.2.1 L1 hits In FusedCache, L1 hits require no shifts and have constant access latency. Fig. 31 shows an example of L1 hit access, in which the target L2 set id is 4. Tag matching determines `E` is the target cache line, which is also at the head location, and thus in the L1 set of this track group. Accessing `E` takes a single step of directly reading or writing the cache line, and updates the LRUs accordingly. All L1 cache lines of the accessed track group are read simultaneously with the tag match to ensure faster L1 reads, while writes must wait until tag matching is complete.

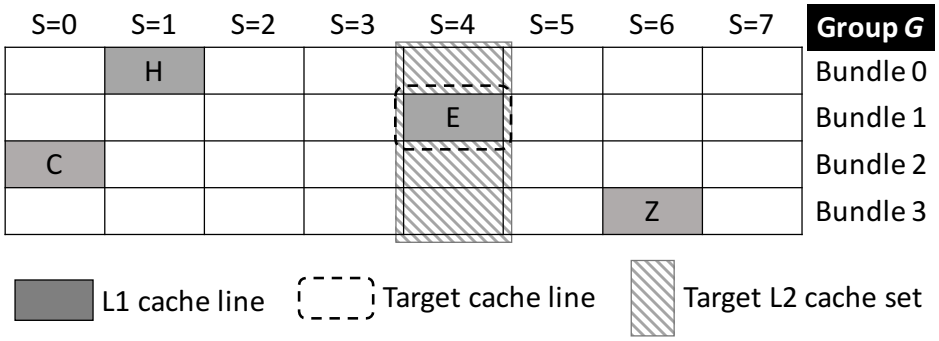


Figure 31: An L1 hit example.

5.1.2.2 L1 misses - L2 hits In this case, the requested line at address $< T, S, G >$ is located in the FusedCache at location (i, S) of group G with the head position of bundle i, P_i, not pointing to set S. For example, in Figure 32(a), if line $< T, 2, G >$ (set index $S = 2$) is accessed and T matches $\tau_{1,2}$ ($i = 1$, $S = 2$, tag corresponding to cache line `D`, stored at set 2, way 1) in group G, then the accessed line, `D`, which is in L2 but not L1, needs to be promoted to L1. This process depends on which of the L1 lines is the least recently used (indicated by LRU_{L1}). Given the head position of Bundle 1, $P_1 = 4$ is at cache line E, there are two cases:

• **Case 1:** The line at location (i, P_i) is the least recently used L1 cache line. In other words, (i, P_i) is the right cache line to evict from L1:

In this case, promoting the addressed line from L2 to L1 is accomplished by simply shifting bundle i to position S. In the example of Figure 32, if `E` is the least recently used L1 cache line, then replacing `E` with `D` in L1 is done by simply shifting the head for bundle 1 from position 4 to position 2, and updating the LRU stacks accordingly.

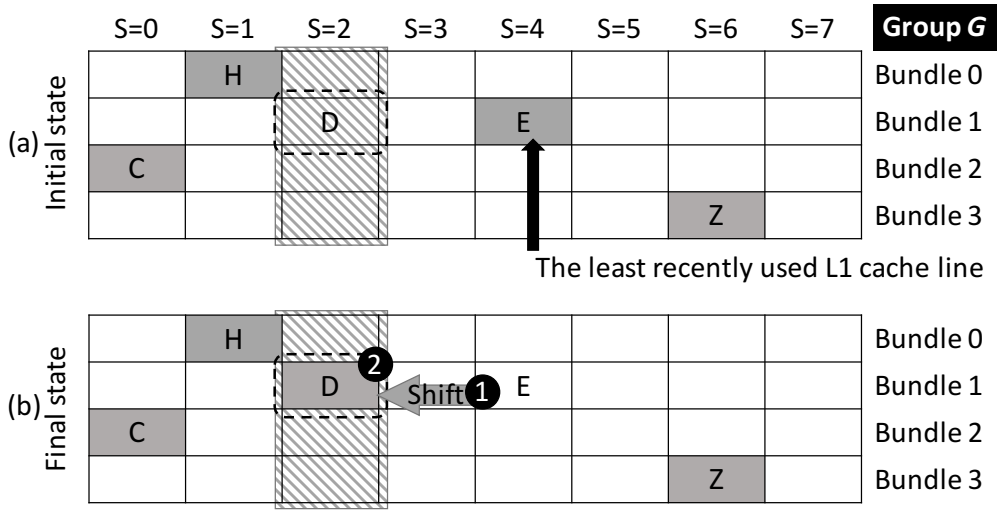


Figure 32: An L2 hit example w/o swap. The "Shift" arrow illustrates the relative position change of the access port. In actual RM, domains shift to the access port, which is logically equivalent.

• **Case 2:** Some line at location (j, P_j), $j \neq i$, is the least recently used L1 cache line. For example, `Z` is the right cache line to evict from L1:

In this case, promote the line at location (i, S) must be promoted to L1, while evicting the line at location (j, P_j) to L2. This can be done by first swapping the lines at locations (i, S) and (j, S), which is possible since they are both in the same associative set, and then shifting bundle j so that the line at location (j, S) is promoted to L1 while the line at location (j, P_j) is evicted from L1. In the example of Figure 33, if the line at (3,6), denoted `Z`, is at the head of LRU_{L1}, and the line at (1,3), denoted `D`, is accessed, then the process of promoting `D` to L1 and evicting `F` from L1 is described as follows:

1. Shift, simultaneously, the heads of bundle 1 and bundle 3 to position 2 [Figure 33(b)]
2. Access the requested line, `D` [Figure 33(b)]
3. Swap lines `D` and `F` [Figure 33(c)]
4. Shift the head of bundle 1 back to position 3 [Figure 33(c)]

During this process, the LRU stack for set 2 as well as LRU_{L1} should be updated to reflect the new access order. Note that although only the shift of bundle 1 and access of

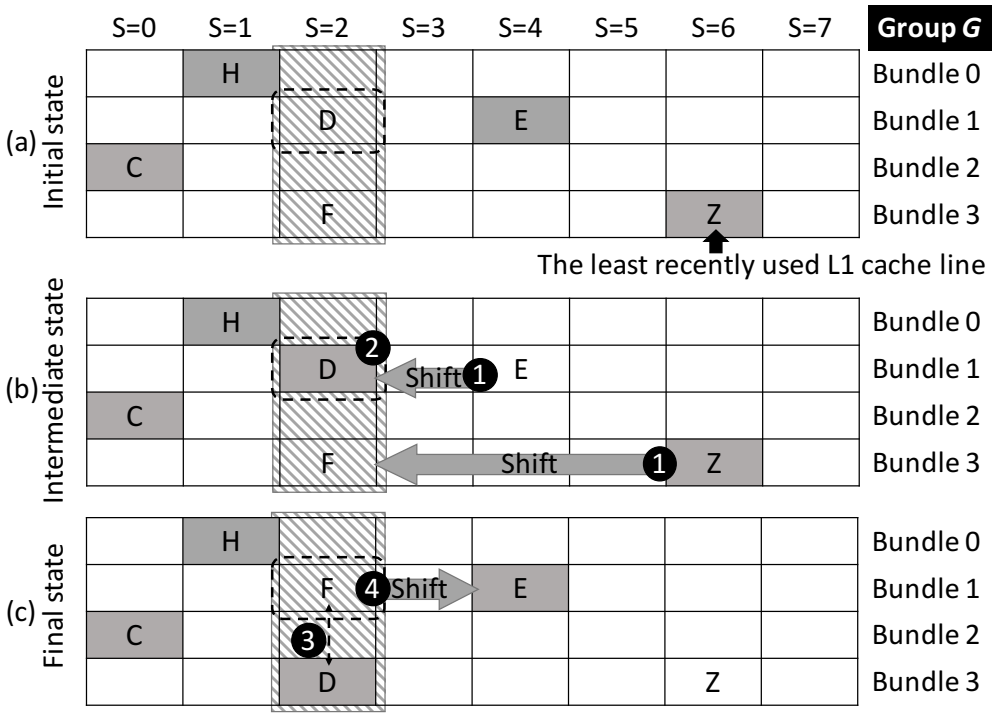


Figure 33: An L2 hit example w/ swap.

D are on the critical path of the cache access, the evaluation assumes that all the bundles in the group are not available for subsequent accesses until the entire promotion/eviction process completes.

5.1.2.3 L2 misses In this scenario, a new cache line is loaded into FusedCache from higher levels of caches or memories, and replaces the least recently used line in set S of L2, denoted as (i, S). To also place the new line in L1, the least recently used L1 cache line has to be evicted from L1 to L2 by shifts. Similar to the previous discussion on L1 and L2 hits, there are three cases:

• **Case 1:** The least recently used line of the target L2 set S is at way (bundle) i and aligned with the head ($P_i = S$). In other words, (i, S) is the right cache line to evict from L2 (also L1[3]), and to be replaced by the new cache line, without any shift operations.

[3]It can be easily proved with contradiction that the least recently used L2 cache line, if it happens to be in L1, is also the least recently used L1 cache line.

Similar to the example shown in Fig. 31, `E` is the cache line at location (i, S) (in this case, $i = 1$, $S = 4$). If `E` is the least recently used L2 cache line in set 4 then the access replaces `E` with the newly brought-in cache line from a higher level of the memory hierarchy, and moves the new cache line in `E` to the tails of both the L2 set LRU and LRU_{L1} stacks (`E` becomes the most recently used cache line).

• **Case 2:** The least recently used line of the target L2 set S is at way (bundle) i and not aligned with the head ($P_i \neq S$). In other words, (i, S) is the cache line to be evicted from L2, while according to LRU_{L1}, (i, P_i) is the cache line to be evicted from L1. In this case, compared to case 1, an additional shift operation is needed.

Similar to the example shown in Fig. 32, `D` is the cache line at location (i, S) (in this case, $i = 1$, $S = 2$). `D` is now the least recently used L2 cache line in set 2 instead of target cache line, thus to evict from L2. `E` is the least recently used L1 cache line in the group G. Thus, the access process includes, firstly, shifting the head of bundle 1 from `E` to `D`, then replacing `D` with the new cache line. Finally, the new cache line in `D` is moved to the tails of both L2 set LRU and LRU_{L1}.

• **Case 3:** The least recently used line of the target L2 set S is at way (bundle) i, and not aligned with the head ($P_i \neq S$), and some line at location (j, P_j), $j \neq i$, is the least recently used L1 cache line. In this case, I want to evict (i, S) from L2, evict (j, P_j) from L1, and place the new cache line in L1 at location (i, P_i).

Similar to the example shown in Fig. 33, `D` is the cache line at location (i, S) (in this case, $i = 1$, $S = 2$). `D` is now the least recently used L2 cache line in set 2 and should be evicted. `Z` is the cache line at location (j, P_j) (in this case, $j = 3$, $P_j = 6$), and is the least recently used L1 cache line in group G and thus, is to be evicted from L1. The access process resembles the four steps in Section 5.1.2.2 Case 2, except that in step 2, `D` is now replaced by the new cache line. Finally, the new cache line in `D` is moved to the tails of both L2 set LRU and LRU_{L1}.

5.1.3 Hybrid associativity

FusedCache is also compatible with heterogeneous L1 and L2 associativity configurations, called *hybrid associativity*. For this case, mapping from virtual to physical space is slightly different from what is discussed above. Specifically, assuming that A_{L1} and A_{L2} are the associativities of L1 and L2, respectively, then instead of spreading ways across different tracks, each track accommodates $\frac{A_{L2}}{A_{L1}}$ ways of a same set. This presumes that A_{L2} is an integer multiple of A_{L1}, which is normally the case in cache hierarchies

For example, a 4-way set associative L1 and 8-way L2 configuration is shown in Figure 34, with track length set as 8. In this figure, S_mW_n represents way n in set m. Cells of the same color are in the same L2 cache set. Each track accommodates two ways of a same set but still has only one L1 cache line. Thus, all the cache and Racetrack operations described for homogeneous associativity configurations may be applied with a simple modification. As cache lines in the same set are swappable and an L2 set no longer occupies a single track position in all track bundles, it becomes possible to conduct a swap operation between different ways depending on the scenario. For example, to swap S_1W_4 with an element from the same set in bundle 2, the cache could swap S_1W_2 if the L1 currently contains an element from L2 set S_0 or S_1W_6 if the L1 currently contains an element from sets S_2 or S_3.

S_0W_0	S_0W_4	S_1W_0	S_1W_4	S_2W_0	S_2W_4	S_3W_0	S_3W_4
S_0W_1	S_0W_5	S_1W_1	S_1W_5	S_2W_1	S_2W_5	S_3W_1	S_3W_5
S_0W_2	S_0W_6	S_1W_2	S_1W_6	S_2W_2	S_2W_6	S_3W_2	S_3W_6
S_0W_3	S_0W_7	S_1W_3	S_1W_7	S_2W_3	S_2W_7	S_3W_3	S_3W_7

Figure 34: An hybrid associativity mapping example.

The aforementioned swap operation requires two cache lines to be in the same set thus the same column of the physical mapping. In hybrid asssociativity configurations, ways of the same set can stretch across different columns and swaps can happen between two ways in different columns. Fig. 35 illustrates an example of L2 hit in the 4-way L1 and 8-way L2 hybrid associativity configuration. The third and fourth columns together contain eight ways of set 1. Similar to the discussion of "Case 2" in Section 5.1.2.2, in order to access

D, head of bundle 1 must shift from E to D, and after the access shift back to E, as E is not supposed to be evicted from L1. The least recently used cache line Z must be evicted instead. In this example, head of bundle 3 no longer needs to shift from Z to X because Y is also in the same set thus swappable with D, and requires one step less shift.

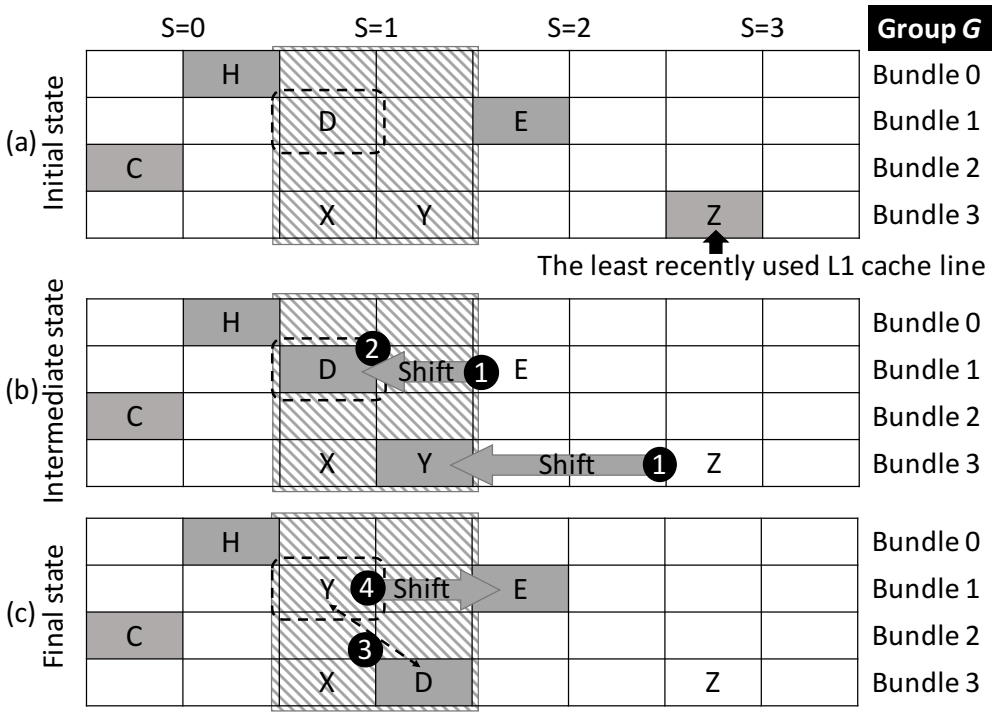


Figure 35: An L2 hit example w/ swap in a hybrid associativity configuration.

5.2 EXPERIMENTAL METHODOLOGY

The fundamental goal of my efforts is to replace traditional SRAM caches with shift-based RM caches. Given the TapeCache proposal to use Racetrack memory for the last-level cache, the static power of the cache hierarchy is greatly reduced and becomes dominated by SRAM private caches near the core (e.g., L1 and L2) in spite of their relatively small capacity compared to the LLC. Thus, I apply RM at all cache levels for the FusedCache scheme to evaluate the effectiveness of my design. I compare my design with three schemes: SRAM-1, SRAM-2, and TapeCache. SRAM1 and SRAM2 employ a private 32KB, 8-way set-associative (SA) SRAM L1, while SRAM2 adds an additional 512KB, 8-way SA, SRAM

Table 3: Latency & energy parameters

Schemes	SRAM-1	SRAM-2		Tape/FusedCache	(Global)
	L1	L1	L2	L1/L2	LLC
Tag access latency (ns)	0.16		0.28	0.28	0.34
Data read latency (ns)	0.94		1.87	0.98	1.21
Data write latency (ns)	0.94		1.87	0.65	0.65
Read energy (pJ/access[4])	6.78		54.49	19.08	61.17
Write energy (pJ/access)	3.76		21.91	7.94	7.94
Leakage power (mW)	65.66		138.49	19.40	25.98
Tag area (mm^2)	0.00443		0.045	0.045	-
Data area (mm^2)	0.0914		1.407	0.0491	-
Total area (mm^2)	0.09583		1.452	0.0941	-
Shift latency (ns)	0.322				
Shift energy (pJ/cell)	0.0617				

L2. TapeCache uses a 512K 4-way SA L1 from single ported RMs, as described in previous work [16, 72]. All schemes use a 4MB, 16-way SA shared unified TapeCache LLC. All caches use a 64-byte line size and LRU replacement. FusedCache and TapeCache are iso-area RM replacements of SRAM-1, and iso-capacity RM replacements of SRAM-2.

I extended the Sniper full system simulator [100] to include a Racetrack FusedCache and a Racetrack LLC. To simulate the impact of the Racetrack structure in Sniper, the state of every track is maintained to determine the access latency based on the actual shift latency. This implementation models the impact of contention between temporally overlapping accesses to cache lines mapped to the same tracks and determines the accesses that can proceed in parallel in the same subarray.

I use NVSim [101] to model the latency, power, and area parameters of SRAM caches including both the tag array and data array, and a modified NVSim to model RM data array.

[4]Each access contains one line, or 512 bits/cells.

