University of Tennessee, Knoxville
Trace: Tennessee Research and Creative Exchange

Doctoral Dissertations | Graduate School

5-2014

Achieving Energy Efficiency on Networking Systems with Optimization Algorithms and Compressed Data Structures

Yanjun Yao
University of Tennessee - Knoxville, yyao9@utk.edu

This Dissertation is brought to you for free and open access by the Graduate School at Trace: Tennessee Research and Creative Exchange. It has been accepted for inclusion in Doctoral Dissertations by an authorized administrator of Trace: Tennessee Research and Creative Exchange. For more information, please contact trace@utk.edu.

To the Graduate Council:

I am submitting herewith a dissertation written by Yanjun Yao entitled "Achieving Energy Efficiency on Networking Systems with Optimization Algorithms and Compressed Data Structures." I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Computer Science.

Qing Cao, Major Professor

We have read this dissertation and recommend its acceptance:

Hairong Qi, Husheng Li, Jindong Tan

Accepted for the Council:
Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

Achieving Energy Efficiency on Networking Systems with Optimization Algorithms and Compressed Data Structures

A Dissertation Presented for the

Doctor of Philosophy

Degree

The University of Tennessee, Knoxville

Yanjun Yao

May 2014

© by Yanjun Yao, 2014

All Rights Reserved.

To my parents, with love and gratitude.

Acknowledgements

Here I want to express my appreciation to those people who contributed to the development of this thesis with either their personal or technical support.

First, I want to express my deepest gratitude to my thesis advisor, Dr. Qing Cao, whose expertise, patience, support, and understanding carried me through my PhD study, and helped me to build my research skills.

Second, I want also to express my thank to my other committee members, Dr. Hairong Qi, Dr. Husheng Li, and Dr. Jindong Tan, who provided valuable comments and constructive suggestions to my research.

Third, I want to express my appreciation to Dr. Xiaorui Wang, who helped me broaden my knowledge and build a solid research foundation.

Fourth, I would like to thank all my friends and research staff at the Laboratory for Autonomous, Interconnected, and Embedded Systems (Lanterns) for providing a vibrant working environment. I especially thank Jilong Liao, Lipeng Wan, Sisi Xiong, Zheng Lu, Zhibo Wan, Xiaodong Wang, Chi Zhang, Yanwei Zhang, Xue Li, Yefu Wang, and Xing Fu for their valuable help and precious friendship.

Last but not least, I want to give me sincerest thanks to my parents for their unconditional love and support.

Abstract

To cope with the increasing quantity, capacity and energy consumption of transmission and routing equipment in the Internet, energy efficiency of communication networks has attracted more and more attention from researchers around the world. In this dissertation, we proposed three methodologies to achieve energy efficiency on networking devices: the NP-complete problems and heuristics, the compressed data structures, and the combination of the first two methods.

We first consider the problem of achieving energy efficiency in Data Center Networks (DCN). We generalize the energy efficient networking problem in data centers as optimal flow assignment problems, which is NP-complete, and then propose a heuristic called CARPO, a correlation-aware power optimization algorithm, that dynamically consolidate traffic flows onto a small set of links and switches in a DCN and then shut down unused network devices for power savings.

We then achieve energy efficiency on Internet routers by using the compressive data structures. A novel data structure called the Probabilistic Bloom Filter (PBF), which extends the classical bloom filter into the probabilistic direction, so that it can effectively identify heavy hitters with a small memory foot print to reduce energy consumption of network measurement.

To achieve energy efficiency on Wireless Sensor Networks (WSN), we developed one data collection protocol called EDAL, which stands for Energy-efficient Delay-aware Lifetime-balancing data collection. Based on the Open Vehicle Routing problem, EDAL

exploits the topology requirements of Compressive Sensing (CS), then implement CS to save more energy on sensor nodes.

Table of Contents

1 Introduction **1**

2 Previous Works **4**

2.1 NP-complete and heuristics 4

2.1.1 Optimal Flow Assignment Problem 4

2.1.2 Open Vehicle Routing Problem 5

2.2 Compressed data structures 6

2.2.1 Compressive Sensing 6

2.2.2 Bloom Filter 7

3 CARPO: Correlation-Aware Power Optimization in Data Center Networks **9**

3.1 Abstract 9

3.2 Introduction 10

3.3 Related Work 14

3.4 Problem Formulation 15

3.4.1 Characteristics of Network Switch 15

3.4.2 Optimal Consolidation and Rate Configuration 16

3.5 Design of CARPO 18

3.5.1 Traffic Correlation Analysis 19

3.5.2 CARPO Framework 23

3.5.3 Correlation-aware Consolidation Algorithm 25

3.5.4 Link Rate Adaptation 27

3.6 Hardware Evaluation 27
3.6.1 Hardware Testbed and Baselines 27
3.6.2 Experiment Results 28
3.7 Simulation Evaluation 31
3.7.1 Experimental Setup 31
3.7.2 Simulation Results for Yahoo! DCP Data Center Workload 32
3.7.3 Simulation results in Large-Scale DCNs 33
3.7.4 Performance with Different Numbers of Sample Points in Correlation Calculation 35
3.8 Discussion 36
3.8.1 Model for ElasticTree 36
3.8.2 Model for CARPO 37
3.8.3 Power Estimation Model Evaluation 38
3.9 Conclusion 39

4 Identifying Frequent Flows in Large Traffic Sets through Probabilistic Bloom Filters 40
4.1 Abstract 40
4.2 Introduction 41
4.3 Related Work 44
4.4 Probabilistic Bloom Filter 46
4.4.1 Programming Interfaces 46
4.4.2 Properties of the PBF Operations 47
4.4.3 Performance Modeling of PBFs 48
4.4.4 Selection of Parameters for PBFs 51
4.4.5 The Maximum Estimative Frequency 56
4.5 Extensions of the PBF 57
4.5.1 C-PBF: Counting PBF Design 57
4.5.2 Selection of Parameters for C-PBF 58

4.5.3 T-PBF: Time-decaying PBF Design 61
4.5.4 Selection of Parameters for T-PBF 64
4.6 Parameter Selection Algorithms are Nash Equilibriums 68
4.7 Evaluation 73
4.7.1 Dataset A: Web Query Log Analysis 74
4.7.2 Dataset B: Network Measurement Dataset Analysis 82
4.8 Conclusion 89

5 EDAL: an Energy-efficient, Delay-aware, and Lifetime-balancing Data Collection Protocol for Heterogeneous Wireless Sensor Networks 91
5.1 Abstract 91
5.2 Introduction 92
5.3 Background 94
5.3.1 Vehicle Routing Problems 94
5.3.2 Compressive Sensing 95
5.4 EDAL Algorithm Design 97
5.4.1 Problem Model 97
5.4.2 Complexity Analysis 99
5.4.3 Centralized Heuristics 101
5.4.4 Distributed Heuristics 105
5.5 Simulation based Evaluation 109
5.5.1 Experimental Settings 110
5.5.2 Algorithm Overhead 111
5.5.3 Experiment Results for Network with Homogeneous Nodes 114
5.5.4 Experiment Results for Network with Heterogeneous Nodes 122
5.5.5 EDAL Application for Sparse Event Detection 127
5.6 Hardware Evaluation 128
5.6.1 Experimental Settings 129
5.6.2 Hardware Experiment Results 130

5.7 Conclusion 132

6 Conclusions **133**

Bibliography **135**

Vita **146**

List of Tables

3.1 Power Consumption of PRONTO 3240 Switch with Different Port Speed. . 16
3.2 Correlation values for flow pairs in Figure 3.8. 25

4.1 Symbols Used in Analysis 48
4.2 Cost for player 1. 69
4.3 Cost for player 2. 69
4.4 Cost for player 3. 70

5.1 Notations of EDAL 98
5.2 Reconstruction error with different node selection probabilities and event sparsity for $N = 256$, $S = 256$ 128

List of Figures

3.1 Average workload for 4 grouped IPs from Yahoo! data centers within 1 day. 20

3.2 CDF distributions of pair-wise correlation coefficient in 5 Yahoo! data centers. 20

3.3 CDF of the normalized data rate for the flows in Figure 3.1. The data rate of each flow is normalized to its own maximum value. 21

3.4 CDF of the 90-percentile of the normalized data rate with 70 flows. 21

3.5 Standard deviation of correlation coefficients between 8 grouped IP flows. Half of the values are removed because of the symmetry. 23

3.6 90-percentile workload of every 10 minutes. Moving average value is calculated with a windows size of 60 minutes. 24

3.7 The proposed CARPO framework. 24

3.8 Flow placement example by CARPO. Sub-figure (a) shows the initial flow assignment. Sub-figure (b) shows the aggregated flow assignment after applying CARPO. s_i is server ID. f_i is flow ID. e_i, a_i, c_i are IDs of $edge$, $aggregation$ and $core$ switches, respectively. Different line color-pattern combinations are different flow assignments. The 90-percentile utilization values are labeled for the four flows. 25

3.9 Hardware testbed with 10 virtual switches (VSs) configured from a production 48-port OpenFlow switch and 8 servers. The VSs are numbered in the same as in Figure 5. 27

3.10 Power consumption of the testbed running the workload in Figure 3.1. . . . 30

3.11 Power savings on the testbed running the workload in Figure 3.1. 30
3.12 Example (*pod-10* fat tree) of network topology for simulations. 31
3.13 Average power savings in simulations running Yahoo! DCP data center workload. 32
3.14 Average packet delay in simulations running Yahoo! DCP data center workload. 32
3.15 Average network power consumption in large scale DCN. 34
3.16 Average packet delay in large scale DCN. 34
3.17 Performance impact of different number of sample points in correlation analysis. 35
3.18 Compare Result with Estimation and Simulation. 38

4.1 Relation between p and the estimation bounds. 52
4.2 Relation between k and the estimation bounds. 52
4.3 Relation between p and θ. 53
4.4 The Markov Transition for the T-PBF 63
4.5 Relation between p and estimation bounds. 66
4.6 Relation between k and estimation bounds. 67
4.7 Relation between m and estimation bounds. 67
4.8 Relation between q and estimation bounds. 68
4.9 The daily pattern for the web query dataset AOL (2006). 74
4.10 Estimations of frequency results of the PBF. 75
4.11 Estimations of frequency results of the C-PBF. 76
4.12 Estimations of frequency results of the CBF. 77
4.13 Estimations of frequency results of the MRSCBF. 77
4.14 Estimations of frequency results of the RSN. 77
4.15 Detect popular key word with the PBF for the web query dataset. 78
4.16 Detect popular key word with the C-PBF for the web query dataset. 79
4.17 Detect popular key word with the CBF for the web query dataset. 80

4.18 Detect popular key word with the MRSCBF for the web query dataset. . . . 80
4.19 Detect popular key word with the RSN for the web query dataset. 80
4.20 Daily frequency trend estimation. 81
4.21 Dataset traffic pattern, generated based on CAIDA (2013) 82
4.22 Dataset flow frequency, generated based on CAIDA (2013) 82
4.23 Comparison between the real frequency and the estimated frequency with the PBF. 83
4.24 Comparison between the real frequency and the estimated frequency with the C-PBF. 84
4.25 Comparison between the real frequency and the estimated frequency with the CBF. 85
4.26 Comparison between the real frequency and the estimated frequency with the MRSCBF. 85
4.27 Comparison between the real frequency and the estimated frequency with the RSN. 86
4.28 Comparison of the real heavy hitter ratio and the detected heavy hitter ratio of the T-PBF. 87
4.29 The average error rate of the PBF with $p = 0.0001$, and different k and m values. 88
4.30 The false positive ratio of the PBF. 88
4.31 The false negative ratio of the PBF. 88
4.32 The memory overhead of the PBF with different k values. 89

5.1 The worst case and optimal solution of RPFIH. 102
5.2 Computational time overhead of the centralized heuristic under different network sizes 112
5.3 Computational time overhead of the distributed heuristic with different gossip ranges in a network with 256 nodes 113
5.4 The average power distribution of each node for the centralized algorithm. . 113

5.5 The average power distribution of each node for the distributed algorithm. . 114
5.6 The average node number used in each period by different algorithms with different delay requirements 115
5.7 The average energy consumption of the network running different routing algorithms 116
5.8 The network lifetime while running different routing algorithms with different delay requirements 116
5.9 The CDF of delay of packets generated in the whole network duration from different routing algorithms 117
5.10 The average energy consumption of the network with CS while running different routing algorithms. 118
5.11 The network lifetime with CS implemented on different routing algorithms. 118
5.12 The CDF of remaining energy of each node while running different routing algorithms 120
5.13 The network lifetime with CS implemented on different routing algorithms with delay constraint as 135 ms 120
5.14 The reconstruction error under different compression rate, data sparsity, and source node number. 121
5.15 The average node number used in each period by different algorithms in the heterogeneous network 122
5.16 The network lifetime while running different routing algorithms in the heterogeneous network 123
5.17 The average energy consumption of the heterogeneous network while running different routing algorithms. 124
5.18 The CDF of delay of packets generated in the heterogeneous network from different routing algorithms. 125
5.19 The lifetime of the heterogeneous network with CS implemented on different routing algorithms 125

5.20 The average energy consumption of the heterogeneous network with CS while running different routing algorithms. 126
5.21 The CDF of remaining energy of each node while running different routing algorithms in the heterogeneous network 127
5.22 Hardware testbed topology. 129
5.23 The network lifetime while running different routing algorithms in the small scale testbed network 130
5.24 The average energy consumption while running different routing algorithms in the small scale testbed network 130
5.25 The CDF of remaining energy of each node while running different routing algorithms in the small scale testbed network 131
5.26 The CDF of delay of packets generated in the whole testbed network duration from different routing algorithms 131

Chapter 1

Introduction

Due to the increasing numbers of Internet users and higher bandwidth services, the Internet traffic keeps on increasing in recent years. As a result of that, the quantity, capacity and energy consumption of transmission and routing equipment is also grown to handle these traffics. Raghavan and Ma (2011) estimated that the annually Internet power consumption is *between 170 and 370 GW*. More specifically, Gupta and Singh (2003) break down the Internet power consumption based on networking device type, and found that 26.4% in total power goes to Internet hubs, 52.9% goes to LAN switches, 2.5% goes to WAN switches, and 18.2% goes to routers. The huge amount of Internet energy leads to tremendous cost, and also postpones the spreading of Internet deployment in the places, where electricity is a rare resource.

More specifically, in recent years, high energy consumption has become one of the most important concerns for large-scale data centers that are rapidly increasing the number of hosted servers. For example, in a 2007 report to the US Congress, the Agency (2007) estimated that the annual data center energy consumption in the US will grow to over 100 billion kWh at a cost of $7.4 billion by 2011. As a result, minimizing the energy consumption of data centers has recently attracted a lot of research efforts. However, current research focuses mostly on computer servers to lower their power consumption, while only few studies have tried to address data center networks (DCNs), which can

account for 10% to 20% of the total energy consumption of a data center Greenberg et al. (2008); Heller et al. (2010). The percentage of network energy can even increase to 50% in future data centers, where servers become more energy-proportional to their workloads Abts et al. (2010).

On the other hand, in wireless sensor networks (WSNs), where limited energy, computation and transmission resources are equipped in sensor nodes, energy efficiency is critical for network lifetime. Prolonging WSN lifetime leads to the less network maintenance and device replacement cost. Especially for those network deployed in the wild field, where is hard for human to access, frequent node energy depletion would be an unpleasant experience.

In the third aspect, the Internet routers, which consume 18.2% of total internet power, is also a nonnegligible part.

Based on all above, developing power efficient algorithms on networking systems is a very important topic. Although a lot of works has been done in this area, there is still a space for us to work on. In that case, in my dissertation, we plan to achieve energy efficiency on networking systems through the following three methodologies:

- **NP-complete problems and heuristics:** the networking energy efficiency problems are usually produced as NP-complete problem, such as spanning tree problem, routing problems, flow problems, and so on. The most prominent characteristic of NP-complete problem is that there are no fast solutions, especially for large scale networks. In that case, the approximation solutions (find an almost optimal solution, instead of searching for an optimal solution), and heuristics (algorithms works reasonably well) are widely used. In our research, we generalize the energy efficiency networking problem in data centers as optimal flow assignment problems, which is NP-complete, and propose a heuristic called CARPO, a correlation-aware power optimization algorithm, that dynamically consolidates traffic flows onto a small set of links and switches.

- **Compressed data structures :** in compressed data structures, their operations are roughly as fast or accurate as conventional data structures for the same problem, but their data sizes are substantially smaller. In that case, implementing compressed data structures on networking systems can achieve energy efficiency on two aspects: on network traffics, compressed data structures can reduce the transmission traffic volume, which decreases the packet transmission power; on the other hand, on networking devices, compressed data structures enables smaller memory usages, which saves energy on memory banks. In our research, we implement the Probabilistic Bloom Filter (PBF) on network routers to detect heavy hitter flows with a small memory size.

- **Combination of the above two :** by carefully design the routing problem in networks, we can develop routes that are more suitable for the further implementation of compressed data structures. As a result of that, two rounds of energy efficiency can be achieved. In our research, we construct routes by solving the open vehicle routing problem to connect as many nodes as possible to the same route, while guarantee the packet delay requirements. The result routes provides a better platform for implementing compressive sensing to reduce transmitted packet number, and also achieve better energy efficiency.

Chapter 2

Previous Works

There are a lot works have been done for saving power on data center networks, wireless sensor networks and network routers with NP complete algorithms and compressed data structures.

2.1 NP-complete and heuristics

2.1.1 Optimal Flow Assignment Problem

Putting network devices to sleep Gupta and Singh (2003); Gupta et al. (2004); Ananthanarayanan and Katz (2008) is the most common method for saving power on Internet. However, none of these studies put their proposed scheme into the data center application context, which have special traffic characteristics. The oversubscription of a multi-tiered DCN leads to short idle periods Abts et al. (2010) of the network, which results in a high transitional power consumption on the network device with periodic sleeping.

In that case, optimal flow assignment, which consolidates traffic flows to a subset of network nodes that consume the minimal power, is the important means for achieving energy-proportional DCNs. The researchers of the *ElasticTree* project Heller et al. (2010) proposed to consolidate traffic flows in a DCN onto a small set of links and switches such that unused network equipment could be turned off for power saving. Although their

consolidation approach was developed based on a real datacenter traces, they assume that the traffic rate of each data flow is approximately a constant. This is not entirely true in the current production datacenters. The second approach for energy-proportional DCN is to adapt the link rate according to the workload of each traffic flow Nedevschi et al. (2008); Abts et al. (2010); Gunaratne et al. (2008). One problem of solely using link rate adaptation is that the power saving gain is relatively small since a large amount of power is consumed by components such as fans and switch fabrics.

2.1.2 Open Vehicle Routing Problem

The vehicle routing problem (VRP) Eksioglu et al. (2009) is a well-known NP-hard problem in operational research. VRP finds routes between a depot and customers with given demands so that the transportation cost is minimized with the involvement of the minimum number of vehicles, while satisfying capacity constraints. With additional constraints, VRP can be further extended to solve different problems, where one of the most important is the vehicle routing problem with time windows (VRPTW) Braysy and Gendreau (2005). This problem happens frequently in the distribution of goods and services, where an unlimited number of identical vehicles with predefined capacity serve a set of customers with demands of different time intervals (time windows). VRPTW tries to minimize the total transportation cost through the minimum number of vehicles, without violating any timing constraints in delivering goods. If vehicles are not required to return back to the depot, and if the time windows are replaced by deadlines, VRPTW can be further extended to the open vehicle routing problem with time deadlines (OVRP-TD) Ozyurt et al. (2006).

As an NP-hard problem, OVRP-TD has inspired many heuristics. Ozyurt et al. (2006) proposed the nearest insertion method, where the farthest node is chosen first to be connected with a route. Then, repeatedly, each selected node chooses the nearest neighbor that has not been assigned a route so far, and connects itself to this neighbor. This procedure repeats until all customers are connected by routes. Solomon (1987) developed the push

forward insertion heuristic (PFIH), which repeatedly selects the customer with the lowest additional insertion cost as the next node, until all customers are routed. Once initial routes have been found, various algorithms Du and He (2012); Cheng and Wang (2009); Chiang and Russell (1996); Ozyurt et al. (2006) are developed to generate near optimal solutions based on simulated annealing Skiscim and Golden (1983), tabu search Tan et al. (2001), or genetic programming Holland (1992).

2.2 Compressed data structures

2.2.1 Compressive Sensing

Compressive sensing (CS) is a technique through which data are compressed during their transmission to a given destination, by exploiting the fact that most sensors may not always have valid data to report when they sample the environment Caione et al. (2012); Wu (2009); Cao et al. (2011); Luo et al. (2009); Zheng et al. (2012); Ling and Tian (2010); Zhu and Wang (2010); Zheng et al. (2011). It works as follows. Consider the case that there are N nodes generating N segments of data. Such data are K-sparse, meaning only K of them are non-zero. We can compress these N pieces of data into M pieces through a linear transformation to reduce the number of packets.

Because CS promises improved energy efficiency and lifetime balancing properties Cao et al. (2011), data gathering protocols have been proposed to exploit CS for better performance. Xiang et al. (2011) proposed a new data aggregation technique derived from CS to minimize the total energy consumption through joint routing and compressed aggregation. Mehrjoo et al. (2010) employed CS and particle swarm optimization algorithms to build up data aggregation trees and decrease communication rate. These two methods require all nodes to contribute sensing data during the data collection phase. On the other hand, Wang et al. (2010) proposed random routing methods based on different network topologies to collect data from a subset of nodes.

2.2.2 Bloom Filter

The Bloom Filter (BF), which is proposed by Burton H. Bloom in 1970 Bloom (1970), is a space efficient randomized data structure that answers the question about if an element is already in a set. The space efficiency is achieved at the cost of false positives (an element is claimed to be part of a set when it is not). The accuracy of a Bloom Filter depends on the filter size m, the number of used hash functions k, and the number of inserted elements n. There are two basic operations: insert, and query. In the insert operation, the k hash functions will hash each element into k bits in m, and set the corresponding bits to 1s. In the query operation, it will check if all k hashed bits for the target element are 1s, and report positive if they are. The false negatives (an element is claimed to not be part of a set when it is) never happens.

After the original BF was proposed, a large number of variants followed. We refer to two surveys Broder and Mitzenmacher (2003); Tarkoma et al. (2012) for detailed description. Among them, the most relevant to this work is the Counting Bloom Filter (CBF), as proposed by Fan et al. (2000), which has m counters along with m bits. This way, CBF can support not only deletion operations, but also frequency queries. However, CBFs are known for their significantly increased memory overhead. Another work, proposed by Shen and Zhang (2008), developed the idea of Decaying Bloom Filter (DBF), which extended the CBF to support the removal of stale elements when new elements are inserted. In our work, we present extensions of PBF for counting and decaying as well.

In recent years, the BF has been widely used in the context of network measurement. Estan and Varghese (2002a) applied CBFs to traffic measurement problems inside routers. The approach was based on the simple idea that if the counter for a flow increases beyond a threshold, it should be considered as a frequent flow. Zhao et al. (2006) used the BF to find local icebergs (items whose frequency is larger than a given threshold) in a distributed manner, and then estimated global icebergs in a central server. Finally, Liu et al. (2012) proposed the Reversible MultiLayer Hashed Counting Bloom Filter(RML-HCBF), whose hash functions select a set of consecutive bits from the original strings as hash values, so

that it may find elephant flows (large and continuous flow) using the counter values and thresholds.

Chapter 3

CARPO: Correlation-Aware Power Optimization in Data Center Networks

3.1 Abstract

Power optimization has become a key challenge in the design of large-scale enterprise data centers. Existing research efforts focus mainly on computer servers to lower their power consumption, while only few studies have tried to address data center networks (DCNs), which can account for 20% of the total power consumption of a data center. In this chapter, we propose CARPO, a correlation-aware power optimization algorithm that dynamically consolidates traffic flows onto a small set of links and switches in a DCN and then shuts down unused network devices for power savings. In sharp contrast to existing work, CARPO is designed based on a key observation from the analysis of real DCN traces that the bandwidth demands of different flows do not peak at exactly the same time. As a result, if the correlations among flows are considered in consolidation, more power savings can be achieved. CARPO also integrates traffic consolidation with link rate adaptation for maximized power savings. Furthermore, CARPO generalizes previous work to present an analytical framework that theoretically estimates how much power can be saved for a given DCN topology and workloads. We implement CARPO on a hardware testbed composed of

10 virtual switches configured with a production 48-port OpenFlow switch and 8 servers. Our empirical results with Yahoo! DCN traces demonstrate that CARPO can save up to 60% of network power for a DCN, while having only negligible delay increases. CARPO also outperforms two state-of-the-art baselines by having approximately 10% and 20% more power savings, respectively. Our simulation results with a trace file of 10 data centers composed of 5,415 servers also show the power efficiency of CARPO in large-scale data centers.

3.2 Introduction

In recent years, high energy consumption has become one of the most important concerns for large-scale data centers that are rapidly increasing the number of hosted servers. For example, in a 2007 report to the US Congress Agency (2007), the Environmental Protection Agency (EPA) estimated that the annual data center energy consumption in the US will grow to over 100 billion kWh at a cost of $7.4 billion by 2011. As a result, minimizing the energy consumption of data centers has recently attracted a lot of research efforts. However, current research focuses mostly on computer servers to lower their power consumption, while only few studies have tried to address data center networks (DCNs), which can account for 10% to 20% of the total energy consumption of a data center Greenberg et al. (2008); Heller et al. (2010). The percentage of network energy can even increase to 50% in future data centers, where servers become more energy-proportional to their workloads Abts et al. (2010). Similar to servers, the networks in data centers are commonly provisioned for the worst-case workloads that rarely occur. As a result, the capacity of a DCN is usually far from being fully utilized. Therefore, significant power* saving can be achieved by making DCNs more energy-proportional to their workloads as well.

It is well known that the power consumption of a network is mostly independent of its workload, mainly because devices, such as switch chips and fans, consume a significant

*Similar to related work Heller et al. (2010); Mahadevan et al. (2009b), we use power and energy interchangeably in this paper, because data center networks are typically required to be always-on. Also, we do not address heat density or electricity costs in this paper.

amount of power even at low loads Heller et al. (2010); Mahadevan et al. (2009b). There are some existing energy-efficient designs for traditional local and wide area networks. For example, some studies have proposed to put network devices, such as switches and routers, into sleep during periods of very low traffic activities Gupta et al. (2004); Nedevschi et al. (2008). While this approach works effectively for traditional networks, it is less applicable to today's typical multi-tiered DCNs, because their high degrees of oversubscription can lead to much shorter idle periods Abts et al. (2010). As a result, putting network devices into sleep in a DCN may cause packets to be buffered or rerouted around the deactivated switches and links. Buffering packets in a DCN for the purpose of energy savings may cause packets to be dropped or backpressure depending on the adopted flow control mechanism Abts et al. (2010). Route changing may lead to considerable overheads for coordination.

To achieve energy-proportional DCNs, two recent studies proposed energy-efficient approaches that are more amenable to DCNs. First, in their work called *ElasticTree*, Heller et al. Heller et al. (2010) studied real traces from a production data center and demonstrated that traffic flows in a DCN can be consolidated onto a small set of links and switches, which are sufficient to serve the bandwidth demands most of time. Second, Abts et al. Abts et al. (2010) proposed a link rate adaptation approach that dynamically estimates the future bandwidth needs of each link and then reconfigures its data rate to achieve power savings.

While traffic consolidation has been demonstrated to be a highly effective way to achieve energy proportionality in DCNs by shutting down unused network devices, existing work consolidates traffic flows in a greedy way and assumes that the bandwidth demand of each data flow can be approximated as a constant during the consolidate process. This is in contrast to the fact that the bandwidth demand of a traffic flow can vary over time. The variations can be significant because the consolidation period normally cannot be very short due to overhead consideration Verma et al. (2009). Therefore, existing work has to use either estimated maximum or average demands to perform consolidation, which can result in either unnecessarily high power consumption or undesired link capacity violations, respectively. A key observation based on the analysis of real DCN traces is

that the bandwidth demands of different flows usually do not peak at exactly the same time. As a result, if the correlations among flows are considered in consolidation, more power savings can be achieved. Another important observation is that the 90-percentile bandwidth demands are usually half or less of the maximum demands. Therefore, if we could avoid consolidating traffic flows that are positively correlated (*i.e.*, peak at the same time) based on 90-percentile demands instead of maximum demands, we may further improve the energy efficiency of traffic consolidation. A recent study Verma et al. (2009) has demonstrated the effectiveness of considering workload correlations in virtual machine consolidation for servers. However, to our best knowledge, no existing work has applied correlation analysis in traffic consolidation in DCNs.