Table 4: Data array read latency breakdown

Latency (ps)	H-tree	Predecoder	Row decoder	Bitline	Sense Amp	Mux
32K SRAM	363.58	63.25	146.01	154.86	139.85	72.69
512K SRAM	1150.11	81.40	231.93	174.27	139.85	93.03
512K RM	273.91	97.68	166.78	161.30	226.29	53.91

The area of one RM track is assumed to be dominated by the area of access transistors due to its 3-D structure and adopt a cell size of $2F^2$ based on previous work [102]. RM shift latency and energy are calculated based on a size $120*720nm^2$ perpendicular magnetic anisotropy (PMA) nanowire [103]. RM reads and writes faster than SRAM due to its smaller peripheral circuitry. Rather than current-based writes, I adopt the more efficient shift-based writes proposed in [15]. Detailed parameters for each memory type based on a 32nm technology feature size are listed in Tables 3 and 4.

For simulation workloads, I used 10 multi-threaded (MT) parallel applications and 7 groups of single threaded applications as multi-programmed (MP) workloads drawn from the Parsec [104] and SPEC-CPU2006 [105] benchmark suites, summarized in Table 5, which also includes each individual application's miss per kilo instructions (MPKI) of L1 cache. Each MT application is executed with four threads and each MP group consists of four applications. Characterization is started from the "region of interest" (ROI) with one billion instructions to warm up the cache followed by one billion for evaluation. For MP workloads, the number of instructions is the sum of all four applications within a group, which means cores running faster applications execute more instructions during the simulation.

5.3 RESULTS

In this section, I evaluate the feasibility of replacing SRAM with RM in the entire cache hierarchy and demonstrate FusedCache's capability of saving energy while providing com-

Table 5: Benchmarks (L1 MPKI)

Parsec: multi-threaded workloads			
MT-1	blackscholes (0.74)	MT-2	canneal (24.08)
MT-3	dedup (26.26)	MT-4	facesim (25.58)
MT-5	ferret (31.15)	MT-6	fluidanimate (8.40)
MT-7	freqmine (12.01)	MT-8	raytrace (6.34)
MT-9	streamcluster (9.63)	MT-10	vips (13.42)

SPEC-CPU2006: multi-programmed workloads	
MP-1	perlbench(48.45), bzip2(40.21), gcc(26.66), bwaves(109.46)
MP-2	gamess(14.97), mcf(8.55), milc(35.03), zeusmp(38.87)
MP-3	gromacs(2.67), cactusADM(30.26), leslie3d(19.96), namd(27.84)
MP-4	gobmk(44.37), dealII(83.17), soplex(34.61), povray(51.99)
MP-5	calculix(25.15), hmmer(90.70), sjeng(87.55), GemsFDTD(5.5)
MP-6	libquantum(7.25), h264ref(21.90), tonto(17.46), lbm(58.12)
MP-7	omnetpp(29.38), astar(15.12), wrf(7.90), sphinx3(12.63)

parable performance with traditional private L1 and L2 SRAM cache systems (SRAM-2). Compared to a private L1 SRAM cache (SRAM-1), in the same physical area FusedCache provides a "free" private L2, which helps to improve performance and reduce energy due to RM's low leakage power.

5.3.1 Performance

Figure 36 shows the system IPC (higher is better) comparison of the four different cache schemes. On average, MT applications show a greater than 5% speedup through FusedCache compared to SRAM-1, more than 10% with MP workloads, and overall better than 7% across all workloads. Most MT applications have low L1 miss rates (less than 10%), which limits the value of the extra L2 cache's performance contribution. MP workloads have larger

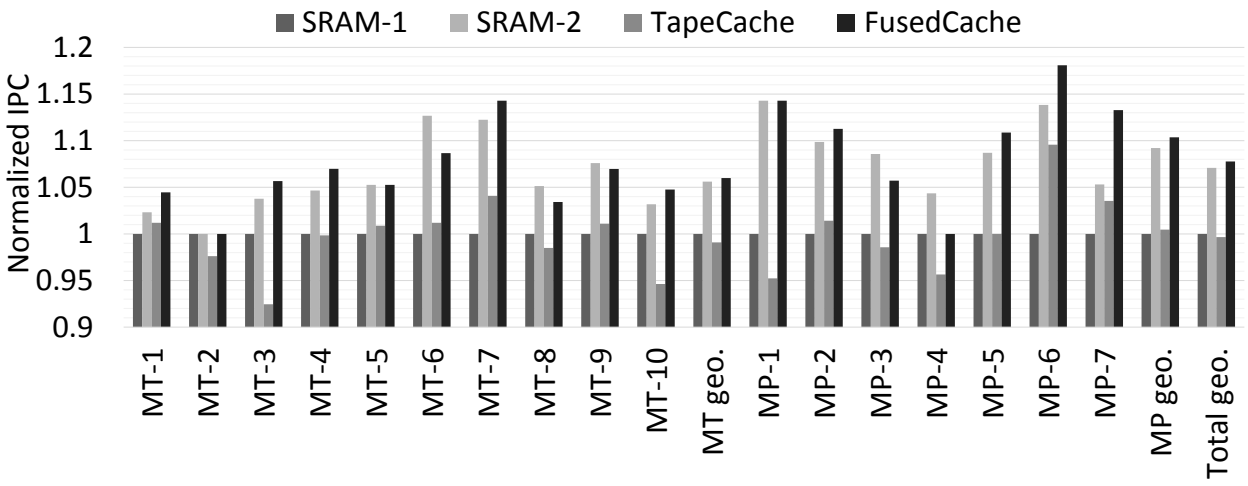


Figure 36: Performance comparison of four cache schemes.

L1 miss variation. FusedCache L2 accesses can be faster than SRAM L2 (e.g., smaller peripheral circuitry), although shift overheads from port contention can reduce the overall access speed of cache lines in FusedCache. During these fused L2 accesses, swap operations do not add latency directly to the critical path (Section 5.1). My simulator records the track busy time for every shift as required by swapping. In contrast to FusedCache, the performance of a TapeCache L1 with 512K capacity is almost indistinguishable from SRAM-1 in spite of the capacity advantage. This is due to an 87% increase in shifting latency over FusedCache for L1 hits. Generally, FusedCache outperforms SRAM-2 for both MT and MP workloads. However, in certain applications (MT-6,8,9 and MP-3,4), SRAM-2 avoids significant FusedCache shift contention. On average, shifting adds about three cycles for each L2 access in these workloads. FusedCache is particularly beneficial for workloads MT-7, and MP-1,6, achieving between 15-18% improvement due to relatively high L1 miss rates and the extra fused L2 cache. Applications such as MT-1,9 and MP-2 see smaller performance improvements as a result of already low L1 miss rates (under 5%), although nearly all L1 misses hit in the fused L2 cache.

5.3.2 Energy

Figure 37 shows the total energy consumption comparison of all levels of caches, both leakage and dynamic. To ensure a fair comparison, the LLC energy is also considered, as schemes with and without private L2 caches have different LLC access behavior. In general, the extra L2 in FusedCache and SRAM-2 helps to reduce LLC accesses. For the FusedCache, two additional power categories, shifts and swaps, are included. All detailed values used for calculation are listed in Table 3. SRAM-2 experiences a severe energy increase from adding a private L2 due to the additional SRAM static power. On average, it consumes more than twice SRAM-1's energy to gain a 7% speedup. In contrast, while providing a similar performance gain to SRAM-2, FusedCache reduces energy over SRAM-1 by one third. Compared to SRAM-2, FusedCache reduces cache energy by 70% on average, and by up to 80% with certain workloads (e.g., MT-2). Compared to SRAM-1, FusedCache reduces energy by 28% and 41% for MT and MP workloads, respectively. Moreover, FusedCache actually saves energy over TapeCache (6%) while outperforming SRAM-2. As shown in Fig 38, in terms of energy-delay-product (EDP), FusedCache achieves a 39%, 70%, and 13% improvement compared to SRAM-1, SRAM-2, and TapeCache, respectively.

Figure 39 shows the breakdown of six power components in FusedCache, where LLC is the sum of all dynamic, leakage, and shift energies of the last level RM cache. The leakage energy of the SRAM tag array and other peripheral circuitry in FusedCache and LLC accounts for 30% and 42% of the total energy consumption on average. Of the remaining components, reads, writes, shifts, and swaps take up to 17%, 2%, 6%, and 3%, respectively. Amongst all 17 workloads, MT-9 has the smallest energy saving with FusedCache and the lowest overhead with SRAM-2 because for this workload, the read dynamic energy dominates the overall consumption, minimizing the leakage advantage.

5.3.3 Sensitivity study

Besides the experiments with the typical settings listed in Section 5.2, I also conduct a sensitivity study to see how FusedCache performs against the other schemes in scenarios with different combination of cache parameters, including capacity, Racetrack length (L2/L1

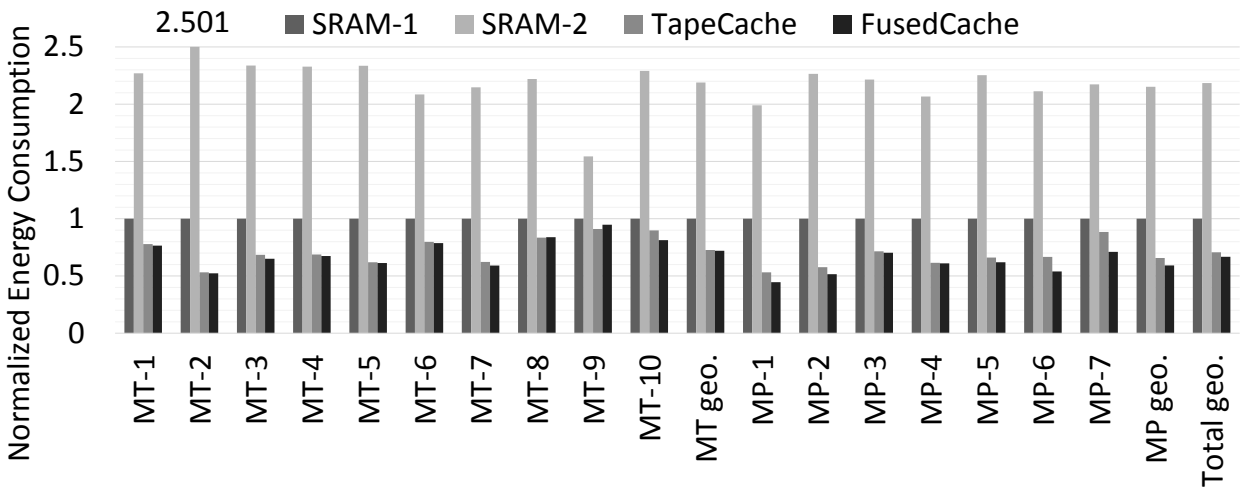


Figure 37: Energy Consumption comparison of four cache schemes.

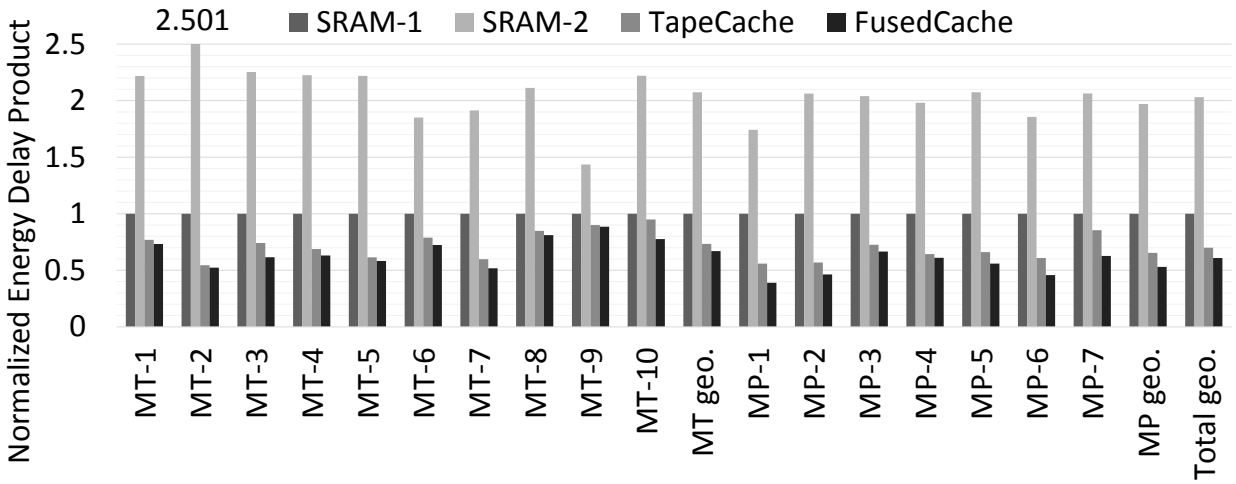


Figure 38: Energy Delay Product (EDP) comparison of four cache schemes.

capacity ratio) and associativity. The results are presented with the geometric mean of normalized IPCs and energy consumption of the 17 applications listed in Table 5.

5.3.3.1 L1 capacity Three scenarios in which the L2/L1 capacity ratio were held constant with different L1 capacities were tested. Note that in the FusedCache scheme, the

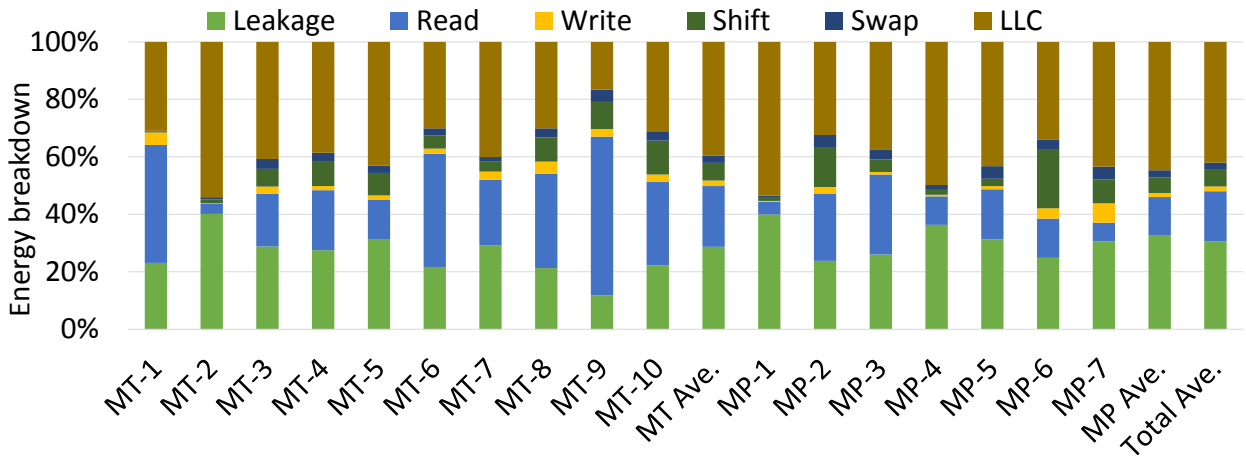


Figure 39: Energy Breakdown of FusedCache scheme (LLC indicates overall energy consumption of the last level cache).

L2/L1 capacity ratio is determined by and equal to the Racetrack length (e.g., a 32KB L1 cache with a 16-bit long Racetrack includes a 512KB L2 cache). Figure 40 shows performance and energy comparison of the three schemes with different L1 capacities: 16KB, 32KB, and 64KB, respectively, with a 32KB SRAM-1 scheme as the baseline. L1 capacity misses increase with the smaller L1 cache (16 KB). When L1 capacity is reduced, the remaining three schemes gain a larger performance advantage over the baseline SRAM-1. The TapeCache configuration increases the L1 capacity, dramatically reducing the L1 miss rate sufficiently to compensate for the shifting latency. The extra L2 capacity of SRAM-2 and FusedCache prevent the high percentage of L1 misses from being serviced by the significantly slower LLC. For the 64K L1, the TapeCache performs poorly due to the shifting delay per access compared to SRAM-1. For all schemes and configurations, FusedCache shows the best performance. As expected, the energy savings of FusedCache compared to two SRAM schemes is related to the overall size of the storage with a smaller advantage for the 16KB SRAM-1 due to smaller leakage power gaps between SRAM and RM data arrays and dramatic (2X) advantage for 64KB SRAM-1.

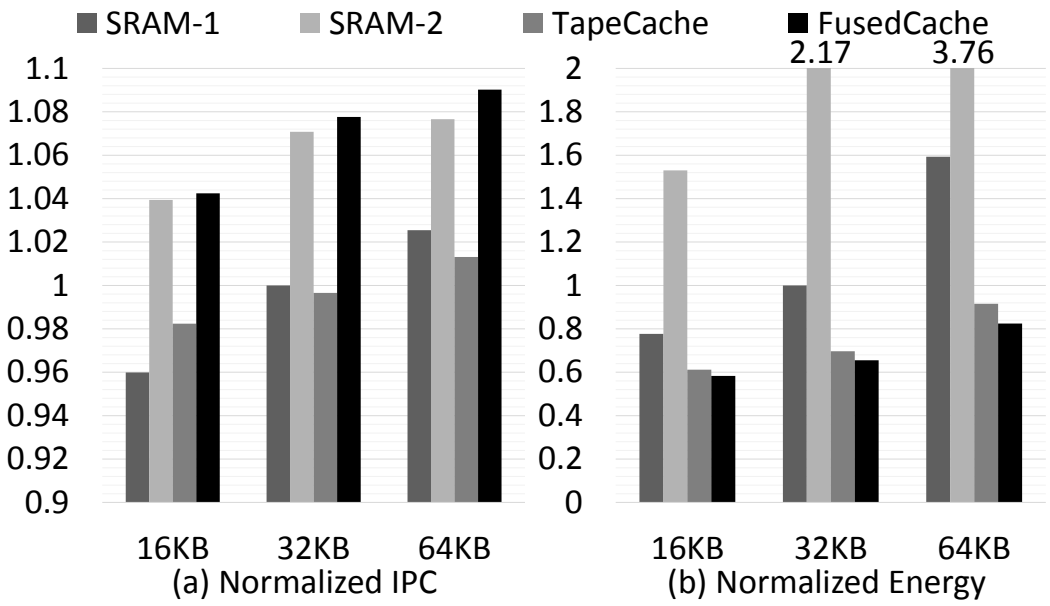


Figure 40: Average IPC and energy consumption: Constant L2/L1 capacity ratio with various L1 cache capacities.

5.3.3.2 L2/L1 capacity ratio As mentioned in Section 5.2, L2/L1 capacity ratio in FusedCache is determined by the physical length of Racetrack. In this study I examine the impact of changing the Racetrack length while keeping L1 size constant at 32KB. The baseline length of 16 means that the L2 capacity is 512KB. Since the L1 capacity stays constant, SRAM-1 offers the same performance and energy consumption across all three scenarios, as shown in Figure 41. In the 8-bit long Racetrack scenario, the smaller L2 capacity leads to lower L2 access latency, which helps SRAM-2 to outperform FusedCache. Moreover, the smaller L2 results in more contention for the single access port in each RM subarray. Leakage power of the single ported Racetrack data array does not dramatically increase when longer tracks are used as it is dominated by the shift and access transistors rather than the magnetic nanowire. FusedCache's iso-capacity replacement SRAM-2, however, suffers from significant energy increase with larger L2 caches. In the length 32 scenario, FusedCache provides slightly better performance while consuming only a quarter of SRAM-2's energy.

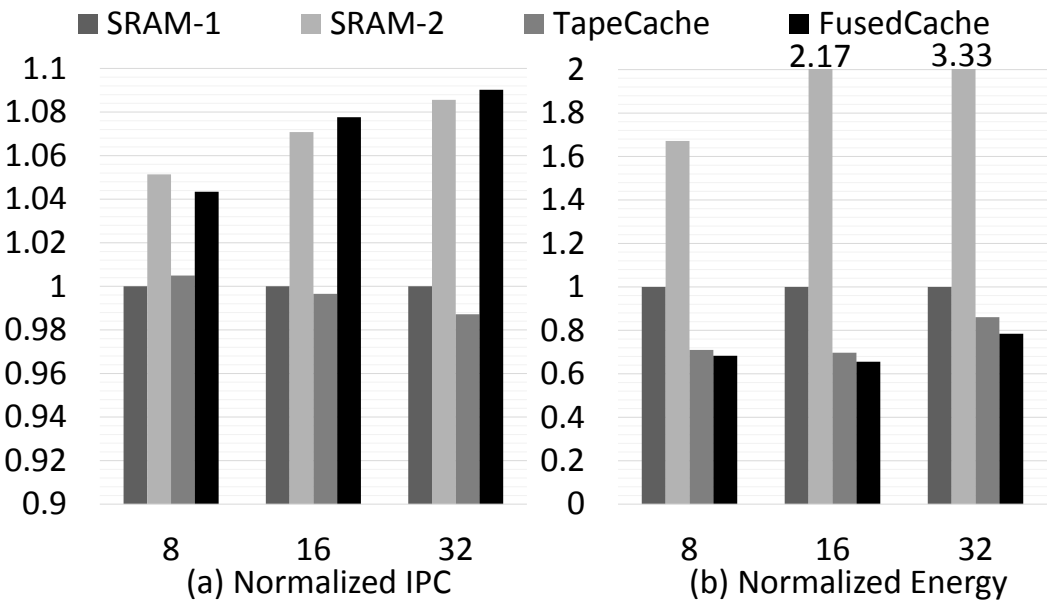


Figure 41: Average IPC and energy consumption: constant L1 capacity and different L2/L1 capacity ratios.

5.3.3.3 Associativity This section presents result comparisons of various L1 and L2 associativity combinations, using the hybrid associativity scheme discussed in Section 5.1.3. The result is shown in Figure 42, where **m-n** represents a configuration with m-way set associative L1 and n-way set associative L2[5]. FusedCache shows its capability of handling the hybrid associativity configuration (e.g., 4-8), and provides similar performance, although the mapping may increase the shift overhead. For example, shifting to other sets, particularly non-adjacent sets, can often require more shifts because multiple ways are stored in consecutive domains along the same track bundle. FusedCache also achieves a better performance improvement compared to SRAM-1 in the direct mapped configuration than set associative configurations, due to the parallel tag and data access hiding some shift overhead. Energy consumption remains similar across all FusedCache configurations.

[5]For SRAM-1 and TapeCache schemes, m-n simply refers to a one-level m-way set associative cache.

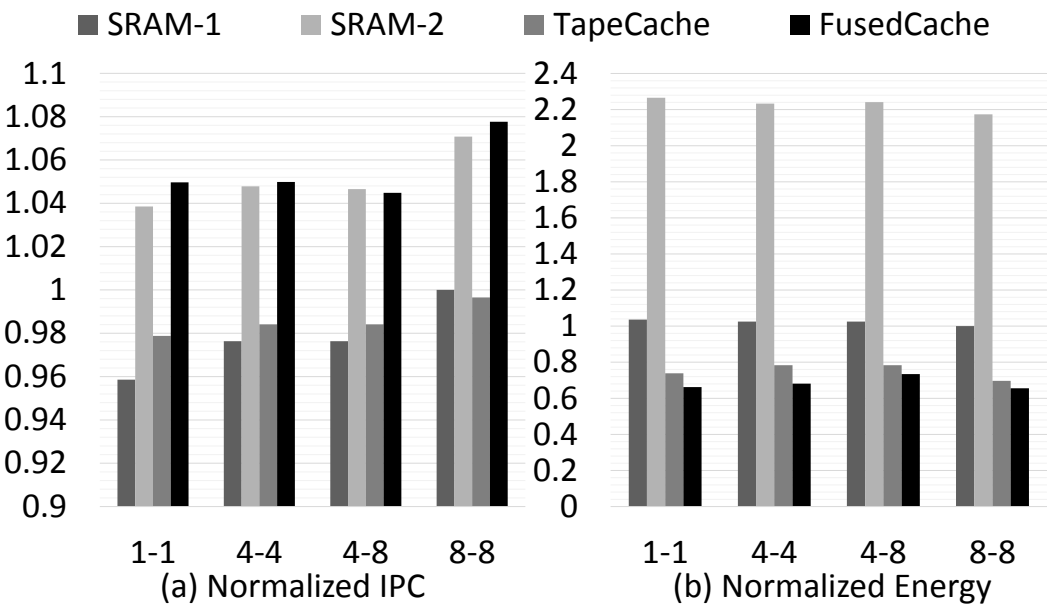


Figure 42: Average IPC and energy consumption: constant L1 capacity and L2/L1 capacity ratios and different L1, L2 associativity combinations.

6.0 MULTILANE RACETRACK CACHES: IMPROVING EFFICIENCY THROUGH COMPRESSION AND INDEPENDENT SHIFTING

6.1 MRC LAST LEVEL CACHE DESIGN

To describe the proposed MRC scheme I begin with a preliminary discussion of the fundamental organization of the RM in MRC, which is consistent with previous work in RM caches. My contributions are then described, and their cooperation within MRC to achieve better performance and energy efficiency for RM cache designs is discussed.

6.1.1 Fundamental Structure

Fig. 43 illustrates the MRC structure. MRC organizes the cache in a similar hybrid organization as FusedCache using SRAM instead of RM for the tag array. First, the tag array is more frequently accessed than the data array due to cache misses and invalidations. Using SRAM prevents a performance degradation due to shifting during the tag lookup. Unlike the one-to-one bit-per-storage circuit that simplifies the array-based organization of SRAM, DRAM, or STT-MRAM, RM's many-to-one bit-per-storage circuit adds another dimension that complicates how logical structures are mapped to the physical structure including cache lines and cache sets. As introduced in Section 2.3, Fig. 6 shows the smallest repeating unit in RM, which can store tens to hundreds of bits [12], all of which share the same access port. Thus, to avoid port contention within a single access, I adopt the same bit-interleaved mapping of TapeCache [16], which is different the FusedCache mapping as described in Section 5.1.

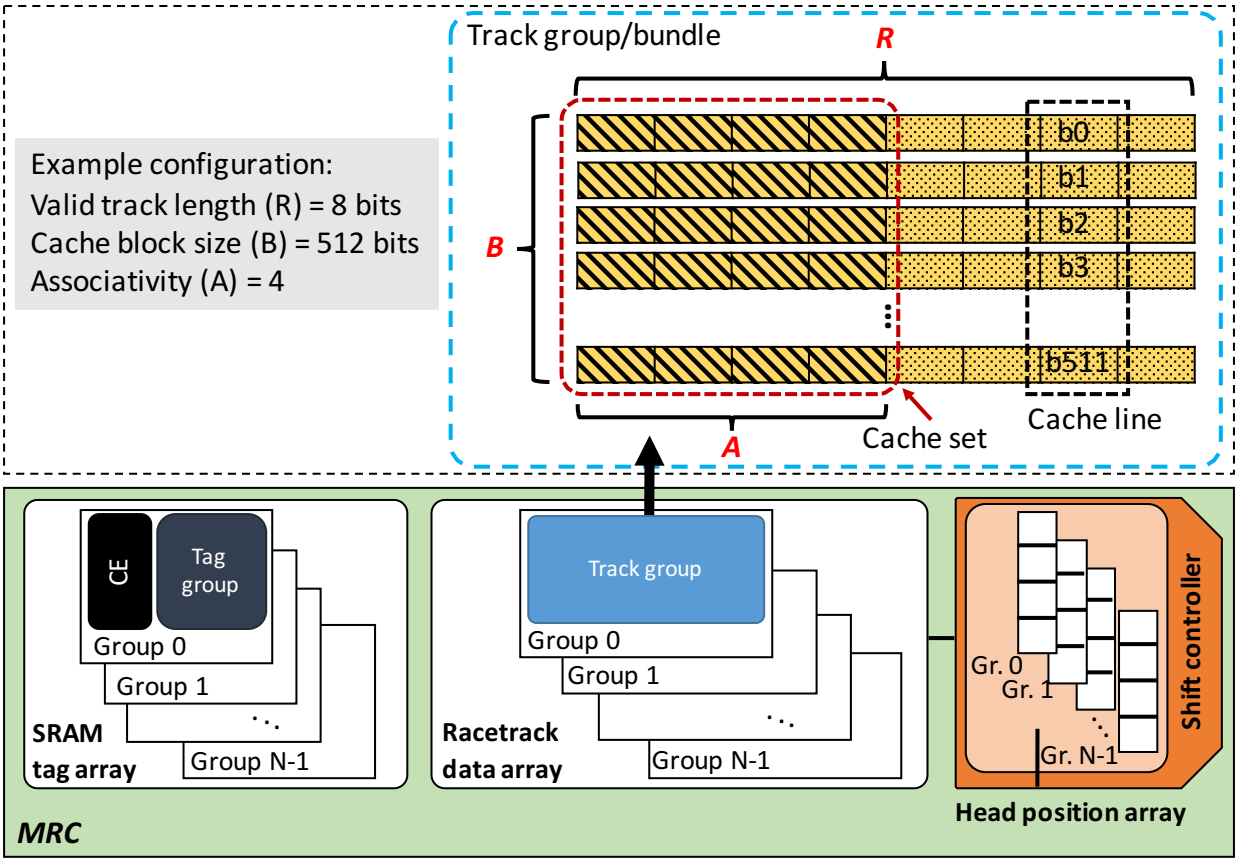


Figure 43: Overview of the MRC structure including an example group with parameters R=8, A=4. B=512.

The upper section of Fig. 43 illustrates the expanded view of an RM "track group", which contains only one "track bundle" defined in Section 5.1. Columns represent cache lines, while rows of the track group are physical tracks. The example shows a track length of 16 with 512 (64*8, i.e., the width of a cache line) parallel tracks in this subarray. $Track_i$ contains the ith bit of each of 16 cache lines stored in the subarray. For simplicity, the track length I refer to in this dissertation refers to the number of valid bits not including auxiliary bits required for preventing data from shift overflow.

Additionally, the area and energy overhead of SRAM tags over RM tags is relatively small since the tag array contributes only 4.8% to the total area of a 1MB cache [16]. This is also consistent with my modeling results summarized in Table 8.

As discussed in prior work [16, 17], there are two typical methods for managing the locational relationship between domains and the access port after each access:

- **Return:** The data stored within the nanowire is returned to its original (home) location after each access. This policy requires no extra logic to store location information but is likely to require more shifts.
- **Stay:** No shift after each access, the data remains in the shifted position until the next access arrives. This policy requires additional logic to store the shift position relative to the home location.

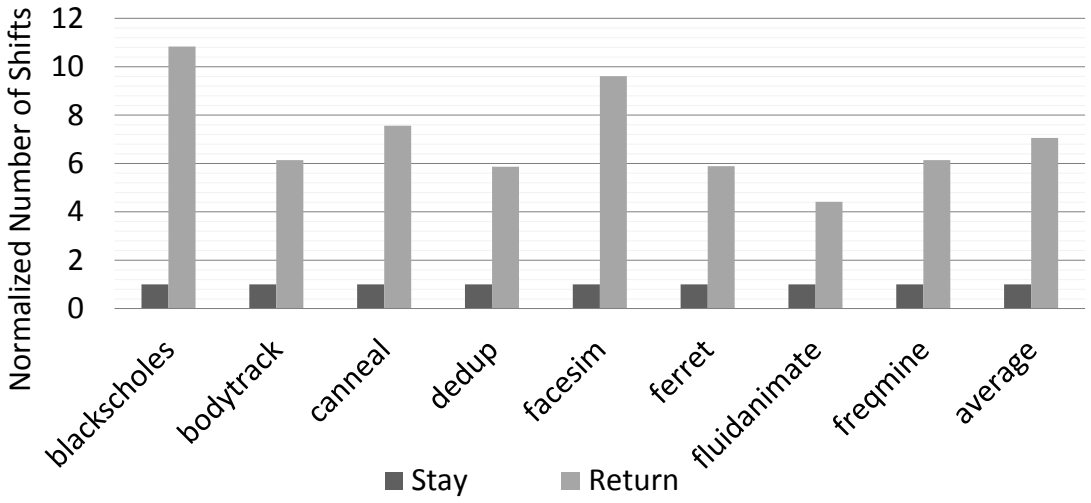


Figure 44: Number of shifts for a 1MB 1P1T Racetrack LLC with track length of 16 (detailed parameters from Tables 7 and 8) with *Return* shifting policy normalized to the same design with *Stay* shifting policy.

Fig. 44 shows the comparison of number of shifts for a subset of Parsec benchmarks [104]. The Return policy results in 7x more shifts than Stay. This result is not unexpected as for 1P1T Racetrack structures the Return policy will typically require a much longer shift distance than would be the case in MP1T structures. As a result, the shift energy overhead would be unacceptably large if the Return policy is employed as the dynamic shift energy using the Stay shift policy already approaches 20% of the overall energy (see Fig. 49). Therefore the MRC LLC design only considers the Stay shift policy.

Table 6: 14 compression schemes (including uncompressed) ordered by sizes after compressing a 64 byte cache line

Scheme	ZERO	REP1	REP2	REP4	REP8
Size (B)	0	1	2	4	8
Scheme	B8D1	REP16	B4D1	B8D2	REP32
Size (B)	15	16	19	22	32
Scheme	B2D1	B4D2	B8D4	UNCOMPRESSED	
Size (B)	33	34	36	64	

6.1.2 Memory In-place Compression

As compression and decompression require energy overheads and the latter lies on the critical path of reading compressed data, I choose lightweight compression methods including ZERO [106], Repeated Values [107], and BDI [80] over more complicated ones with better compression ratios. Unlike previous cache compression schemes which attempt to increase effective cache capacity [80] I use compression to save read, write, and shift energy while accelerating accesses to the RM LLC. A compressed cache line has fewer bits to write than the uncompressed line, and thus requires less energy to write and read. With in-place compression, this energy saving is accomplished without complicating the addressing mechanism as the storage capacity for an uncompressed line remains allocated to each cache line (64 Bytes in this case) even when only a fraction of that space is needed to store a compressed line. Details of the compression algorithms have been explained in Section 7.1.1.

Initially, 14 effective compression schemes from the literature [106, 107, 80] were evaluated from in-place compression. Table 6 shows a breakdown of the compression effectiveness for subset of the benchmark applications. Zero compression applies when the data is zero. If the data repeats every x bytes then it can be compressed by the REPx scheme. If I divide a cache line into y-byte subblocks and all offsets between the first subblock with each of the remaining subblocks being within a z-byte range, then the line is compressible by the

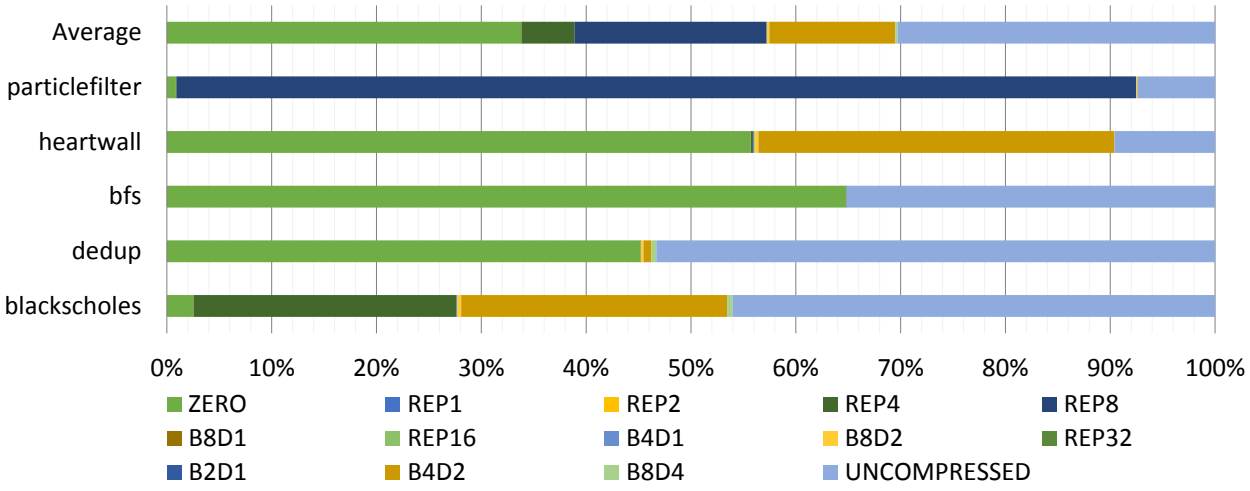


Figure 45: Breakdown of occurrences of 14 compression schemes (including uncompressed) in 5 applications with small input sets.

ByDz (y-byte base, z-byte delta) scheme. In many cases, a cache line may be compressed by multiple schemes. Of the valid schemes, the results report the compression scheme with the smallest size by prioritizing the 14 schemes by compression ratio.

Many schemes can, of course, be included to achieve the best possible overall compression ratio without affecting the performance, as all compression units can function in parallel [80]. However, each compression/decompression scheme requires additional hardware and energy overhead. Further, as shown in Table 6, several schemes have similar compressed sizes and Fig. 45 indicates that several schemes are rarely applied. Thus I select the four dominating compression schemes (ZERO, REP4, REP8, B4D2) to go with uncompressed data to be used in the MRC cache. To store the compression mechanism and to represent the five compression states, I add three extra bits to each element of the tag array. I modify the LLC tag structure to accommodate the extra bits as shown in Fig. 46. The *compression scheme detector* selects the most effective of the valid schemes (or leaves the data uncompressed) and generates the compression code stored in the modified tag structure. The *block aligner* is added to the flow when skewed alignment in considered.