In this paper, we propose CARPO, a correlation-aware power optimization algorithm that consolidates traffic flows based on correlation analysis among flows in a DCN. Another important feature of CARPO is to integrate traffic consolidation with link rate adaptation for maximized power savings. The integration is formulated as an optimal flow assignment problem, which is known to be NP-Complete. The optimal consolidation solution and data rate of each link in the DCN are computed using a linear programming tool. To reduce the computation complexity, we then propose a heuristic algorithm to find a consolidation and rate configuration solution with acceptable overheads. Furthermore, CARPO generalizes previous work to present an analytical framework that theoretically estimates how much power can be saved by ElasticTree or CARPO for a given DCN topology and workloads. As a result, a DCN administrator can use CAPRO to compare the power savings of different topologies and choose one that can provide the desired trade-offs between power and performance.

Specifically, the contributions of this paper are as follows:

- By analyzing the correlation of the traffic flows in real DCN traces from Yahoo! production data centers, we observe that the correlation analysis can be used in traffic consolidation to achieve more power savings.

- We integrate the two energy-saving approaches amenable to DCNs, traffic consolidation and link rate adaptation, for maximized power savings. The integration is formulated as an optimal flow assignment problem. We then propose a heuristic algorithm to find a consolidation and rate configuration solution with acceptable overheads.

- We generalize previous work and CARPO to present an analytical framework that theoretically estimates how much power can be saved by ElasticTree or CARPO for a given DCN topology and workloads.

- We implement CARPO on a hardware testbed composed of 10 virtual switches configured with a production 48-port OpenFlow switch and 8 servers. Our empirical results with Yahoo! DCP traces demonstrate that CARPO can save up to 60% of network power for a DCN, while having only negligible delay increases. CARPO also outperforms two state-of-the-art baselines by having approximately 10% and 20% more power savings, respectively.

- We also develop a simulation platform based on OPNET OPNET Technologies (2010) to evaluate CARPO. Our results with another trace file of 10 data centers composed of 5,415 servers also show the power efficiency of CARPO in large-scale data centers.

The rest of this paper is organized as follows. Section 3.3 reviews the related work. Section 3.4 formulates the integration of traffic consolidation and link rate adaptation. Section 3.5 analyzes the correlations of traffic flows in the Yahoo! DCN traces and presents the CARPO framework and the proposed correlation-aware algorithm. Section 3.6 introduces the implementation of our hardware testbed and the evaluation results. Section 3.7 presents the simulation results in large-scale data centers. Section 3.9 concludes the paper.

3.3 Related Work

There have been several studies on the power saving of network equipment. Gupta et al. Gupta and Singh (2003) began to explore the possibility of putting network devices into sleep, such as switch and router, when the traffic activities are low to save the network device power. They also proposed an abstract sleep model in Gupta et al. (2004), which is used to calculate the power savings by putting the network devices into sleep. Ananthanarayanan et al. Ananthanarayanan and Katz (2008) uses a time prediction window to predict the packet coming time such that the network device can be shut down when there is no packets coming. However, none of these studies put their proposed scheme into the data center application context, which have special traffic characteristics. The oversubscription of a multi-tiered DCN leads to short idle periods Abts et al. (2010) of the network, which results in a high transitional power consumption on the network device with periodic sleeping. In contrast to all the previous projects, we study the network power saving within the data center network, in which our switches do not wake up frequently with the change of network workloads.

DCN research has recently received a lot of attention. For example, novel DCN architectures have been proposed in Guo et al. (2008); Al-Fares et al. (2008); Guo et al. (2009), but those projects do not address power consumption. Two recent studies have proposed different approaches to achieve energy-proportional DCNs. The first approach is *traffic consolidation* proposed in *ElasticTree* Heller et al. (2010). The researchers of the *ElasticTree* project proposed to consolidate traffic flows in a DCN onto a small set of links and switches such that unused network equipment can be turned off for power saving. Although their consolidation approach is developed based on a real datacenter traces, they assume that the traffic rate of each data flow is approximately a constant. This is not entirely true in the current production datacenters. In contrast to the consolidation methods in the *ElasticTree*, we analyze the statistical characteristics of datacenter traffics first based on the real datacenter traces. The statistical analysis provides a guide to us for the traffic

consolidation, leading to a better energy saving solution. The second approach for energy-proportional DCN is to adapt the link rate according to the workload of each traffic flow Nedevschi et al. (2008); Abts et al. (2010); Gunaratne et al. (2008). One problem of solely using link rate adaptation is that the power saving gain is relatively small since a large amount of power is consumed by components such as fans and switch fabrics. In this paper, we propose to integrate link rate adaptation onto the traffic consolidation, such that more power saving gain can be achieved.

3.4 Problem Formulation

In this section, we first analyze the power characteristics of network switches. Based on the analysis, we formulate the integration of traffic consolidation and link rate adaptation as a constrained optimization problem.

3.4.1 Characteristics of Network Switch

A general purpose network switch is commonly composed of chassis, switching fabric, line-cards and ports. Switch chassis includes cooling equipment, such as fans, which consumes fixed amount of power. The switching table is maintained by the switching fabric while the line-card maintains buffers all the incoming and outgoing packets. Ports contain the networking circuitry consumes. According to the measurements from Mahadevan et al. (2009a), the idle power consumption of a 48-port edge LAN Switch ranges from 76W to 150W. Around 40 more Watts is to be added if the switch is working under maximum capacity. These measurements clearly indicate that the each switch port only consumes 1-2 Watts, while the switch chassis, fabric and line-cards consume most of the power. Therefore, compared with strategies that only fine tune the transmission rate of each port, more significant power savings can be achieved if unnecessary switches can be turned off.

On the other hand, Benson et al. studied the data center link utilization characteristics in Benson et al. (2009) for real production data center and found that the link utilization of

Table 3.1: Power Consumption of PRONTO 3240 Switch with Different Port Speed.

PRONTO 3240 Switch	Port Speed (Mbps)			
	None Active	10	100	1000
Power (W)	67.7	70.7	80.2	111.5

the aggregation layer links is only close to 8% during 95% of the time, while the average link utilizations of the edge layer links and the core layer links are approximately 20% and 40%, respectively. and the link utilization of links is approximately 40%. The low link utilization provides us a large space to consolidate traffic flows from different links, such that a smaller number of links and switches are required to provide services to all the existing traffic workloads.

To verify the power consumption characteristics on the switch, we measure the power consumption on a PRONTO 3240 OpenFlow enabled switch with 48 10/100/1000Mbps Ethernet ports. We tune the port speed to different values (inactive/10/100/1000Mbps) at different round of measurement and measure the total switch power consumption. The measurement results are listed in Table 3.1. We see that the switch itself with no active ports consumes more than half of the total power when all the ports are set to the maximum speed. Power consumption only varies about 10-20 Watts between each level of port speed. Note that the measured value is the power consumed when there is no workload on the switch. According to Mahadevan et al. (2009a), 100% utilization only increases the power consumption by about 5%.

3.4.2 Optimal Consolidation and Rate Configuration

We now formulate the flow consolidation problem as an optimal flow assignment problem. The final goal is to assign traffic flows into the network such that the traffic constraints are satisfied and optimal power is achieved. This problem is a NP-complete problem for integer flows. The overall idea of our design is to select the minimum subset of network devices, which can provide enough capacity for all traffic in the network.

We first define the following notations:

- P_{switch}: the power of one switch, including power consumed by both chassis and port;
- $P_{chassis}$: the power consumed by switch chassis;
- P_{port_d}: the dynamic power of each port, which is proportional to the link utilization;
- P_{port_f}: the dynamic power of ports in full link capacity;
- tr: the data rate on the link that connects to the port, where tr^{in} represents the data rate of traffic sent to the port, and tr^{out} denotes the data rate of traffic send from the port; and
- lc: the maximum link capacity.

Assume we have n switches, each switch has p ports, and the consolidation period is T, the problem can be formulated as follows:

$$\min \sum_{t=0}^{T} \sum_{i=0}^{N} P_{switch_{i_t}} \tag{3.1}$$

$$P_{switch} = P_{chassis} + P_{port_d} \tag{3.2}$$

where

$$P_{port_d} = P_{port_f} \times \frac{\sum(tr^{in} + tr^{out})}{lc} \tag{3.3}$$

under the following constraints:

- For each time point, the link capacity is not exceeded

$$tr_i \leq lc_i. \tag{3.4}$$

- For each switch, the number of packets that are received equals to the number of packets that are sent out

$$\sum_{i=0}^{p} tr_i^{in} = \sum_{i=0}^{p} tr_i^{out}. \tag{3.5}$$

- When a port or a switch is off, it consumes zero power.
- When all ports of a switch are off, the switch is off.
- All traffic flows are assigned to paths in the network.

To mathematically implement the formal model, we use MathProg mathematical programming language and solve it using GLPK (GNU Linear Programming Kit) mathematical programming software. The formal model is translated into a mixed-integer programming (MIP) model for mathematical programming purpose. The decision of usage for each link is defined as a binary variable. If a link is used, the corresponding binary variable is 1, otherwise it is 0. Status of each switch is also represented by a binary variable, where 1 indicates the switch is turned on while 0 means it is turned off. Constraints are described in details in previous formal model description. Optimization goal is to minimize total power consumption over the network of switches connected by links.

While the optimal solution can give us a theoretical upper bound for power savings, it cannot be used in practice because 1) it assumes perfect knowledge of future bandwidth demands of the traffic flows by reading the information from the DCN traces, and 2) its computation complexity is too high.

3.5 Design of CARPO

Due to the lack of perfect knowledge about the bandwidth demands of the traffic flows in a DCN in practice, we now propose CARPO to estimate the correlations among different flows in traffic consolidation. In this paper, we assume a centralized power manager, which allocates path for each traffic flow, adapts the link rate for each port, and turns off unused switches and ports. In very large DCNs, CARPO can be extended to work in a distributed way, which is our future work.

3.5.1 Traffic Correlation Analysis

To justify the design methodology of CARPO in later sections, we now present our analysis on traffic traces from real production data centers of Yahoo! Yahoo! (2008). The traffic traces in our analysis contain network flows between end users and Yahoo servers. The end users are located outside the DCN, while the Yahoo servers are located in one of the Yahoo data centers. There are 5 Yahoo data centers located in Dallas (DAX), Washington DC (DCP), Palo Alto (PAO), Hong Kong (HK) and United Kingdom (UKL). Each entry in the trace files contains the timestamp, source IP address, destination IP address, source port, destination port, protocol, number of packets and number of bytes transferred from the source to destination. The trace data is collected by three boarder routers and contains both the request (inbound) traffic flows and response (outbound) traffic flows.

In this paper, we focus the analysis on the response (outbound) traffic flows sent from data centers. The outbound traffic flows can be identified from the port number used at the source IP as presented in Yahoo! (2008). We assume that response packet sent by a tier-1 server is fetched from a corresponding database server in the DCN. In that case, the response traffic flows can be converted to traffic flows between end server and database server within DCN. With virtualization technology, each physical server usually accommodates several virtual machines. In the traffic trace in our analysis, there are more than 2 million server IPs in each data center. To make the size of data center reasonable, we group the traffic traces of IPs with same prefix together to represent the real traffic on a physical server. The prefix length we use to group IP addresses is 8 bits, which leads to around 100 servers in each data center. We further manually screen out the aggregated workload that is atypical for the data center network.

Figure 3.1 shows a traffic flows example of 4 grouped flow workload from the final 70 grouped IPs in the DCP data center within one day. We see that although traffic flows are constantly changing, they do not always peak together. Some of the workload show strong correlation such as workload with IP prefix 58 and 108, while others show negative

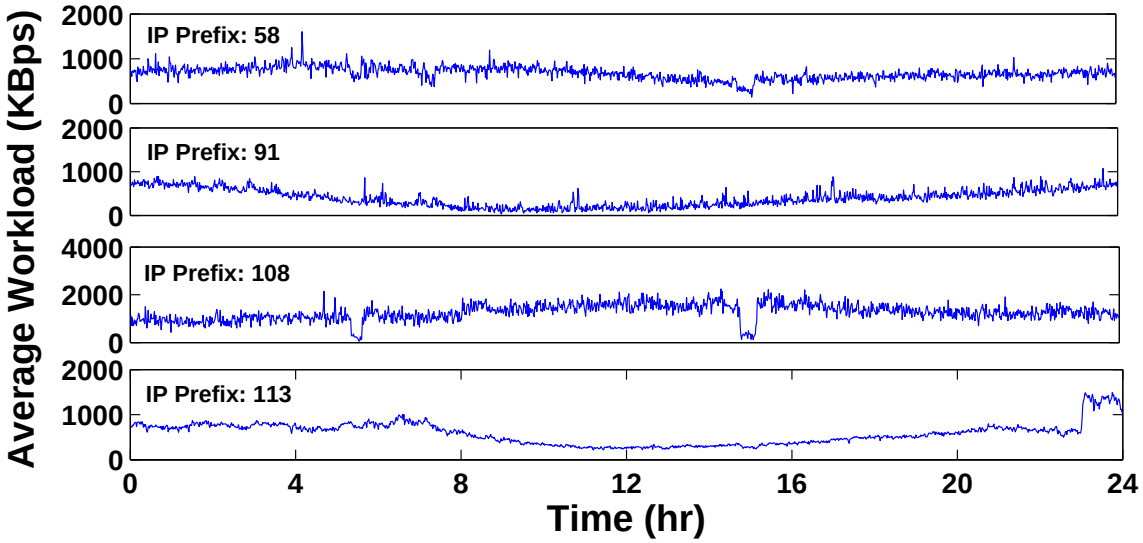


Figure 3.1: Average workload for 4 grouped IPs from Yahoo! data centers within 1 day.

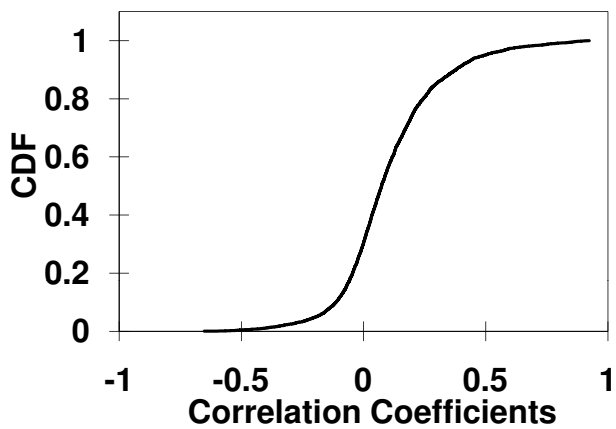


Figure 3.2: CDF distributions of pair-wise correlation coefficient in 5 Yahoo! data centers.

correlation, such as workload with IP prefix 113 and 108, where their traffic flows show opposite increasing and decreasing trend.

To quantify the correlation relationship of each flow pair, we calculate the Pearson Correlation Coefficient between each flow by

$$r_{xy} = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{\sqrt{n\sum x_i^2 - (\sum x_i)^2}\sqrt{n\sum y_i^2 - (\sum y_i)^2}} \tag{3.6}$$

where x and y are two flows. We sample each traffic flow every one second to calculate the correlation coefficient.

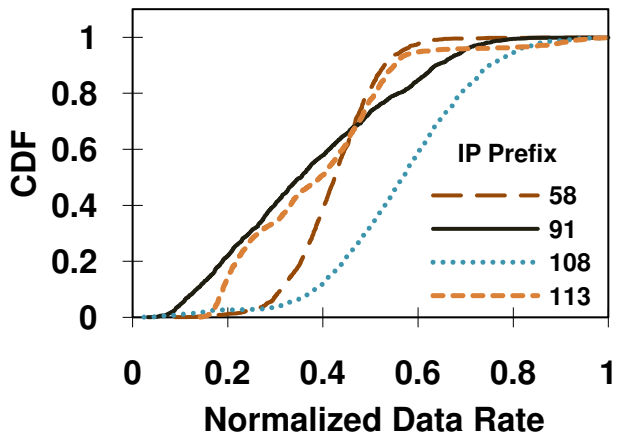


Figure 3.3: CDF of the normalized data rate for the flows in Figure 3.1. The data rate of each flow is normalized to its own maximum value.

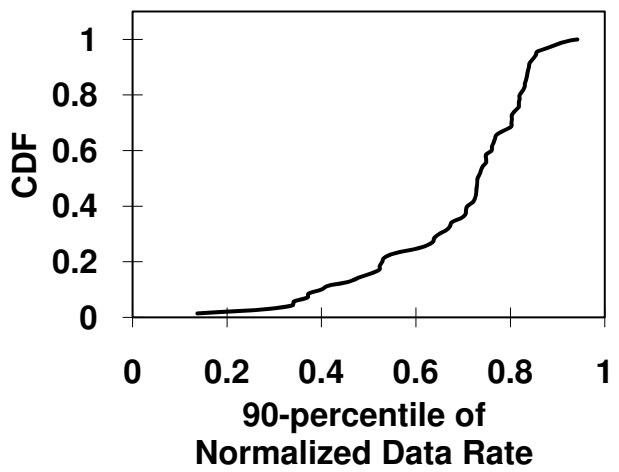


Figure 3.4: CDF of the 90-percentile of the normalized data rate with 70 flows.

The CDF distribution of all the pair-wise correlation coefficients in Yahoo! data center is shown in Figure 3.2. We see that more than 80% of the traffic flows are loosely correlated or even negatively correlated together with a correlation value of less than 0.3. This distribution leads to the follows:

Observation 1: *traffic flows within a data center are usually loosely correlated together, and so usually do not peak at the same time.*

Next we study the statistical properties of traffic flows in data centers. Figure 3.3 shows CDF distribution of the normalized data rate of each traffic flow from the 4 flows in Figure 3.1. The data rate is normalized to each flow's own maximum data rate. From Figure 3.3

we see that two out of the four flows has a 90-percentile normalized data rate less than 50% of their own maximum data rates, while 90-percentile value for the other two flows are also less than 80% of their own maximum. We also plot out the CDF of all the 70 flows' 90-percentile normalized data rates in Figure 3.4. We see that more than half of flows have a 90-percentile data rate less than half of their own peak value. Based on these statistical data, we conclude that

Observation 2: *the 90-percentile of the link utilization for most flows are less than 50% of their own maximum, which suggests that if traffic flow consolidation is based on the off-peak values for link utilization, more power savings can be achieved.*

Based on the previous two observations, we design CARPO, a correlation aware power optimization scheme for the DCN. In general, CARPO periodically calculates the correlations between flows and consolidates traffic flows based on the 90-percentile of each traffic flow's peak demand, such that only a small set of network is needed to hold all the traffic flows while the rest of the network can be shut down for saving power. A more detail design methodology is presented in later sections.

We further explore the stability of the correlation values and the statistical property of traffic flows to demonstrate that the correlation analysis based on history data can be applied to traffic consolidation in the next period. To study the stability of correlation values, we plot the standard deviation of correlation coefficients of 8 flow pairs in Figure 3.5 without loss of generosity. We see that the standard deviations of each flow pair are all less than 0.15, which indicates the correlation values are stable over time. Figure 3.6 shows the 90-percentile link utilization value of one example flow during the first 4 hours of the trace. We see that the 90-percentile value shows high variation, which is not ideal for predict the future link utilization. However, we calculate the moving average of 90-percentile value with a window of 60 minutes. We see that the moving average 90-percentile value can track the real 90-percentile value well and smooth the variation. Thus, for the workload with high varying 90-percentile workload, we use the moving average 90-percentile value to aggregate traffic flows. Note that both the correlation and the 90-percentile value are re-calculated every 10 minutes with one sample point per second.

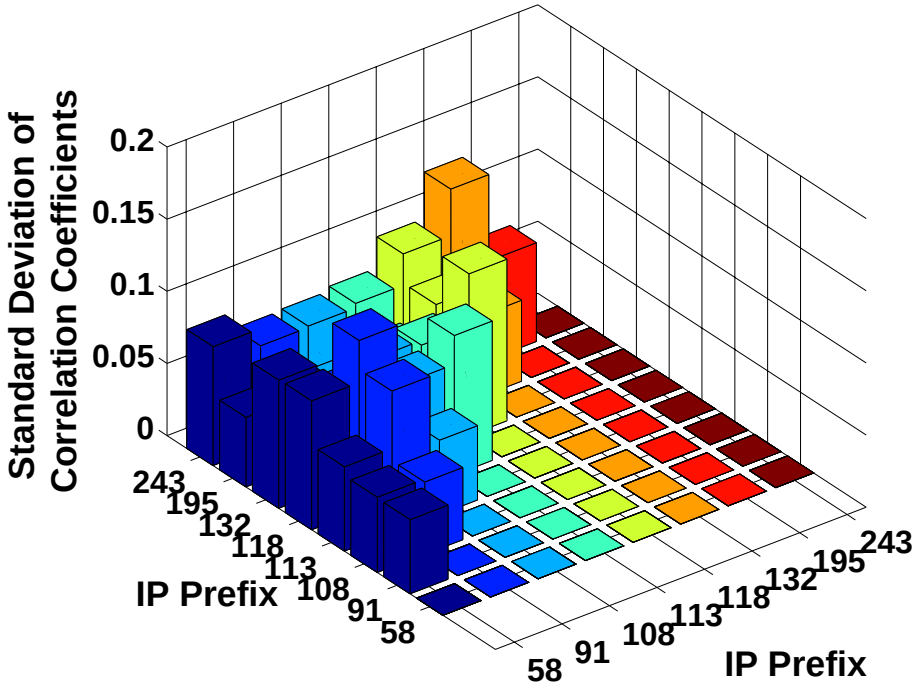


Figure 3.5: Standard deviation of correlation coefficients between 8 grouped IP flows. Half of the values are removed because of the symmetry.

3.5.2 CARPO Framework

The key novelty of CARPO is the adoption of correlation analysis for improved traffic consolidation. In general, as shown in Figure 3.7, CARPO takes three steps, *Correlation Analysis*, *Traffic Consolidation* and *Link Rate Adaptation* to perform the power optimization for DCN. In the first step, CARPO takes the historic network data as input and analyzes the correlation relationship between different traffic flows using the method introduced in Section 3.5.1. In the second step, CARPO consolidates the traffics under the link capacity constraint based on the correlation coefficients from the previous analysis. After the consolidation, unused switches and ports are turned off for power saving. In the last step, CARPO adapts the data rate of each active link to the demand of the consolidated traffic flows on that link, such that more power savings can be reached for the DCN.

Figure 3.8 presents a simple example to illustrate how CARPO works. In this example, there are four traffic flows: $f_1 : s1 \rightarrow s5$, $f_2 : s2 \rightarrow s6$, $f_3 : s3 \rightarrow s7$ and $f_4 : s4 \rightarrow s8$. The correlation value between each flow pair is listed in Table 3.2. The normalized

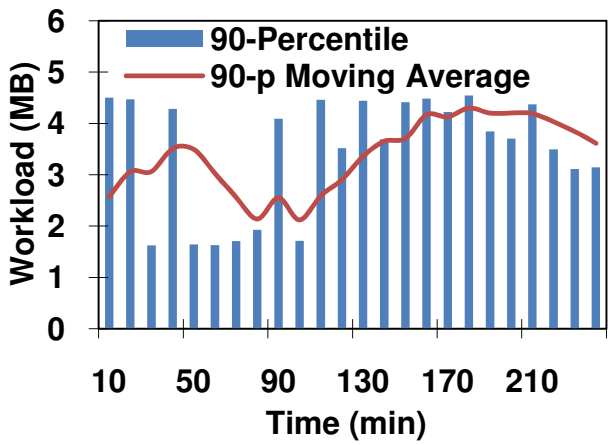


Figure 3.6: 90-percentile workload of every 10 minutes. Moving average value is calculated with a windows size of 60 minutes.

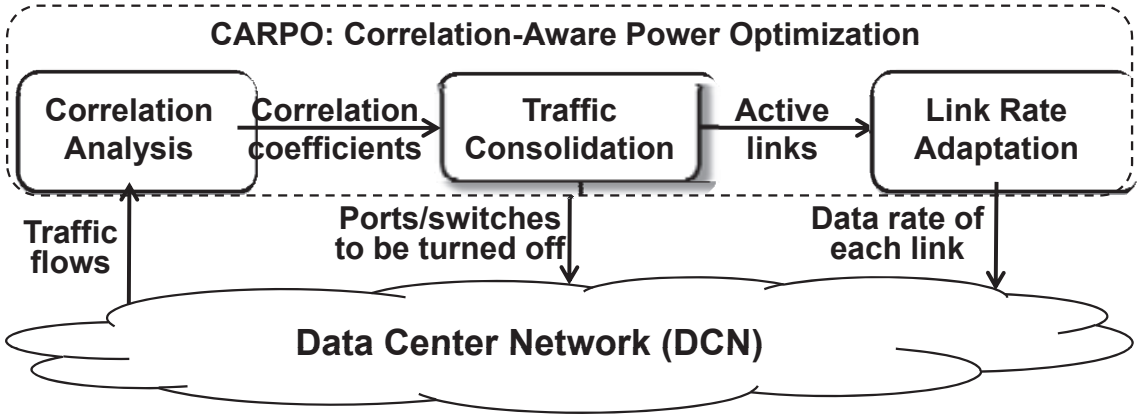


Figure 3.7: The proposed CARPO framework.

90-percentile capacity demand of each flow is labeled on the flow's source machine in Figure 3.8(a). Figure 3.8(a) shows an initial flow placement setting of the network. Figure 3.8(b) is the flow placement result after we apply CARPO on the initial flow setting. More specifically, flow f_1 and f_2 are aggregated together since they have low correlation (-1) and their aggregated 90-percentile data rate does not violate the link capacity. Flow f_3 cannot be aggregated with f_1 and f_2 because the aggregated 90-percentile data rate violates the capacity constraint of link between switch a_1 and c_1. Although the total bandwidth demand of flow f_1, f_2, and f_4 does not violate the capacity constraint, f_4 should not be aggregated to f_1 and f_2 neither because it has a high correlation (+1) with f_1, which means f_1 and f_4 is going to peak together, resulting in high probability of link capacity violation. Finally,

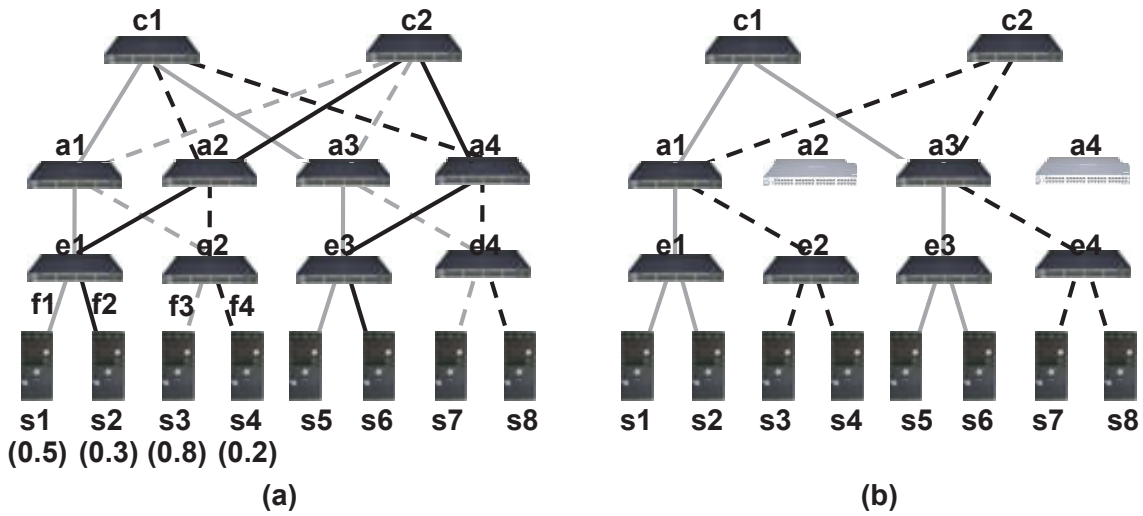


Figure 3.8: Flow placement example by CARPO. Sub-figure (a) shows the initial flow assignment. Sub-figure (b) shows the aggregated flow assignment after applying CARPO. s_i is server ID. f_i is flow ID. e_i, a_i, c_i are IDs of $edge$, $aggregation$ and $core$ switches, respectively. Different line color-pattern combinations are different flow assignments. The 90-percentile utilization values are labeled for the four flows.

Table 3.2: Correlation values for flow pairs in Figure 3.8.

Flow Pairs	f_1,f_2	f_1,f_3	f_1,f_4	f_2,f_3	f_2,f_4	f_3,f_4
Correlation	-1	-1	1	-1	-1	-1

since f_3 and f_4 have negative correlation and their aggregated 90-percentile data rate does not violate the capacity constraint, they should be aggregated together. After replacing and aggregating all the flows, switch $a1$ and $a4$ can be turned off to save power, since they do not serve any workload.

3.5.3 Correlation-aware Consolidation Algorithm

The most important component of CARPO is a correlation-aware heuristic algorithm that dynamically consolidates traffic flows. We assume that the link in our DCN is duplex with same capacity for upstream and downstream flow. The algorithm we designed is based on greedy-bin packing algorithm, where we greedily assign as most traffic flows as possible to a single path. The pseudo code of our algorithm is presented in Algorithm 1. The algorithm

Algorithm 1 Correlation-aware Traffic Consolidation

Require: Flow list $F = \cup\{f_i\}$ with n flows, correlation threshold Cor_{th}, link list l, link capacity c and link list $PATHL$ for each available path.