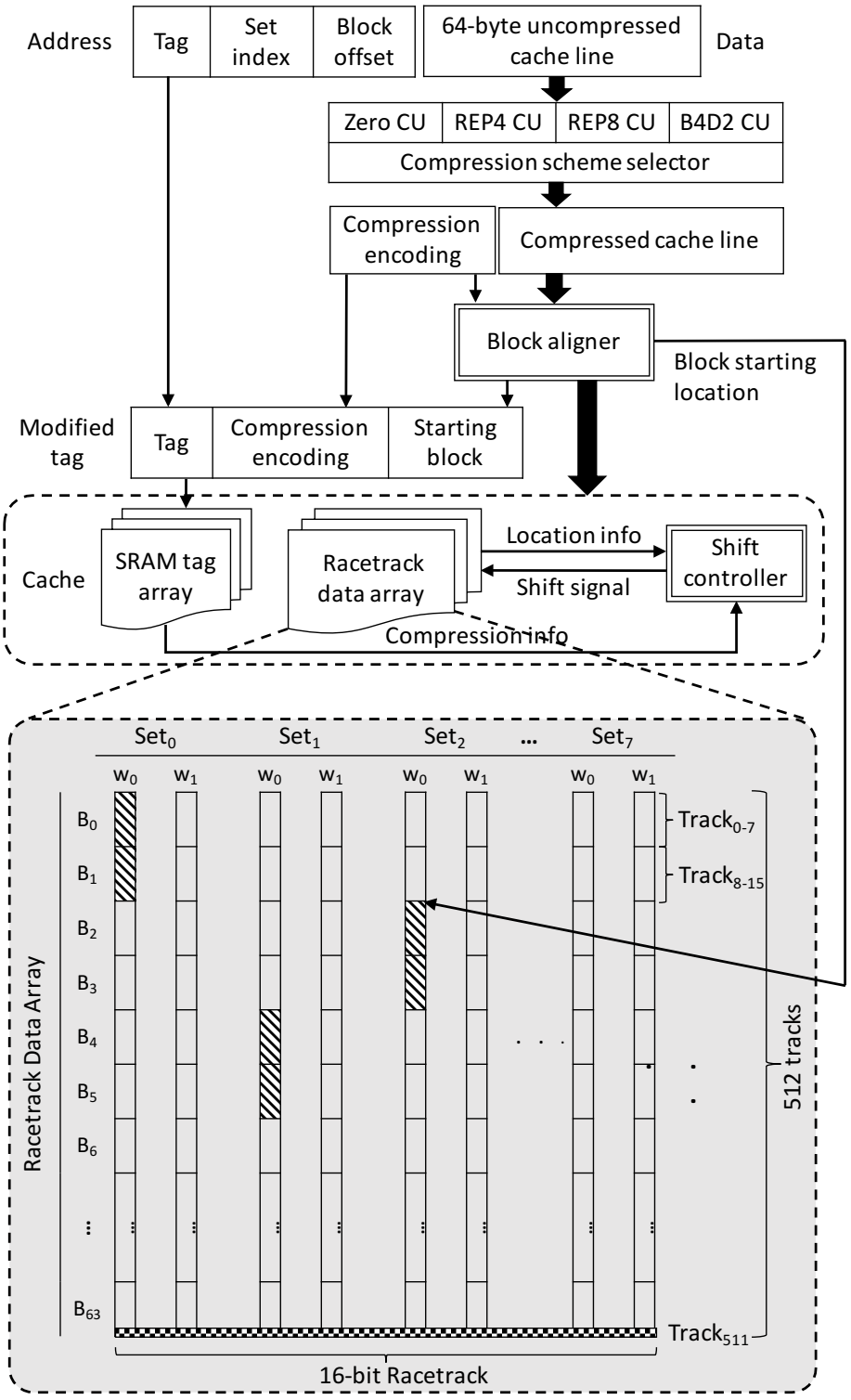


Figure 46: Operational flow of a write access w/ compression to the Racetrack cache (top) and logical view of a 1KB subarray of n-way (in this case, n = 2) set-associative MRC cache data array with 64-byte block size (bottom).

6.1.3 Independent Shifting

As discussed in Section 6.1.2, a cache line can require 0, 4, 8, 34, and 64 bytes for each of the compression configurations, respectively. Thus I propose to independently shift every 16 tracks or 2 bytes (the greatest common divisor of the 4 non-zero sizes) instead of shifting the whole block. For instance, in order to access a cache line compressed by the REP4 scheme, I only need to shift 32 tracks (or I do *not* need to shift 480 tracks) to be aligned with the access ports to read the cache line. Thus, considerable shift energy can be saved in this way through compression combined with independent shifting.

Each shift controller implementation can be reduced to the complexity of a 4-bit adder which consumes 39.84fJ per operation according to a LEAP adder design [108]. In my MRC, to shift a 64 byte uncompressed cache line, 32 independent controllers are needed. According to my evaluation with 20 applications, the average compression ratio is 32%, thus 11 (0.32*32) controllers are used, on average, per access, making the controller energy consumption 0.438pJ (11*39.84fJ). While each access requires an average of 467 shifts which consumes 28.81pJ per my evaluation, the shift controllers consume a nominal overhead of approximately 1.5% of the total shift energy.

6.1.4 Skewed Alignment

In-place compression and independent shifting are effective in saving dynamic power but do not provide a performance improvement. As mentioned in Section 6.1.2, compressed cache lines are placed in original storage area allocated for a full cache line. However, as compressed cache lines reduce the number of bits to be written, it is not necessary to align them with the beginning of the storage cell. Consider the example shown in Figure 46 when there are three overlapping accesses to three different cache lines in the same RM subarray: way W_0 of each of the first three sets of the subarray. In TapeCache, two accesses must stall until one finishes, as all the bits on the same track share a single port. After each access completes, the tracks must be shifted by at least two domains to allow the next cache line to be accessed at the port. Due to the spatial locality of access present in many applications, this contention turns out to be very common and can result in significant performance degradation. In this same

scenario, if the three cache lines can be compressed into the sizes of the shaded bars shown in Fig. 46 and their alignment skewed accordingly, by leveraging independent shifting the Set_0W_0 stored on $Track_{0-15}$ can be accessed by $Port_{0-15}$ without affecting the access to the Set_1W_0 on $Track_{32-47}$ or the Set_2W_0 on $Track_{16-31}$. Shifts are still required to move them to the access port location but their cell alignment is no longer (necessarily) overlapping thus reducing the subarray contention and leading to both energy and performance improvement.

In such a scenario, a critical step is to determine where the compressed cache lines are to be skewed in the storage cell. This is complicated by the need to determine the skewed alignment when the cell is written while the most contention may be generated by reads with a potentially significantly different access pattern. I considered two schemes, *random* and *round-robin*. Round-robin was selected due to its simpler hardware and it turned out to perform better than random. The top part of Fig. 46 shows the basic operational flow of a write to the MRC LLC. The round-robin *block aligner* determines the skew and requires a negligible overhead due to the limited input/output range (0, 2, 4, ..., 62) [109]. To store the skewed alignment location, I require an additional four bits to be added to the three bits for the compression code in the tag array. These extra bits are considered for both area and static power overheads as described in the next section.

6.2 EXPERIMENTAL METHODOLOGY

To evaluate the effectiveness of the compression with independent shifting for Racetrack-based LLC I extended the Sniper full system simulator [100] to include a Racetrack LLC structure. To simulate the impact of the Racetrack structure in Sniper I modified the LLC structure to maintain the Racetrack state and to determine the access latency based on the actual shift latency. The implementation models the impact of contention between temporally overlapping accesses to cache lines mapped to the same tracks and determines the accesses that can proceed in parallel in the same subarray.

I simulated a 64-core processor with 1MB LLC for all memory models operating at a core frequency of 3GHz. For latency, power, and area parameters I use CACTI [110] to

Table 7: Global architecture parameters

CPU	64 cores, 3GHz, 4-dispatch width
L1 Cache	32KB I-cache, 32KB D-cache, 4-way, 64B block size, LRU replacement
L2 Cache	Unified, 1MB, 8-way, 64B block size, LRU replacement
Compression	1-cycle decompression latency 9.96fJ per bit per addition/subtraction for compression/decompression unit

model SRAM, NVSim [101] to model STT-MRAM, and a modified NVSim to model RM. I assume that the area of one RM track is dominated by the area of access transistors due to the 3-D structure (see Figure 6 and adopt the cell size of $2F^2$ [102]. RM shift latency and energy are calculated based on a size 120*720nm^2 perpendicular magnetic anisotropy (PMA) nanowire [103]. RM reads and writes faster than STT-MRAM thanks to its smaller peripheral circuitry. Faster shift-based writes have also been proposed for RM [15]. My studies indicate that independent shifting is equally applicable to both shift and current based writing strategies achieving comparable performance gains. This is because the shifts optimized by MRC dominate the total latency of accesses. Detailed parameters for each memory type based on a 32nm technology feature size are listed in Table 8[1]. A LEAP adder design [108] with energy consumption of 9.96 fJ/bit per addition operation with an assumed 1-cycle latency for each decompression operation as reported in [80] is adopted to model the 4 compression schemes. Detailed configurations are listed in Table 7.

For simulation workloads, I use 20 parallel applications drawn from the Parsec [104] and Rodinia [111] benchmark suites. Each application is executed with 64 threads and characterization is started from the parallel section or "region of interest" (ROI) with 1 billion instructions to warm up the cache followed by 1 billion for evaluation.

[1]All memory types use SRAM tag arrays for fairest possible comparisons.

Table 8: LLC parameters of different memory technologies

Memory technology	SRAM	STT-MRAM	RM	MRC
Tag access latency (ns)	0.34			
Data read latency (ns)	1.89	1.69	1.35	
Data write latency (ns)	1.89	3.91	3.25	
Shift latency (ns)	N/A	N/A	0.643	
Read energy (pJ/access)	142.25	57.74	60.03	
Write energy (pJ/access)	67.76	129.37	121.56	
Shift energy (pJ/cell)	N/A	N/A	0.0617	
Leakage power (mW)	166.9	15.42	8.66	9.38
Tag area (mm^2)	0.12	0.12	0.12	0.14
Data area (mm^2)	2.39	0.65	0.062	
Total area (mm^2)	2.51	0.77	0.18	0.2

6.3 RESULTS

As previous work has demonstrated, the tremendous advantages of employing RM is to improve cache capacity and improved performance due to reduced miss rates [16, 17, 71]. MRC is compared against existing RM cache proposals (e.g., TapeCache), which utilize a similar logical organization. Thus, I consider isocapacity replacements for comparison and include SRAM and STT-MRAM mainly for reference. Evaluation results of a 1MB LLC are presented both to remain consistent with conservative SRAM capacities and to ensure that benchmark simulations do not become largely cache bound[2]. For both RM and MRC, track lengths of both 16 and 32 elements are considered because they reflect the most popular track sizes in previous work[3].

[2]Simulations with LLC sizes of 4MB and higher often become cache bound due to the limited workset sizes of even the larger benchmark workloads.

[3]The 1T1P assumption makes even longer track lengths possible which would increase track contention and the benefits of MRC.

6.3.1 Performance

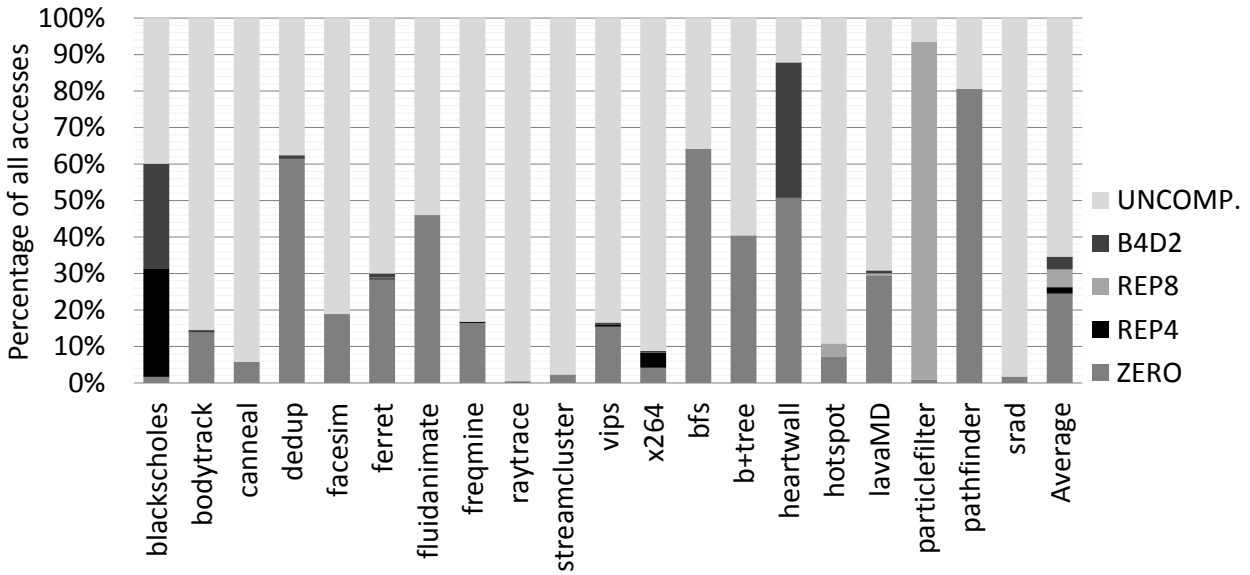


Figure 47: Breakdown of occurrences of 5 compression schemes (including blocks that cannot be compressed) in executing one billion instructions in 20 different applications.

Fig. 48 (top) shows the system performance comparison of six different LLC schemes. For most applications, SRAM outperforms RM and STT-MRAM due to the isocapacity assumption and SRAM's considerable write speed advantage. In cases where the writes are essentially cache bound in the L1 (e.g., ferret, fluidanimate, streamcluster, and srad), STT-MRAM and RM outperform SRAM due to their lower read latency. However, shifting overheads tend to counteract the latency improvement from the smaller peripheral circuitry of the more dense RM (e.g., pathfinder), deflecting the average performance lower than STT-MRAM. However, by compressing cache lines and interleaving shift operations at finer granularity, MRC is able to improve the performance of RM by more than 5% for both 16-bit and 32-bit length tracks. Thus, the performance improves by 4% and 3.5% for 16-bit and 32-bit tracks, respectively, over SRAM. In some applications, such as dedup and pathfinder, dramatic (i.e., over 10%) IPC improvements result from the high compression ratios. Conversely, in applications such as raytrace and srad, MRC degrades performance of RM because the compression/decompression overheads yield relatively low compression

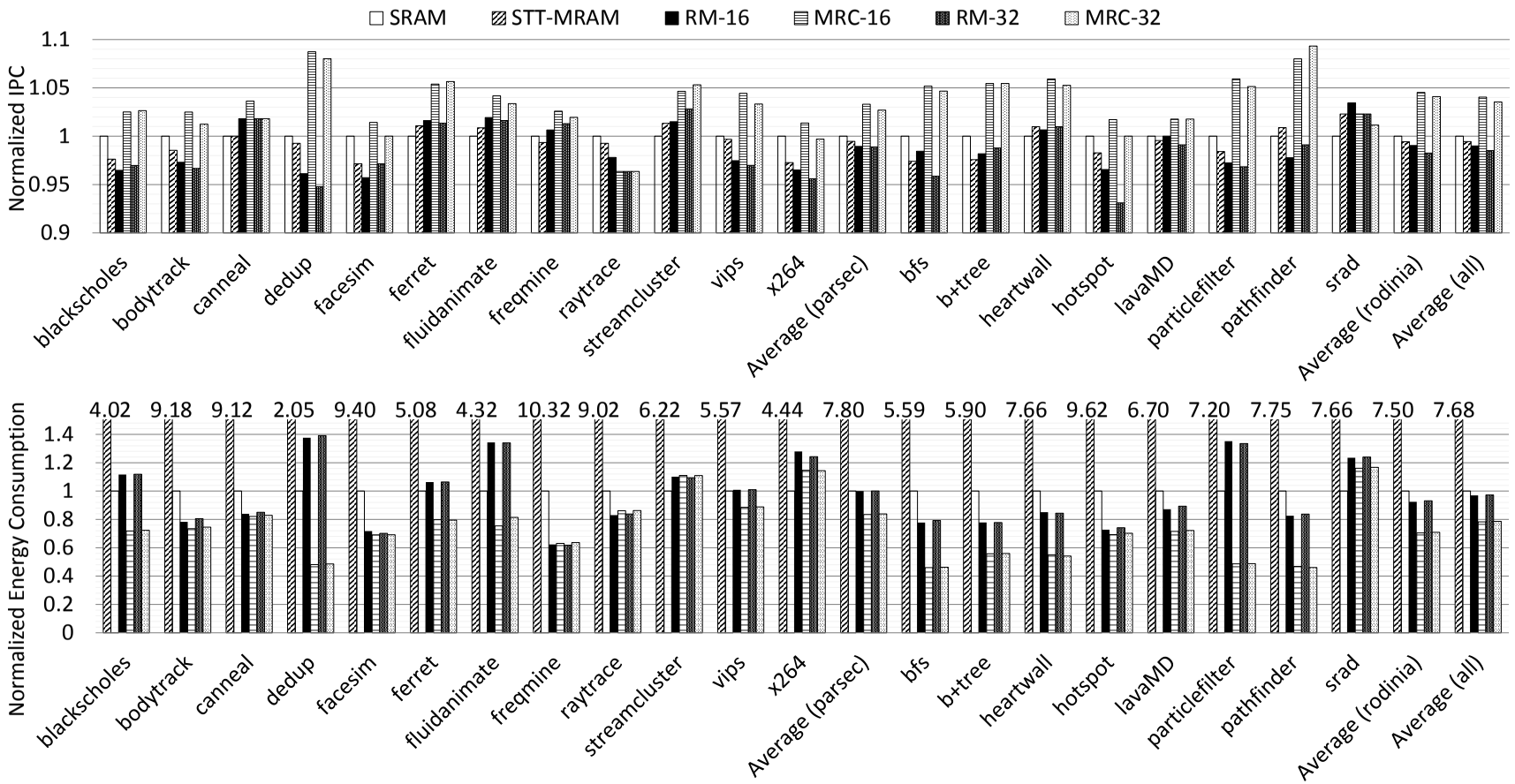


Figure 48: Performance (top) and energy consumption (bottom) comparison of 6 LLC schemes: SRAM, STT-MRAM, RM w/ 16-bit long track, MRC w/ 16-bit long track, RM w/ 32-bit long track, MRC w/ 32-bit long track. For each application, IPCs are normalized the SRAM scheme, energy consumption is normalized to the STT-MRAM scheme.

rations. As shown by the compression breakdown of the four schemes in Fig. 47, most blocks in these two applications are uncompressed. For these rare applications with low compression ratio, the compression/decompression logic could be bypassed to achieve similar performance to RM.

6.3.2 Energy

Fig. 48 (bottom) shows the energy consumption results. Non-volatility helps STT-MRAM and RM to achieve significant energy savings compared to SRAM: 7.68x on average. Compression further reduces dynamic power, especially of expensive write operations. Numbers of shifts are also greatly reduced by compression and the capability of shifting independently in a group of 16 tracks, as partially indicated in Fig. 44. Additionally, the shorter execution time of MRC saves leakage power. On average, more than 19% energy savings are seen for both MRC-16 and MRC-32 compared to RM-16 and RM-32, respectively.

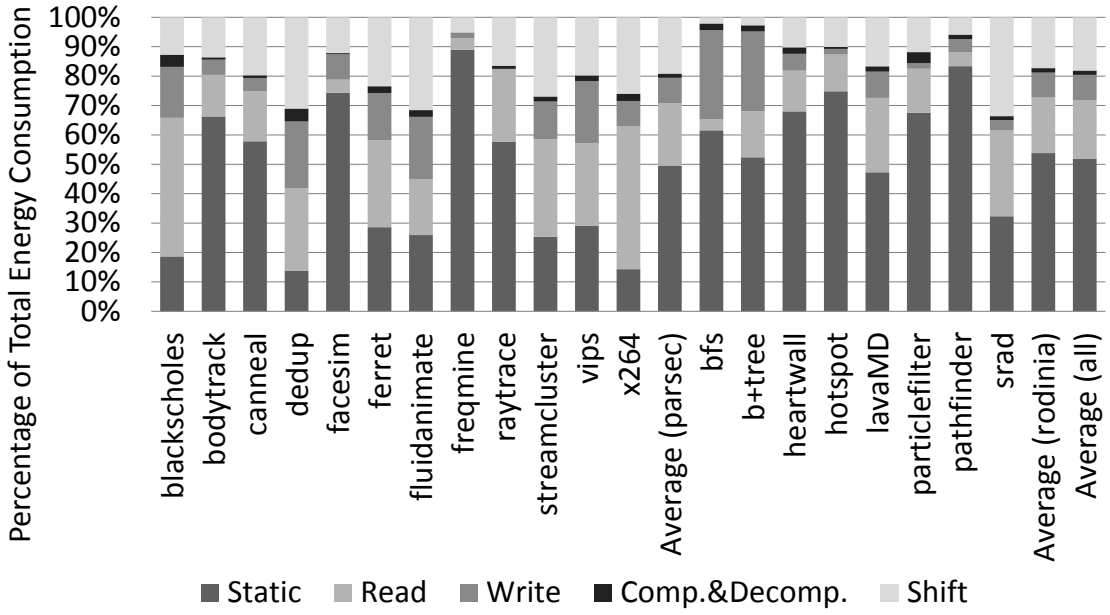


Figure 49: Energy breakdown of a 1MB Racetrack LLC w/ compression (16-bit long track, 64 program threads).

The LLC energy consumption of the MRC-16 scheme is divided into five parts as presented in Fig. 49. The static energy consumed by the SRAM tag array and other peripheral

circuitry accounts for 51.5% of the total consumption. Of the remaining energy, reads, writes take up 20.2% and 8.5% respectively, shift operations consume 18.3%, and lightweight compression/decompression consumes only 1.5%.

7.0 IMPROVING EFFICIENCY OF WIRELESS SENSOR NETWORKS THROUGH LIGHTWEIGHT IN-MEMORY COMPRESSION

7.1 WSN MEMORY AND NETWORK CO-DESIGN FOR LOW-OVERHEAD COMPRESSED COMMUNICATION

In this section, the collaborative designed in-place memory compression and network interface approach to reduce the overhead of software compression for communication in a WSN are described. The next section covers the utilization of lightweight in-place compression in memory and the following sections describe the reorganization of sensor data to be more effectively compressed and, finally, how packets are constructed for transmission from the compressed memory blocks.

7.1.1 Lightweight In-place Compression

In contrast to even the simplest software compression approaches, lightweight compression can be tightly integrated in the cache or memory controller with negligible overheads [83]. However, popular lightweight compression methods, such as *ZERO* [106], *Repeated Value* [107], and *BDI* [80], trade this simplicity for restrictions on what data patterns can be compressed and often achieve a significantly reduced compression ratios compared to software approaches.

For example, ZERO compression simply identifies blocks that store all '0's for each bit and typically replace it with a single '0'. Similarly, repeated value compression partitions a block into smaller sub-blocks and in cases where all the sub-blocks hold identical data, replace it with a single copy of the data. A 64-byte block compressed with REP4 contains

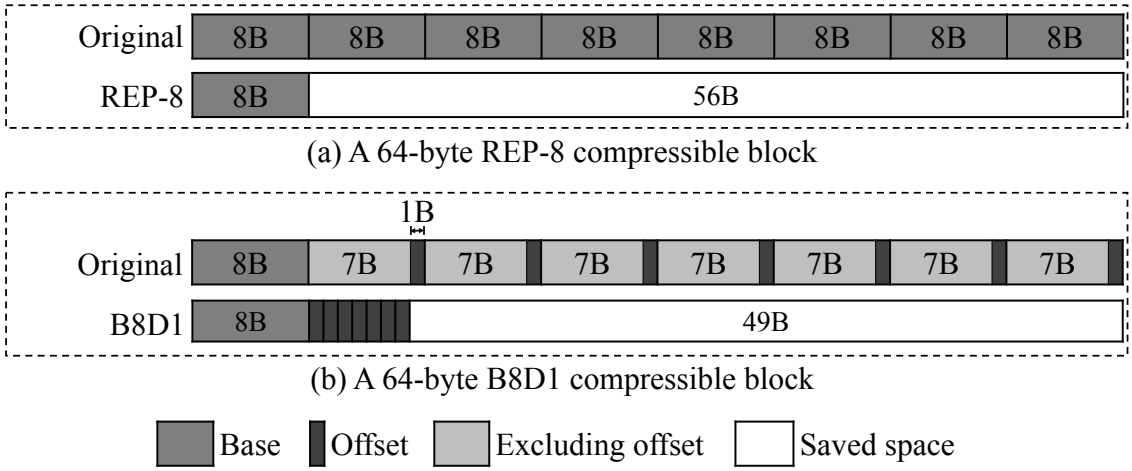


Figure 50: Before and after comparison of compression schemes: (a) REP-8 (b) B8D1.

sixteen 4-byte sub-blocks with exactly the same value. As mentioned in Section 2.4.1, BDI compression leverages low dynamic range patterns exhibited by contiguous sub-blocks within a data block. Like repeated value compression, BDI partitions a large block into multiple smaller sub-blocks. However, BDI is more flexible as it encodes the original block with a *Base* sub-block and multiple offsets (Δs).

Depending on the content, a block can be compressed with different schemes, including: 8-Byte Base + 2-Byte Δ (B8D2), 8-byte Base + 1-byte Δ (B8D1), 4-byte Base + 2-byte Δ (B4D2), 4-byte Base + 1-byte Δ (B4D1), etc. Additionally, both ZERO and Repeated Value compression can be treated as special cases of BDI: ZERO is B0D0 (no Base or Offset information is needed), while REP-n is BnD0 (only an n-byte Base is required).

Fig. 50 shows two lightweight compression schemes applied on a 64-byte data block (a common size of a cache line): REP-8 (B8D0) [Fig. 50 (a)] and B8D1 [Fig. 50 (b)]. Recall, REP-8 requires all eight, 8-byte sub-blocks to have the same content, such that one sub-block (plus a compression code) can represent the whole block. B8D1 is less restrictive and requires that the highest seven bytes of each 8-byte sub-block have the same content. Thus, it is necessary only to store the first 8-byte sub-block intact as the base and the remaining seven sub-blocks' lowest byte as offsets from the base (plus a compression coding) to represent

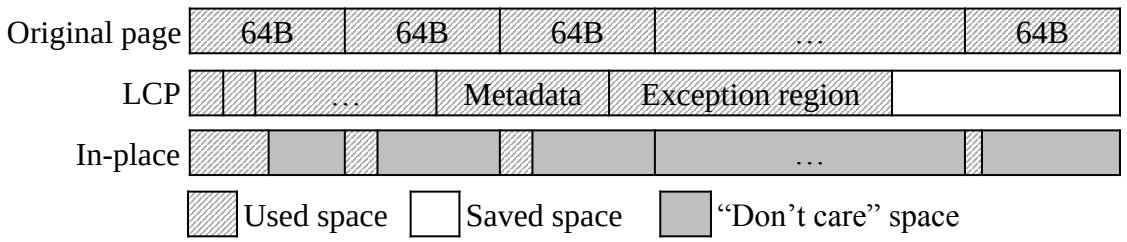


Figure 51: Comparison of space saving compression with LCP with in-place compression for a memory page.

the entire 64-byte block. The compression code may be stored in a variety of ways, such as in a metadata block/page [81, 82], at the end of unused space of each compressed block [83] similar to ECC parity bits in the memory, or alongside of tags in caches [72].

As illustrated in Fig. 51, compression to increase memory density (e.g., LCP) concatenates compressed data blocks in order to reduce the page size requirement. Blocks within a page must be compressed using the same scheme, the incompressible blocks are placed in the *exception region* and indexed by *metadata*. This approach introduces addressing complexity and may cause significant overhead when exceptions occur. For example, if a block is overwritten and the previous compression scheme no longer applies, the compressed block size changes and all blocks concatenated after this one have to shift accordingly. Moreover, if a compressed page increased from just under 1K to just over 1K, a page fault exception is triggered to create a 2K page. In contrast, in-place compression does not concatenate compressed data and leaves each block in its original position. As shown in Fig. 51, space for redundant data is not eliminated, but treated as "Don't Care" space and not utilized. This leaves the memory addressing and offsets intact while clearly marking the critical data for decompression. Benefits of in-place compression have been demonstrated by several prior work as mentioned in Section 2.4.1.

In this work, lightweight compression configurations of BDI are utilized with the eighth encoding being uncompressed. They are prioritized according to the compressed block sizes as shown in Table 9. These seven BDI compression configurations were selected based on

Table 9: 8 compression schemes (including uncompressed scheme) ordered by sizes after compressing a 64-byte data block (smaller compressed sizes have higher priories).

Scheme	ZERO	REP-4	REP-8	B8D1
Encoding	111	110	101	100
Size (byte)	0	4	8	15
Scheme	B4D1	B8D2	B4D2	UNCOMPRESSED
Encoding	011	010	001	000
Size (byte)	19	22	34	64

their relationship to the common sizes of the data primitives used in the WSN applications (e.g., 32-bit integer and floating point, 64-bit string, etc.) and their sensor datasets. The number of bits required to select the correct encoding scheme (i.e. three bits) is determined by the number of compression schemes used (i.e. eight schemes). The detailed encoding metadata organization is adopted to be consistent with the communication protocol in the WSN. I demonstrate this for the TCP segment standard in Section 7.1.3. In the next section I introduce another technique which can increase the compression ratio of sensor data using lightweight BDI compression.

7.1.2 Source-Aware Layout Reorganization (SALR)

As explained in Section 2.4, we recall that data sensed from the same source often demonstrates strong temporal correlation. In this section, I propose a technique called *Source-Aware Layout Reorganization* (SALR) that leverages this correlation property to better compress data from a collection of sensors in order to improve the transfer efficiency of WSNs. I explain this concept in the context of a fairly typical example.

```
<work_unit>
  <electricity_usage>
    2012-08-08 13:00:00,11.0,7.4,11.8,5.8,5.2,
    1.2,8.1,19.7,4.8,50.0,34.0,22.3,181.3,0
  </electricity_usage>
```

```
  <grid_mix>
    0.70,0.00,0.13,0.17,0.01,0.00
  </grid_mix>
</work_unit>
<work_unit>
  <electricity_usage>
    2012-08-08 14:00:00,10.2,7.3,11.7,6.2,5.2,
    1.0,8.0,19.5,4.8,51.0,32.0,22.0,179.0,0
  </electricity_usage>
  <grid_mix>
    0.70,0.00,0.13,0.17,0.01,0.00
  </grid_mix>
</work_unit>
<work_unit>
  <electricity_usage>
    2012-08-08 15:00:00,11.1,7.3,13.8,6.1,5.7,
    1.1,7.6,20.7,4.8,50.0,32.3,22.3,182.8,0
  </electricity_usage>
  <grid_mix>
    0.70,0.00,0.13,0.17,0.01,0.00
  </grid_mix>
</work_unit>
                              ...
```

Listing 7.1: An input data set example

Listing 7.1 shows the structure of an XML file correlating electrical usage and electrical generation for a commercial building. The input data is relationally organized according to the data and time. Data collected from different sensors at the same time is put into the same workunit, which represents the smallest repeating unit of data sets. In this example, data is collected hourly from 14 electricity meters in the building. The second element "grid_mix" is the national average electrical mix [112] which was used to break down the electrical consumption from the building into its fractional generation by fuel type (coal, petroleum, natural gas, nuclear, etc.). This breakdown is used in a dynamic life cycle assessment (DLCA) to calculate carbon emissions along with electricity usage, since each type of fuel releases different amounts of pollutants [95].

Fig. 52 shows how the original structure of a packet containing 16 workunits where data is organized and correlated by time points. Each workunit contains an eight-byte string type timestamp, 14 four-byte floating point values for electricity usage data, and six, four-byte floating point values representing the electrical grid mix data. Fig. 53 illustrates the layout of this packet if directly placed in contiguous memory. Each small square represents one

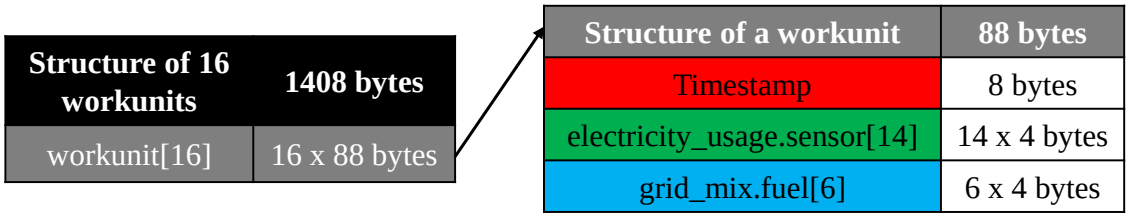


Figure 52: Original structure of a 16 workunit structure.

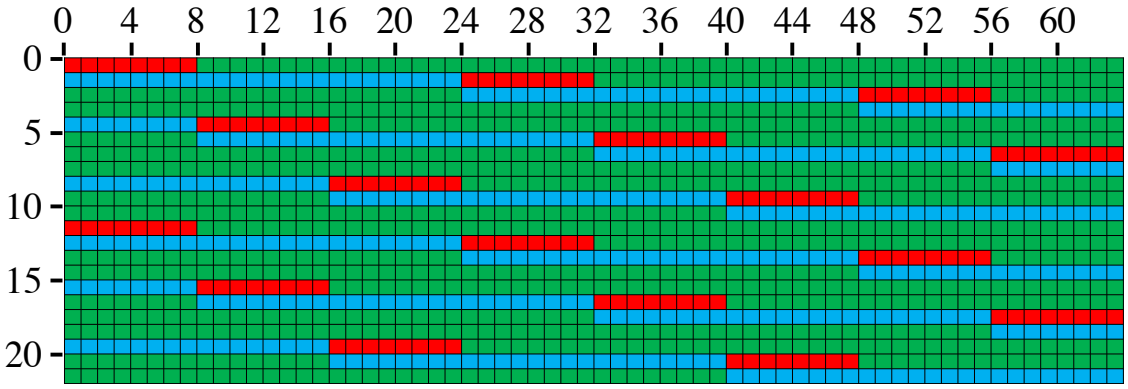


Figure 53: Original layout (byte map) of a 16 workunit packet.

byte, and each row has 64 bytes to represent a typical cache-line and page sub-block. Red segments are timestamp strings; green segments are electricity usage data; blue segments are grid mix data. Values of different data types have minimal correlation and as a result, BDI lightweight compression schemes which rely on the low dynamic range of neighboring blocks do not provide good compression results on this structure.

Sensed data from the same sensor target, such as temperature, humidity, and CO_2 concentration of a room tend to have much better range correlation. Thus, I propose SALR to group data from the same sensor at consecutive time points next to each other, while still maintaining the order according to the timestamp. Fig. 54 shows the SALR-optimized data structure and Fig. 55 shows the byte map in which each row contains the same type of data. In the following section I demonstrate the advantage of using the SALR layout in WSN data transfers.

Structure of 16 workunits for in-place compression	1408 bytes
Timestamp[16]	16 x 8 bytes
electricity_usage.sensor[14][16]	14 x 16 x 4 bytes
grid_mix.fuel[6][16]	6 x 16 x 4 bytes

Figure 54: Reorganized structure of a 16 workunit structure.

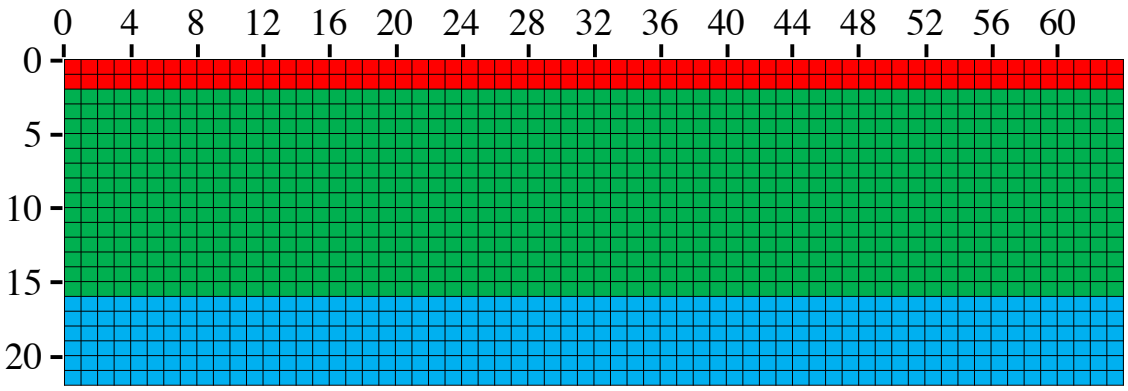


Figure 55: Reorganized layout (byte map) of a 16 workunit packet.

7.1.3 Memory Network Compression Co-Design

Recalling that lightweight compression can be efficiently implemented in hardware and natively integrated into the memory controller with negligible overhead (see Section 7.1.1), this natively compressed data can be sent directly from memory over the network in a WSN in place of using software compression. In this section, the system designed to translate the compressed data into network packets in the WSN context is explained.