Ensure: Final path of each flow $PATHF$

1: $Cor[n][n] = CORRELATION(F)$;
2: $rate[n] = Ninety_Percentile(F)$;
3: **while** $F \neq NULL$ **do**
4: **for** j = 1 to m **do**
5: **for** $\forall f_i$ that can take path j **do**
6: **if** $c[k] - rate[i] \geq 0\ \forall l[k] \subset PATHL[j]$ **then**
7: **if** $Cor[i][i'] \leq Cor_{th}\ \forall f_{i'} \subset PATHF[j]$ **then**
8: $PATHF[j] = PATHF[j] \cup \{f_i\}$;
9: $F = F - \{f_i\}$;
10: $c[k] = UPDATE(PATHL[j], rate[i])$ $(\forall l[k] \subset PATHL[j])$;
11: **end if**
12: **end if**
13: **end for**
14: **end for**
15: **end whilereturn** $PATHF$

takes the flow list F, the link list l, link capacity c, path link list $PATHL$ for each available path, and the correlation threshold Cor_{th} as input. Note that $PATHL$ is a path list, where each entry is the link of each path. The path list is ordered in the order from left to right based on the network topology. In Algorithm 1, Line 1-2 initializes the correlation value (Cor) between all flow pairs, and the data rate ($rate$) of each flow. Line 3-15 assigns each flow to a path. More specifically, Line 6 calculates if the adding the current flow f_i violates the available link capacity of the chosen *path* j. Line 7 checks if the correlation between flow f_i and the flows existing on the chosen path j meets the correlation requirements. If both of these two requirements are satisfied, the flow is assigned to the chosen path and the available link capacity of each link along the path is updated (Line 8-10). Note that the available link capacity is updated by the true 90-percentile link utilization value of the aggregated traffic after the new flow is assigned in each step. Program terminates when all the flows are assigned.

The complexity of the algorithm is determined by the number of switches in the network and the number of flows that the network need to serve. Assume the network has V switches

and serves n traffic flows, the worst case number of paths that a flow can take is at the order of $O(V^2)$ when the core switches and the aggregation switches are fully connected. Therefore, the complexity of the algorithm is $O(nV^2)$;

3.5.4 Link Rate Adaptation

We have seen that correlation-aware traffic consolidation provides us an efficient way to save DCN power. The algorithm we designed is essentially based on the bin-packing algorithm. As a result, links may not be fully utilized after flows are consolidated. We further propose to adapt the data rate of each link to the consolidated traffic data rate, such that the waste of link power consumption can be reduced, leading to more overall power savings. The switches commonly used by DCNs is manufactured with a large range of tunable link rate, as studied in Abts et al. (2010). After each step of the consolidation, the link rate is adapted according to the combined flow data rate for each link. In our project, we assume the switch is a commodity switch with link rates available at 10Mbps, 100Mbps and 1000Mbps, as shown in Table 3.1 in Section 3.4.1.

3.6 Hardware Evaluation

In this section, we first introduce the hardware testbed used for our hardware evaluation. We then present the hardware evaluation results based on the Yahoo! trace data.

3.6.1 Hardware Testbed and Baselines

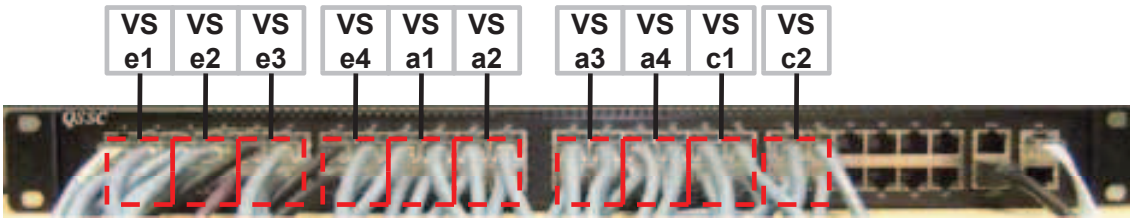


Figure 3.9: Hardware testbed with 10 virtual switches (VSs) configured from a production 48-port OpenFlow switch and 8 servers. The VSs are numbered in the same as in Figure 5.

We build our hardware testbed with 8 servers serving 4 flows, a Pronto 3240 OpenFlow enabled switch, and a desktop as the flow manager. The topology of the network in the testbed is the same as shown in Figure 3.8. Each server is equipped with two AMD Athlon(tm) 64 X2 Dual Core processors with 4G. To form the 10 switches in our testbed, we divide the Pronto switch into 10 virtual switches, each with 4 ports as shown in Figure 3.9. A desktop is connected to the Pronto switch via a RJ45 to DB9 console cable, serving as a centralized manager. The centralized manager calculates the new flow assignments every 10 minutes and sends the new paths configuration to the switch. The switch then updates the internal flow table and directs each flow accordingly.

To evaluate the performance of our $CARPO$ strategy, we compare our results with the results using the following baseline schemes:

- First baseline in our evaluation is *ElasticTree* Heller et al. (2010). *ElasticTree* assumes that all traffic flows are constant bit rate traffic in one flow assignment period. Similar to our algorithm, it consolidates the traffic flow with first-fit greedy bin-packing algorithm from the leftmost path in the network.

- *GoogleP* Abts et al. (2010) is the second baseline. *GoogleP* predicts the future bandwidth demand of each link, and then tunes each port rate to the lowest level that can meet the demand.

Besides the above two baselines, we also present the evaluation results of *CARPO-C*, the flow consolidation approach only considering the correlation without the link rate adaptation. All the evaluation results from different power saving schemes are compared with the performance of the *Optimal* solution we derived from the mixed integer programing approach presented in Section 3.4.

3.6.2 Experiment Results

The 4 traffic flows in the experiment are the ones in Figure 3.1. Each flow is served by one pair of servers, leading to 8 servers in total. Sources of the four flows are located at the left half of the topology as shown Figure 3.8, while the other half are served as destinations.

The performance metrics we evaluate in the section are the power consumption on the switch and the power saving that each power management scheme can achieve. According to the CARPO framework, the flow configuration in the DCN needs to be updated every 10 minutes and the unused switches and ports should be shut down. However, since our testbed is established on a single physical switch, divided into 10 virtual switches, virtual switches cannot be shut down separately. Therefore, to calculate the power consumption of the testbed, we measure the power consumption of the physical switch with no ports turned on, which is 66.7 W, leading to 6.67W for each virtual switch. We also measure the power consumption of a single active port at different speeds, which are 1W at 1 Gbps, 0.3 W at 100 Mbps and 0.15 W at 10 Mbps. By having these measurements, we can calculate the power consumption of $CARPO$ for any network configuration by

$$P = 6.67 \cdot N_s + \sum_i (P_i \cdot N_i) \tag{3.7}$$

where N_s is number of active virtual switches, P_i is the single active port power at speed i and N_i is the corresponding number of active port at that speed. To calculate the power consumption of *CARPO-C* and *ElasticTree*, we only need to change the second item in Equation 3.7 to $1 \cdot N_p$, where N_p number of active ports, assuming all ports can provide the maximum 1Gbps throughput. For *GoogleP*, the power consumption is $66.7 + 1 \cdot N_p$ since no virtual switch can be turned off at any time.

The power consumption of the entire workload is shown in Figure 3.10. We see that *GoogleP* consumes the most power since it only adapts the link rate to running workload and saves power from the port. The majority of the power is consumed by the switches, where *GoogleP* is not capable of saving power. *ElasticTree* has a higher power consumption, compared with *CARPO* and *CARPO-C*, since it only uses a sample from the last period as a constant bit rate for the next consolidation period. Different from *ElasticTree*, *CARPO-C* considers the correlation and 90-percentile value of the workload when consolidating traffic, leading to a lower power consumption. *CARPO* has the lowest

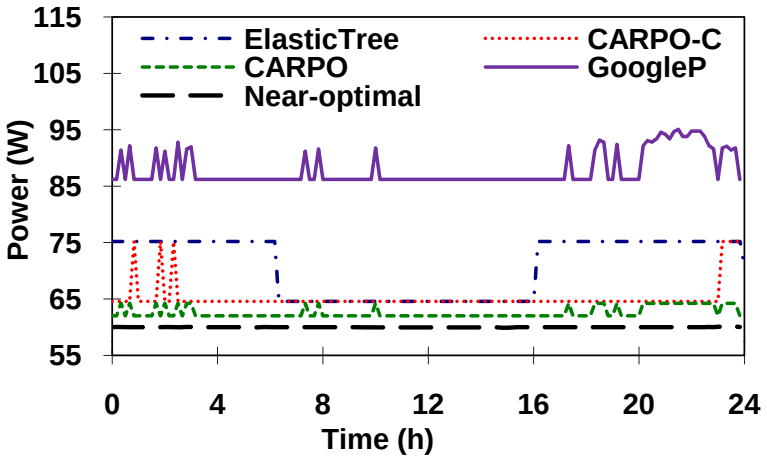


Figure 3.10: Power consumption of the testbed running the workload in Figure 3.1.

power consumption and the closest performance to the *Optimal* power consumption with the flow assignment solution from the mixed integer programming algorithm.

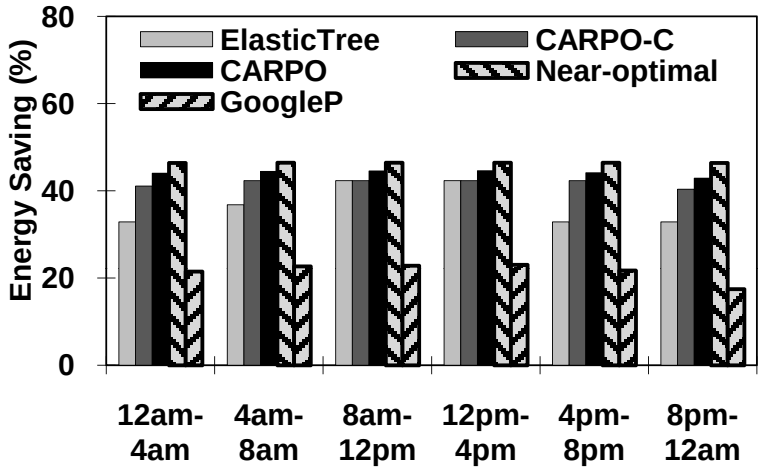


Figure 3.11: Power savings on the testbed running the workload in Figure 3.1.

Figure 3.11 shows the average power savings over time for different power management solution, compared with the power consumption with no power management for the network. We see that on average *CARPO-C* has a higher power saving than *ElasticTree* and *GoogleP* by 8.5% and 25.9%, respectively. Integrated with link rate adaptation, *CARPO* performs better than *CARPO-C* with only flow consolidation based on correlation. *CARPO* outperforms the two baslines by 12.4% and 30%, respectively.

3.7 Simulation Evaluation

In this section, we present the evaluation result from simulations with different network setup, including a network with the 70 grouped IP flows from Yahoo! DCP data center and a large-scale network with workload converted from a server utilization trace of 10 data centers with 5,415 servers. All the simulation are conducted with the OPNET OPNET Technologies (2010) network simulator.

3.7.1 Experimental Setup

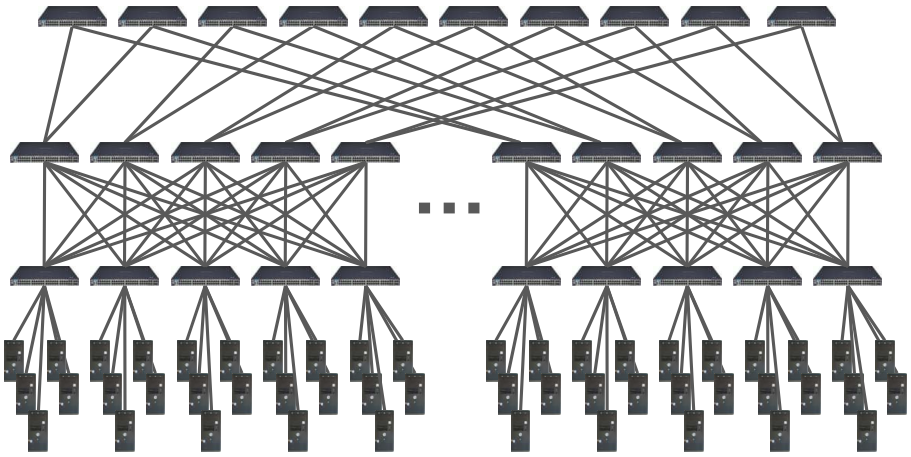


Figure 3.12: Example (*pod-10* fat tree) of network topology for simulations.

In all of our experiment, we the *fat tree* structure to construct the network. A *pod-k* fat tree structure is composed of k core switches, $\frac{k^2}{2}$ aggregation switches and $\frac{k^2}{2}$ edge switches. Figure 3.12 shows an example of a *pod-10* fat tree structure, which can hold 250 end hosts at most. If each flow needs to use a pair of dedicated servers, *pod-10* structure can hold 125 traffic flows.

For simulations running Yahoo! DCP data center workloads with 70 grouped IP flows, we use the *pod-10* fat tree structure to construct the network. For the large-scale simulation with 5,415 servers, we use *pod-28* fat tree network structure. The baselines used in simulations are the same as those in hardware evaluations.

3.7.2 Simulation Results for Yahoo! DCP Data Center Workload

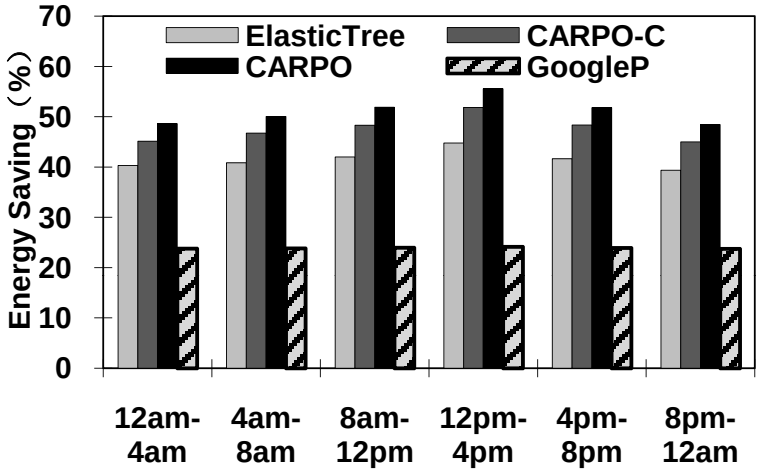


Figure 3.13: Average power savings in simulations running Yahoo! DCP data center workload.

Figure 3.13 shows the power savings from the simulations running the 70 grouped IP data flows from Yahoo! DCP Data Center. We see that *CARPO* has the highest power saving among all the power management scheme. More specifically, *CARPO* has 12.3% and 21.1% more power savings than *ElasticTree* and *GoogleP*, respectively. Compared with *CARPO-C* without link rate adaptation, *CARPO* shows 2.6% more power savings.

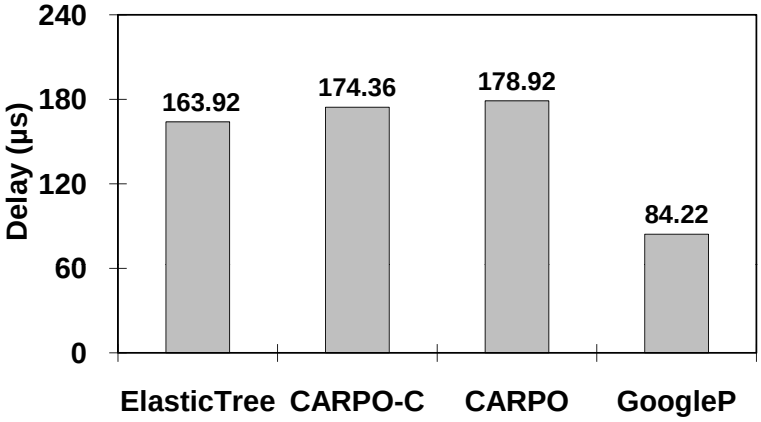


Figure 3.14: Average packet delay in simulations running Yahoo! DCP data center workload.

The average packet delay is presented in Figure 3.14. Compared with *ElasticTree*, power management by *CARPO* only incurs around 15μs more delay to each packet on average, which is acceptable considering the amount of more power savings. The little more delay of *CARPO* comes from the variations in correlation and 90-percentile workload value, which could lead to a little more violations to the link capacity.

3.7.3 Simulation results in Large-Scale DCNs

In this set of simulation experiment, we use a different network workload derived from a much larger data center utilization trace. The trace we use in this experiment comes from a data center server CPU utilization trace Wang et al. (2009), which consists of the CPU utilization 5,415 servers for 7 days. The granularity of the trace is 15 minutes per sample point.

To make use of this CPU utilization trace for our network simulation, we need to convert it to the network flow trace. We achieve this conversion by using a model for the relationship between CPU utilization and the data transfer speed on the server over the network. The model is established from real measurement on a small web service testbed with two servers. One server works as the client, which sends the request to the other server providing web service. The web service is an emulation from a real web service following the specification given in SPEC (2009). When the web server gets a request, it replies a message. The message size is picked from 1KB, 10KB, 100KB and 1MB, with the probability 35%, 50%, 14% and 1% as specified in SPEC (2009). We gradually increase the number of requests per second sent from the client and collect the CPU utilization of the web server with different replied file size. We found that the request number per second has an approximately linear correlation with the CPU utilization on our testbed. Therefore, we use a linear function to approximate this relationship as

$$R = 2124.76 \times CU - 116.25 \tag{3.8}$$

where R denotes request number, and CU represents CPU utilization. By Equation 3.8, we can compute the request number given any CPU utilization in the trace and then estimate the data rate of each flow between.

To accommodate the 5,415 servers, we use a fat tree structure with *pod-28*, which consists of 392 aggregation switches, 392 edge switches and 28 core switches. Each edge switch is connected to at most 14 servers. A destination server is randomly picked for a source server.

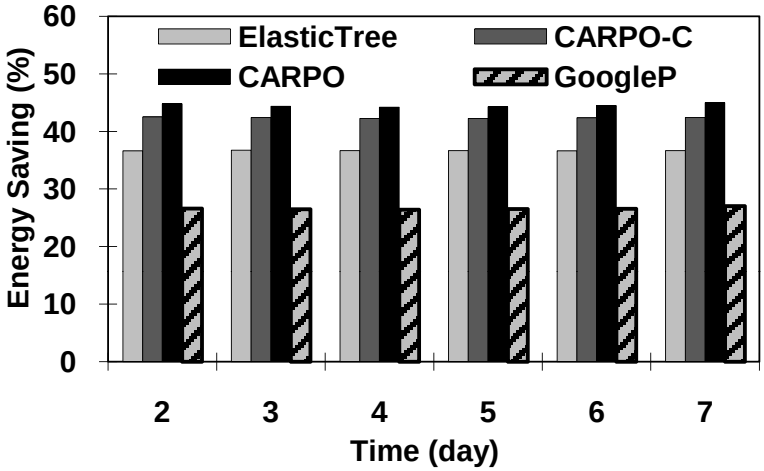


Figure 3.15: Average network power consumption in large scale DCN.

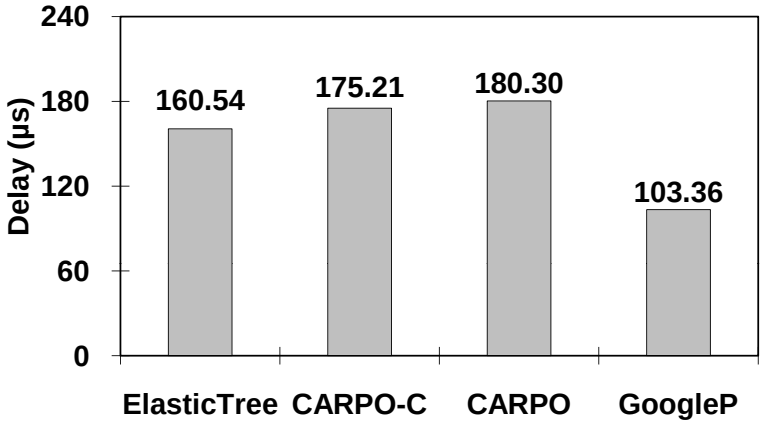


Figure 3.16: Average packet delay in large scale DCN.

Figure 3.15 shows the power savings that each power management scheme can achieve. *CARPO* shows the best power savings among all the four different strategies. More

specifically, *CARPO* outperforms *CARPOL*, *ElasticTree* and *GoogleP* by saving 3.6% 12.4% and 24.4% more power, respectively. Note that the power savings for all the four schemes are quite stable over different days. This is because the data center workload has small variation among different days. Figure 3.16 shows the average packet delay for each of the four schemes. We see that *CARPO* only has an increase of packet delay by less than 20μs compared with *ElasticTree* and by around 70μs compared with *GoogleP*, which is significantly smaller than the 1ms, the typical round-trip-time in data center Vasudevan et al. (2009). All the data in this experiment demonstrate that *CARPO* can effectively save more power on a large-scale network, with only very low increase in the packet delay.

3.7.4 Performance with Different Numbers of Sample Points in Correlation Calculation

In the design of CARPO, we calculate the correlation between each flow pair every time when we run the consolidation algorithm. This is a computational overhead that needs to be addressed. The time complexity of calculating all the correlation coefficients with n flows and m sample points for each flow is $O(mn^2)$. Since the number of flow is fixed for each data center network, the overhead can be decreased if using less sample points for the calculation.

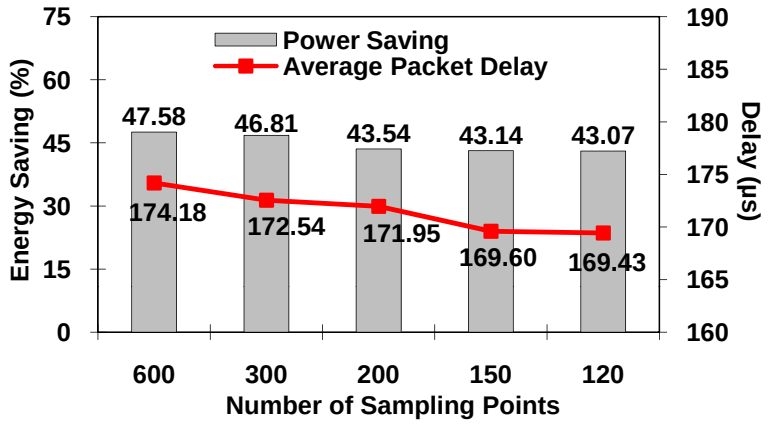


Figure 3.17: Performance impact of different number of sample points in correlation analysis.

Figure 3.17 shows the power saving performance of $CARPO$ with different numbers of sample points used to calculate the correlation coefficients in each step. We see that with more sample points used, more power savings can be reached. This is because more sample points gives the algorithm a better resolution of the workload, leading to a low correlation coefficient between two flows. With a smaller correlation, more flows can be aggregated onto a single path. If using a smaller number of sample points for the calculation, the variation of the workload is smoothed out, which leads to a larger correlation, thus less flows to be aggregated together. However, as shown in Figure 3.17, with more sample points, the average packet delay will increase. It is because the more flow aggregated onto a single path, the higher possibility of capacity violation to occur.

3.8 Discussion

As shown in the previous section, the percentage of power we can save from ElasticTree by using our algorithm fits into a parabolic curve. The curve for the different topologies are different; therefore, we propose a power saving estimation model for both ElasticTree and CARPO. This model tells how much power those two algorithms will consume when compared to the network without power saving algorithms implemented, by taking the traffic data rates, correlation coefficients, and topology type as inputs.

3.8.1 Model for ElasticTree

We first propose a model for ElasticTree. This model takes the sample traffic data rates for all traffic and network topology description as inputs, and prints out the value of the power of ElasticTree over the power of the network that implements no power saving algorithms.

Currently, this model only works for symmetric topologies, such as fat tree with pods and 2N topology. The model takes traffic data rates as inputs, and for each part of the topology, decides how many upper layer switches are needed by only taking the cross layer traffic workload into consideration. However, to decide how many upper layer switches are

necessary is found by simply computing the value of the sum of the cross layer traffic over the link capacity. However, this may lead to the split of traffic flows, which is bad for TCP traffics. To compensate, we use the following formula

$$P = P_c \times N_c + P_a \times N_a + P_e \times N_e \tag{3.9}$$

$$N_c = \max_j \{\max\{\frac{\sum T_{j,i}}{L}, \sum_j O_{j,i}\}\} \tag{3.10}$$

$$N_a = \sum_p N_p \tag{3.11}$$

$$N_p = \max_i \{\max\{\frac{\sum T_{i,s}}{L}, \sum_j O_{i,s}\}\} \tag{3.12}$$

$$\begin{aligned} C_{a,b} = (&1, tr_{a,b} > 0.5 \\ &0, tr_{a,b} < 0.5.) \end{aligned} \tag{3.13}$$

where P is the power of the network, P_c is the power of one core switch, P_a is the power of one aggregation switch, P_e is the power of one edge switch, N_c is the number of core switches, N_a is the number of aggregation switches, N_e is the number of edge switches, N_p is the number of aggregation switches in each symmetric subpart, $T_{j,i}$ is the data rate of traffics sent from node i to node j, and $C_{j,i}$ indicates how much traffic goes from node i to node j with a data rate larger than 0.5, and L is the link capacity.

3.8.2 Model for CARPO

We further extend the former model to take the correlation coefficients between the traffic as the third input and estimate how much power our algorithm needs to consume.

As our algorithm updates the aggregated traffic data rate according to the correlation between the traffic, we add a preprocess procedure on top of the model for ElasticTree, which curves the traffic data rates according to the correlation coefficients of all the traffic

that might be consolidated onto the same switch. Currently, we use the following formula

$$Tr_k = Tr_k \times \frac{1 + \frac{\sum cor_{l,m}}{Num_c}}{2} \tag{3.14}$$

where Dr_k is the data rate of kth traffic, $cor_{l,m}$ is the correlation coefficient between traffic l and m, and Num_c is the number of traffic that might be consolidated onto the same switch.

3.8.3 Power Estimation Model Evaluation

To evaluate how accurate this model is, we run the model on the 10 random traces introduced in former section on four different topologies, fat tree with 12 pods, fat tree with 16 pods, fat tree with 20 pods, and fat tree with 24 pods.

The estimated number of switches generated by our model for fat tree with 12 pods is shown in Figure 3.18.

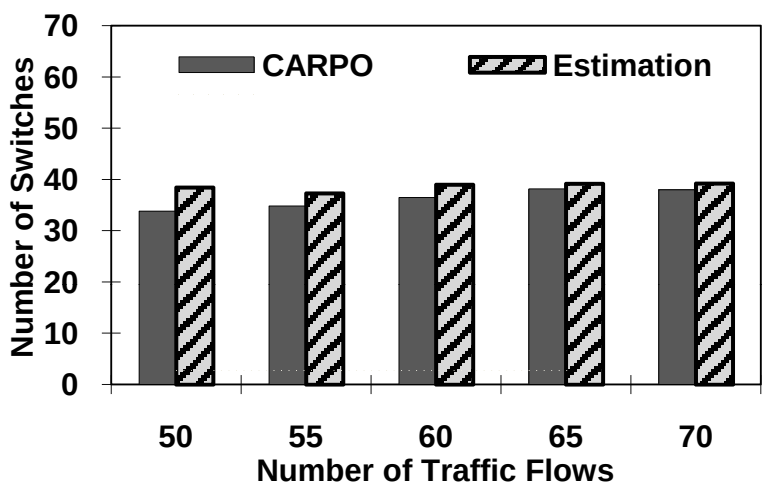


Figure 3.18: Compare Result with Estimation and Simulation.

We can see that, although the model cannot always find out the number of switches exactly as the simulation result, but the different is no more than 4%

3.9 Conclusion

In this paper, we have presented CARPO, a correlation-aware power optimization algorithm that dynamically consolidates traffic flows onto a small set of links and switches in a DCN and then shuts down unused network devices for power savings. In sharp contrast to existing work, CARPO is designed based on a key observation from the analysis of real Yahoo! DCN traces that the bandwidth demands of different flows do not peak at exactly the same time. As a result, if the correlations among flows are considered in consolidation, more power savings can be achieved. CARPO also integrates traffic consolidation with link rate adaptation for maximized power savings. Furthermore, CARPO generalizes previous work to present an analytical framework that theoretically estimates how much power can be saved for a given DCN topology and workloads. We implement CARPO on a hardware testbed composed of 10 virtual switches configured with a production 48-port OpenFlow switch and 8 servers. Our empirical results with Yahoo! DCN traces demonstrate that CARPO can save up to 60% of network power for a DCN, while having only negligible delay increases. CARPO also outperforms two state-of-the-art baselines, ElasticTree Heller et al. (2010) and GoogleP Abts et al. (2010), by having approximately 10% and 20% more power savings, respectively. Our simulation results with a trace file of 10 data centers composed of 5,415 servers also show the power efficiency of CARPO in large-scale data centers.