Fig. 56 illustrates my proposal for integrating lightweight compression hardware in the memory, and seamlessly wrapping compressed memory pages into reduced payload of TCP segments. In fact, my co-design concept is hardware compression-type agnostic and is compatible with schemes such as SOCO, Memzip which compress to improve metrics such as energy efficiency as well as LCP, which concatenates pages to increase effective memory

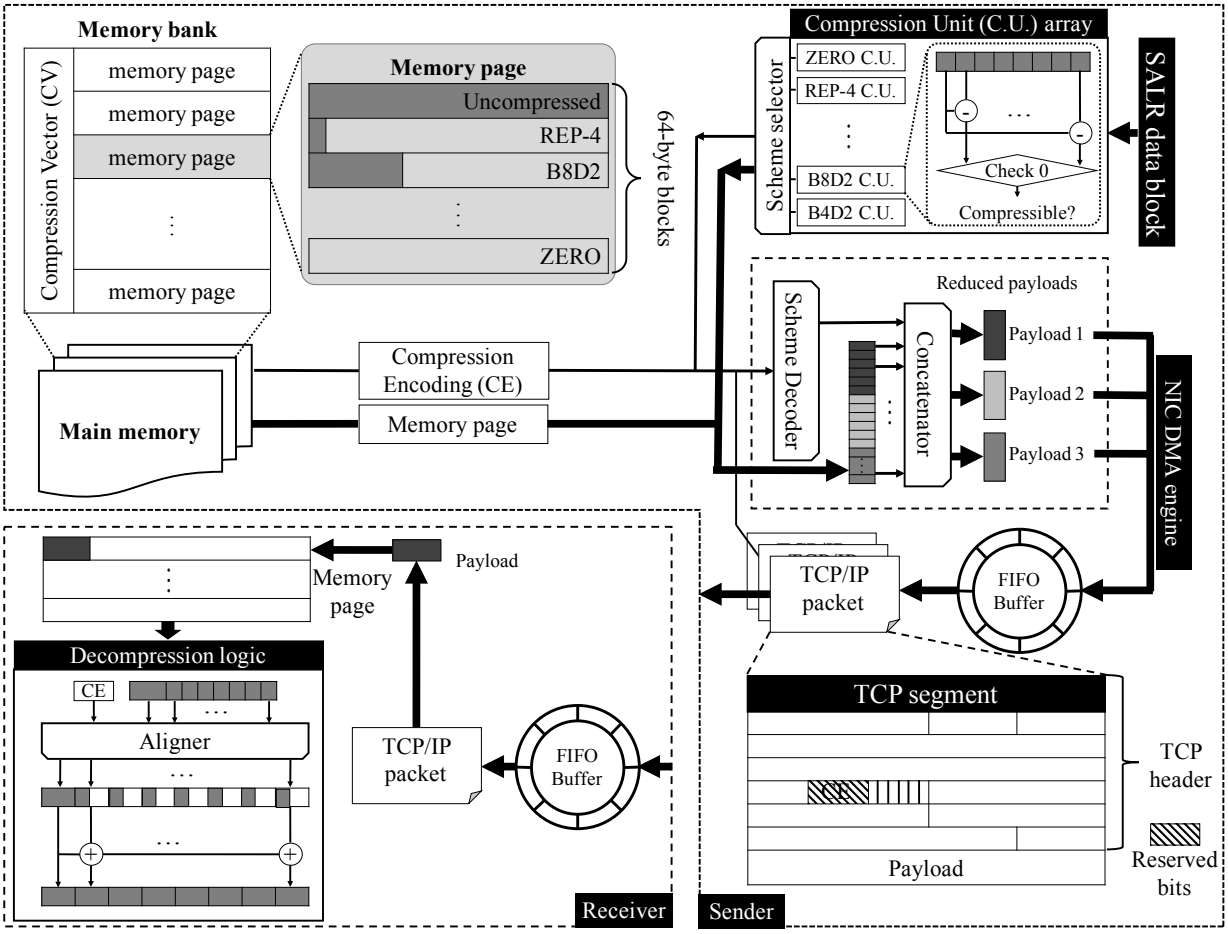


Figure 56: Memory network co-design and work flow of compressed transfer in WSN.

capacity. In-place compression (Memzip/SOCO) help to reduce the bytes written to the cache/memory, thus consumes less memory bandwidth or dynamic power of write operations [82, 83]. Space saving compression [81] increases the overall effective density by utilizing smaller pages to store compressed data allowing more pages to be stored in memory.Without loss of generality, I present the design using an in-place compression approach while mentioning the difference with a space saving approach (LCP) and evaluate both in Section 7.2.

I explain how my co-design scheme works starting from the initial point of acquisition and storage of sensor data into memory. Seven *compression units* (CUs) of the CU array

(top right of Fig. 56) simultaneously examine the content of incoming data block, and output whether it is compressible with the corresponding scheme, along with the compressed block. Thanks to the low latency nature of lightweight compression, compression circuits are simple, comprised of simple adder logic [80]. Compression scheme selection is handled with priority encoding (combinational) logic and, as a result, encoding takes only a few cycles. This nominal delay tends not to impact system performance as write operations are typically not on the critical path of memory accesses. In my experiments, the impact of compression on the overall system performance is negligible. Based on the seven outputs and the priority order, the *scheme selector* picks the best available compression scheme (or "uncompressed") and outputs the encoding to a metadata storage location, which can be stored either in non-addressable memory space similar to ECC [83], within a metadata space within the page [81], or within a separate metadata page [82]. The case of non-addressable storage is shown using the same terminology as previous work regarding a compression vector (CV) [83] (top left of Fig. 56). This technique is easily applied to systems storing the compression metadata within the compressed page or within a compression metadata page.

The dark gray area of data blocks in the memory page of Fig. 56 represents the useful bytes after compression (e.g., the second block is REP-4 compressible, thus, only the lowest four bytes are useful). For space saving compression, page sizes would be non-uniform in size, so the operating system would have to support multiple page sizes simultaneously.

Sending data starts with a read operation loading the compression encoding (CE) from the metadata region (e.g., CV) and the data bytes from memory pages, respectively. The compression code is used to mask which bytes of the data block contain useful data and concatenates them into a shorter block. For space saving compression, this is already done, and the code along with the datasize tells the system how many subpage blocks are contained in the compressed section.

There are 6 reserved bits in TCP segment header (bottom right of Fig. 56), which are used to store the CE. The value is initially set to zero by default, which indicates uncompressed data. Thus, uncompressed blocks naturally become regular TCP packets. A compressed block is embedded into the segment as payload and wrapped with source and destination IP addresses into a IP packet ready for transfer.

My approach groups consecutive blocks into the same TCP packet if they share the same compression scheme. For example, a memory page is partitioned into three payloads (compressed with three different schemes), and are segregated into three packets, as shown in Fig. 56. The *maximum transmission unit* (MTU) or largest TCP packet size of IEEE 802.11 wireless standard is 7981 byte [113], which is larger than a typical 4K page, making it possible to transfer an entire memory page if all of its 64-byte blocks are compressible with the same compression scheme. Of course, for a memory page where every consecutive block has a different compression code, this can create a significant amount of overhead. However, as illustrated in Figs. 53 and 55, my SALR approach reorganizes similar data together. As a result, my studies found that most consecutive cache line sized data blocks were actually compressed with the same scheme because of that.

Upon receiving a packet, the receiver node (bottom left of Fig.56) extracts and stores the compressed payload along with its CE (again, metadata location varies across different designs) into the contiguous memory region. A memory page is then restored after one or more (in this case, three) packets arrive. To access the received data, the memory controller reads CE and decompress the compressed page accordingly with simple logic as is native to the lightweight compression scheme.

7.2 EXPERIMENTS AND RESULTS

To evaluate the impact of my proposed co-design memory and network compression, I applied different methods of lightweight compression with and without SALR on a variety of data logging benchmarks relevant to sensor networks including hourly and monthly temperature and precipitation data from NOAA 1981-2010 climate normals [18] and 1976-2014 Nasdaq composite index data [114] and an in house sensor network data as described in Section 7.1.2. I implemented my approach using real hardware consisting of Asus/Google Nexus 7 (2012) tablets evaluating the transfer time of seven benchmarks of data via Bluetooth and WiFi Direct connections. I compared both lightweight in-place and space saving compression with software (ZIP) compression applied to both original and SALR data layouts.

7.2.1 Impact of SALR

Examining the NOAA and NASDAQ data, the original format for hourly temperature is as shown in Fig. 57(a) and the original daily or monthly Nasdaq index data is formatted as shown in Fig. 57(b). Considering the NOAA data, station ids, dates, and hourly readings are represented in different data structures, and thus each field should be grouped consecutively. When applying SALR, temperatures of different stations at the same time of day are treated as being from the same source. Regarding the financial data, different data fields such as daily values of opening, closing, low, and high may tend to be within a similar range, but the volume field will destroy range locality if grouped by date rather than field. In the remainder of this section, I examine the compressibility of seven data groups, temperature, precipitation, and Nasdaq financial data for with both hourly and monthly data, along with my in-house wireless sensor network data (MCSI) in both original and SALR formats, using LCP and in-place lightweight compression, and compare them with the popular ZIP software compression approach.

Station id	Date	1st hour data	2nd hour data	…	24th hour data

(a) NOAA hourly data.

Date	Open	High	Low	Close	Volume	Adj. Close

(b) Nasdaq daily index data.

Figure 57: Data formats.

Fig. 58 shows the occurrence breakdown of the adopted eight lightweight schemes in seven data groups, with the original and SALR optimized data layouts. Nearly all of the data is originally uncompressible with the lightweight compression, due to the mixed alignment of different data types. MCSI data is an exception because the electricity data from several meters contains a significant number of zero readings, particularly on weekends when the building is not in heavy use, leading to 25% of the data having some lightweight compress-

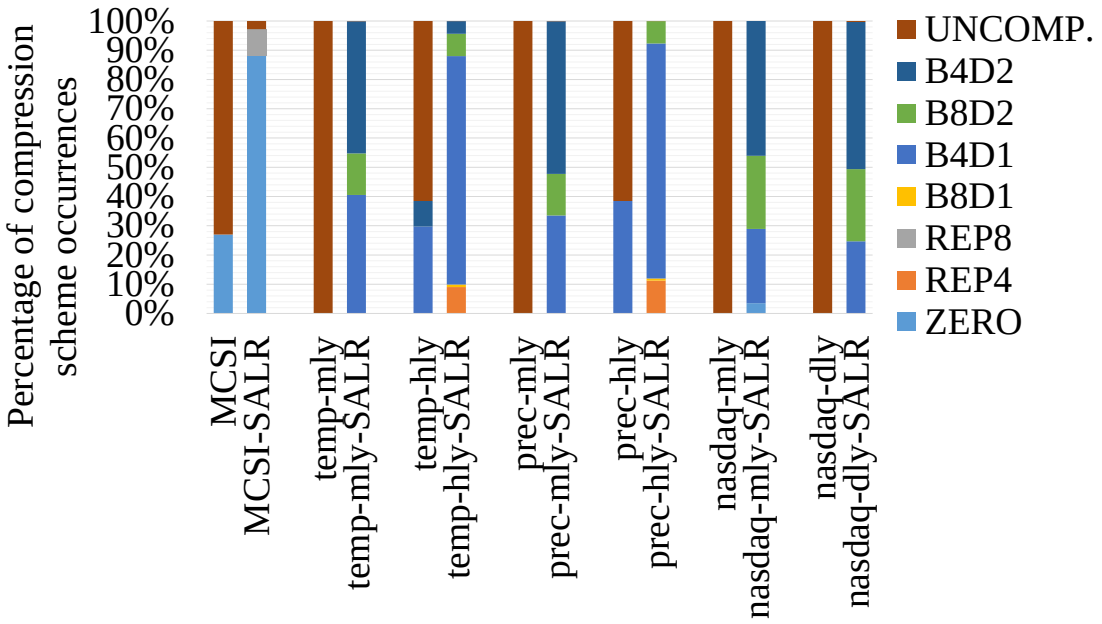


Figure 58: Breakdown of eight lightweight compression schemes.

ibility. After realigning the elements in the original data structures according to their sources (SALR), all of the data sets become compatible with at least one of the seven compressible schemes.

As mentioned in Section 2.4.1, lightweight compression typically cannot compete with traditional software-based approaches in terms of achievable compression ratio. Thus, I compared the lightweight compression techniques with the popular ZIP software compression on both original and SALR data layouts. The results are summarized in Fig. 59. Hourly data, which tends to have less fluctuations among neighboring values (sub-blocks), has a better compression ratios than monthly data[1]. While ZIP compression achieved an almost 6X compression ratio for the original data set, lightweight compression was extremely ineffective. SALR was extremely effective for achieving higher lightweight compression ratios, bring space saving compression up from no compression to about 2X compression and in place compression up to nearly 4X compression. As expected, ZIP outperforms lightweight compression thanks to its more complicated encoding, and in-place compression outperforms LCP thanks to its finer compression granularity. LCP, though uses the same compression

[1]Nasdaq data is an exception, probably due to that fact that, unlike temperature or precipitation, stock prices do not follow certain physical laws and thus, are more random.

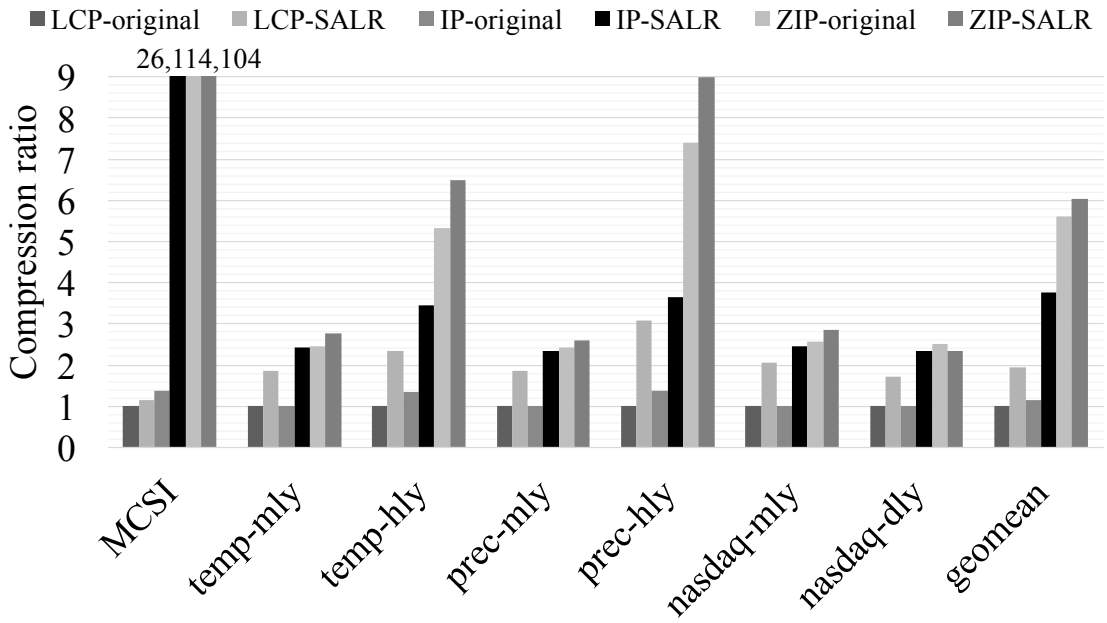


Figure 59: LCP, in-place (IP) and ZIP compression ratio for original and SALR data layouts. Compression ratio is $\frac{rawSize}{compressedSize}$ (higher is better).

algorithms (ZERO, Repeated Value, BDI), must apply the same compression scheme across an entire page. Somewhat surprisingly, I found that SALR also helped to slightly improve the result of ZIP compression although for MCSI and Nasdaq daily datasets it caused a slight degradation.

7.2.2 Compressed transfer in a WSN

With the proposed design in Section 7.1.3, lightweight compression can be naturally integrated in WSN packet transfers with minimal overhead. The time needed to use software compression in the construction of packet is an overhead that is eliminated by in-memory lightweight compression and should be considered as an overhead. My experiments added the software compression time to the total transfer time of the ZIP compression scenarios, determining the compression and decompression times by running them on the Nexus 7 tablet as well. Results were the average of three runs for each communication scenario. The average values are listed in Table 10. Due to the randomness of real-world wireless transfer

Table 10: ZIP compression and decompression time on a Nexus 7 (2012) tablet

Time (ms)		MCSI	temp-mly	temp-hly	prec-mly	prec-hly	nasdaq-mly	nasdaq-dly
Compression	Original	15.33	274.67	216.00	261.33	167.33	11.00	87.00
	SALR	14.67	228.67	126.67	240.00	117.00	9.67	91.67
Decompression	Original	15.67	70.67	103.00	68.33	113.00	6.33	72.00
	SALR	16.33	64.00	93.33	67.67	92.33	6.67	56.00

tests, instead of getting each transfer time by individual experiment with wide variance, I did fairly exhaustive experimentation to determine an average transfer rate used for all experiments. This rate for Bluetooth was 73.2 kB/s, from which I calculated the total transfer time for all scenarios.

Fig. 60 shows the Bluetooth transfer time of seven groups of data files with four different "compression" and "data layout" combinations. The "LCP-original" scenario requires the longest transfer time for all files as it produces the lowest compression ratios and serves as my baseline. All results of the remaining combinations are normalized to this scenario. In 5 out of 7 data benchmarks, ZIP compressed transfer, even with the slight boost from SALR-optimized layout, takes longer time than in-place compression (IP). On average, compared to "IP-original," "IP-SALR" reduces 70% of the transfer time thanks to the high compressibility advantage from SALR. Compared to the two ZIP schemes, "IP-SALR" reduces the transfer time by 16.3% and 7.3%, respectively, due to the savings of the compression and decompression overheads (see Table 10).

I considered similar scenarios using the faster WiFi-Direct for point-to-point connectivity in the mobile device constructed WSN. WiFi has a wider range and much higher transmit rate than Bluetooth [115]. The results for Wifi-Direct are shown in Fig. 61. With the higher transmission rate, the overhead of compression and decompression is amplified. Even though the ZIP schemes have much smaller compressed data, their total transfer time is close to the baseline, while SALR helps to reduce the transfer period by 12.5%. My proposed co-design with in-place compression scheme, "IP-SALR", achieves an impressive 65.4% transfer time reduction compared to the "ZIP-SALR" scheme for this configuration.