Chapter 4

Identifying Frequent Flows in Large Traffic Sets through Probabilistic Bloom Filters

4.1 Abstract

In many network applications, accurate traffic measurement is crucial for bandwidth management and detecting security threats such as DoS (Denial of Service) attacks. In such cases, traffic is usually modeled as a collection of flows, which are identified based on certain features such as IP address pairs. One central problem is to identify those "heavy hitter" flows, which account for a large percentage of total traffic, e.g., at least 0.1% of the link capacity. However, the challenge for this goal is that keeping an individual counter for each flow is too slow, costly, and non-scalable. In this chapter, we describe a novel data structure called the Probabilistic Bloom Filter (PBF), which extends the classical bloom filter into the probabilistic direction, so that it can effectively identify heavy hitters. We analyze the performance, tradeoffs, and capacity of this data structure, as well as developing two extensions to improve its accuracy and flexibility. We use real network traces collected on a web query server and a backbone router to test the performance of the PBF, and

demonstrate that this method can accurately keep track of all objects' frequencies, including those frequent websites and flows, so that heavy hitters can be identified with constant time computational complexity and low memory overhead.

4.2 Introduction

In managing today's complex Internet backbones, accurate traffic monitoring and measurement is crucial for many applications, including short-term purposes such as security needs (e.g., detecting traffic hot-spots, intrusions, and cyber-attacks) and long-term traffic engineering purposes (e.g., rerouting common traffic, expanding the capacity for frequently chosen links) Sarrar et al. (2012); Li et al. (2012); Estan and Varghese (2002b). One central problem in such applications is to identify heavy hitters, i.e., those most frequent flows, by keeping track of flow frequencies based on real-time traffic. Given that the number flows between commercial end host pairs can be extremely large Sarrar et al. (2012); Estan et al. (2006), however, keeping a counter for each flow usually requires more memory than available on limited hardware resources, such as routers. Existing methods of commercial solutions have addressed this problem through sampling and counting, such as NetFlow Cisco (2010), where one of N packets is sampled and counted. However, such methods can only sample a small portion of the entire traffic, leading to inaccurate results and over- or under-estimates.

Given such challenges, in this paper, we address the problem on how to efficiently construct estimates for the frequencies of all traffic flows so that heavy hitters can be identified. To this end, we aim to build an approximate histogram of all traffic flows with limited memory space, so that we can easily identify whether a flow is "heavy" when it is encountered in the ongoing traffic. The key assumption in our approach is that, for traffic management purposes, approximate knowledge on flows' frequency is already sufficient to identify heavy flows, as long as the frequency estimates can provide reliable upper and lower bounds associated with their most likely values. On the other hand, for some flows,

if we need to obtain their accurate frequencies, we can add a few extra counters to serve such needs separately.

Our work in this paper is enabled by extending a compact, hashing-based data structure called the *bloom filter*. A Bloom Filter (BF) is a data structure that is designed to answer a query on whether an element exists in a set. Its basic idea is to hash an element to k different locations in a bit array, and sets these locations to all 1s when inserting this element to the set. Being a randomized method, it allows for false positives, but the space savings often outweigh its drawbacks. BFs were originally introduced for database applications, but recently they have received great attention also in the networking area (see Broder and Mitzenmacher (2003); Tarkoma et al. (2012) as two surveys).

Based on the bloom filter, we investigate how to store *frequency estimations*, rather than set memberships, in a similar manner. Note that previous work has addressed the "accurate" version of this problem by proposing *counting bloom filters (CBF)* Fan et al. (2000); Broder and Mitzenmacher (2003); Tarkoma et al. (2012), where the bit vector is replaced by a counter vector. The cost is that the CBF design usually consumes memory space that is one order of magnitude higher than the original BF. In this paper, as we are concerned with the constrained memory of devices such as routers, our extension is still based on bit vectors rather than counter vectors. Specifically, we present a probabilistic version of the bloom filter and its operations, so that we can provide estimates of flow frequencies using a small amount of memory, based on which we can provide reliable identification of heavy hitter flows. Formally, we define the problem as follows: given a multi-set S, we would like to identify those items that appear for more than f times. Note that items may be provided in a stream, as is the case of traffic flows, where IP addresses in headers are used to denote flows.

The central idea of our design, by extending the classical bloom filter, is it performs probabilistic counting operations. Therefore, we call this new data structure as the *Probabilistic Bloom Filter* (PBF). The key difference is that whenever an item is inserted, instead of flipping the hash locations from 0 to 1, we flip them with a pre-set probability of p. Such a paradigm shift does not need any extra memory space compared to the

standard bloom filter. Therefore, it is still highly compact and feasible to implement on memory-constrained devices. We then model the performance of the PBF rigorously through probabilistic analysis, and we outline our major contributions as follows:

- We present the Probabilistic Bloom Filter, which allows non-deterministic queries on item existence and frequency in data sets. We provide the PBF's APIs and demonstrate how they can be used by applications.
- We quantitatively study the performance of the PBF through analytical approaches, where we derive closed form results regarding its capacity, parameter selection, and query performance.
- We extend the PBF into two variants: a counting PBF (C-PBF) and a time-decaying PBF (T-PBF), for additional application needs..
- We evaluate the performance of the PBF with realistic Internet traffic datasets collected from a backbone router for one hour of time to demonstrate its effectiveness.

We emphasize that our approach is fundamentally probabilistic and approximate. In fact, it may not be able to always return accurate estimations, and may, if poorly designed, fail to identify heavy hitters in three ways: it can miss some large flows, it can wrongly insert some small flows to the report, or it may give an inaccurate estimate of some large flows. We call these three types of errors: false negatives, false positives, and counting errors. We demonstrate that there exits intricate tradeoffs between performance and overhead: the higher the requirement of accuracy, the more memory overhead the design will incur in practice. Our work also quantifies such tradeoffs through theoretical analysis.

While our evaluations are based on network-related datasets such as web query logs and traffic traces, the methods in this paper should be general enough to be applied to many application domains. For example, counting the number of events based on their types in large-scale datasets is a widely used building block for data analysis in computational

sciences. The recent discovery of the Higgs boson using the LHC ATLAS collaboration (2012); CMS Collaboration (2012), for example, draws conclusions based on statistical analysis for collected particle collision events. Properly configured versions of PBFs can be applied for such purposes when TB scale of streaming data need to be analyzed in nearly real time under computational and memory constraints.

The remainder of this paper is organized as follows: We survey related work in Section 4.3. The problem formulation and design are described in Section 4.4. The extensions of the PBF are presented in Section 4.5, and the performance evaluation is given in Section 4.7. We provide conclusions in Section 4.8.

4.3 Related Work

In this section, we describe the related work in three parts: first the original Bloom Filter design, then its variants, and finally, recent progress on traffic flow sampling and counting in Internet routers.

The bloom filter, which is proposed by Burton H. Bloom in 1970 Bloom (1970), is a space efficient randomized data structure that answers the question on whether an element is in a set. There are two basic operations: insert and query. Its space efficiency is achieved at the cost of false positives (an element is claimed to be inside a set when it is not). The accuracy of a bloom filter depends on the filter size m, the number of hash functions k, and the number of inserted elements n. The false negatives (an element is reported as not in a set when it is) never happens. Although originally conceived for database applications, the bloom filter recently has also received great attention in the networking area Broder and Mitzenmacher (2003); Tarkoma et al. (2012); Chen et al. (2013); Rottenstreich and Keslassy (2012); Donnet et al. (2012); Moreira et al. (2012); Chen et al. (2012); Sarela et al. (2011); Debnath et al. (2011).

In its initial design, the BF did not address the issues of element duplicates, as it only considers simple sets. No matter how many times an item appears in a set, it is counted only once in the constructed BF. Counting Bloom Filters (CBFs) Fan et al. (2000);

Broder and Mitzenmacher (2003); Tarkoma et al. (2012) have been designed to address this issue. They are based on the same idea as BFs, but they adopted fixed size counters (also called bins) instead of single bits in its vector design. When an item is inserted, the corresponding counters are increased, hence, duplicate information is maintained rather than lost. However, CBFs are different from our approach in that they are fundamentally deterministic, as they keep accurate counts of the number of duplicates for an item. Therefore, CBFs require memory overhead that is usually an order of magnitude higher than common BFs, which makes them less scalable to a large number of flows. Another related work to ours, proposed by Shen et al. Shen and Zhang (2008); Dautrich and Ravishankar (2013), developed the idea of the Decaying Bloom Filter, which extended the Counting Bloom Filter to support the removal of stale elements when new elements are inserted. However, our design does not use as much memory as the decaying bloom filter for processing the same amount of data. Finally, Kumar et al. Kumar et al. (2006) proposed the Space-Code Bloom Filter (SCBF) (later extended to a multi-resolution version called the MRSCBF), which used a filter made up of a fixed number of groups of hash functions. During the insertion operation, one group of hash functions was randomly chosen for the element. For query, the number of groups containing the element was counted to estimate the frequency. However, as only open formulas were provided, the estimation of frequency was done by looking up a pre-computed table, which was very computationally intensive to be built. Such efforts differ in our work in that we present closed-form results on modeling the performance of the proposed data structures. In our evaluation, we compare with both the CBF and the MRSCBF as they are the state-of-the-art baselines.

In recent years, the bloom filter has been widely used in the context of network measurement Sarrar et al. (2012). Estan et al. Estan and Varghese (2002a) applied Counting Bloom Filters to traffic measurement problems inside routers. The approach was based on the simple idea that if the counter for a flow increases beyond a threshold, it should be considered as a frequent flow. Zhao et al. Zhao et al. (2006) used the Bloom Filter to find local icebergs (items whose frequency is larger than a given threshold) in a distributed manner, and then estimated global icebergs in a central server. Finally, Liu et al. Liu

et al. (2012) proposed the Reversible MultiLayer Hashed Counting Bloom Filter(RML-HCBF), whose hash functions select a set of consecutive bits from the original strings as hash values, so that it may find elephant flows (large and continuous flow) using the counter values and thresholds. In contrast, the PBF we propose is based on approximate counting methods rather than accurate ones, thus saving on the memory overhead and processing speed.

4.4 Probabilistic Bloom Filter

In this section, we describe the design of the probabilistic bloom filter (PBF). We first present its programming interfaces, followed by the operations between multiple PBFs, and finally, an analysis of its properties, capacity, and performance.

4.4.1 Programming Interfaces

Algorithm 2 The PBF Insert Algorithm

```
1: procedure INSERT(x)                                  ▷ Insert operation
2:     for j = 1 → k do
3:         i ← h_j(x)
4:         random_i ← Uniform(0, 1)
5:         if random_i < p then
6:             B_i ← 1
7:         end if
8:     end for
9: end procedure
```

The PBF provides two programming APIs, an insert operation and a frequency query operation, as illustrated in Algorithm 2 and Algorithm 3. For the insert operation, the primary change compared to a conventional bloom filter is that it uses a new parameter p to decide whether to flip a bit from 0 to 1, when new items are inserted. Note that as an optimization, we do not need to read the bit's value before we set it to 1, thereby reducing the number of memory accesses for the insert operation. For the frequency query algorithm,

Algorithm 3 The PBF Frequency Query Algorithm

```
1: procedure FREQUENCY(x)                    ▷ Frequency test operation counter ← 0
2:     for j = 1 → k do
3:         i ← h_j(x)
4:         if B_i == 1 then
5:             counter++
6:         end if
7:     end for
8:     f ← estimation(counter)
9:     return f
10: end procedure
```

it adds up the number of 1s in the k bits as determined by the hashing functions, and uses statistical inference methods to obtain an approximate frequency of the data item in the data set. We will describe the details of this estimation operation in Section 4.4.3.

4.4.2 Properties of the PBF Operations

There are several operations for multiple PBFs, including the union and halving of PBFs, which are facilitated by the bit-vector nature of PBFs. Given two multi-sets S_1 and S_2, suppose that they are represented by two PBFs, $B1$ and $B2$. We can calculate the PBF that represents the union set $S = S_1 \cup S_2$ by taking the OR of their PBFs: $B = B1 \vee B2$ assuming that the bit vector length m and the hash functions are identical. The merged filter B represents the aggregate frequency of an item belonging to S_1 or S_2 as belonging to the set S.

The second operation is halving. If the PBF size m is divisible by 2, halving allows us to store the original multiset in a shorter bit vector. This can be achieved by bitwise ORing the first and second halves of the original PBF's bit vector together. To insert or query the new PBF, the range of the hashing functions also needs to be updated by applying the $mod(m/2)$ operation to their outputs.

Table 4.1: Symbols Used in Analysis

f	The frequency threshold of heavy-hitter items
k	The number of hashing functions
m	The length of the bit-vector
n	The total number of flows or items in a dataset
p	The probability for setting a bit to 1
y	The expected number of 1s in the bit-vector
$\hat{y}$	The observed number of 1s in the bit-vector
θ	The probability that a bit has been set to 1

4.4.3 Performance Modeling of PBFs

Because the PBF introduces one additional parameter p, it has different properties compared to the original BF. In this section, we model the performance of the PBF by studying the relationship between the frequency of items and the number of bits that are flipped in the bit vector. Table 4.1 shows the notations we use in the following analysis.

We first consider what happens if there is just one item. We assume that this item has hashed positions that are distributed uniformly, as is true for the hashing functions we choose in our implementation. The probability that it flips a particular bit is p/m. Therefore, for a given bit, the probability that it is not set to 1 is given by

$$1 - \frac{p}{m}. \tag{4.1}$$

With the PBF, there are actually k hashing functions for inserting any item. Hence, the probability that none of them will set a specific bit to 1 is given by

$$(1 - \frac{p}{m})^k. \tag{4.2}$$

After inserting n items into the bloom filter, the probability that a given bit is still zero is going to be

$$P(n, 0) = (1 - \frac{p}{m})^{kn}. \tag{4.3}$$

Thus, the probability of a bit being set to 1 is

$$P(n,1) = 1 - (1 - \frac{p}{m})^{kn}. \tag{4.4}$$

The expected number of 1s of these k bits, denoted as $g(p,m,n,k)$, is therefore

$$g(p,m,n,k) = (1 - (1 - \frac{p}{m})^{kn}) * k \approx (1 - e^{-\frac{kpn}{m}}) * k. \tag{4.5}$$

Note that this approximation for the expected value is true only when p/m is sufficiently small. This constraint is generally true because our picked m is usually large, and p is usually much smaller than 1. Observe that $P(n,1)$ exists for every bit regardless of what items are inserted. Therefore, this value corresponds to a type of "background noise", which means that some bits will be set due to other items being inserted. Notice that when m increases, the background noise will decrease. If n increases, the noise increases. To obtain the frequency estimation of items, we have to take this noise into account.

Next, for a certain item that appears f times, it will invoke the *insert* API f times. Therefore, the probability for any bit of k bits mapped by hash functions to still be zero is

$$P(f,0) = (1-p)^f \approx e^{-pf}, \text{and} \tag{4.6}$$

the probability of this bit to be 1 is

$$P(f,1) = 1 - (1-p)^f \approx 1 - e^{-pf}. \tag{4.7}$$

Again, we assume that $p \ll 1$, which is true for our selection of p values. Clearly, whether a bit is set to 1 is determined by two factors: the probability that it is set to 1 by an item's insert operation as illustrated by $P(f,1)$, and the probability of the background noise as illustrated by $P(n-f,1)$ (in Eq. 4.4). These two factors are generally independent of each other. Therefore, the probability for an element to remain 0 as (since neither

background noise or repeated items has set it to 1) is given by

$$P(n-f,0) \times P(f,0) \approx e^{-\frac{pk(n-f)}{m}} \times e^{-pf}. \tag{4.8}$$

Next, as each bit can be considered as a Bernoulli experiment, its "success" probability θ can be considered as the event that a bit has been set as 1. Here we denote that

$$\theta = 1 - P(n-f,0) \times P(f,0) \tag{4.9}$$

Therefore, denote the total number of 1s as Y, we have that

$$Y|\theta \sim Bin(k,\theta). \tag{4.10}$$

Therefore, we know that $E(Y|\theta) = k\theta$ and $V(Y|\theta) = k\theta(1-\theta)$. If we denote the expected number of bits that are set to 1 in the k mapped bits for an element in the PBF as y, we therefore have

$$\begin{aligned} y = (1 - P(n-f,0) \times P(f,0)) \times k = \\ k(1 - e^{-\frac{pk(n-f)}{m}} \times e^{-pf}). \end{aligned} \tag{4.11}$$

Based on the observation results for the proportion of bits that have been set, we can denote its value as $\hat{y}$, and define $\hat{\theta} = \hat{y}/k$. We can use $\hat{\theta}$ as an unbiased estimator of θ, and since θ is derived based on the frequency f, we can then estimate f as follows (by solving the Equation 4.11 for f)

$$f = \frac{knp + m \ln\left[1 - \frac{\hat{y}}{k}\right]}{(k-m)p}. \tag{4.12}$$

Next, we calculate the confidence interval for f, by approximating it using a normal distribution based on the central limit theorem. This is also the so-called Wald Method, whose formula for confidence interval is given by

$$\hat{\theta} \pm z_{\frac{1}{2}\alpha} \sqrt{\frac{1}{n}\hat{\theta}\left(1-\hat{\theta}\right)} \tag{4.13}$$

where $\hat{\theta}$ is the proportion of successes, and $z_{\frac{1}{2}\alpha}$ is the critical z value with a tail area of $\frac{1}{2}\alpha$ of the standard normal curve. Based on this formula, we can derive the lower and upper bounds for f as shown below:

$$f_{min} = \frac{knp + m \ln \left[\frac{k-\hat{y}}{k} + \sqrt{\frac{\left(1-\frac{k-\hat{y}}{k}\right)(k-\hat{y})}{k^2}} z_{\frac{1}{2}\alpha} \right]}{(k-m)p} \tag{4.14}$$

$$f_{max} = \frac{knp + m \ln \left[\frac{k-\hat{y}}{k} - \sqrt{\frac{\left(1-\frac{k-\hat{y}}{k}\right)(k-\hat{y})}{k^2}} z_{\frac{1}{2}\alpha} \right]}{(k-m)p} \tag{4.15}$$

As an illustrative example, suppose we want to filter data traffic with $100K$ flows, where frequent flows are defined as those with a frequency to be at least one percent of the total, i.e., $1K$ flows. We pick a bloom filter size of $m = 2M$ bits ($1M = 10^6$), and let $k = 1000$. We select p as 0.0006 (we will explain this later in parameter selection). In this case, the frequent flow is expected to have 467 bits in 1000 bits set as 1. Conversely, if indeed 467 bits are set, the estimated number of flows is 999, with a 95% confidence interval as $[905, 1098]$. On the other hand, if somehow the value of $\mu - 2\sigma = 435$ bits are set, the estimated flows is 902, with a 95% confidence interval as $[813, 995]$*.

4.4.4 Selection of Parameters for PBFs

One critical challenge in using the PBF for analyzing data sets is that it needs to set several parameters properly, such as m, k, and p. Choosing such parameters improperly will reduce its capabilities and increase errors. Further, due to the probabilistic nature of the PBF, its parameters need to be set differently compared to conventional bloom filters. Therefore, in this section, we study how to set the parameters for the PBF, and define the concept of its capacity.

*Note that this interval does not contain 1000 as it is a 95% confidence interval. One may want to use the 99% confidence interval to capture a larger range. In this case, the 99% interval gives the bound as $[797, 1013]$.

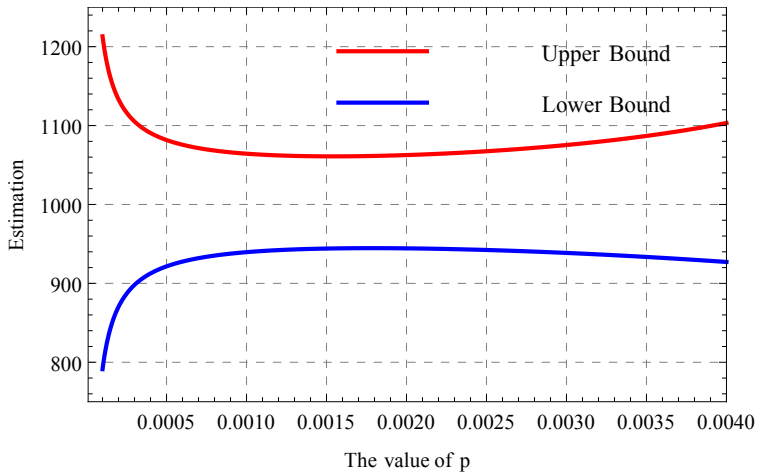


Figure 4.1: Relation between p and the estimation bounds.

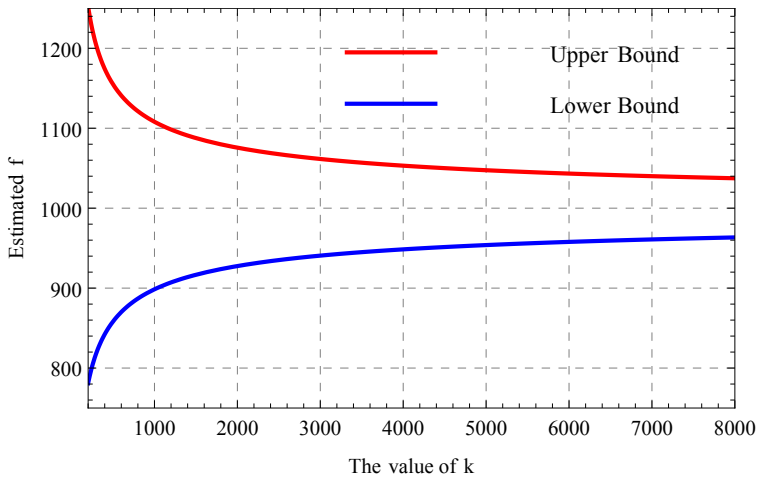


Figure 4.2: Relation between k and the estimation bounds.

Problem Formulation: Given a known number of items n and the threshold for frequent flows f, how do we choose m, k, and p properly? Similarly, given m, k, and p, how do we estimate the PBF's capacity to handle large n and f?

To answer this question, we first find the constraints for m, k, and p, and try to optimize the model performance by minimizing m and k, which correspond to the memory overhead (m) and computational overhead (k).

The first constraint for choosing the right parameters is to limit the background noise as shown in Equation 4.5. As m increases, the background noise will decrease, assuming a

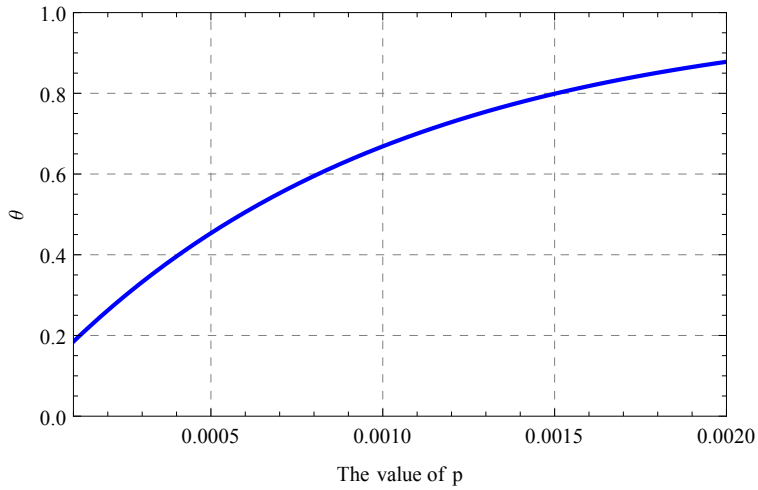


Figure 4.3: Relation between p and θ.

fixed k. Therefore, to keep the noise below a certain threshold ϵ, we require:

$$g(p, m, n, k)/k \leq \epsilon, \tag{4.16}$$

so that

$$m \geq \frac{-knp}{\log(1-\epsilon)}. \tag{4.17}$$

Note that we assume that ϵ is chosen as an appropriately low threshold, e.g., 0.1.

The second constraint concerns the estimation accuracy as shown in Equation 4.12. To ensure the estimation accuracy, the observed value of $\hat{y}$ should not be too small or too large compared to k. Since Equation 4.11 gives the expected value of y, we require that the ratio between y and k should lie between ϵ and $1-\epsilon$, therefore,

$$\epsilon \leq \frac{k(1 - e^{-\frac{pk(n-f)}{m}} \times e^{-pf})}{k} \leq 1 - \epsilon. \tag{4.18}$$

Based on this result, we can then obtain the estimation range of f, which is denoted by the lowest possible f value and the highest possible f value that can be accurately

estimated, as a function of k, m, and p as

$$f_{lowerbound} = \frac{knp + \log(1-\epsilon)m}{(k-m)p}, \tag{4.19}$$

$$f_{upperbound} = \frac{knp + \log(\epsilon)m}{(k-m)p}, \tag{4.20}$$

also, observe that the real value of f should be located within this estimation range, we have that

$$f_{lowerbound} \leq f \leq f_{upperbound}. \tag{4.21}$$

This formula therefore gives the third constraint. To illustrate the meanings of these constraints, especially on how they affect the choices of p, we consider the following way to illustrate possible choices of parameter values. To minimize m, we simply set m as the lower bound, using Equation 4.17, i.e., $m = \frac{-knp}{\log(1-\epsilon)}$. Then we can establish the following constraints for p as

$$-\frac{\log(1-\epsilon)}{n} \leq p \leq \frac{(n-f)\log(1-\epsilon) - n\log(\epsilon)}{nf}. \tag{4.22}$$

On the other hand, the value of k can be chosen based on two considerations. First, if k is too large, it will incur too much computational overhead. Second, the value of k will affect the confidence interval calculated in Equations 4.14 and 4.15. The reason is that different values of k will lead to different lower and upper bounds, and their difference will be varying. To illustrate this, assume that we have chosen p as the upper bound, and m as the lower bound, and we set $\epsilon = 0.1$, we can calculate the ratio between the confidence interval of f to the estimated f as follows

$$Ratio = \frac{f_{max} - f_{min}}{f}. \tag{4.23}$$

Assuming $f \leq \epsilon n$, meaning that the frequency under estimation is not too large compared to the total number of items, n, we can find that

$$Ratio \approx 0.46 \ln \left(0.1 + 0.59\sqrt{\frac{1}{k}}\right) - 0.46 \ln \left(0.1 - 0.59\sqrt{\frac{1}{k}}\right). \tag{4.24}$$

We can verify that under this setting, this ratio decreases monotonically with k increases. In particular, if we let $Ratio \leq 1/2$, we can find that $k \geq 141$. Therefore, we suggest k must be chosen not lower than 150 to ensure good performance. Note that this value of k only incurs moderate overhead in terms of computation. Recent research on fast string hashing algorithms Kaser and Lemire (2012) has demonstrated optimized hashing functions can achieve a hashing throughput of a fraction of a CPU cycle per byte. Therefore, we can use a k value of larger than 1000 on multi-core workstations without having performance bottlenecks, as hashing a 100 byte string (e.g., a packet header) 1000 times only takes less than 0.1ms.

So far, we have outlined the way we should select parameters. To summarize, the procedure is as follows:

Algorithm 4 The PBF Parameter Selection Algorithm

```
1: procedure SELECT(n,f)                                   ▷ n and f as known
2:     Calculate the bounds for p, and use its upper bound if possible (Equation 4.22)
3:     Select a modest value for k (assuming k ≥ 150)
4:     Calculate the lower bound for m (Equation 4.17)
5:     return p, k, and m
6: end procedure
```

Note that we generally require p to be chosen as its upper bound for the reason that a larger p will increase the accuracy of estimation, as long as this p value does not go beyond its upper bound. We now use a numerical example to illustrate possible choices for p, k, and m. Assume that $n = 100K$ and $f = 1000$. If $\epsilon = 0.1$, according to Equation 4.22, we have $1.05 \times 10^{-6} \leq p \leq 0.0022$. We now evaluate the effects of p. By keeping m bounded according to Equation 4.17 and $k = 2000$ (k can also take any other constant value), we change the value of p and study its effects on the confidence interval. The results are shown

in Figure 4.1. As illustrated, a larger p indeed leads to better confidence intervals, as long as the value of p does not go over its upper bound. In fact, the narrowest confidence interval in Figure 4.1 is precisely when $p = 0.0022$. After p goes over this upper bound (specified by Equation 4.22), the confidence interval becomes larger again. The underlying reason for this interesting phenomenon is that when p is too large, it will over-saturates the bit vector too early, hence impairing the performance of estimations.

Next we investigate the effects of m. Suppose that we have chosen a moderate value of p for general cases, where p is chosen as 0.0006. We hope to see how k can affect changes in confidence intervals, by changing k from 200 to 8000. As shown in Figure 4.2, a larger k will lead to better confidence intervals, at the cost of more computational overhead.

Finally, we illustrate the effects of p on θ, which is the percentage of the 1s in the k bits after all flows are inserted. Figure 4.3 shows the results. Observe that as long as p is chosen according to Equation 4.22, θ will be bounded by $[\epsilon, 1 - \epsilon]$, which is expected. On the other hand, if p is larger than the upper bound, θ will be over-saturated; if p is lower than the lower bound, θ will be under-saturated.

4.4.5 The Maximum Estimative Frequency

Once the PBF parameters are chosen, given properly chosen p, k, and m, one critical problem is its estimation errors. Given that our goal of this paper is to identify heavy-hitters, we are most concerned about the errors that are caused by over-saturation of bit vectors, i.e., when the frequency of the flow is too high. Motivated by this observation, we next define the concept of "Maximum Estimative Frequency (MEF)", which is defined as follows:

Maximum Estimative Frequency (MEF): Given a set of PBF parameters, what is the maximum value of f that it can still estimate accurately, where the k bit vector is at most saturated for $1 - \epsilon$?

To derive MPF, we use Equation 4.20, and replace m with the result from Equation 4.17. Therefore, the maximum value that f can reach is given by

$$f_{MEF} = \frac{n(\log(1-\epsilon) - \log(\epsilon))}{np + \log(1-\epsilon)}. \tag{4.25}$$

In reality, for specific PBF settings, the value of p is only a single value. Hence, if we set p as a constant, for a large n, we then have

$$\lim_{n \to +\infty} f_{mpf} = \frac{\log(1-\epsilon) - \log(\epsilon)}{p}. \tag{4.26}$$

Hence, this approximation can be used to estimate the real MEF that a PBF setting can handle. If $\epsilon = 0.1$, the capacity is roughly $2.2/p$. For example, if $p = 0.001$, this PBF can count up to cardinality of around $2,200$. For all items with a frequency above this threshold, the PBF can still report that they are *at least* $2,200$, but their real value is not reported due to bit vector saturations.

4.5 Extensions of the PBF

In this section, we describe two extensions of the PBF for potential applications: a counting PBF (C-PBF) and a time-decaying PBF (T-PBF).

4.5.1 C-PBF: Counting PBF Design

In this section, we introduce the C-PBF, a counting variant of the PBF that extends its capability with more memory usage. The idea of the C-PBF is simple: it replaces each bit in the PBF with a w-bit counter. Whenever an item is inserted, instead of deciding on whether flipping one bit from 0 to 1, it will determine with a probability of p whether to increase this w-bit counter. Formally, this updated algorithm is shown in Algorithm 5.

We can now derive the performance of the C-PBF. Observe that for each counter, it may be updated by two sources: the background noise caused by other items and the

Algorithm 5 C-PBF Insert Algorithm

```
1: procedure INSERT(x)                                  ▷ Insert operation
2:     for j = 1 → k do
3:         i ← h_j(x)
4:         Counter_i ← B[i]
5:         random_i ← Uniform(0, 1)
6:         if random_i < p then
7:             Counter_i = Counter_i + 1
8:         end if
9:     end for
10: end procedure
```

insertion operations of the frequent flows. Therefore, if we denote these two sources with two random variables X_1 and X_2, we have

$$X_1 \sim Bin(k(n-f), p/m), \tag{4.27}$$

$$X_2 \sim Bin(f, p), \tag{4.28}$$

where $X = X_1 + X_2$. In this scenario, we can use the Poisson distribution to approximate these two distributions, so that the sum of the distributions will remain Poisson. In that case, the size of counter is bounded as $\max_{i=1}^{m} \log_2(X_i)$. Based on this result, considering that we observe a total of k counters (assuming k hashing functions), we can denote their values as x_i, where $i \in [1, k]$. We can then use the average of these observations on x_i to estimate the MLE of f as

$$\hat{f} = \frac{knp - m\hat{x}}{(k-m)p}, \text{ where } \hat{x} = \sum\nolimits_{i=1}^{n} x_i/n. \tag{4.29}$$

4.5.2 Selection of Parameters for C-PBF

Although C-PBF uses counters for predicting the element frequencies, it is still critical to choose proper ranges for m, k, and p to increase prediction capabilities and reduce errors. The problem we need to solve is formulated as follows: given a known number of items

and the threshold for frequent flows f, how do we choose m, k, and p properly? The same as what we discussed in PBF, to answer this question, we also need to find the constraints for m, k, and p to optimize the system performance by minimizing m and k to achieve minimal memory overhead (m) and computational overhead (k).

Similar to PBF, limiting the amount of background noise is the first constraint for choosing the right parameters. For a fixed k, the background noise will decrease, while m increases. In that case, to keep the background noise below a certain threshold ϵ', we need:

$$\frac{Bin(kn, p/m)}{2^s} < \epsilon' \tag{4.30}$$

so that

$$m > \frac{knp}{2^s \epsilon'} \tag{4.31}$$

On the other hand, as the memory is limited in networking devices, the size of the C-PBF is bounded by the memory size ms and the counter size s. For a fixed counter size s, the size of m should be bounded as:

$$\frac{knp}{2^s \epsilon'} < m \leq \frac{ms}{s} \tag{4.32}$$

Secondly, we concern the prediction accuracy. To ensure the prediction accuracy, we observe that the counter value should not under saturate or over saturate the counter. Based on the expected counter value, we require that its ratio with maximum counter value should lie between ϵ' and $1 - \epsilon'$, such that

$$\epsilon' \leq \frac{kn\frac{p}{m} + fp}{2^s} \leq 1 - \epsilon' \tag{4.33}$$

Assume that ϵ' is chosen as an appropriately low threshold, e.g., 0.1. With the second constraint, we can obtain the prediction range of f as a function of k, m, and p as:

$$f_{lowerbound} = \frac{knp - m2^s \epsilon'}{(k - m)p} \tag{4.34}$$

$$f_{upperbound} = \frac{knp - m2^s(1-\epsilon')}{(k-m)p} \tag{4.35}$$

and we assume that the real value of f should be located within this prediction range, such that

$$f_{lowerbound} \leq f \leq f_{upperbound} \tag{4.36}$$

In that case, when we minimize m and set up $m > \frac{knp}{2^s\epsilon'}$, we can establish the constraints for p as:

$$\frac{2^s\epsilon'}{n} < p \leq \frac{f2^s\epsilon' + n2^s - 2n2^s\epsilon'}{fn} \tag{4.37}$$

On the other hand, the value of k needs to be carefully selected. If k is too large, it would incur too much computation overhead, and also the background noise would be large, as the probability of different elements mapping to the same filter bucket would be large. However, if k is too small, the deviation of counter value would be large. Now, assume that $p = \frac{f2^s\epsilon'+n2^s-2n2^s\epsilon'}{fn}$, $m = \frac{knp}{2^s\epsilon'}$, and $\epsilon' = 0.1$, then the ratio between the confidence interval of f to the predicted f is

$$Ratio = \frac{f_{max} - f_{min}}{f} \tag{4.38}$$

Assume $f \leq n\epsilon'$, the counter value is in Poisson Distribution with λ, where the $E(X) = Var(X) = \lambda$, then

$$X_{max} = (\lambda+1)(1 - \frac{1}{9(\lambda+1)} + \frac{1.96}{3\sqrt{\lambda+1}})^3 \tag{4.39}$$

$$X_{min} = \lambda(1 - \frac{1}{9\lambda} - \frac{1.96}{3\sqrt{\lambda}})^3 \tag{4.40}$$

then

$$f_{max} = \frac{2^s\epsilon' * n - X_{max}n}{2^s\epsilon' - n\frac{f2^s\epsilon'+n2^s-2n2^s\epsilon'}{fn}} \tag{4.41}$$

$$f_{min} = \frac{2^s\epsilon' * n - X_{min}n}{2^s\epsilon' - n\frac{f2^s\epsilon'+n2^s-2n2^s\epsilon'}{fn}} \tag{4.42}$$

In that case

$$Ratio = \frac{X_{min} - X_{max}}{2^s \epsilon' - X} \tag{4.43}$$

As

$$\lambda = fp + \frac{k(n-f)p}{m} = 2^s(1-\epsilon') \tag{4.44}$$

Then

$$\begin{aligned} Ratio = -1.25(&\frac{0.9(1 - \frac{0.123457}{2^s} - \frac{0.688674}{2^{\frac{s}{2}}})^3 2^s}{2^s} - \\ &\frac{(1 + 0.9 * 2^s)(1 + \frac{0.653333}{\sqrt{1+0.9*2^s}} - \frac{0.123457}{1.11111+2^s})^3}{2^s}) \end{aligned} \tag{4.45}$$

Observe from the above formulas, the $Ratio$ is not related to k values. In that case, we choose $k = \frac{m}{n} \ln 2$, which is the same as what has been done in the classical BF.

4.5.3 T-PBF: Time-decaying PBF Design

Algorithm 6 T-PBF Decaying Algorithm

```
procedure DECAY(x)                                  ▷ Decay operation
    for Every T seconds do
        for j = 1 → k do
            i ← h_j(x)
            if then B[i] == 1
                random_i ← Uniform(0, 1)
                if random_i < q then
                    B[i] ← 0
                end if
            end if
        end for
    end for
end procedure
```

In this section, we introduce the second variant, the T-PBF, a time-decaying variant of the PBF that allows it to *forget* those frequent items that appeared in the distant past. The idea of the T-PBF is to introduce a new operation, decaying, which flips bits from 1 to 0

with a probability q over each time epoch T. This can be considered as an approximation for the *delete* operation. Formally, this updated algorithm is shown in Algorithm 6.

We now demonstrate the long term behavior of the T-PBF. For simplicity, we consider operations in epochs, and the decaying operation only occurs at the end of each epoch.

Observe that for each bit, for each epoch, it may either start with bit 1 or 0. If it starts with 1, it may be flipped to 0 at the end of the epoch with a probability of q. However, if it starts with 0, it may be flipped to 1 first with a probability shown in Equation 4.11, and then flipped back to 0 with a probability of q. Therefore, we can use a discrete time Markov chain to describe these operations. In particular, because the transitions exhibit different probabilities at the beginning and the end of each epoch, we can model it with a time-inhomogeneous chain with a seasonal variation. In this case, we have that the seasonal period $d = 2$, and the transition probability as the following (assuming $n \geq 0$, while $2n$ and $2n+1$ stand for the index of epochs)

$$P(2n) = \begin{array}{c c} & \begin{array}{cc} 0 & 1 \end{array} \\ \begin{array}{c} 0 \\ 1 \end{array} & \begin{bmatrix} 1-\alpha & \alpha \\ 0 & 1 \end{bmatrix} \end{array}, \quad P(2n+1) = \begin{array}{c c} & \begin{array}{cc} 0 & 1 \end{array} \\ \begin{array}{c} 0 \\ 1 \end{array} & \begin{bmatrix} 1 & 0 \\ \beta & 1-\beta \end{bmatrix} \end{array},$$

where we have:

$$\alpha = 1 - e^{-\frac{pk(n-f)}{m}} \times e^{-pf}, \text{and} \tag{4.46}$$

$$\beta = q. \tag{4.47}$$

To analyze this seasonal chain, we can add a supplementary variable to create a homogeneous chain. The new chain contains four states: $A(0,a), B(0,b), C(1,a), D(1,b)$. The new transition matrix is shown below, followed by the corresponding Markov chain illustration.

$$P(n) = \begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} 0 & 1-\alpha & \alpha & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \beta & 1-\beta & 0 \end{bmatrix} \end{array}$$

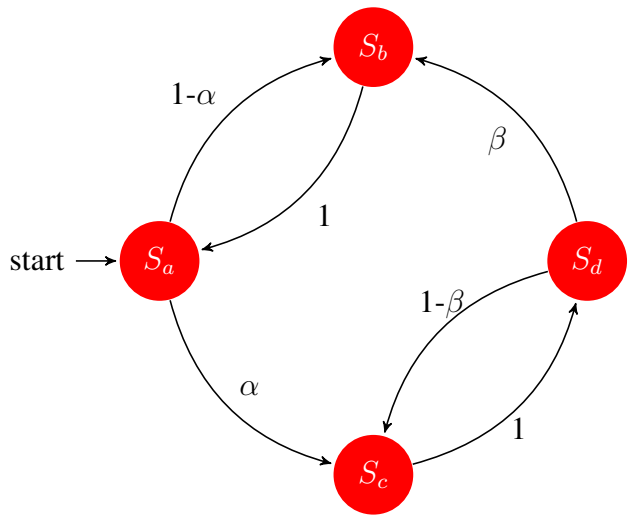


Figure 4.4: The Markov Transition for the T-PBF

This Markov chain has four communicating classes, A, B, C, and D. They are all recurrent classes, as well. In the long term, this chain has the following stationary distribution:

$$P(s) = \begin{cases} \frac{\beta}{2\alpha+2\beta} & \text{for } s \equiv A, \\ \frac{\beta}{2\alpha+2\beta} & \text{for } s \equiv B, \\ \frac{\alpha}{2\alpha+2\beta} & \text{for } s \equiv C, \\ \frac{\alpha}{2\alpha+2\beta} & \text{for } s \equiv D. \end{cases}$$

As expected, the results show that in the long term, if a flow is no longer available, f will decrease to 0, and its α will quickly converge to $1-e^{-\frac{pkn}{m}}$, which is smaller. Therefore,

in the long run, most bits will be reset to 0 (as demonstrated by the increased probability of states A and B in Figure 4.4. This validates the design of the time-decaying nature of the T-PBF. On the other hand, if a flow re-appears with a large f, its transition will mostly stay in states C and D, which means that more bits will be set due to the flow existence. Note that the epoch of the T-PBF can also be dynamic: as the PBF will lose the ability of prediction when all k bits are set to 1s, when a large percentage of k bits for any element are set to 1, the decaying process can be triggered.

4.5.4 Selection of Parameters for T-PBF

To performance accurate frequency estimations with T-PBF, it is critical to choose proper values for parameters, such as m, k, p, and q. As T-PBF extends PBF by introducing one additional parameter q, and the decaying operations are only triggered at the end of each period, we can use the same algorithms to choose the values of m, k, and p, based on what discussed in Section 4.4.4. As a result of that, in the following part, we will discuss how to choose the value for q.

For an element, at the end of the first epoch, the probability of a bit is still 1 is

$$\theta_1^{'} = \theta_1 * (1 - q) \tag{4.48}$$

where $theta$ is as defined in Equation 4.9.

Then at the end of the second epoch, the probability of a bit is 1 would be a joint effect of the first two epoch, therefore

$$\theta_2^{'} = (\theta_2 + \theta_1^{'} - \theta_2 * \theta_1^{'}) * (1 - q) \tag{4.49}$$

Finally, at the end of the epth epoch, the probability of a bit is 1 would be

$$\theta_{ep}^{'} = (\theta_{ep} + \sum_{i=1}^{ep-1} \theta_i^{'} - \theta_{ep} \prod_{i=1}^{ep-1} \theta_i^{'}) * (1 - q) \tag{4.50}$$

as in long term, $\theta_{ep}\prod_{i=1}^{ep-1}\theta_i^{'} \rightarrow 0$, we can have the estimative probability as

$$\theta_{ep}^{'} \approx \sum_{i=1}^{ep}\theta_i(1-q)^{ep-i} \tag{4.51}$$

To guarantee the prediction accuracy, we need to prevent the over saturation of the k bits for any element, therefore,

$$\forall \sum_{i=1}^{ep} y_i(1-q)^{ep-i} \leq k(1-\epsilon) \tag{4.52}$$

where ep is the number of pasted epochs, ϵ is a threshold with a small value, and y_i is the number of bits set to 1s with the insertion of an element in the ith epoch. As a result of above requirements, we can rewrite this equation as

$$\max \sum_{i=1}^{ep} y_i^{'}(1-q)^{ep-i} \leq k(1-\epsilon) \tag{4.53}$$

where $y_i^{'}$ is the number of 1 getting from the most frequent element. By setting $y_i^{'} = y_{max} = \max_{i=1}^{ep}\max_{j=1}^{t} y_{ij}$, which is the maximum number of 1s set by the element with the largest frequency in all epochs, where t is the number of elements in each epoch, then

$$\sum_{i=1}^{ep} y_{max}(1-q)^{ep-i} \leq k(1-\epsilon) \tag{4.54}$$

hence

$$y_{max}\frac{1-(1-q)^{ep}}{q} \leq k(1-\epsilon) \tag{4.55}$$

then

$$(1-e^{-\frac{pk(n-f_{max})}{m}}e^{-pf_{max}})\frac{1-(1-q)^{ep}}{q} \leq 1-\epsilon \tag{4.56}$$

in that case, for a long run, where $(1-q)^{ep} \rightarrow 0$, then

$$\frac{1-e^{-\frac{pk(n-f_{max})}{m}}e^{-pf_{max}}}{1-\epsilon} \leq q \leq 1 \tag{4.57}$$

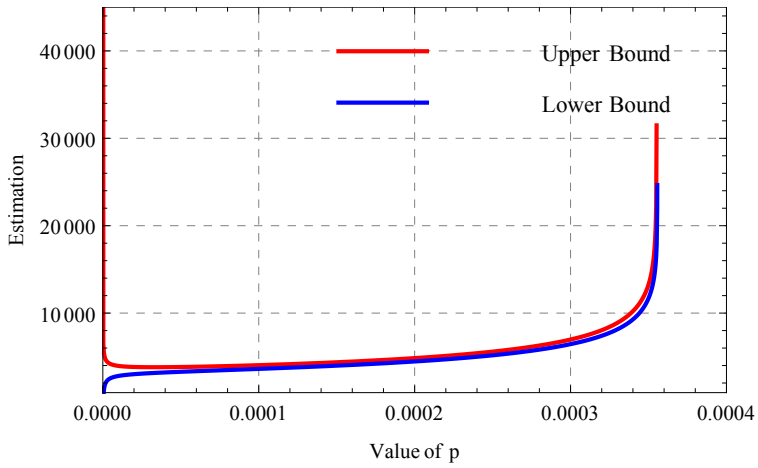


Figure 4.5: Relation between p and estimation bounds.

To validate the chosen range of p, k, m, and q, we will show the effect of fixing any three parameters and changing the rest one. Assume that $n = 1M$, $ep = 30$, and a certain element happens $f = 1k$ in each period. Based on the upper mentioned formulas, we get $4.5 * 10^{-8} \leq p \leq 3.39 * 10^{-4}$, $k \geq 150$, $m \geq 1.2 * 10^{7}$, and $0.285 \leq q \leq 1$. We now evaluate the effects of p, by keeping $m = 1.2 * 10^{7}$, $k = 4000$, and $q = 0.3$, at the end of the 30th period. As shown in Figure 4.5, if p is too small, the confidence range is unbounded. On the other hand, if p is too large, the T-PBF will be saturated and loose its estimation ability before the end of the 30th period. Secondly, we evaluate the effect of k, by fixing $m = 1.2 * 10^{7}$, $p = 3.2 * 10{-}4$, and $q = 0.3$. As shown in Figure 4.6, the larger the k, the smaller the confidence interval. Thirdly, the effect of m is evaluated by fixing $k = 4000$, $p = 3.2 * 10{-}4$, and $q = 0.3$. As shown in Figure 4.7, a m smaller than the lower bound would lead to a bad estimation accuracy. Finally, we evaluate the effect of q, by fixing $m = 1.2 * 10^{7}$, $k = 4000$, and $p = 3.2 * 10{-}4$. As shown in Figure 4.8, if q is smaller than its lower bound, the T-PBF would be saturated and loose its estimation ability before the end of the 30th period. On the other hand, if q is very large, the insertions of the element in the past periods take little effect on the estimated frequency of the current period.

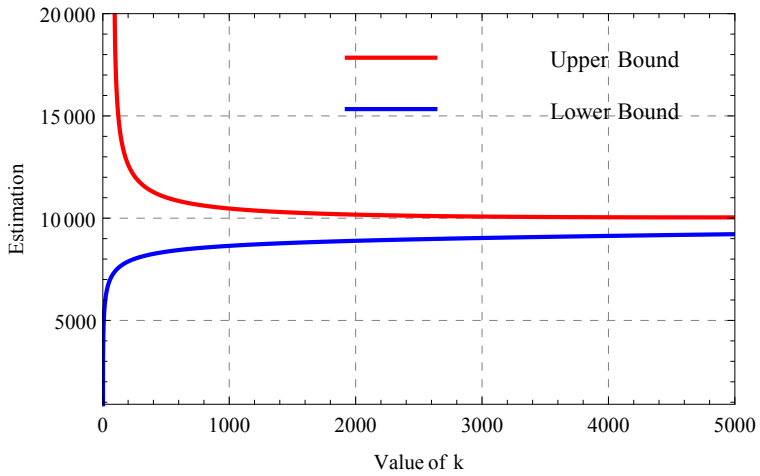


Figure 4.6: Relation between k and estimation bounds.

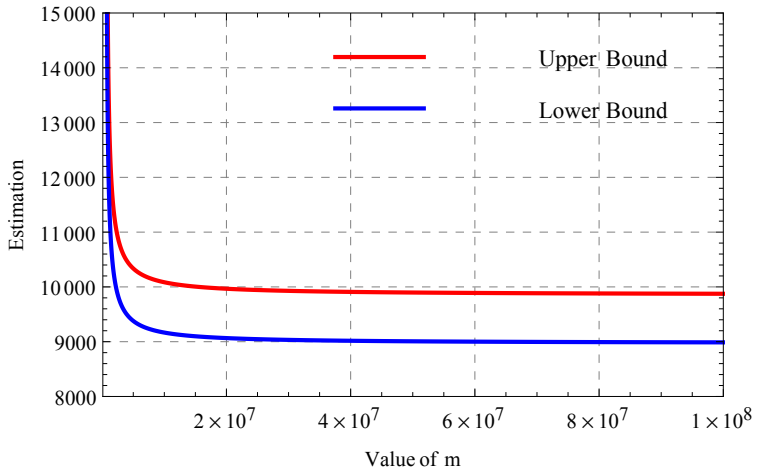


Figure 4.7: Relation between m and estimation bounds.

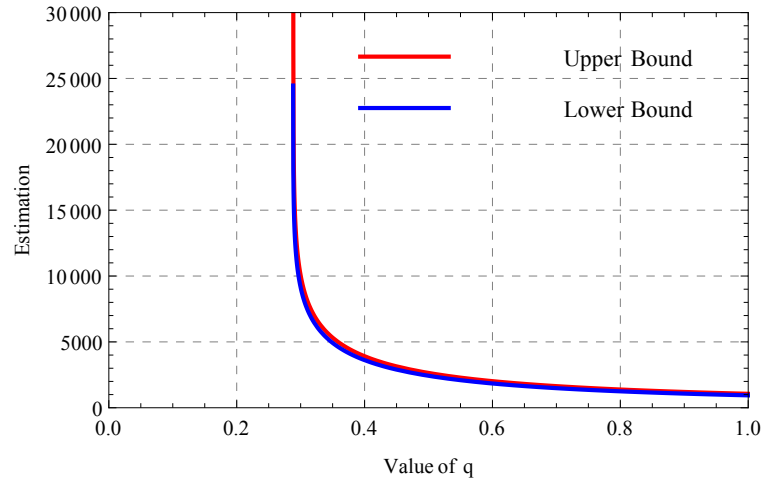


Figure 4.8: Relation between q and estimation bounds.

4.6 Parameter Selection Algorithms are Nash Equilibriums

As discussed in the previous sections, the estimation accuracy of the PBF and T-PBF would be bad, if we choose any other parameters following the predefined rules, and choose the rest one parameter disobey the rules. In that case, if we treat the rules of choosing parameters as the equilibrium strategies in a non-cooperative game, then we can claim that:

Theorem 4.1. *The parameter selection algorithms for PBF is a Nash Equilibriums Robert (1992).*

Proof. For selecting parameters of PBF, we can define a normal form game as a triple $(N, (S_i)_{i \in N}, (c_i)_{i \in N})$, where

- N is the set of players, and $n = 3$;
- S_i is the set of strategies of player i. Each player can choose to follow or not follow a specific rule of choosing a parameter, where $S_1 = \{FollowP, NFollowP\}$, $S_2 = \{FollowK, NFollowK\}$, and $S_3 = \{FollowM, NFollowM\}$;
- $S = S_1 \times S_2 \times S_3$ is the set of states;

Table 4.2: Cost for player 1.

	(FK,FM)	(NFK,FM)	(FK,NM)	(NK,NM)
FP	C(p,k,m)	C(p,k',m)	C(p,k,m')	C(p,k',m')
NFP	C(p',k,m)	C(p',k',m)	C(p',k,m')	C(p',k',m')

Table 4.3: Cost for player 2.

	(FP,FM)	(NFP,FM)	(FP,NFM)	(NFP,NFM)
FK	C(p,k,m)	C(p',k,m)	C(p,k,m')	C(p',k,m')
NFK	C(p,k',m)	C(p',k',m)	C(p,k',m')	C(p',k',m')

- a state is $s = (s_1, ..., s_n) \in S$;
- $c_i : S \rightarrow \Re$ is the cost function of player $i \in N$. In state s player i has a cost of $c_i(s)$. The cost is defined as the difference between the upper and lower estimation bounds, which represents the estimation accuracy.

In general, the cost value is computed as

$$C(p, k, m) = f_{max} - f_{min} = \frac{m(\ln \epsilon - \ln(1 - \epsilon))}{(k - m)p} \tag{4.58}$$

Based on the Nash Theorem Robert (1992), which claimed that every finite normal form game has a mixed Nash equilibrium, the PBF parameter selection game has a mixed Nash equilibrium. In that case, in the following part, we will show that the state as shown in the PBF parameter selection algorithm is the Nash equilibrium for the game.

As described in the game, each player has two strategies as follow or not follow the rule of choosing a specific parameter. For player 1, the cost for the game is as shown in Table 4.2. where FP is the abbreviation of $FollowP$, and NFP represents $NFollowP$. Other abbreviations follow the same format. Following the same method, the cost for player 2 and player 3 are as shown in Table 4.3 and Table 4.4.

Now, we will find the best response strategy of the all three players. As mentioned in the previous section, $-\frac{\log(1-\epsilon)}{n} \leq p \leq \frac{(n-f)\log(1-\epsilon)-n\log(\epsilon)}{nf}$, $k \geq 150$, and $m \geq \frac{-knp}{\log(1-\epsilon)}$, then $p' < -\frac{\log(1-\epsilon)}{n}$ or $p' > \frac{(n-f)\log(1-\epsilon)-n\log(\epsilon)}{nf}$, $k < 150$, and $m < \frac{-knp}{\log(1-\epsilon)}$. Let us start

Table 4.4: Cost for player 3.

	(FP,FK)	(NFP,FK)	(FP,NFK)	(NFP,NFK)
FM	C(p,k,m)	C(p',k,m)	C(p,k',m)	C(p',k',m)
NFM	C(p,k,m')	C(p',k,m')	C(p,k',m')	C(p',k',m')

from player 1, whose costs for choosing different strategies are as follows:

$$C_1(FP) = C(p,k,m) + C(p,k',m) + C(p,k,m') \tag{4.59}$$

$$+C(p,k',m') \tag{4.60}$$

$$C_1(NFP) = C(p',k,m) + C(p',k',m) + C(p',k,m') \tag{4.61}$$

$$+C(p',k',m') \tag{4.62}$$

When the k bits mapped by an element is saturated, we define the difference between upper and lower bounds of the estimated frequency as ∞, as the PBF cannot do the estimation computation anymore. In that case, if we choose $p' > \frac{(n-f)\log(1-\epsilon)-n\log(\epsilon)}{nf}$, then the $C_1(NFP) = \infty$, where the $C_1(NFP) > C_1(FP)$. On the other hand, when $p' < -\frac{\log(1-\epsilon)}{n}$, it is obviously that $C_1(NFP) > C_1(FP)$, as the cost is monotonically increasing with the decreasing of p. Hence, the best response strategy for player 1 is $FollowP$.

Following the same procedure, the costs for player 2 under different strategies are:

$$C_1(FK) = C(p,k,m) + C(p',k,m) + C(p,k,m') \tag{4.63}$$

$$+C(p',k,m') \tag{4.64}$$

$$C_1(NFK) = C(p,k',m) + C(p',k',m) + C(p,k',m') \tag{4.65}$$

$$+C(p',k',m') \tag{4.66}$$

As the cost is monotically increasing with the decreasing of k value, and $k > k'$, then $C_1(FK) < C_1(NFK)$. In that case, the best response strategy for player 2 is $FollowK$.