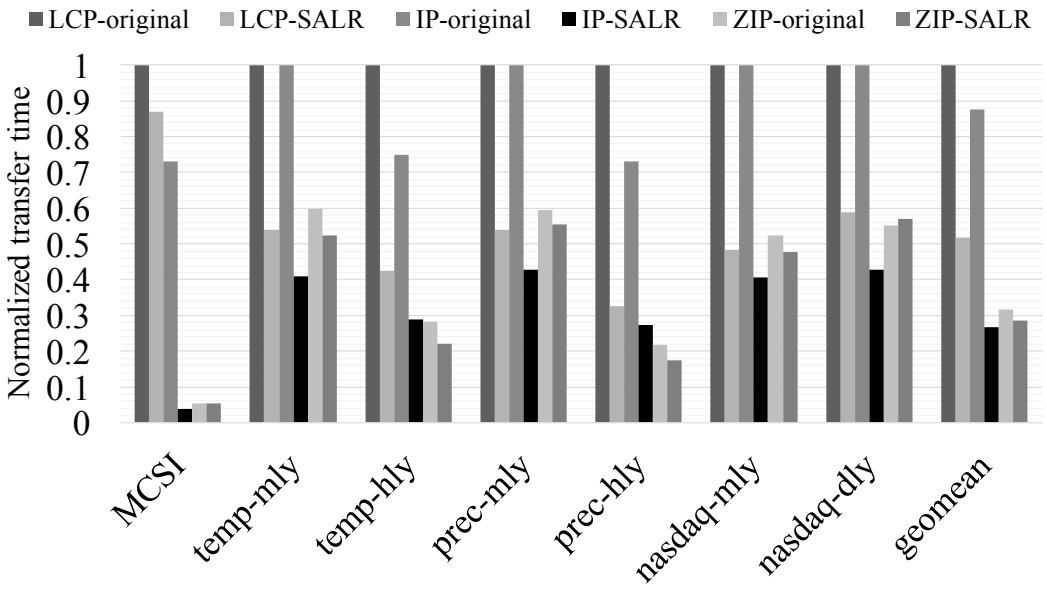


Figure 60: Bluetooth transfer time of LCP, in-place (IP), and ZIP compression, with original and SALR layouts (normalized to LCP-original).

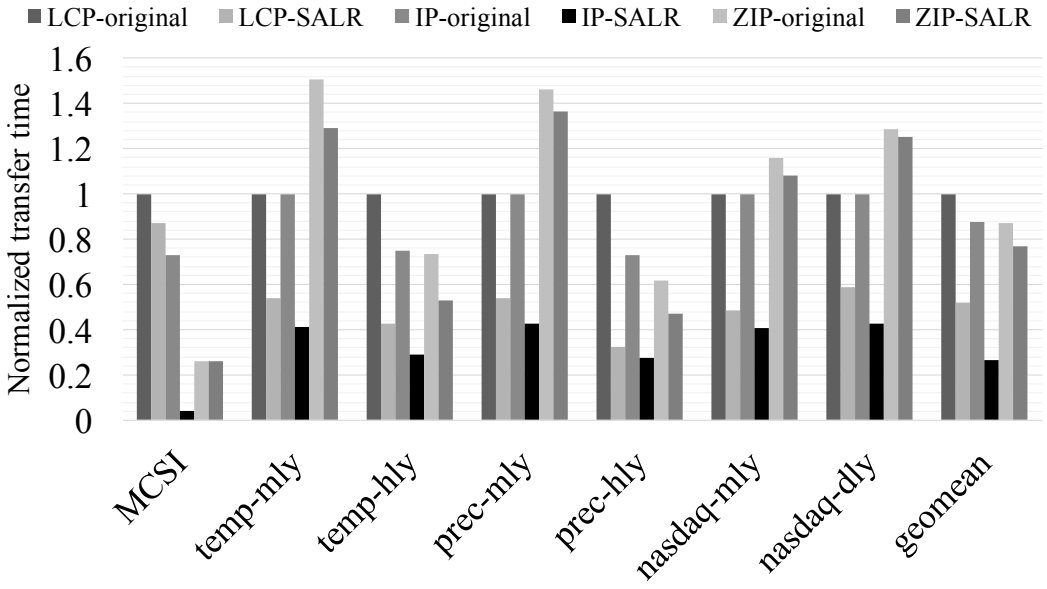


Figure 61: WiFi-Direct transfer time of LCP, in-place (IP), and ZIP compression, with original and SALR layouts (normalized to LCP-original).

8.0 CONCLUSION AND FUTURE WORK

8.1 CONCLUSION

In this dissertation I explore the notion of leveraging the ubiquity of palmtop computers to build highly flexible, efficient, and capable wireless sensor network systems. To accomplish this I develop a lightweight grid computing and self organizing communication system layer for these devices. To improve the energy efficiency and storage capability to support this computing and communication system layer I propose the use of next generation non-volatile memories throughout the memory hierarchy as well as two novel approaches to optimize these structures from the lowest level of the cache hierarchies to higher levels such as last level caches and main memory. Finally, I create a direct link between these architectural advancements and the communication layer with a memory/network co-design approach that leverages lightweight compression used to improve the efficiency of the memory to improve the communication efficiency over multiple types of wireless connections.

• **Ocelot:** I introduce the Ocelot system, a commodity palmtop device based, distributed computing engine. I demonstrate in a case study a green building and in a larger context of an integral component of my dynamic life cycle assessment work, that Ocelot is a robust and energy efficient platform. My experiments show that Ocelot clients can save 86% of energy comparing to traditional computers just during times when tasks are being computed, and I still see a 67% energy saving in tests based on efficient compression in data communication. By assigning tasks to a large amount of clients, significant speedups can be achieved.

• **Lynx:** I also present the Lynx system, a self-organizing wireless sensor network with palmtop devices. The Lynx system alone provides a smarter yet reliable approach to configure the wireless sensor network. The self-organizing features are great guides for me in

designing and implementing WSNs in real world hardware, reducing human intervention in both configuration and maintenance. As shown in the experimental results, WiFi-Direct connection provides better speed due to the direct P2P link and higher transmitting rate. With the shorter radio range and slower speed of Bluetooth, Lynx still achieves a reasonable result. Combining Ocelot, it saves up to 88% of energy compared to the previous Ocelot-only system, let alone the more power-hungry computer platform such as BOINC. The flexibility that Lynx-Ocelot system provides in communication and distributed computing has huge potential for applications in areas ranging from health care to big data collection and processing. The global network map maintenance and multi-hop routing modules laid the foundation for possible future directions such as real world testing of different routing algorithms.

• **FusedCache:** I also present a lower level cache design named FusedCache, a naturally inclusive dual-level private cache which fully utilizes RM's storage density. While prevailing RM research work focuses on sacrificing high density to improve performance by aligning multiple ports along one track, I try to fully explore the feasibility of adopting a simpler single-port track structure in low level caches. One critical challenge is to provide fast, uniform L1 access with non-uniform access pattern. My proposed design naturally fuses two levels of caches, with constant access latency at one level and variable latency at the next. The experimental results demonstrate that using the best current RM cache organization (TapeCache) is ineffective in improving performance by leveraging additional capacity. In contrast, when compared to an iso-area SRAM cache replacement, FusedCache improves performance by more than 7% and reduces energy by 33%. Compared to an iso-capacity two-level SRAM cache replacement, FusedCache saves cache energy by 69% while providing similar performance.

• **MRC:** I also present MRC, an energy efficient Racetrack-based LLC design using lightweight compression with independent shifting and skewed alignment to accelerate cache accesses. While prevailing RM research topics focus on optimizing the MP1T structure, I attempt to fully explore the potential of the original 1P1T structure for its higher density yet lower complexity. Two major challenges to this work include Racetrack's energy hungry write operations and its costly pseudo-sequential access pattern. To address these, cache lines were

compressed using lightweight schemes, and store the compressed data in its original place without concatenation, but realign the starting locations to reduce contention for the limited ports. The experimental results demonstrate the effectiveness of the independent shifting at reducing both unnecessary waiting time for port resources and energy consumption from read, write and shift operations. Besides, the drastic on-chip cache area reduction achieved by the 1P1T structure makes it promising to employ RM in mobile or smaller form factor embedded systems than stationary computing platforms.

• **Co-design:** Compression turns out to be an effective and sometimes necessary approach to improve the usability of both high level WSN application and low level cache/memory designs. I have demonstrated that software-based compression, though takes extra time to compress and decompress, is able to significantly reduce data transfer time over the Lynx network. However, software compression have considerable overhead of compressing (before transfer) and decompressing (after transfer). To explore the potential of leveraging low overhead hardware-based compression in WSN transfers, I present a network and memory co-design combining memory-level in-place compression with network-level packet transfer. As prior work has demonstrated that in-place compression is effective in saving bandwidth and dynamic write power, especially for emerging memories, I extends this approach particularly to improve the efficiency of WSN transfer. By natively compressing the data in the memory and sending reduced payload, lightweight compression outperforms traditional software-based approach which has expensive compression and decompression overhead. Additionally, I propose SALR, a technique specially designed to improve the compressibility of sensor data. Experiments demonstrate that, in terms of compression ratio, SALR improves the traditional software-based ZIP compression by 7.3%, and hardware-based lightweight compression by 230%. The integration of these two orthogonal techniques greatly reduces total transfer time of various types of sensor data in a WSN. Using Bluetooth connections, my proposed scheme achieves a 7.3% reduction in transfer time compared to the second best scheme. And this advantage grows significantly to 65.4% with faster WiFi-Direct connections.

8.2 FUTURE WORK

8.2.1 Future Work in Environmental Impact Evaluation

Caches require about 20% of the total energy consumption of a modern processor [116], which in a regular usage scenario, consumes about 14% of the total battery power of a typical palmtop computer [117]. On the other hand, in a typical four-core system, the proposed RM cache designs are able to achieve an 83% cache (three-level) energy reduction, leading to a 2.3% overall system energy saving. As described in Section 3.7, LCA provides a comprehensive view of the environmental impact by a process or product, including global warming potential, acidification, and ozone depletion. With additional information on grid mix and palmtop device market penetration, it becomes possible to evaluate the broader environmental impact of my architecture-level work using LCA. Assuming that each of the 34,934 students enrolled at the University of Pittsburgh in 2014 had a palmtop device, the adoption of my proposed cache designs would have saved a total of 17,889 kWh electricity. This can be broken down into the 16,601 kWh that would not have been consumed by the tablets and 1,288 kWh of electricity that would have been required for the electricity generation process to generate that 16,601 kWh that would have been required by the tablets. This correlates to a potential reduction of 10 tons of greenhouse gas emissions (CO_2 equivalent) each year.

Conducting LCA to study the broad impact of those designs on a larger population would be a potentially interesting future direction. Moreover, in specific areas such as distributed computing and wireless sensor data communication, the proposed Ocelot and Lynx platforms, as well as the memory network co-design, can further reduce the energy consumption of palmtop devices, the environmental impact of which can be explored similarly.

8.2.2 Future Work in WSN System Improvements

- **Smart hybrid wireless communication:** Lynx is independent of the low level wireless connectivity, and is capable of either sending data over Bluetooth, WiFi-Direct, or other protocols available in the system. A possible future direction is smart hybrid wireless communication. Bluetooth is known to have a shorter signal coverage and a lower

transfer speed than WiFi-Direct, but consumes less power, and therefore is better suited to transfer small quantities of data. By intelligently switching between all possible connectivities, it is possible to achieve a wider coverage, a reasonable transfer speed, and an optimal energy efficiency.

- **Storage reliability:** The distributed database management system (DDMS) and distributed file management system (DFMS) for Ocelot task execution over Lynx networks are implemented in a lightweight fashion to prove the concept of pure P2P distributed computing. There is a battery of existing work on distributed database topics such as localization of data, concurrency control, data replication, and failures, etc. A possible future direction is to improve the data reliability in Lynx by integrating these techniques. On the other hand, Lynx can be used as a real-world testbed for WSN evaluations of these techniques.
- **Transmission reliability:** Forward error correction (FEC) is commonly used to correct errors in data transmission by sending redundant information encoded as an error correcting code (ECC). A possible future direction for lightweight in-place compression is to improve transmission reliability in conjunction with FEC. The compressed data itself takes less space thus shorter ECC to encode. Moreover, the saved space can be utilized to store the code in order to save both bandwidth and transmission time.

BIBLIOGRAPHY

[1] M. Meeker and L. Wu, "Kpcb internet trends 2013," in *Internet Trends D11 Conference*, 2013.

[2] D. P. Anderson, "Boinc: A system for public-resource computing and storage," in *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on*. IEEE, 2004, pp. 4–10.

[3] A. Van Someren and C. Atack, *The ARM RISC Chip: a programmer's guide*. Addison-Wesley, 1994.

[4] J. S. JESD209, "Low power double data rate (lpddr) sdram specification," *JEDEC Solid State Technology Association*, vol. 8, 2007.

[5] R. Heydon, *Bluetooth low energy: the developer's handbook*. Prentice Hall, 2012.

[6] V. J. Reddi, B. C. Lee, T. Chilimbi, and K. Vaid, "Mobile processors for energy-efficient web search," *ACM Transactions on Computer Systems (TOCS)*, vol. 29, no. 3, p. 9, 2011.

[7] M. Mouly, M.-B. Pautet, and T. Foreword By-Haug, *The GSM system for mobile communications*. Telecom publishing, 1992.

[8] A. J. Viterbi, *CDMA: principles of spread spectrum communication*. Addison Wesley Longman Publishing Co., Inc., 1995.

[9] N. R. T. Definition, "Nfc forum technical specification," 2006.

[10] S. Bluetooth, "Specification of the bluetooth system, version 1.1," *http://www.bluetooth.com*, 2001.

[11] W. Alliance, "Wi-fi standards," 2007.

[12] S. S. Parkin, M. Hayashi, and L. Thomas, "Magnetic domain-wall racetrack memory," *Science*, vol. 320, no. 5873, pp. 190–194, 2008.

[13] A. Annunziata, M. Gaidis, L. Thomas, C. Chien, C. Hung, P. Chevalier, E. O'Sullivan, J. Hummel, E. Joseph, Y. Zhu, *et al.*, "Racetrack memory cell array with integrated

magnetic tunnel junction readout," in *2011 International Electron Devices Meeting*, 2011.

[14] Y. Zhang, W. Zhao, J.-O. Klein, D. Ravelsona, and C. Chappert, "Ultra-high density content addressable memory based on current induced domain wall motion in magnetic track," *IEEE Transactions on Magnetics (TMAG)*, vol. 48, no. 11, pp. 3219 –3222, nov. 2012.

[15] R. Venkatesan, M. Sharad, K. Roy, and A. Raghunathan, "Dwm-tapestri-an energy efficient all-spin cache using domain wall shift based writes," in *Proc. of DATE*, 2013, pp. 1825–1830.

[16] R. Venkatesan, V. Kozhikkottu, C. Augustine, A. Raychowdhury, K. Roy, and A. Raghunathan, "Tapecache: a high density, energy efficient cache based on domain wall memory," in *Proc. of ISLPED*, 2012, pp. 185–190.

[17] Z. Sun, W. Wu, and H. Li, "Cross-layer racetrack memory design for ultra high density and low power consumption," in *Proc. of DAC*, 2013, pp. 53:1–53:6.

[18] NOAA, "NOAA's 1981-2010 Climate Normals," https://www.ncdc.noaa.gov/data-access/land-based-station-data/land-based-datasets/climate-normals/1981-2010-normals-data, 2013, [Online; accessed 31-May-2015].

[19] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer, "Seti@home: an experiment in public-resource computing," *Commun. ACM*, vol. 45, no. 11, pp. 56–61, Nov. 2002.

[20] D. Stainforth, J. Kettleborough, A. Martin, A. Simpson, R. Gillis, A. Akkas, R. Gault, M. Collins, D. Gavaghan, and M. Allen, "Climateprediction.net: Design principles for public-resource modeling research," in *In 14th IASTED International Conference Parallel and Distributed Computing and Systems*, 2002, pp. 32–38.

[21] I. Statistics, "Key ict indicators for developed and developing countries and the world (totals and penetration rates)," *URL: http://www. itu. int/ITUD/ict/statistics/at_glance/KeyTelecom. html*, vol. 29, p. 2012, 2011.

[22] A. Mohamud, "Eu5 smartphone penetration reaches 55 percent in october 2012," *comScore Press Release, Dec*, 2012.

[23] comScore, "comscore reports september 2012 u.s. mobile subscriber market share)," http://www.comscore.com/Insights/Press_Releases/2012/11/comScore_Reports_September_2012_U.S._Mobile_Subscriber_Market_Share, [Online; accessed 31-May-2015].

[24] Z. Alliance, "Zigbee specification," 2006.

[25] W. Alliance, "Wifi-direct," http://www.wi-fi.org/discover-and-learn/wi-fi-direct, [Online; accessed 31-May-2015].

[26] D. Camps-Mur, A. Garcia-Saavedra, and P. Serrano, "Device to device communications with wifi direct: overview and experimentation," *IEEE Wireless Communications Magazine*, 2012.

[27] C. Gomez, J. Oller, and J. Paradells, "Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology," *Sensors*, vol. 12, no. 9, pp. 11 734–11 753, 2012.

[28] F. Ye, H. Luo, J. Cheng, S. Lu, and L. Zhang, "A two-tier data dissemination model for large-scale wireless sensor networks," in *Proceedings of the 8th annual international conference on Mobile computing and networking*. ACM, 2002, pp. 148–159.

[29] C. A. Boano, M. Zúñiga, J. Brown, U. Roedig, C. Keppitiyagama, and K. Römer, "Templab: A testbed infrastructure to study the impact of temperature on wireless sensor networks," in *Proceedings of the 13th international symposium on Information processing in sensor networks*. IEEE Press, 2014, pp. 95–106.

[30] D. Ganesan, R. Govindan, S. Shenker, and D. Estrin, "Highly-resilient, energy-efficient multipath routing in wireless sensor networks," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 5, no. 4, pp. 11–25, 2001.

[31] J. N. Al-Karaki and A. E. Kamal, "Routing techniques in wireless sensor networks: a survey," *Wireless communications, IEEE*, vol. 11, no. 6, pp. 6–28, 2004.

[32] K. Akkaya and M. Younis, "A survey on routing protocols for wireless sensor networks," *Ad hoc networks*, vol. 3, no. 3, pp. 325–349, 2005.

[33] D. Karaboga, S. Okdem, and C. Ozturk, "Cluster based wireless sensor network routing using artificial bee colony algorithm," *Wireless Networks*, vol. 18, no. 7, pp. 847–860, 2012.

[34] C.-F. Huang and Y.-C. Tseng, "The coverage problem in a wireless sensor network," *Mobile Networks and Applications*, vol. 10, no. 4, pp. 519–528, 2005.

[35] C. Zhu, C. Zheng, L. Shu, and G. Han, "A survey on coverage and connectivity issues in wireless sensor networks," *Journal of Network and Computer Applications*, vol. 35, no. 2, pp. 619–632, 2012.

[36] S. Goel and T. Imielinski, "Prediction-based monitoring in sensor networks: taking lessons from mpeg," *ACM SIGCOMM Computer Communication Review*, vol. 31, no. 5, pp. 82–98, 2001.

[37] S. Bandyopadhyay and E. J. Coyle, "An energy efficient hierarchical clustering algorithm for wireless sensor networks," in *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, vol. 3. IEEE, 2003, pp. 1713–1723.