Now let us have a look at player 3, whose costs under different strategies are as follows:

$$C_1(FM) = C(p, k, m) + C(p', k, m) + C(p, k', m) \tag{4.67}$$

$$+C(p', k', m) \tag{4.68}$$

$$C_1(NFM) = C(p, k, m') + C(p', k, m') + C(p, k', m') \tag{4.69}$$

$$+C(p', k', m') \tag{4.70}$$

As the cost function can be rewritten as $C(p, k, m) = \frac{\ln \epsilon - \ln(1-\epsilon)}{\frac{kp}{m} - p}$, whose value increases with the decrease of m, on the other hand $m > m'$, then $C_1(NFM) > C_1(FM)$. In that case, the best response strategy is $FollowM$ for player 3.

In that case, $\{FollowP, FollowK, FollowM\}$ is a Nash equilibrium for the game. In another word, the parameter selection algorithm of PBF is a Nash equilibrium. □

On another hand, we can also claim that

Theorem 4.2. *The parameter selection algorithms for T-PBF is a Nash Equilibriums Robert (1992).*

Proof. Floowing the same procedure of the upper proof, we first set up a normal form game for the parameter selection game of T-PBF as a triple $(N, (S_i)_{i \in N}, (c_i)_{i \in N})$, where

- N is the set of players, and $n = 4$;
- S_i is the set of strategies of player i. Each player can choose to follow or not follow a specific rule of choosing a parameter, where $S_1 = \{FollowP, NFollowP\}$, $S_2 = \{FollowK, NFollowK\}$, $S_3 = \{FollowM, NFollowM\}$, and $S_4 = \{FollowQ, NFollowQ\}$;
- $S = S_1 \times S_2 \times S_3$ is the set of states;

- a state is $s = (s_1, ..., s_n) \in S$;
- $c_i : S \to \Re$ is the cost function of player $i \in N$, which is defined as the difference between the upper and lower estimation bounds.

The cost can be computed as:

$$C(p, k, m, q) = f_{max} - f_{min} = \tag{4.71}$$

$$\frac{m \ln \left[\frac{k-\hat{y}}{k} - 1.96\sqrt{\frac{\left(1-\frac{k-\hat{y}}{k}\right)(k-\hat{y})}{k^2}} \right]}{(k-m)p} - \tag{4.72}$$

$$\frac{m \ln \left[\frac{k-\hat{y}}{k} + 1.96\sqrt{\frac{\left(1-\frac{k-\hat{y}}{k}\right)(k-\hat{y})}{k^2}} \right]}{(k-m)p} \tag{4.73}$$

where

$$\hat{y} = (1 - e^{-\frac{pk(n-f_{max})}{m}} e^{-pf_{max}}) \frac{1-(1-q)^{ep}}{q} \tag{4.74}$$

We have proved previously, that the best response strategies for the first three players are $FollowP$, $FollowK$, and $FollowM$ separately. In the following part, we will show that the best response strategy for player 4 is $FollowQ$.

In general, the costs of player 4 under different strategies are as follows:

$$C_1(FQ) = C(p, k, m, q) + C(p', k, m, q) + C(p, k', m, q) \tag{4.75}$$

$$+C(p, k, m', q) + C(p', k', m, q) + C(p', k, m', q) \tag{4.76}$$

$$+C(p, k', m', q) + C(p', k', m', q) \tag{4.77}$$

$$C_1(NFQ) = C(p, k, m, q') + C(p', k, m, q') + C(p, k', m, q') \tag{4.78}$$

$$+C(p, k, m', q') + C(p', k', m, q') + C(p', k, m', q') \tag{4.79}$$

$$+C(p, k', m', q') + C(p', k', m', q') \tag{4.80}$$

As we discussed in Section 4.5.4, if q is smaller than the lower bound of the computed range, the T-PBF will be saturated and lose its estimation ability in a long run. As mentioned above, $\frac{1-e^{-\frac{pk(n-f_{max})}{m}}e^{-pf_{max}}}{1-\epsilon} \leq q \leq 1$, then $q' < \frac{1-e^{-\frac{pk(n-f_{max})}{m}}e^{-pf_{max}}}{1-\epsilon}$. In that case, choosing q' will saturated the bloom filter. As a result of that, the best response strategy for player 4 is $FollowQ$.

To sum up, $\{FollowP, FollowK, FollowM, FollowQ\}$ is a Nash equilibrium for the game. In another word, the parameter selection algorithm of T-PBF is a Nash equilibrium.

□

4.7 Evaluation

In this section, we evaluate the performance of the PBF and its extensions, the C-PBF and the T-PBF, using one web query dataset and one real Internet traffic dataset. We compare the performance of our proposed algorithms with the following three baselines, in terms of estimation accuracy and memory usage:

- Counting Bloom Filter (CBF) Fan et al. (2000): as a well-known extension of the Bloom Filter, CBF uses counters to replace bits in each filter bucket. If the frequency of an element grows large, CBF requires allocation of larger memory space to hold all counters.

- Multi-Resolution Space-Code Bloom Filter (MRSCBF) Kumar et al. (2006): the second extension, MRSCBF, employs multiple filters operating at different resolutions, where the frequency estimation is performed by looking up this number in a pre-computed lookup table.

- Random Sampled Netflow (RSN) Cisco (2010): RSN processes only one randomly selected packet out of n sequential packets, and then estimate the frequency for each sampled element based on its individual counter.

We emphasize that our approach is fundamentally probabilistic and approximate. Therefore, it may not be able to always return accurate estimations, and may fail to identify

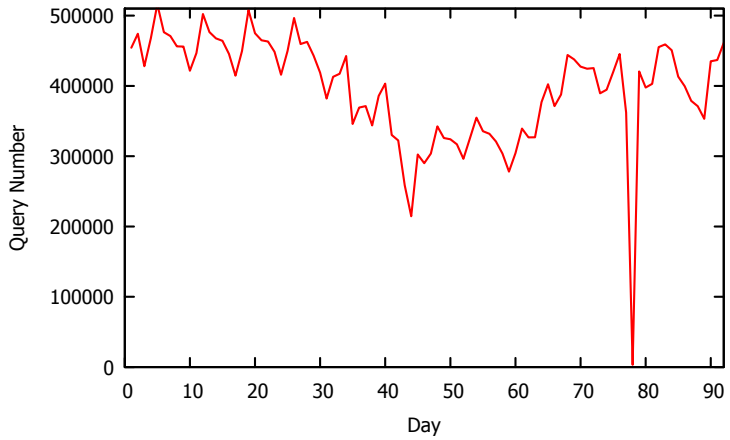


Figure 4.9: The daily pattern for the web query dataset AOL (2006).

heavy hitters in three ways: it may give inaccurate estimates, it may wrongly insert some small flows to the report, and it may miss some large flows. We call these three types of errors: estimation errors, false positives, and false negatives. We define these metrics as following:

- Estimation error ratio: this metric is defined as the difference, in percentage, between the estimated frequency and the real frequency.
- False positive ratio: the false positive ratio is defined as the percentage of falsely reported heavy-hitters, whose frequency are actually below f, among all flows whose frequency are lower than f.
- False negative ratio: the false negative ratio is defined as the percentage of falsely un-reported flows, whose frequency are actually above f, among all flows whose frequency are above f.

4.7.1 Dataset A: Web Query Log Analysis

Our first evaluation dataset is a public web query dataset AOL (2006), where web query logs, with $20M$ web queries, are collected from $650K$ users over a period of three months. Each web query contains the user ID, searching keyword(s), a timestamp, and the web link

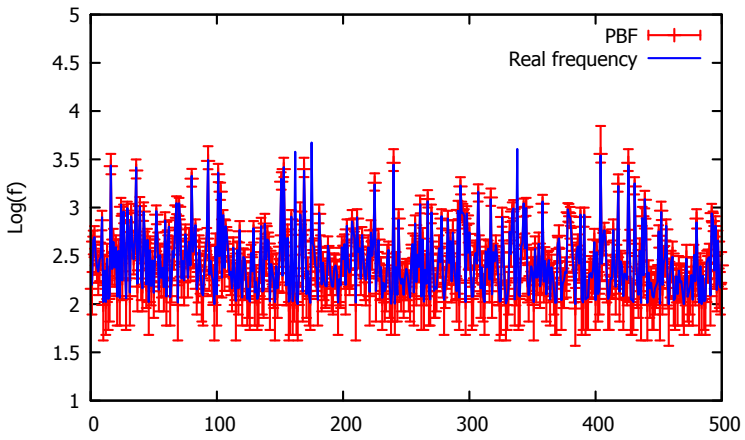


Figure 4.10: Estimations of frequency results of the PBF.

that this user picks. The daily workload pattern is plotted in Figure 4.9, which shows a seven-day repeating pattern on the number of queries per day. In the following, we will use the PBF and the C-PBF to detect popular websites and key words, and use T-PBF to show the trend of the daily query number for the website *www.google.com*.

Note that due to the limited size of the dataset, this task can be easily finished with any conventional method. However, our goal is to demonstrate that our proposed methods can achieve a better memory usage while introducing only limited errors. The limited size of this dataset allows us to evaluate the performance of our proposed methods easily.

Detecting Popular Websites

The dataset includes a total of $378,087$ websites selected by users, where around 0.45% of them have a frequency above 100. We define the popular websites as those that appear for more than 100 times.

To detect popular websites, we first feed the dataset to a PBF with $k = 150$, $p = 0.001$, and $m = 6M$, where the parameters are selected based on the method in Section 4.4.4. Note that the total memory use is only $0.75MB$ ($6M$ bits). Figure 4.10 compares the estimated frequency and the real frequency of the first 500 popular websites. Observe that the estimations are matching real frequencies closely, with an average estimation error computed using the formula $\frac{f_{estimated}-f_{real}}{f_{real}}$ as 4.7%. The primary source

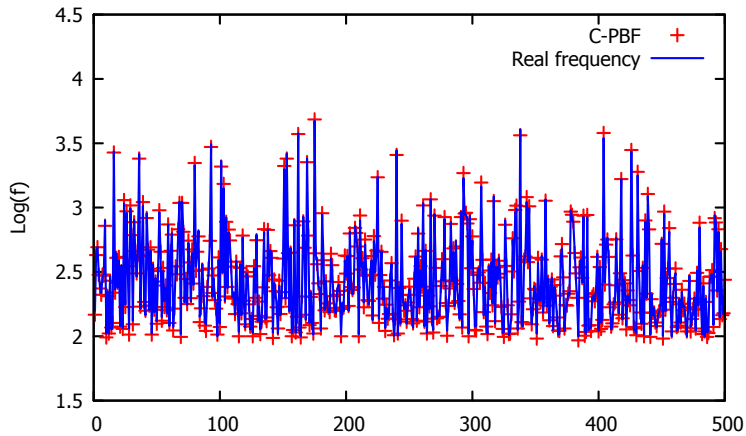


Figure 4.11: Estimations of frequency results of the C-PBF.

of inaccuracies comes from the randomness when flipping a bit from 0 to 1. Also note that Figure 4.10 shows the upper and lower bounds of f estimations, which have much larger errors compared to the estimated value of f (the estimate of f is calculated based on Equation 4.12).

We also test the same dataset with the C-PBF, and the performance is comparable. Specifically, for C-PBF, we set up $k = 50$, $p = 0.03$, $m = 0.6M$. The counter size of C-PBF is set to 10 bits, so that the total memory overhead is the same as PBF. The average estimation error of C-PBF is 4.9%, as shown in Figure 4.11. With this setting of C-PBF, the maximum frequency that can be estimated is $34,105$, according to Equation 4.29, which is sufficient for this dataset.

Next, we compare the performance of PBF/C-PBF with the baselines. The first baseline is CBF, where we set it up with the most suitable parameters. We choose to set the counter size to be 16 bits to accommodate the most frequent flows. Compared to the PBF, the CBF consumes *16 times of the memory* as the PBF and C-PBF under similar settings. The advantages of CBF lie in that it does not provide approximate answers, as demonstrated by the Figure 4.12, where no confidence intervals are plotted. The second baseline is MRSCBF, where we set it up with $l = 32$, $r = 5$, and $m_i = \frac{k_i n l}{\ln 2}$, where $k = \{3, 4, 6, 6, 6\}$ as mentioned in Kumar et al. (2006). As shown in Figure 4.13, the average estimation error of MRSCBF is 10.1%, which is twice of the error of PBF and C-PBF. On the other hand,

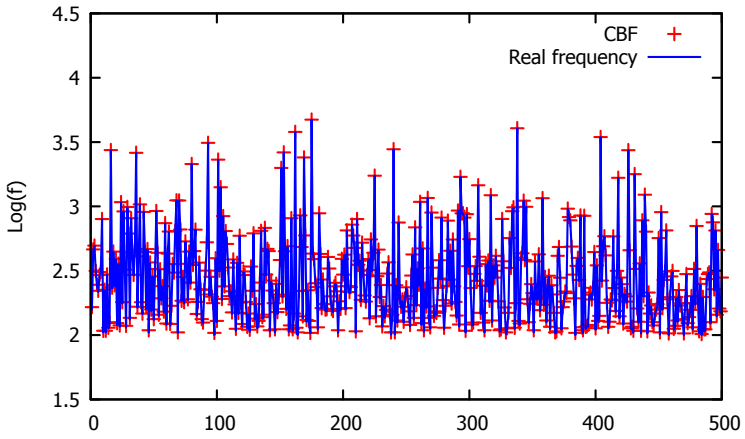


Figure 4.12: Estimations of frequency results of the CBF.

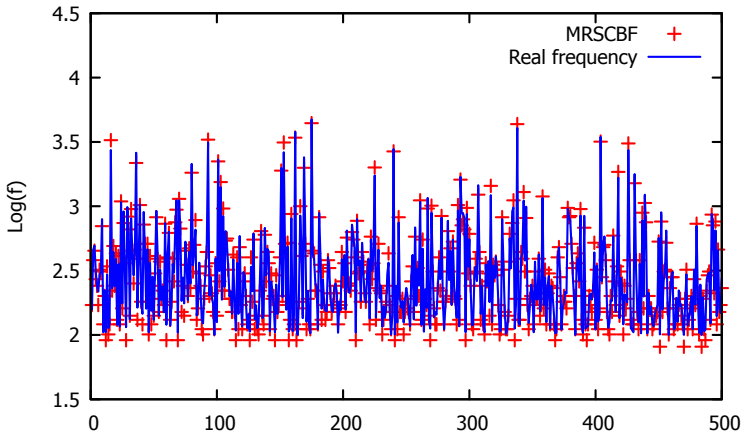


Figure 4.13: Estimations of frequency results of the MRSCBF.

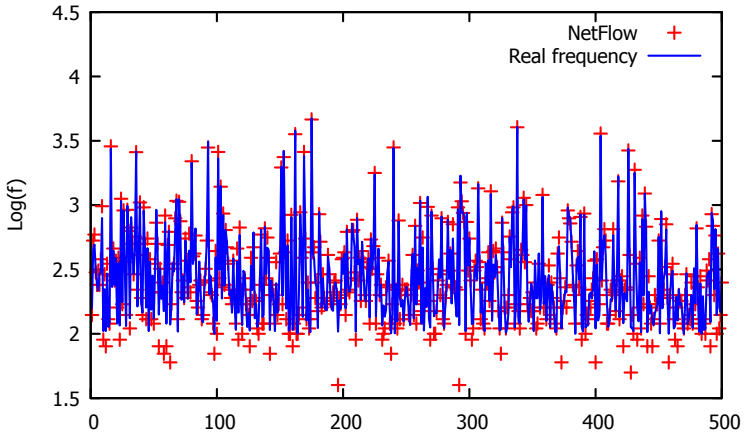


Figure 4.14: Estimations of frequency results of the RSN.

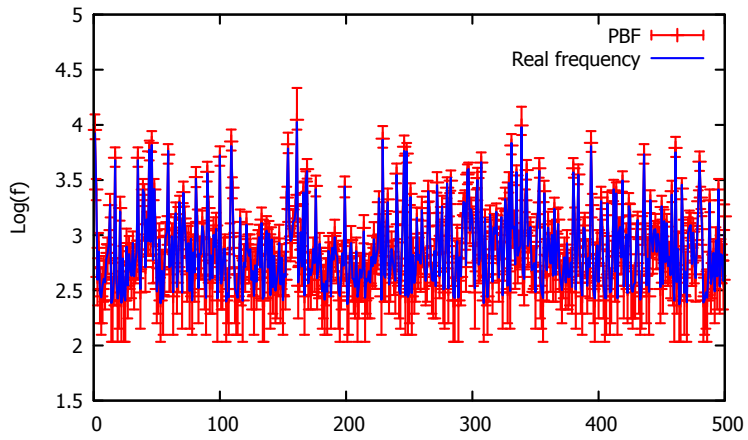


Figure 4.15: Detect popular key word with the PBF for the web query dataset.

besides a long and offline pre-computation phase to get the look up table, the MRSCBF takes $\sum_{i=1}^{5} m_i = 436,371,391$ bits memory space, which is *72.7 times of the memory* as the PBF and C-PBF. The only advantage of MRSCBF is that once the lookup table is built, it only takes a few CPU cycles to estimate the frequency. In PBF, however, a constant k hashing functions need to be calculated. The third baseline is RSN, where we randomly sample one packet out of ten packets. As a result of the low sample rate, RSN only takes $11,063,808$ bits memory space, which is *1.8 times of the memory* as the PBF and C-PBF. According to Figure 4.14, the average estimation error of RSN is 11.3%. The major sources of error are the randomness of sampling, and frequency estimation based on the sample counters and the predefined sample rate. Better estimation error can be achieved, at the cost of taking more memory space.

Detecting Popular Key Word

In this dataset, there are $580,392$ different key words used for searching websites, where around 1.69% of them have a frequency above 100. We define the popular key words as those that appear for more than 100 times.

To detect popular key words, we first feed the dataset to a PBF with $k = 150$, $p = 0.0005$, and $m = 8M$, where the parameters are selected based on the method in Section 4.4.4. Note that the total memory use is only $1MB$ ($8M$ bits). Figure 4.15

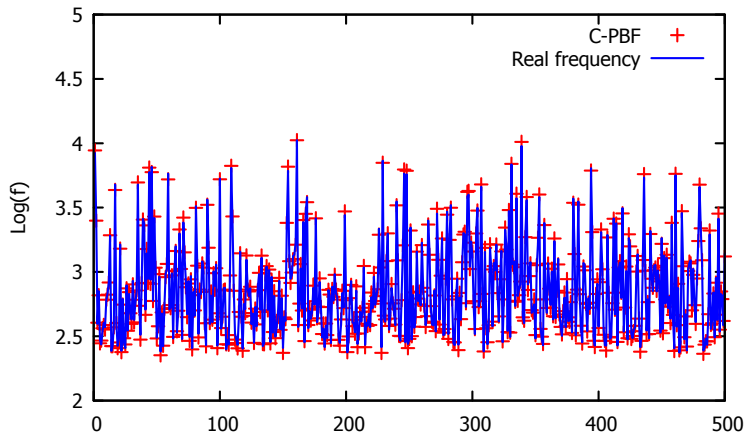


Figure 4.16: Detect popular key word with the C-PBF for the web query dataset.

compares the estimated frequency and the real frequency of the first 500 popular key words. In general, the estimations are matching real frequencies closely, with an average estimation error as 4.9%, where the primary source of inaccuracies is the randomness when flipping a bit from 0 to 1.

On the other hand, we also test the same dataset with the C-PBF, and get a comparable performance. For C-PBF, we set up $k = 50$, $p = 0.02$, $m = 0.8M$. The counter size of C-PBF is set to 10 bits, such that the total memory overhead is the same as PBF. The average estimation error of C-PBF is 3.5%, as shown in Figure 4.16. As the maximum frequency that can be estimated is $34,105$, according to Equation 4.29, there is no missing point in the figure.

Next, we compare the performance of PBF/C-PBF with the baselines. Compare to the first baseline CBF, which is set up with the most suitable parameters. We choose to set the counter size to be 18 bits to accommodate the most frequent flows. Compared to the PBF, the CBF consumes *18 times of the memory* as the PBF and C-PBF under similar settings, as demonstrated by the Figure 4.17. For the second baseline MRSCBF, we also set up $l = 32$, $r = 5$, and $m_i = \frac{k_i nl}{\ln 2}$, where $k = \{3, 4, 6, 6, 6\}$. As shown in Figure 4.18, the average estimation error of MRSCBF is 9.7%, which is twice of the error of PBF and C-PBF. On the other hand, besides a long and offline pre-computation phase, the MRSCBF takes $\sum_{i=1}^{5} m_i = 669,862,928$ bits memory space, which is *83.7 times of the memory* as

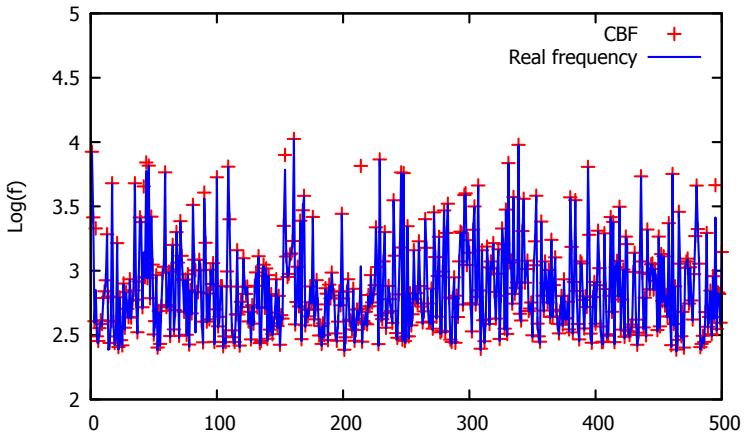


Figure 4.17: Detect popular key word with the CBF for the web query dataset.

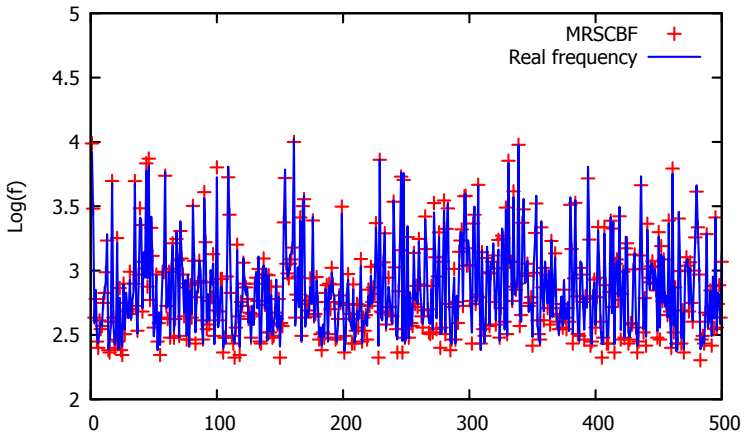


Figure 4.18: Detect popular key word with the MRSCBF for the web query dataset.

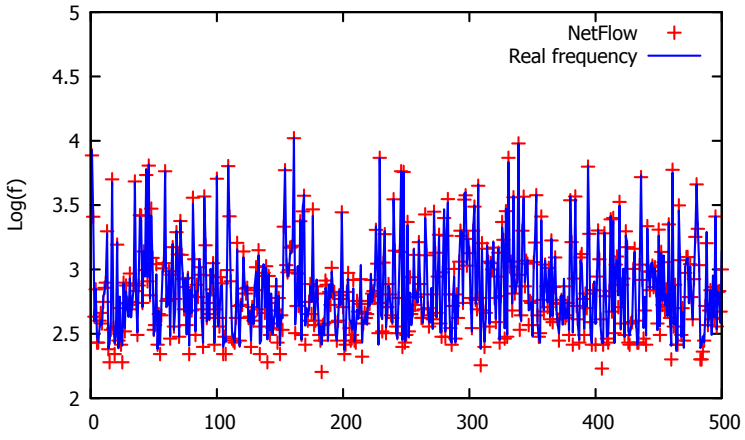


Figure 4.19: Detect popular key word with the RSN for the web query dataset.

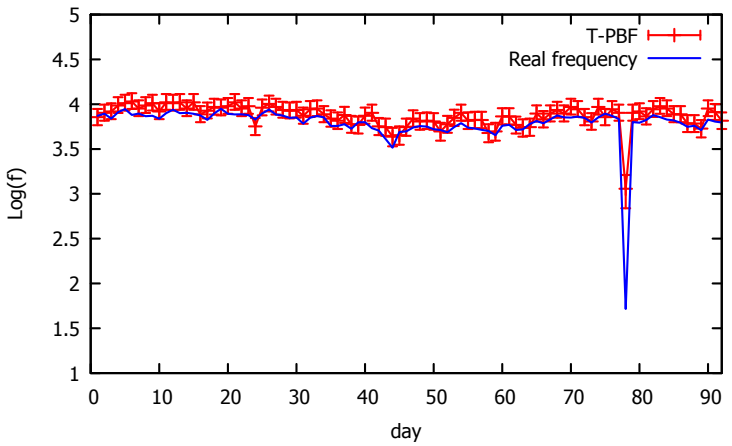


Figure 4.20: Daily frequency trend estimation.

the PBF and C-PBF. The only advantage of MRSCBF is that once the lookup table is built, it only takes a few CPU cycles to estimate the frequency, which is smaller compare to a constant k hashing functions need to be calculated in PBF. For the third baseline RSN, we also choose the sample rate as 0.1. As a result of that, RSN takes $18,487,258$ bits of memory space, which is *2.3 times of the memory* as the PBF and C-PBF. On the other hand, as shown in Figure 4.19, the average estimation error of RSN is 14.2%, which is around three times of the error of PBF and C-PBF.

Frequency Trend of a Popular Website

For the long-term detection, we can use T-PBF to reveal the trend of frequency of a popular website. Here we choose Google as our observing target. For the T-PBF, we set up $k = 150$, $p = 0.001$, $m = 6M$, and $q = 0.8$. We trigger the decay operation every time we finish processing one day's data. As shown in Figure 4.20, in general, T-PBF can estimate the frequency trend of the Google website well. On the other hand, when sudden changes occur, T-PBF cannot adapt fast enough due to the existence of previous epochs' data, which take time to fully decay in the filter.

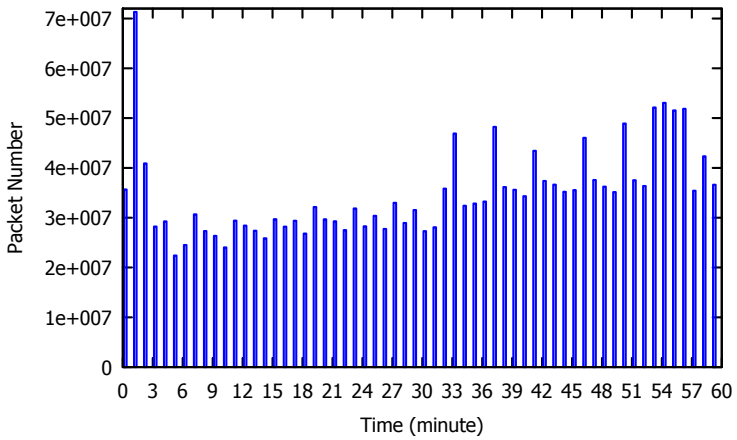


Figure 4.21: Dataset traffic pattern, generated based on CAIDA (2013)

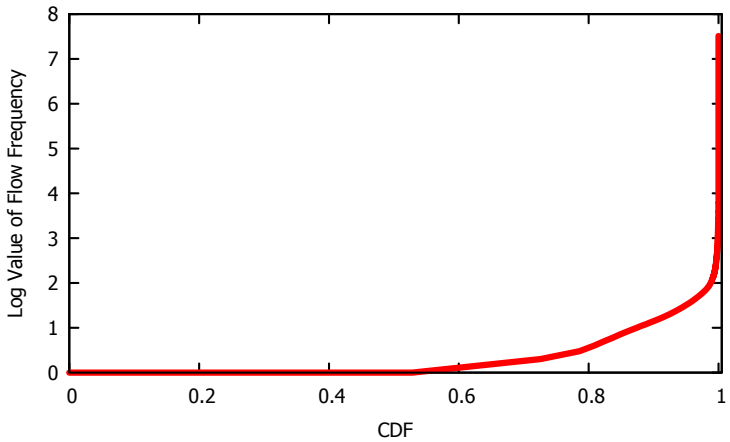


Figure 4.22: Dataset flow frequency, generated based on CAIDA (2013)

4.7.2 Dataset B: Network Measurement Dataset Analysis

In our second case study, we use PBF to analyze heavy-hitters from Internet traffic traces. Our evaluation dataset contains passive traffic traces from CAIDA's equinix-chicago and equinix-sanjose monitors on high-speed Internet backbone links CAIDA (2013). The dataset spans one hour of activity. First, we analyze the general traffic pattern of the trace, by counting the total number of packets collected in each minute. The traffic patterns are shown in Figure 4.21. In the following, we will use the PBF and the C-PBF to detect the heavy hitter flows, and use the T-PBF to detect the long-term trends of flow frequencies.

To differentiate between flows, we use pairs of source and destination IP addresses as the key for each flow. We next count the frequency of each flow in the whole dataset, where the results are shown in Figure 4.22. Observe that most of the flows have a frequency of only once or twice, but a small number of flows exist with a large frequency. In our following experiments, we define a threshold of $1,000$ for those heavy hitters. These flows account for 0.12% of the total $96,854,555$ flows, where the maximum frequency is $32,404,064$.