[38] M. A. Sharaf, J. Beaver, A. Labrinidis, and P. K. Chrysanthis, "Tina: a scheme for temporal coherency-aware in-network aggregation," in *Proceedings of the 3rd ACM international workshop on Data engineering for wireless and mobile access*. ACM, 2003, pp. 69–76.

[39] J.-H. Chang and L. Tassiulas, "Maximum lifetime routing in wireless sensor networks," *IEEE/ACM Transactions on Networking (TON)*, vol. 12, no. 4, pp. 609–619, 2004.

[40] G. Anastasi, M. Conti, M. Di Francesco, and A. Passarella, "Energy conservation in wireless sensor networks: A survey," *Ad hoc networks*, vol. 7, no. 3, pp. 537–568, 2009.

[41] M. Erol-Kantarci and H. T. Mouftah, "Wireless sensor networks for cost-efficient residential energy management in the smart grid," *Smart Grid, IEEE Transactions on*, vol. 2, no. 2, pp. 314–325, 2011.

[42] K. Sohrabi, J. Gao, V. Ailawadhi, and G. J. Pottie, "Protocols for self-organization of a wireless sensor network," *IEEE personal communications*, vol. 7, no. 5, pp. 16–27, 2000.

[43] J. Zhao, R. Govindan, and D. Estrin, "Computing aggregates for monitoring wireless sensor networks," in *Sensor Network Protocols and Applications, 2003. Proceedings of the First IEEE. 2003 IEEE International Workshop on*. IEEE, 2003, pp. 139–148.

[44] S. Soro and W. B. Heinzelman, "Prolonging the lifetime of wireless sensor networks via unequal clustering," in *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*. IEEE, 2005, pp. 8–pp.

[45] Y. Zhao, J. Wu, F. Li, and S. Lu, "On maximizing the lifetime of wireless sensor networks using virtual backbone scheduling," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 23, no. 8, pp. 1528–1535, 2012.

[46] S. Feng and E. Seidel, "Self-organizing networks (son) in 3gpp long term evolution," *Nomor Research GmbH, White Paper*, 2008.

[47] H. Hu, J. Zhang, X. Zheng, Y. Yang, and P. Wu, "Self-configuration and self-optimization for lte networks," *Communications Magazine, IEEE*, vol. 48, no. 2, pp. 94–100, 2010.

[48] Y. Zhang, W. Zhao, D. Ravelosona, J.-O. Klein, J. Kim, and C. Chappert, "Perpendicular-magnetic-anisotropy cofeb racetrack memory," *Journal of Applied Physics*, vol. 111, no. 9, p. 093925, 2012.

[49] X. Jiang, L. Thomas, R. Moriya, and S. Parkin, "Discrete Domain Wall Positioning due to Pinning in Current Driven Motion along Nanowires," *Nano Letters*, vol. 11, no. 1, pp. 96–100, 2010.

[50] T. Koyama, D. Chiba, K. Ueda, K. Kondou, H. Tanigawa, S. Fukami, T. Suzuki, N. Ohshima, N. Ishiwata, Y. Nakatani, K. Kobayashi, and T. Ono, "Observation of

the Intrinsic Pinning of a Magnetic Domain Wall in a Ferromagnetic Nanowire," *Nature Materials*, vol. 10, no. 3, pp. 194–197, 2011.

[51] L. Thomas, R. Moriya, C. Rettner, and S. Parkin, "Dynamics of magnetic domain walls under their own inertia," *Science*, vol. 330, no. 6012, pp. 1810–1813, 2010.

[52] S. Fukami, T. Suzuki, K. Nagahara, N. Ohshima, Y. Ozaki, S. Saito, R. Nebashi, N. Sakimura, H. Honjo, K. Mori, C. Igarashi, S. Miura, N. Ishiwata, and T. Sugibayashi, "Low-Current Perpendicular Domain Wall Motion Cell for Scalable High-Speed MRAM," in *2009 Symposium on VLSI Technology*, 2009, pp. 230–231.

[53] R. Nebashi, N. Sakimura, Y. Tsuji, S. Fukami, H. Honjo, S. Saito, S. Miura, N. Ishiwata, K. Kinoshita, T. Hanyu, T. Endoh, N. Kasai, H. Ohno, and T. Sugibayashi, "A content addressable memory using magnetic domain wall motion cells," in *2011 Symposium on VLSI Circuits (VLSIC)*, June 2011, pp. 300–301.

[54] J. Sampaio1, L. O'Brien, D. Petit, D. E. Read, E. R. Lewis, H. T. Zeng, L. Thevenard, S. Cardoso, and R. P. Cowburn, "Coupling and Induced Depinning of Magnetic Domain Walls in Adjacent Spin Valve Nanotracks," *Journal of Applied Physics (JAP)*, vol. 113, p. 133901, 2013.

[55] Y. Zhang, W. Zhao, J.-O. Klein, C. Chappert, and D. Ravelosona, "Implementation of magnetic field assistance to current-induced perpendicular-magnetic-anisotropy racetrack memory," *Journal of Applied Physics*, vol. 115, no. 17, p. 17D509, 2014.

[56] N. Ben-Romdhane, W. Zhao, Y. Zhang, J. Klein, Z. Wang, and D. Ravelosona, "Design and analysis of racetrack memory based on magnetic domain wall motion in nanowires," in *Nanoscale Architectures (NANOARCH), 2014 IEEE/ACM International Symposium on*. IEEE, 2014, pp. 71–76.

[57] O. Boulle, L. Buda-Prejbeanu, E. Jué, I. Miron, and G. Gaudin, "Current induced domain wall dynamics in the presence of spin orbit torques," *Journal of Applied Physics*, vol. 115, no. 17, p. 17D502, 2014.

[58] A. Khvalkovskiy, V. Cros, D. Apalkov, V. Nikitin, M. Krounbi, K. Zvezdin, A. Anane, J. Grollier, and A. Fert, "Matching domain-wall configuration and spin-orbit torques for efficient domain-wall motion," *Physical Review B*, vol. 87, no. 2, p. 020402, 2013.

[59] A. Bernand-Mantel, L. Herrera-Diez, L. Ranno, S. Pizzini, J. Vogel, D. Givord, S. Auffret, O. Boulle, I. M. Miron, and G. Gaudin, "Electric-field control of domain wall nucleation and pinning in a metallic ferromagnet," *Applied Physics Letters*, vol. 102, no. 12, p. 122406, 2013.

[60] I. Polenciuc, A. Vick, D. Allwood, T. Hayward, G. Vallejo-Fernandez, K. O'Grady, and A. Hirohata, "Domain wall pinning for racetrack memory using exchange bias," *Applied Physics Letters*, vol. 105, no. 16, p. 162406, 2014.

[61] R. Venkatesan, V. Kozhikkottu, C. Augustine, A. Raychowdhury, K. Roy, and A. Raghunathan, "Tapecache: a high density, energy efficient cache based on domain wall memory," in *Proceedings of the 2012 ACM/IEEE international symposium on Low power electronics and design*. ACM, 2012, pp. 185–190.

[62] R. Nebashi, N. Sakimura, Y. Tsuji, S. Fukami, H. Honjo, S. Saito, S. Miura, N. Ishiwata, K. Kinoshita, T. Hanyu, T. Endoh, N. Kasai, H. Ohno, and T. Sugibayashi, "A content addressable memory using magnetic domain wall motion cells," in *Symposium on VLSI Circuits (VLSIC)*, Jun. 2011, pp. 300–301.

[63] W. Zhao, N. Ben Romdhane, Y. Zhang, J.-O. Klein, and D. Ravelosona, "Racetrack memory based reconfigurable computing," in *Faible Tension Faible Consommation (FTFC), 2013 IEEE*. IEEE, 2013, pp. 1–4.

[64] L. Thomas, S.-H. Yang, K.-S. Ryu, B. Hughes, C. Rettner, D.-S. Wang, C.-H. Tsai, K.-H. Shen, and S. Parkin, "Racetrack memory: A high-performance, low-cost, non-volatile memory based on magnetic domain walls," in *IEEE International Electron Devices Meeting (IEDM)*, Dec. 2011, pp. 24.2.1–24.2.4.

[65] A. Iyengar and S. Ghosh, "Modeling and analysis of domain wall dynamics for robust and low-power embedded memory," in *Proceedings of the The 51st Annual Design Automation Conference on Design Automation Conference*. ACM, 2014, pp. 1–6.

[66] N. Muralimanohar, R. Balasubramonian, and N. Jouppi, "Optimizing nuca organizations and wiring alternatives for large caches with cacti 6.0," in *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 2007, pp. 3–14.

[67] J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N. S. Kim, T. M. Aamodt, and V. J. Reddi, "Gpuwattch: Enabling energy optimizations in gpgpus," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*. ACM, 2013, pp. 487–498.

[68] H. Jang, B. S. An, N. Kulkarni, K. H. Yum, and E. J. Kim, "A hybrid buffer design with stt-mram for on-chip interconnects," in *Networks on Chip (NoCS), 2012 Sixth IEEE/ACM International Symposium on*. IEEE, 2012, pp. 193–200.

[69] Z. Sun, W. Wu, and H. Li, "Cross-layer racetrack memory design for ultra high density and low power consumption," in *Design Automation Conference (DAC), 2013 50th ACM/EDAC/IEEE*. IEEE, 2013, pp. 1–6.

[70] M. Mao, W. Wen, Y. Zhang, Y. Chen, and H. H. Li, "Exploration of gpgpu register file architecture using domain-wall-shift-write based racetrack memory," in *Proc. of DAC*, 2014, pp. 1–6.

[71] M. Sharad, R. Venkatesan, A. Raghunathan, and K. Roy, "Multi-level magnetic ram using domain wall shift for energy-efficient, high-density caches," in *Proc. of ISLPED*, 2013, pp. 64–69.

[72] H. Xu, Y. Li, R. Melhem, and A. K. Jones, "Multilane racetrack caches: Improving efficiency through compression and independent shifting," in *Design Automation Conference (ASP-DAC), 2015 20th Asia and South Pacific*. IEEE, 2015, pp. 417–422.

[73] R. Venkatesan, S. G. Ramasubramanian, S. Venkataramani, K. Roy, and A. Raghunathan, "Stag: Spintronic-tape architecture for gpgpu cache hierarchies," in *Proc. of ISCA*. IEEE, 2014, pp. 253–264.

[74] L. P. Deutsch, "Gzip file format specification version 4.3," 1996.

[75] W. B. Pennebaker and J. L. Mitchell, *JPEG: Still image data compression standard*. Springer Science & Business Media, 1993.

[76] K. Brandenburg and M. Bosi, "Overview of MPEG audio: Current and future standards for low bit-rate audio coding," *J. Audio Eng. Soc.*, vol. 45, no. 1/2, pp. 4–21, January/February 1997.

[77] I. E. Richardson, *H. 264 and MPEG-4 video compression: video coding for next-generation multimedia*. John Wiley & Sons, 2004.

[78] J. Yang, Y. Zhang, and R. Gupta, "Frequent value compression in data caches," in *Proc. of the 33rd annual ACM/IEEE international symposium on Microarchitecture*, 2000, pp. 258–265.

[79] A. R. Alameldeen and D. A. Wood, "Frequent pattern compression: A significance-based compression scheme for l2 caches," *Dept. Comp. Scie., Univ. Wisconsin-Madison, Tech. Rep*, vol. 1500, 2004.

[80] G. Pekhimenko, V. Seshadri, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, "Base-delta-immediate compression: practical data compression for on-chip caches," in *Proc. of PACT*, 2012, pp. 377–388.

[81] G. Pekhimenko, V. Seshadri, Y. Kim, H. Xin, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, "Linearly compressed pages: a low-complexity, low-latency main memory compression framework," in *Proc. of MICRO*, 2013, pp. 172–184.

[82] A. Shafiee, M. Taassori, R. Balasubramonian, and A. Davis, "Memzip: Exploring unconventional benefits from memory compression," *HPCA-20*, 2014.

[83] Y. Li, H. Xu, R. Melhem, and A. K. Jones, "Space oblivious compression: Power reduction for non-volatile main memories," in *GLSVLSI*. IEEE, 2015.

[84] H. Xu, M. M. Bilec, L. A. Schaefer, A. E. Landis, and A. K. Jones, "Ocelot: A wireless sensor network and computing engine with commodity palmtop computers," in *Green Computing Conference (IGCC), 2013 International*. IEEE, 2013, pp. 1–8.

[85] J. Kusuma, L. Doherty, and K. Ramchandran, "Distributed compression for sensor networks." in *ICIP (1)*, 2001, pp. 82–85.

[86] L. Xiang, J. Luo, and A. Vasilakos, "Compressed data aggregation for energy efficient wireless sensor networks," in *Sensor, mesh and ad hoc communications and networks (SECON), 2011 8th annual IEEE communications society conference on*. IEEE, 2011, pp. 46–54.

[87] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[88] C. Jardak, J. Riihijärvi, F. Oldewurtel, and P. Mähönen, "Parallel processing of data from very large-scale wireless sensor networks," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*. ACM, 2010, pp. 787–794.

[89] M. C. Vuran, Ö. B. Akan, and I. F. Akyildiz, "Spatio-temporal correlation: theory and applications for wireless sensor networks," *Computer Networks*, vol. 45, no. 3, pp. 245–259, 2004.

[90] P. Garbacki, D. H. Epema, and M. Van Steen, "The design and evaluation of a self-organizing superpeer network," *Computers, IEEE Transactions on*, vol. 59, no. 3, pp. 317–331, 2010.

[91] Apple, "Apple push notification service," http://developer.apple.com/library/mac/#documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/ApplePushService/ApplePushService.html, [Online; accessed 31-May-2015].

[92] Google, "Google cloud messaging for android," http://developer.android.com/google/gcm/index.html/, [Online; accessed 31-May-2015].

[93] T. Blasing, L. Batyuk, A.-D. Schmidt, S. Camtepe, and S. Albayrak, "An android application sandbox system for suspicious software detection," in *Malicious and Unwanted Software (MALWARE), 2010 5th International Conference on*, oct. 2010, pp. 55 –62.

[94] T. Malmqvist, M. Glaumann, S. Scarpellini, I. Zabalza, A. Aranda, E. Llera, and S. Díaz, "Life cycle assessment in buildings: The enslic simplified method and guidelines," *Energy*, vol. 36, no. 4, pp. 1900–1907, 2011.

[95] W. O. Collinge, L. Liao, H. Xu, C. L. Saunders, M. M. Bilec, A. E. Landis, A. K. Jones, and L. A. Schaefer, "Enabling dynamic life cycle assessment of buildings with

wireless sensor networks," in *Sustainable Systems and Technology (ISSST), 2011 IEEE International Symposium on*. IEEE, 2011, pp. 1–6.

[96] M. M. Bilec, R. J. Ries, and H. S. Matthews, "Life-cycle assessment modeling of construction processes for buildings," *Journal of infrastructure systems*, vol. 16, no. 3, pp. 199–205, 2009.

[97] S. Junnila and A. Horvath, "Life-cycle environmental effects of an office building," *Journal of Infrastructure Systems*, vol. 9, no. 4, pp. 157–166, 2003.

[98] Qualcomm, "Alljoyn," https://www.alljoyn.org/about, [Online; accessed 31-May-2015].

[99] S. Verma, A. Robinson, and P. Dutta, "Audiodaq: turning the mobile phone's ubiquitous headset port into a universal data acquisition interface," in *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*. ACM, 2012, pp. 197–210.

[100] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sniper: exploring the level of abstraction for scalable and accurate parallel multi-core simulation," in *Proc. of SC*, 2011, p. 52.

[101] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, "Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *TACD*, vol. 31, no. 7, pp. 994–1007, 2012.

[102] C. Augustine, A. Raychowdhury, B. Behin-Aein, S. Srinivasan, J. Tschanz, V. K. De, and K. Roy, "Numerical analysis of domain wall propagation for dense memory arrays," in *Proc. of IEDM*, 2011, pp. 17–6.

[103] S. Fukami, T. Suzuki, N. Ohshima, K. Nagahara, and N. Ishiwata, "Micromagnetic analysis of current driven domain wall motion in nanostrips with perpendicular magnetic anisotropy," *Journal of Applied Physics*, vol. 103, no. 7, p. 07E718, 2008.

[104] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: Characterization and architectural implications," in *Proc. of PACT*, 2008, pp. 72–81.

[105] J. L. Henning, "Spec cpu2006 benchmark descriptions," *ACM SIGARCH Computer Architecture News*, vol. 34, no. 4, pp. 1–17, 2006.

[106] L. Villa, M. Zhang, and K. Asanović, "Dynamic zero compression for cache energy reduction," in *Proc. of MICRO*, 2000, pp. 214–220.

[107] Y. Sazeides and J. E. Smith, "The predictability of data values," in *Proc. of MICRO*, 1997, pp. 248–258.

[108] M. Alioto and G. Palumbo, "Analysis and comparison on full adder block in submicron technology," *IEEE Trans. on VLSI Systems*, vol. 10, no. 6, pp. 806–823, 2002.

[109] E. S. Shin, V. J. Mooney III, and G. F. Riley, "Round-robin arbiter design and generation," in *Proc. of the 15th international symposium on System Synthesis*, 2002, pp. 243–248.

[110] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "Cacti 6.0: A tool to model large caches," *HP Laboratories*, 2009.

[111] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *Proc. of IISWC*, 2009, pp. 44–54.

[112] USDOE, "Electric power annual 2009 - state data tables," http://www.eia.gov/electricity/data/state/, [Online; accessed 23-March-2015].

[113] I. S. Association *et al.*, *IEEE Standard for Information Technology-Telecommunications and Information Exchange Between Systems-Local and Metropolitan Area Networks-Specific Requirements: Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*. IEEE, 2001.

[114] Yahoo, "1976-2014 Nasdaq composite index," http://finance.yahoo.com/q?s=%5EIXIC, 2015, [Online; accessed 31-May-2015].

[115] E. Ferro and F. Potorti, "Bluetooth and wi-fi wireless protocols: a survey and a comparison," *Wireless Communications, IEEE*, vol. 12, no. 1, pp. 12–26, 2005.

[116] R. Hameed, W. Qadeer, M. Wachs, O. Azizi, A. Solomatnikov, B. C. Lee, S. Richardson, C. Kozyrakis, and M. Horowitz, "Understanding sources of inefficiency in general-purpose chips," in *ACM SIGARCH Computer Architecture News*, vol. 38, no. 3. ACM, 2010, pp. 37–47.

[117] A. Carroll and G. Heiser, "An analysis of power consumption in a smartphone." in *USENIX annual technical conference*, vol. 14, 2010.

[118] P. Teehan and M. Kandlikar, "Comparing embodied greenhouse gas emissions of modern computing and electronics products," *Environmental science & technology*, vol. 47, no. 9, pp. 3997–4003, 2013.