Detecting Heavy Hitter Flows

First, we evaluate the estimation accuracy of the PBF. We focus on the data traces of the first 5 minutes when we apply PBF and C-PBF in this study. Later we use T-PBF to handle the entire hour of data. The reason is that networked devices such as routers will most likely process datasets in a streaming manner, exactly as what T-PBF does, instead of processing the entire dataset in one operation.

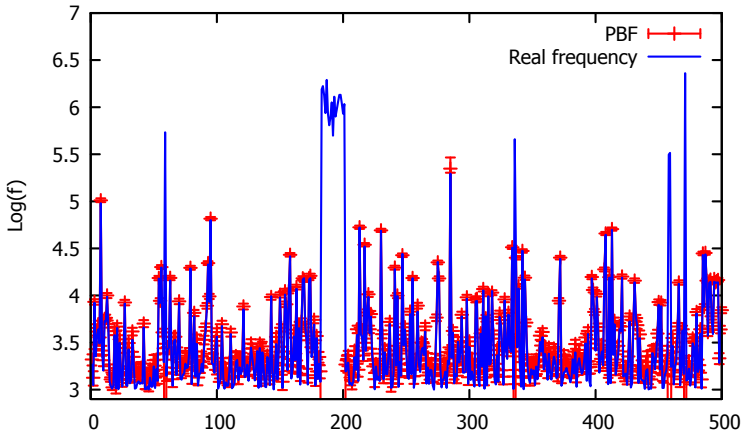


Figure 4.23: Comparison between the real frequency and the estimated frequency with the PBF.

For the first 5 minutes, there are $187,116,831$ packets, which belong to $15,454,076$ different flows. The number of flows with a frequency larger than $1,000$ is $13,569$. As there are limited resource on networking devices, we set $p = 0.00005$, $k = 4,000$, and $m = 800M$ bits, according to the method in Section 4.4.4. The comparison between the

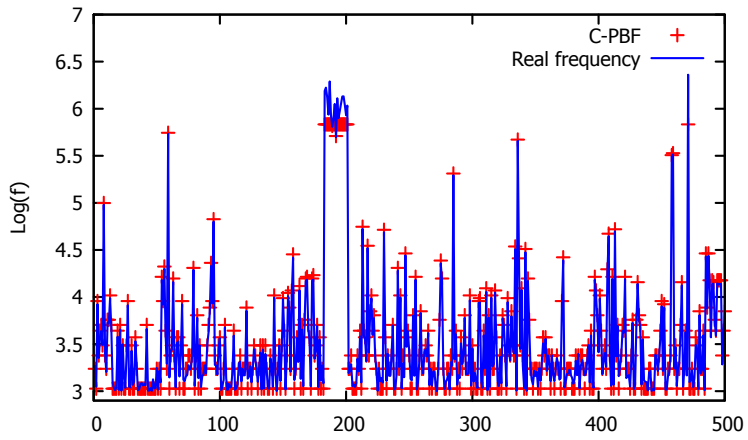


Figure 4.24: Comparison between the real frequency and the estimated frequency with the C-PBF.

real frequency and the estimated frequency of the first 500 frequent flows (sorted by time) are shown in Figure 4.23. Observe that the estimated frequency and the real frequency are almost perfect matching each other. The only exceptions are those cases when the PBF loses its estimation ability as it's k hash bits are saturated with 1s, leading it to miss certain data points. However, we argue that such missing points have no effects on the identification of heavy hitters, as these missing points correspond to heavy flows with a high certainty.

Next, we use the same first 5 minute data to evaluate the estimation accuracy of the C-PBF. To fit the C-PBF into networking devices, we set up the size of the counter to be 10 bits, so that $m = 80M$, $k = 400$, and $p = 0.0015$. As shown in Figure 4.24, the C-PBF can estimate the frequency of flows equally accurately compared to PBF. On average, the estimated frequency is 3.83% larger than the real frequency, caused by the inherent approximate nature of C-PBF counting designs. This accuracy is comparable to the original PBF design.

Next, we compare with the baselines including the CBF, the MRSCBF, and the RSN. For the CBF method, we set $k = 4000$ and $m = 800M$. As shown in Figure 4.25, the estimated frequencies are also close to the real frequencies, with an average error as 3.76%. However, to prevent counter overflows, we have to set each counter to occupy 22 bits, which

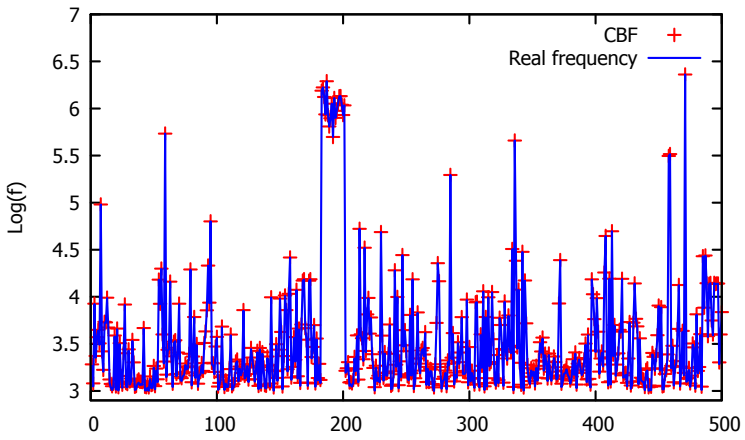


Figure 4.25: Comparison between the real frequency and the estimated frequency with the CBF.

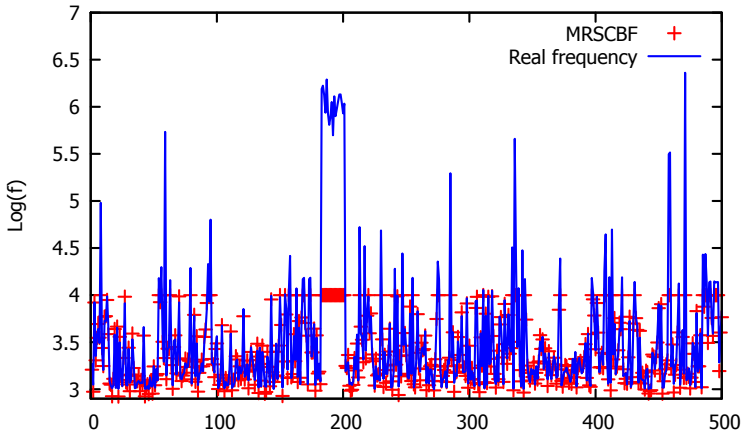


Figure 4.26: Comparison between the real frequency and the estimated frequency with the MRSCBF.

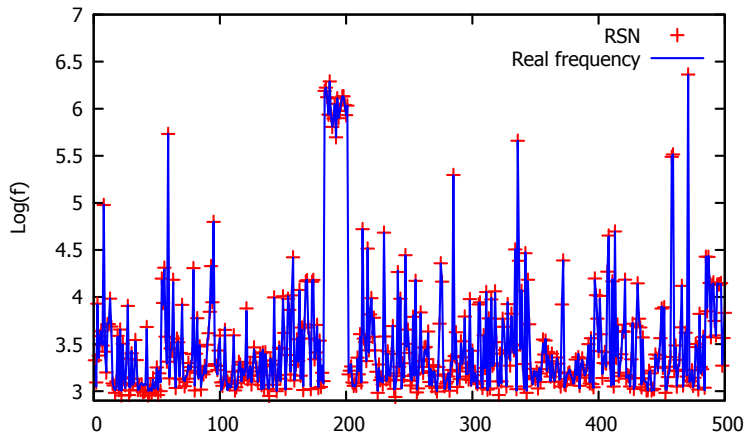


Figure 4.27: Comparison between the real frequency and the estimated frequency with the RSN.

means that in terms of memory overhead, the CBF takes *22 times of the memory* compared to PBF and C-PBF to deliver accurate counting performance. For the second baseline MRSCBF method, we set $l = 32$, $r = 6$, and $k = \{3, 4, 6, 6, 6, 6\}$ to accommodate the most frequent flows. To reduce the computation time of the look-up table, we set the maximum estimated frequency as $10,000$, which has no effects on detecting heavy hitter flows. This is why the estimated frequency is never larger than $10,000$ in the Figure 4.26. Excluding the flows with frequencies more than $10,000$, the average estimation error of MRSCBF is 9.3%, which is more than twice of the error of PBF and C-PBF. On the other hand, it needs $\sum_{i=1}^{6} m_i = 22,117,154,656$ bits of memory to hold this MRSCBF, which means that the MRSCBF takes *27.6 times of the memory* compared to PBF and C-PBF to estimate flow frequencies. For the RSN method, we set sample rate as 0.1. We can observe from Figure 4.27 that the average estimation error is 4.61%. On the other hand, RSN takes $1,304,986,970$ bits of memory space for this large dataset, which is *1.63 times of the memory* compared to PBF and C-PBF.

Long-term Heavy Flow Detection with T-PBF

We next investigate the performance of long-term flow detection with the T-PBF on the whole dataset. For the T-PBF, we set $p = 0.00005$, $k = 4000$, $m = 800M$, and $q = 0.3$.

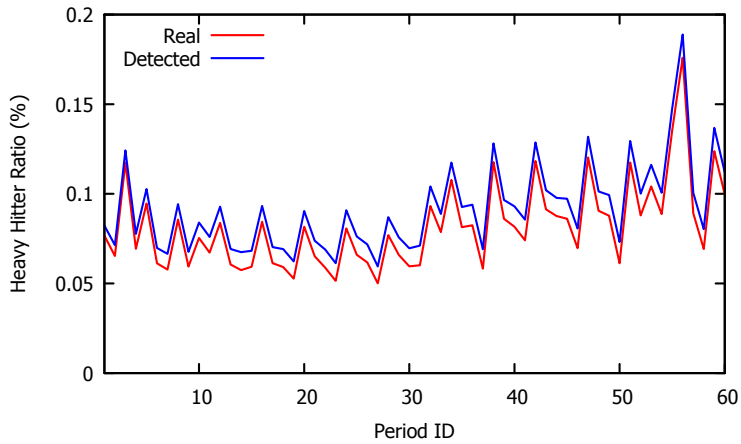


Figure 4.28: Comparison of the real heavy hitter ratio and the detected heavy hitter ratio of the T-PBF.

The decay operation is triggered after processing one minute's traffic data. We calculated the detected heavy hitter ratio (the number of detected heavy flows over the total number of flows), the false positive ratio, and the false negative ratio for each period. For both the false positive ratio and the false negative ratio, the threshold on frequency is set as 1000. As shown in Figure 4.28, the detected heavy hitter ratio on average is 6.48% larger than the real heavy hitter ratio. The source of this inaccuracy comes from the accumulation of flow frequencies in the previous periods. In general, the T-PBF works as expected, as we can observe that it triggers small estimation errors and low memory overhead.

Effects of Parameter Selections

We next perform experiments to evaluate the effects of k on estimation errors, the false positive ratio, and the false negative ratio. With a given k, the computational overhead and query delay is constant. In the experiments, we choose p to be 0.00005 and 0.0001, respectively, and change k from 200 to 2000 with a step size 50. The value of m is computed from Equation 4.17 with different k and p values. As shown in Figure 4.29, the estimation error decreases with the increasing k values. This observation is consistent with our theoretical evaluation shown in Figure 4.2.

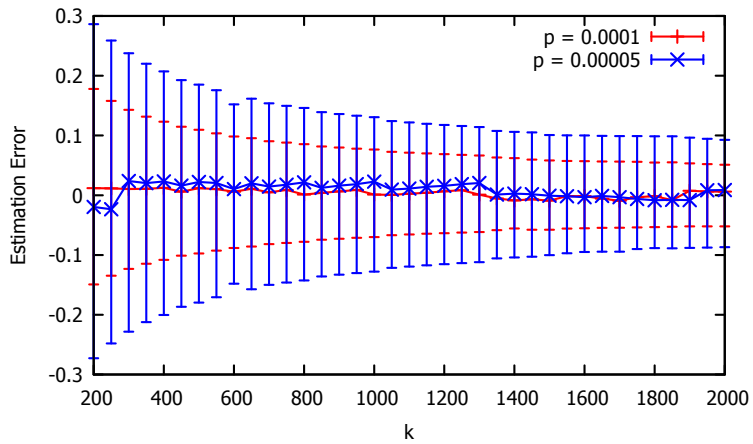


Figure 4.29: The average error rate of the PBF with $p = 0.0001$, and different k and m values.

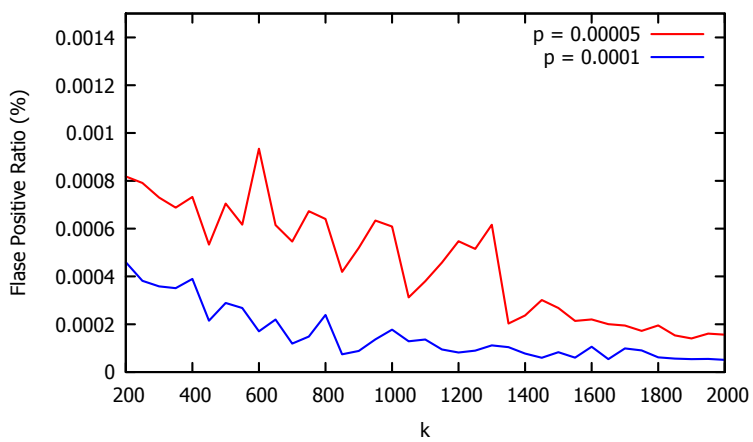


Figure 4.30: The false positive ratio of the PBF.

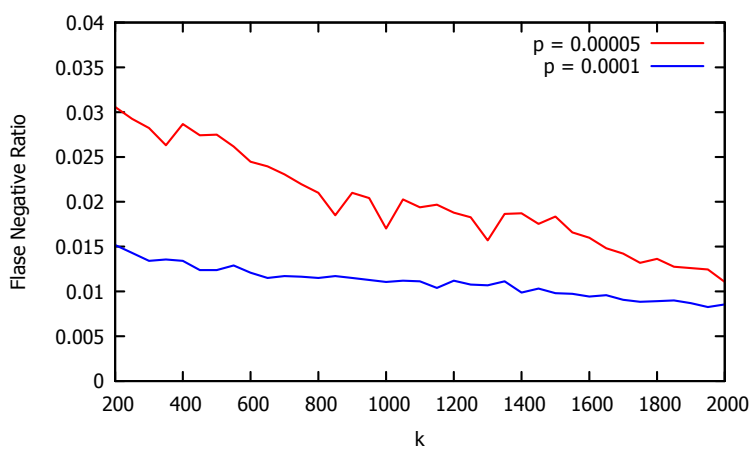


Figure 4.31: The false negative ratio of the PBF.

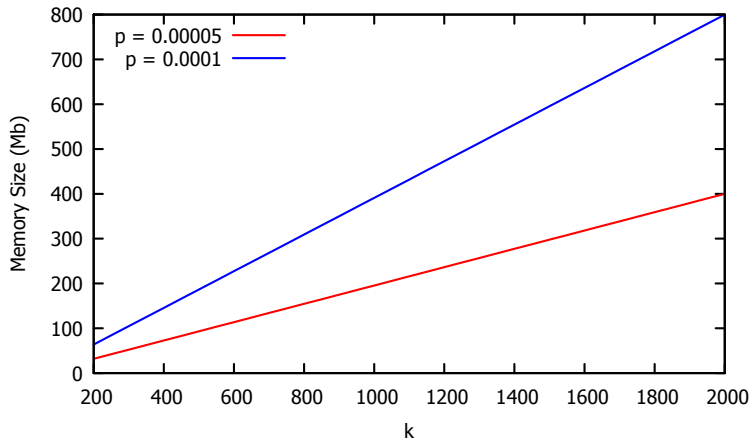


Figure 4.32: The memory overhead of the PBF with different k values.

On the other hand, as shown in Figure 4.30, the maximum false positive ratio is 0.000009, which is acceptably small. The false negative ratio shown in Figure 4.31 has a maximum value of 0.03, which is also acceptably small. Note that the false negative ratio is larger than the false positive ratio, because the number of heavy hitter flows is much smaller than the number of non-heavy-hitter flows. We also observe that by increasing k, the false positive and false negative ratios are decreasing. Finally, with the increasing of k and p, the needed memory size is increasing as shown in Figure 4.32. This explains the tradeoff between the error rates and the memory overhead: a larger memory overhead typically leads to a better performance.

4.8 Conclusion

In this paper, we develop the probabilistic bloom filter (PBF), which extends conventional bloom filters to perform probabilistic counting operations. We provide the PBF's APIs to demonstrate how they can be used by applications, and quantitatively investigate the performance of the PBF through analytical approaches. The derived closed form results answer our questions regarding the capacity, accuracy, and parameter selection of the PBF. Finally, we also extend the PBF into two variants: a counting PBF (C-PBF) and a time-decaying PBF (T-PBF), for additional application needs. Our evaluation results based

on two realistic datasets show that this design outperforms the existing, state-of-the-art approaches.

To the best of our knowledge, our work in this paper is the first probabilistic bloom filter that is designed to count large volume of data with adjustable capacity and accuracy. We hope our work can stimulate future work in this direction, and provide a basis for investigations towards better methods based on probabilistic counting and bloom filters.

Chapter 5

EDAL: an Energy-efficient, Delay-aware, and Lifetime-balancing Data Collection Protocol for Heterogeneous Wireless Sensor Networks

5.1 Abstract

Our work in this paper stems from our insight that, recent research efforts on open vehicle routing (OVR) problems, an active area in operations research, are based on similar assumptions and constraints compared to sensor networks. Therefore, it may be feasible that we could adapt these techniques in such a way that they will provide valuable solutions to certain tricky problems in the WSN domain. To demonstrate that this approach is feasible, we develop one data collection protocol called EDAL, which stands for **E**nergy-efficient **D**elay-**a**ware **L**ifetime-balancing data collection. The algorithm design of EDAL

leverages one result from OVR to prove that the problem formulation is inherently NP-hard. Therefore, we proposed both a centralized heuristic to reduce its computational overhead, and a distributed heuristic to make the algorithm scalable for large scale network operations. We also develop EDAL to be closely integrated with compressive sensing, an emerging technique that promises considerable reduction in total traffic cost for collecting sensor readings under loose delay bounds. Finally, we systematically evaluate EDAL to compare its performance to related protocols in both simulations and a hardware testbed.

5.2 Introduction

In recent years, wireless sensor networks (WSNs) have emerged as a new category of networking systems with limited computing, communication, and storage resources. A WSN consists of nodes deployed to sense physical or environmental conditions for a wide range of applications, such as environment monitoring Tolle et al. (2005), scientific observation Werner-Allen et al. (2006), emergency detection Li and Liu (2009), field surveillance Vicaire et al. (2009), and structure monitoring Xu et al. (2004). In these applications, prolonging the lifetime of WSN and guaranteeing packet delivery delays are critical for achieving acceptable quality of service.

Many sensing applications share in common that their source nodes deliver packets to sink nodes via multiple hops, leading to the problem on how to find routes that enable all packets to be delivered in required time frames, while simultaneously taking into account factors such as energy efficiency and load balancing. Many previous research efforts have tried to achieve trade-offs in terms of delay, energy cost, and load balancing for such data collection tasks Liu et al. (2010); Yang Wengi (2011). Our key motivation for this work stems from the insight that, recent research efforts on open vehicle routing (OVR) problems are usually based on similar assumptions and constraints compared to sensor networks Eksioglu et al. (2009); Braysy and Gendreau (2005); Ozyurt et al. (2006). Specifically, in OVR research on goods transportation, the objective is to spread the goods to customers in finite time with the minimal amount of transportation cost. One may

wonder, naturally, if we treat packet delays as delivery time of goods, and energy cost as delivery cost of goods, it may be possible to exploit research results in one domain to stimulate the other.

Motivated by this observation, our work in this paper develops EDAL, an **E**nergy-efficient **D**elay-**A**ware **L**ifetime-balancing data collection protocol. Specifically, EDAL is formulated by treating energy cost in transmitting packets in WSNs in a similar way as delivery cost of goods in OVR, and by treating packet latencies similar to delivery deadlines. We then prove that the problem addressed by EDAL is NP-hard. To reduce its computational overhead, we introduce both a centralized meta-heuristic based on tabu search Tan et al. (2001), and a distributed heuristic based on ant colony gossiping, to obtain approximate solutions. Our algorithm designs also take into account load balancing of individual nodes to maximize the system lifetime. Finally, we integrate our algorithm with compressive sensing, which helps reduce the amount of traffic generated in the network. We evaluate both approaches using large-scale simulations with NS-3 NS-3 (2011) as well as a small-scale hardware testbed, and present the evaluation results.

As an extension to our conference paper Yao et al. (2013), which only considered the case of homogeneous sensor network deployments, as reflected by its evaluation that focused on delay and energy efficiency, in this article, we systematically address the very different research challenges of heterogeneous sensor networks to significantly strengthen our design. More specifically, our major contributions in this journal paper are as follows:

- We extend the data collection protocol called EDAL Yao et al. (2013), which employs the techniques developed for OVR in operations research to find the minimum cost routes to deliver packets within their deadlines, to a more comprehensive and general version in the context of heterogeneous networks. The problem formulation is proven to be NP-hard.

- We modified the algorithm design for both the Tabu Search in our centralized heuristic and the status gossiping component in the distributed heuristic to not

only make them suitable for heterogeneous sensor networks, but also improve their performance and stability in actual deployments.

- We consider the challenge brought by sparse event detection, and add a systematic set of experiments for understanding and evaluating the compressive sensing reconstruction errors under different compression rate, data sparsity, and the number of source nodes.

- Besides the simulations in a sensor network with heterogeneous nodes, we also evaluate the performance of the proposed protocols on the IRIS sensor nodes to demonstrate its advantages.

The remainder of the paper is organized as follows. Section 5.3 includes the background about open vehicle routing problem, compressive sensing, and other related works. Section 5.4 describes the details about the centralized heuristic and the distributed heuristic. The simulation results on NS-3 NS-3 (2011) as well as comparisons with baseline protocols are presented in section 5.5. Section 5.6 shows the comparative results on a small scale hardware testbed. Finally, Section 5.7 concludes the paper.

5.3 Background

In this section, we describe the background knowledge on both the vehicle routing problems and compressive sensing.

5.3.1 Vehicle Routing Problems

The vehicle routing problem (VRP) Eksioglu et al. (2009) is a well-known NP-hard problem in operational research. VRP finds routes between a depot and customers with given demands so that the transportation cost is minimized with the involvement of the minimal number of vehicles, while satisfying capacity constraints. With additional constraints, VRP can be further extended to solve different problems, where one of the

most important is the vehicle routing problem with time windows (VRPTW) Braysy and Gendreau (2005). This problem occurs frequently in the distribution of goods and services, where an unlimited number of identical vehicles with predefined capacity serve a set of customers with demands of different time intervals (time windows). VRPTW tries to minimize the total transportation cost through the minimum number of vehicles, without violating any timing constraints in delivering goods. If vehicles are not required to return back to the depot, and if the time windows are replaced by deadlines, VRPTW can be further extended to the open vehicle routing problem with time deadlines (OVRP-TD) Ozyurt et al. (2006).

As an NP-hard problem, OVRP-TD has inspired many heuristics. Ozyurt et al. Ozyurt et al. (2006) proposed the nearest insertion method, where the farthest node is chosen first to be connected with a route. Then, repeatedly, each selected node chooses the nearest neighbor that has not been assigned a route so far, and connects itself to this neighbor. This procedure repeats until all customers are connected by routes. Solomon Solomon (1987) developed the push forward insertion heuristic (PFIH), which repeatedly selects the customer with the lowest additional insertion cost as the next node, until all customers are connected. Once initial routes have been found, various algorithms Du and He (2012); Cheng and Wang (2009); Chiang and Russell (1996); Ozyurt et al. (2006) are developed to generate near optimal solutions based on simulated annealing Skiscim and Golden (1983), tabu search Tan et al. (2001), or genetic programming Holland (1992).

5.3.2 Compressive Sensing

Once routes have been found using EDAL, we further refine the data collection efficiency through an emerging technique called *compressive sensing (CS)*. CS is a technique through which data is compressed during their transmission to a given destination, by exploiting the fact that most sensors may not always have valid data to report when they sample the environment Caione et al. (2012); Wu (2009); Cao et al. (2011); Luo et al. (2009); Zheng

et al. (2012); Ling and Tian (2010); Zhu and Wang (2010); Zheng et al. (2011), especially for nodes deployed in stable environments with rare and infrequent events to be detected.

CS works as follows. Consider the case that there are N nodes generating N segments of data. Such data is K-sparse, meaning only K of them are non-zero. We can compress these N pieces of data into M pieces through a linear transformation, such as Equation 5.1, to reduce the number of packets, where $K < M \ll N$. Formally we have:

$$y = \Phi x \tag{5.1}$$

where y is a $M \times 1$ column vector, Φ is a $M \times N$ matrix, and x is a $N \times 1$ column vector. As $M \ll N$, recovering x from y is an ill-posed problem. However, as long as $M \geq K \log N$, x can be accurately reconstructed with very high probability through $l_1 - norm$ minimization Donoho and Tsaig (2006).

Because CS promises improved energy efficiency and lifetime balancing properties Cao et al. (2011), data gathering protocols have been proposed to exploit CS for better performance. Xiang et al. Xiang et al. (2011) proposed a new data aggregation technique derived from CS to minimize the total energy consumption through joint routing and compressed aggregation. Mehrjoo et al. Mehrjoo et al. (2010) employed compressive sensing and particle swarm optimization algorithms to build up data aggregation trees and decrease communication rate. These two methods are different from EDAL in that they require all nodes to contribute sensing data during the data collection phase. On the other hand, Wang et al. Wang et al. (2010) proposed random routing methods based on different network topologies to collect data from a subset of nodes, which is a similar application scenario as EDAL. However, EDAL achieves better energy efficiency because it optimizes the number of constructed routes such that the total number of packets is decreased. We further compare the performance of EDAL with that reported in Wang et al. (2010) in the evaluation section to show that a better gain in energy efficiency is achieved because it exploits the topological requirements of compressive sensing.

5.4 EDAL Algorithm Design

In this section, we propose the EDAL algorithm. First, we describe the problem model, and how we can convert existing approaches in OVR research to sensor networks. Next, given that this problem is NP-complete, we develop both centralized and distributed heuristics for obtaining approximate solutions.

5.4.1 Problem Model

Our formulation of the problem follows a similar approach with those in the literature. We assume that there are N heterogeneous sensor nodes deployed, which are modeled by a connectivity graph of $G = (V, E)$, where E represents wireless links between nodes. For different types of nodes, the radio bandwidth and transmission power are different. All links are assumed to be directional, and each is associated with a metric q representing its link quality. To perform sensing tasks, there are M nodes selected as sources. All packets must be sent to the sink within the required deadline, where different types of nodes have their own deadline requirements. The objective function of the delivery tasks is that all packets need to be delivered with the minimum total cost. The lifetime of a node is defined as the time for it to deplete its energy. A list of these definitions is shown in the Table 5.1.

Based on these notations, for each link $l_{ij} \in E$ and each route k, we define x_{ijk} as

$$x_{ijk} = \begin{cases} 1, & \text{if route } k \text{ contains link } l_{ij} \\ 0, & \text{otherwise} \end{cases} \tag{5.2}$$

Next we initialize c_{ij} for links with appropriate values. If the link quality is poor, then the link cost should be proportionally higher. On the other hand, to meet our goal of lifetime balancing, it is appropriate to assign a higher weight to those links connecting nodes with less remaining energy, so that they will be less frequently selected by the algorithm during execution. Finally, those nodes that consume more energy for transmitting packets are less

Table 5.1: Notations of EDAL

N	Total number of nodes
M	Total number of source nodes
E	Total number of links
K	Total number of routes
L	Maximum level of node energy
$E_{max,s}$	Total energy of node in type s
T_s	The transmission power of node in type s
tp_i	The node type of node i
p_{ki}	The ith node on path k
P_{kp}	The pth packet transmitting on path k
pt_{kp}	The packet type of the pth packet transmitting on path k
l_{ij}	The link connecting node i and j
q_{ij}	The link quality of the link l_{ij}
c_{ij}	The weight of the link l_{ij}
t_{ij}	The time for transmitting a packet over l_{ij}
e_i	The current remaining energy of node i
l_i	The current energy level of node i
d_r	The delay requirement of packet in type r
t	The processing time on node i

likely to be selected. Based on this intuition, we develop the following formula to assign c_{ij} with proper values:

$$c_{ij} = \frac{L - \min l_i, l_j}{q_{ij} \times q_{ji}} \times T_{tp_i} \times t_{ij} \tag{5.3}$$

where:

$$l_i = \left\lceil L \times \frac{e_i}{E_{max,tp_i}} \right\rceil \tag{5.4}$$

where Equation 5.4 defines a step equation for computing the remaining energy level of node i. The ceiling value is computed to differentiate between complete energy depletion and near-complete energy depletion. Together, Equations 5.3 and 5.4 ensure that those nodes with less remaining energy, poor communication links, or more transition energy will have a lower chance of being selected as forwarders.

We now formulate our optimization objective, i.e., delivering all packets to the destination under the constraint that no packet deadline is violated, as follows:

$$\min \sum_{k \in K} \sum_{i \in N} \sum_{j \in N} c_{ij} x_{ijk} \quad \text{s.t.,} \tag{5.5}$$

$$\sum_{j \in N} x_{0jk} = 1, \quad \forall k \in K, \tag{5.6}$$

$$\sum_{i \in N} x_{ihk} - \sum_{j \in N} x_{hjk} = 0, \quad \forall h \in (N - \{p_{k1}\}), \forall k \in K, \tag{5.7}$$

$$\sum_{i \in p_k, j \in p_k} t + \frac{t_{ij}}{q_{ij}} < \forall_{p \in P_k} d_{pt_{kp}}, \quad \forall k \in K. \tag{5.8}$$

where the objective function 5.5 minimizes the total communication cost (if two approaches lead to the same cost, the one with lower number of participation nodes should be chosen), and the constraints 5.6, 5.7, and 5.8 ensure that 1) all routes must end at the sink; 2) the number of routes joining into a node should be the same as the number of routes leaving from it, unless the node is the first or the last of a route; and 3) the time for the packets being transmitted on the routes should not violate packet delay requirements.

5.4.2 Complexity Analysis

In this section, we prove that the aforementioned formulation is NP-hard.

Theorem 5.4.1. *The problem of finding the minimum cost routes to deliver packets within their deadlines, as defined in the previous section, is NP-hard.*

Proof. To prove this fact, we need to select a known NP-hard problem, and show that in polynomial steps, it can be reduced to our problem. The particular NP-hard problem we select is the open vehicle routing problem with time deadlines (OVRP-TD) Ozyurt et al. (2006), which is a variant of vehicle routing problem with time windows (VRPTW) Braysy and Gendreau (2005). This problem aims to find the least cost routes from one point to a

set of scattered points, and has been proven as NP-hard. Formally, this problem is defined as follows: given a graph $G = (V, E)$ with $n + 1$ vertices V and a set of edges E. Let V contain 1 depot node and n customer nodes that need to be served within specified time windows. Each edge in E has a nonnegative weight, d_{ij}, and a travel time, tr_{ij}. Specifically, tr_{ij} includes the service time on node i, which we denote as ts_i, and the transportation time from node i to node j, which we denote as tl_{ij}. The objective is to minimize the total travel cost with the smallest number of routes.

We now show that OVRP-TD can be reduced to our problem within polynomial steps. The graph G in OVRP-TD can be easily transformed to a corresponding sensor network topology by representing vertices with sensor nodes. The depot corresponds to the sink node, and the customers correspond to the source nodes. The cost of the edges, d_{ij}, is a little tricky to handle. Specifically, we need to solve equation 5.3 by adjusting the values of l_i, l_j, or the link quality q properly. On the other hand, however, the link quality q is actually determined since it is related to the transmission time from i to j. That is, given tl_{ij} as a known parameter in the OVRP-TD formulation, we can obtain the appropriate value of q by enforcing that t_{ij}/q_{ij} (in WSN formulation) $= tl_{ij}$ (in OVRP-TD formulation). Recall that t_{ij} is the minimum transmission time of a packet over link l_{ij}. When links are unreliable, multiple transmissions are needed to ensure reliable delivery. Because each transmission is independent, the expected number of transmission rounds is $1/q_{ij}$. Therefore, the total transmission time is t_{ij}/q_{ij}. Since t_{ij} is a fixed parameter depending on the radio hardware and bandwidth, we can decide appropriate q_{ij} for each link from tl_{ij}. After that, we are able to obtain the appropriate $l_i(j)$ values according to equation 5.3.

The remaining formulation is straightforward. The nonnegative transportation cost of each edge in E represents the cost of path connecting two source nodes with the edge. The path cost is computed based on the minimum remaining energy of the adjacent nodes. The time window for each customer is (t_s, t_d), where t_s is the start time of the window, and t_d is the end of the time window. If we set $t_s = p_i$, and $t_d = p_i + d$, where p_i is the start time of the ith period, and d is the packet delay constraint, then the time window in OVRP-TD is transformed to delay bounds in the WSN domain. In this way, we have transformed

OVRP-TD to a special case of EDAL problem formulation in polynomial time. Given that OVRP-TD is NP-hard, the problem defined by EDAL must also be NP-hard.

□

Algorithm 7 Heuristic Algorithm based on Revised Push Forward Insertion

Input: Topology graph G, the source node set S, the deadline set D, the remaining time of packets RT, and the sink node t

Output: A set of routes R with the minimum cost

```
Set candidate list L = ∅ and R = ∅
Calculate the minimum path cost of all source nodes s_i ∈ S to the sink t using the Dijkstra's algorithm
Put all nodes in the source set S into the candidate list L
Find the node s_new that has the maximum path cost to the sink from L, and assign the global variable s_m = s_new
while L ≠ ∅ do
    Remove the node s_new from L
    Assign the remaining time of packet generated by s_new based on the packet type and D, and append the value into RT
    for all node s_i ∈ L do
        Compute the incremental delay d_incr = DELAY(s_new, s_i) + DELAY(s_i, t)
        Compute the insertion cost as PATHCOST(s_new, s_i) + PATHCOST(s_i, t)
        If the insertion cost is the lowest, and the delay d_incr ≤ min_{∀RT_i∈RT} RT_i, pick s_i as s_new
        for all remaining time RT_i ∈ RT do
            RT_i = RT_i − DELAY(s_new, s_i)
        end for
    end for
    if No candidate s_new is found then
        Put the currently found route into R
        Start a new route construction procedure
        Clear RT
    end if
end while
Return R as the output
```

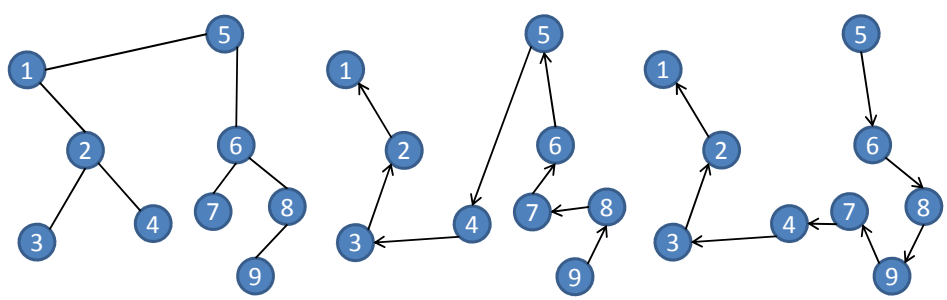
5.4.3 Centralized Heuristics

Given that we have proven the problem of data collection with deadline constraints as NP-hard, we now present heuristic solutions to reduce its computational overhead. In this

section, we propose a centralized meta-heuristic that employs tabu search Tan et al. (2001) to find approximate solutions. We assume that M nodes have been selected as sources at the beginning of each data collection period. The heuristic algorithm consists of two phases: route construction, which finds an initial feasible route solution, and route optimization, which improves the initial results using the tabu-search optimization technique.

In the route construction phase of this algorithm, we present a heuristic algorithm based on the revised push forward insertion (RPFIH) method, as shown in Algorithm 7. The original push forward insertion algorithm was proposed by Solomon (1987), and we modify it to fit the needs of wireless sensor network. At the beginning of RPFIH, for each node, the minimum cost path to the sink is found. RPFIH then finds the node that has the largest path cost to the sink, and incrementally selects candidate nodes with the lowest additional insertion cost. For each candidate node, RPFIH also checks its feasibility by making sure that the overall delay requirement is met. If no candidate node can guarantee the delay, RPFIH initializes a new route with the node that has the largest path cost to the sink in the remaining sources, and repeats this process until all sources are connected with the sink. Finally, RPFIH generates a set of found routes as the final output.

We now analyze the time complexity of RPFIH. As the Dijkstra's algorithm is used, it takes $O(E \log N)$ time to find a minimum weight path between two nodes. In RPFIH, a maximum number of $\frac{(M-1)\times M}{2}$ paths between source nodes need to be computed. Therefore, the overall time complexity is $O(M^2 E \log N)$.



(a) MST **(b)** R **(c)** R^*

Figure 5.1: The worst case and optimal solution of RPFIH.

Next, we demonstrate the following result concerning the approximation ratio of RPFIH.

Theorem 5.4.2. *RPFIH is a polynominal-time 2-approximation algorithm for the VRPTW.*

Proof. We have already shown that RPFIH is a polynominal-time algorithm with time complexity as $O(M^2 E \log N)$. Let R^* denote the optimal routes for the given set of source nodes, and R denote the routes generated by RPFIH. When the delay bound is very tight, each source node must follow the minimum cost shortest path toward the destination. In that case, we can expect that approximately $R^* \approx R$, and $C(R^*) \approx C(R)$, where C represents the total cost of the routes.

On the other hand, if the delay bound is very loose, VRPTW is equivalent to VRP. Furthermore, if the vehicle capacity is not restricted, the lower bound on the cost of an optimal route is the weight of the minimum spanning tree T Cormen et al. (2001) of source nodes, where $C(T) \leq C(R^*)$. On the other hand, in the worst case, we can observe that R becomes a pre-order tree walking of T, while the insertion cost of nodes are ordered in the pre-order tree walking sequence, as shown in Figure 5.1. Since a full walk W will travel through every edge of T exactly twice, we know that $C(W) = 2C(T) \leq 2C(R^*)$. As R is a route that is equal to W where the last link is deleted, we have $C(R) \leq C(W) \leq 2C(R^*)$. Hence, RPIFH is a 2-approximation algorithm. □

Algorithm 8 The Centralized Heuristic Algorithm in EDAL

Input: The list of routes R from RPFI
Output: A list of optimized routes OR

```
Initialize Tabu move BF M-BF with zeroes and candidate list CL = {R}
while Total number of steps is less than M: do
    Perform λ-interchange LSD based intensification on each route in CL
    if A better route is found: then
        Record the partial solution into R
    else
        Perform λ-interchange LSD based diversification on each route in CL
    end if
end while
Output the best solution found so far in R
```

While RPFI generates a list of routes, they are by no means optimal in the sense of the overall cost and delay. We next optimize the initial solution using tabu search. Tabu search is a popular memory-based search strategy for guiding search beyond locally optimal points. To reduce the memory usage for keeping records of found routes in large networks, we replace the Tabu move list in the original design with Tabu move BF. Specifically, tabu search keeps the following data structures:

- Tabu move BF M-BF: this is a BF with fixed size to keep the recent moves, so that problems such as repetition and cycling can be avoided.
- Candidate list CL: this is another list that stores the best solutions found so far by the search process, ranked by their total route cost.
- Maximum number of iterations M: this is a parameter defined to guarantee the termination of iterations.

In our tabu search implementation, we adopt the λ-interchange local search descent (LSD) method, which uses a systematic insertion and swapping of nodes between routes to produce mutations of the current solution. Up to λ nodes can be exchanged. For example, if $\lambda = 2$, a total of eight interchange operations are possible, including $(0, 1)$, $(1, 0)$, $(1, 1)$, $(0, 2)$, $(2, 0)$, $(1, 2)$, $(2, 1)$ and $(2, 2)$, where (i, j) means to choose i nodes in route r_1 and swap it with j nodes in route r_2, while r_1 and r_2 may not necessarily be different. The tabu search exploits LSD in two steps: intensification and diversification. In intensification, the algorithm implements the 2-interchange LSD procedure on each route individually to find the best potential order of nodes. The diversification step enables the algorithm to search out of the local optimum by making random 2-interchange operations between routes so that better routes that are combinations of the original ones can be found. The detailed steps of this algorithm are shown in Algorithm 8.

5.4.4 Distributed Heuristics

One problem with the centralized heuristic algorithm we have developed in EDAL is that it requires information to be collected from each node to a centralized one. In distributed sensor networks, this step will typically incur additional overhead. Therefore, it is usually desirable to distribute the algorithm computation into individual nodes. In this section, we develop a distributed heuristics algorithm for EDAL, where at the beginning of each period, each source node independently chooses the most energy-efficient route to forward packets.

Our developed algorithm is based on the ant colony optimization Chen et al. (2006) and geographic forwarding. It consists of two phases: status gossiping and route construction. In the status gossiping phase, each source node sends forward ants spreading its current status, including its remaining energy level, toward its neighbor source nodes within H hops. Meanwhile, the status data of nearby nodes is collected by each source node with the received backward ants. During the gossip phase, the ants are forwarded with a modified geographic forwarding routing protocol, which chooses the node with the maximum remaining energy while making geographical progress towards the destination as the next hop. Once a node collects status information of all its nearby sources, it enters the route construction phase, and runs RPFIH distributedly based on collected nearby neighbor status, and the estimation of node status outside the immediate neighborhood. The overall algorithm is shown in Algorithm 9.

More specifically, the algorithm works as follows. At the beginning of each period, each source node predicts which nearby nodes to be the source nodes, based on the given random seed for each nearby node. Then the node generates forward ants (an ant is represented as one or more packets) targeting at each of its nearby source nodes. The role of the forward ant is that it explores the path and collects information along the travel, and the role of the backward ant is that it travels back to the source node and informs their pass-by nodes to update their knowledge with the collected information.

Algorithm 9 The Ant Colony Based Gossiping Algorithm

Input: Topology graph G, the source node s, the nearby source node set S_s

Output: s spread its status to nearby neighbors, and collects status of nearby neighbors

1: **for all** $s_i \in S_s$ **do**
2: s generates a forward ant to s_i, where the ant holds a tuple as $< source, destination, intermediate_node >$, where $intermediate_node$ is a tuple $< ID, role, energy_level >$
3: **end for**
4: **if** Node $n \notin S_s$ receive the forward ant a **then**
5: n generates an $intermediate_node$ tuple in, and saves in into the payload of a
6: n forward a with the modified geographic forwarding routing protocol
7: **end if**
8: **if** Two forward ants from source nodes s_i and s_j meet at node $n \notin S$ **then**
9: Ants exchange information
10: n generates backward ants towards s_i and s_j
11: **else if** Node $n \in S_s$ **then**
12: n generates backward ants towards s
13: **else**
14: n picks $s_{new} \in S_n$, where $\textsc{Distance}(s_{new}, s) > \textsc{Distance}(n, s)$
15: Repeat the process, until a source node is found
16: **end if**
17: **if** A backward ant travels to a node n **then**
18: n updates neighbor status with backward ant payload
19: **end if**

When a relay node gets a forward ant, it selects the neighbor node with the most remaining energy to make progress to the destination as the next hop, and sends the ant out. The forward ant collects the information of the status and remaining energy level of each encountered node along the path. The backward ant is released under one of three cases: first, the forward ant meets another ant sent from other source nodes, where they exchange information with each other immediately; second, the initial target of the ant has been reached, and it is found to be a source node; third, the initial target is reached, but it is not a source node. Instead, a newly picked one along the path is. In each of these cases, the backward ant will be sent along the traveled path of the forward ant, and each node along the path will be updated with the collected information carried by the backward ant. With ant colony gossip, one advantage is that we can now reduce the information collected by nodes by making the collected status more relevant. The computation complexity of ant-colony based gossip is at most $4H^2$ in the worst case, where H is the maximum number of hops in the gossip range.

At the end of gossip phase, each source node s collects a list of source nodes, S, and the cost of e edges W_s. Each node in S can be inserted into the route in the route construction phase. For W_s, it contains the costs of e edges traveled by all ants sent or received by node s.

The route construction phase is based on the RPFIH introduced in previous section. For each source node, it triggers the RPFIH if no other nearby source node with a longer distance to the sink is detected in the ant colony gossip phase. As all nodes start with a fixed amount of energy according to the node type, the source node can accurately estimate the status of nearby nodes. In that case, the minimal weight path from a source node to a nearby source node can be calculated with the currently held information. The tricky part is how to find the minimal weight path to the sink, so that the source can examine if the newly formed route violates the delay constraint. We solve this problem by letting each source node first compute the minimal weight path to each of the nodes on the border of its gossip range that make geographical progress toward the sink, and estimate the weight of the path from that node to the sink, so that it can choose the one with minimal total path

Algorithm 10 The Distributed Heuristic for EDAL

Input: Topology graph G, the source node s, the neighbor source node set S, the deadline set D, the remaining time of packets RT, and the sink node t

Output: Constructed routes with $s_i \in S$ with the minimum insertion cost such that D is not violated

1: Run the ant-colony based gossip to collect neighborhood status
2: Estimate the minimum path cost of s and all $s_i \in S$ to the sink t using the Dijkstra's algorithm
3: Put all nodes in the source set S into the candidate list L
4: **if** $\forall \textsc{Distance}(s, t) > \textsc{Distance}(s_i, t)$ **then**
5: goto 14
6: **end if**
7: **if** Route construction packet rc is received **then**
8: Extract partially constructed route pr from rc, and the minimum remaining time of packets d_m of pr
9: **if** s is already assigned a route **then**
10: Send a packet to inform the previous source node, and terminate
11: **end if**
12: Remove $n \in pr$ from S, and goto 14
13: **end if**
14: **for all** Node $s_i \in L$ **do**
15: Compute the incremental total delay $d_{incr} = \textsc{Delay}(s, s_i) + \textsc{Delay}(s_i, t)$
16: Compute the insertion cost as $\textsc{PathCost}(s, s_i) + \textsc{PathCost}(s_i, t)$
17: If the insertion cost is the lowest, and the delay $d_{incr} \leq d_m$, append s_i to pr
18: Update the remaining time of each packet i as $r_i = r_i - \textsc{Delay}(s, s_i)$
19: Send a construction packet to s_i with payload pr and $d_m = \min r_i$
20: **end for**
21: **if** No candidate s_i is found **then**
22: Choose t as the next node, and send construction packet to t
23: Send construction packets with empty route to each $s_i \in S$
24: **end if**

weight. Assume that there is a path p, which takes n hops from node s to the sink, in the gossip phase, node s knows the edge weight of the first k hops as c, then the cost of the whole path is computed as:

$$C_s = \sum_{1<w<k} c_w + (n-k) * \frac{\sum_{w \in W_i} W_{iw}}{e} \tag{5.9}$$

That is, by using the number of hops and the average cost per link, the source can estimate the whole path cost from itself to the sink. The overall distributed algorithm is shown in Algorithm 10. Note that here, a source node i will be triggered to select the next target node by either receiving a route construction packet or being selected as the farthest node to sink. The algorithm terminates after all source nodes are included into their own route. In the route construction phase, only the neighborhood status information is taken into consideration to find the minimum cost path between nearby source nodes. Therefore, the computation complexity for Dijkstra's algorithm is reduced to $O(H^2 \log H^2)$. Overall, the total complexity of the distributed heuristic is $O(H^2) + O(H^2 \log H^2)$, which is $O(H^2 \log H^2)$, where H is the size of gossip range in the number of hops.

5.5 Simulation based Evaluation

In this section, we present the performance evaluation results on a large network topology with a simulation platform. To evaluate EDAL, we implement both the centralized heuristic (C-EDAL) and the distributed heuristic (D-EDAL) described in Section 5.4, and compare their performance in terms of network lifetime, selected nodes, and packet delay, with and without the integration of compressive sensing, to two selected baselines. The network lifetime is defined as the time for critical nodes to deplete their energy in the network. The details are shown in the following parts of this section.

5.5.1 Experimental Settings

In order to understand the network performance of EDAL under different delay requirements, we simulate our design in NS-3 NS-3 (2011). In the simulation, a uniform network topology with 256 sensor nodes is chosen as the simulation environment. On average, each node has at least four adjacent neighbors to communicate with. To accurately reflect true radio properties, we adopt unreliable links in our simulations. The link quality of all links is set to be 0.9 for our comparison purposes. While we acknowledge that more complicated radio communication patterns can be used, adopting this relatively simple radio model is already sufficient to demonstrate the performance differences between our approach and alternative baselines. The sink node is placed outside the grid and between the middle two columns. We assume that a data collection task has been deployed, which is executed periodically with a period length of 2 minutes. At the beginning of each period, a random collection of nodes are selected as sources. Once they have sensing data to report, either C-EDAL or D-EDAL is triggered to generate new routes based on the current selection of source nodes. In D-EDAL, we set the gossip range to be 3 hops.

We compare the performance of C-EDAL and D-EDAL with the following two routing baselines:

- Minimum spanning tree (MST) routing: this is a widely used, conventional routing algorithm of WSNs, where a minimum spanning tree is constructed for collecting data to the sink.

- Location-aware random routing (LRR) Wang et al. (2010): this algorithm works in a similar way to EDAL in the sense that it also focuses on collecting data from a subset of sources. It also integrates a level of randomness in its design, which makes the comparison particularly interesting since EDAL exploits AI based search to introduce similar randomness.

As the goal of EDAL is to connect all source nodes with minimum total cost under the constraint that it aims to achieve a balance between packet delay requirements and lifetime balancing, we compare EDAL with baselines on the following metrics:

- Network lifetime: this metric is computed as the ratio of network lifetime of different algorithms to the network lifetime of MST, which is taken as the standard unit.
- Average selected node number: it is collected as the number of nodes used to form routes under different delay bounds.
- Average energy consumption: it is measured as the average energy consumption of the whole network in each period.
- Node remaining energy: this metric is generated as the percentage of remaining energy from the full battery on each node.
- Packet delays: it is the time consumed for transmitting the packet from the source to the destination.

The energy-efficiency performance is well evaluated with the first three metrics, while the lifetime-balancing and delay-aware performances are clearly indicated with the corresponding fourth and fifth metrics separately. We also integrate compressive sensing with EDAL to achieve a better gain in energy efficiency, by exploiting the topological requirements of compressive sensing.

5.5.2 Algorithm Overhead

In this section, we first evaluate the average time consumed to finish one round of algorithm computation to show the scalability and practicability of our algorithm.

We first discuss the scalability of the centralized heuristic, based on computational time overhead vs. network size, as shown in Figure 5.2. This analysis provides a sense of feasibility for implementing the centralized heuristic in a real sensor network. Observe that, although centralized heuristic provides better performance, it is infeasible to run the

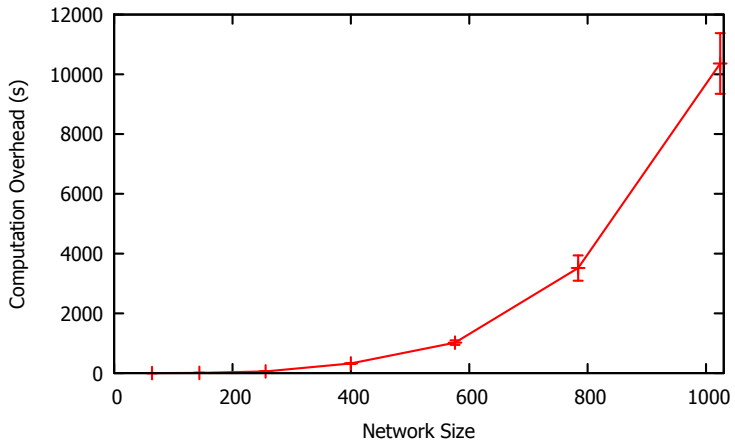


Figure 5.2: Computational time overhead of the centralized heuristic under different network sizes

centralized heuristic for large topologies with more than 400 nodes, as the computation takes longer than the communication period.

We also measured the average time used by each source node for building the routes in a distributed way, based on time overhead vs. gossip range, as shown in Figure 5.3. The computation overhead of the distributed heuristic on each node is tightly correlated with the gossip range, while the algorithm completion time is tightly correlated with the network size. In such case, we collected the time for finishing algorithm computation on each node with different gossip ranges on a uniform network of 256 nodes. Apparently, the larger the gossip range is, the more network status needs to be collected. However, this also leads to longer time to finish computation, which, in fact, will be too long for gossip ranges larger than 5. In this section, we choose the gossip range as 3.

Secondly, we compare the memory usage of using original Tabu search and our modified Tabu search algorithm, which replace the Tabu move list to Tabu move Bloom Filter. After implementing our modified Tabu search algorithm, the memory usage of the centralized heuristic is reduced by 24.9%. This is because Bloom Filter is more space efficient than lists. On the other hand, the memory usage of the distributed heuristic is reduced by 17.1%. The memory saving on centralized heuristic is larger than the distributed

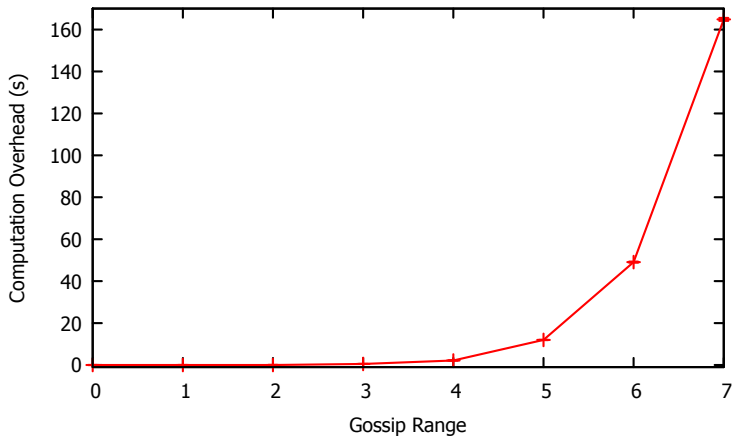


Figure 5.3: Computational time overhead of the distributed heuristic with different gossip ranges in a network with 256 nodes

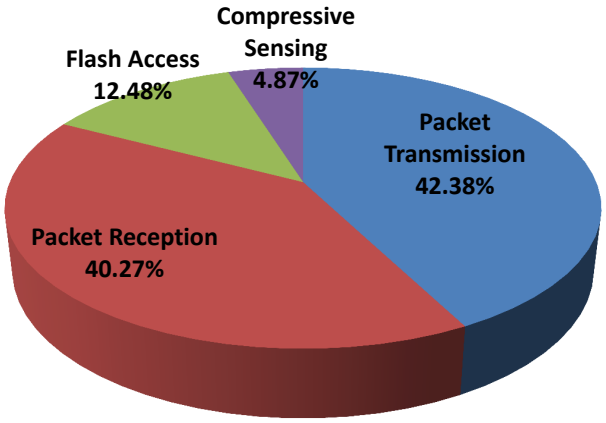


Figure 5.4: The average power distribution of each node for the centralized algorithm.

heuristic is because the number of possible routes found by each node in the distributed algorithm is small in nature, such that there is no large space of memory to be saved.

Third, we evaluate the percentage of energy consumed on each node for different operations. The operations include packet transmission, packet reception, flash access, algorithm computation, and compressive sensing. As shown in Figure 5.4, for each node in the centralized algorithm, on average, the packet transmission takes 42.38% of the total energy, packet reception takes 40.27%, flash access takes 12.48%, and compressive sensing takes 4.87%. There is no energy consumed for algorithm computation, as the algorithm is computed on the central station. On the other hand, for distributed algorithm,

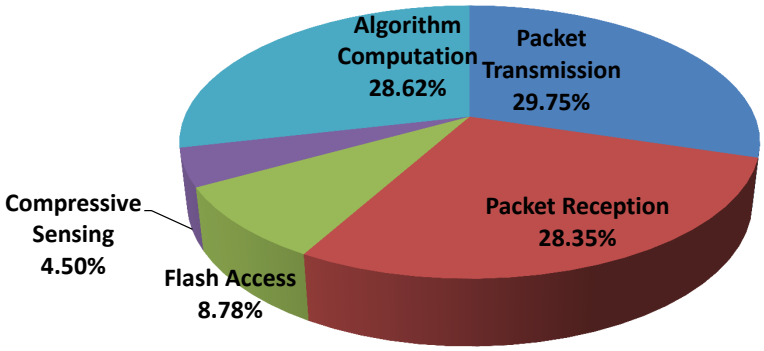


Figure 5.5: The average power distribution of each node for the distributed algorithm.

as shown in Figure 5.5, in the total energy consumption for each node on average, packet transmission takes 29.75%, packet reception takes 28.35%, flash access takes 8.78%, algorithm computation 28.62%, and compressive sensing takes 4.5%.

5.5.3 Experiment Results for Network with Homogeneous Nodes

EDAL without CS

To evaluate the energy efficiency and lifetime balancing effects of EDAL, we first run a set of experiments with EDAL under different delay requirements. The delay bounds of packets are set to be 45 ms, 67.5 ms, 90 ms, 112.5 ms, 135 ms, 157.5 ms, 180 ms, and 202.5 ms separately for each run of experiments. As the chosen baselines do not take delay requirements into consideration, their results do not have big variations on network lifetime and energy consumption under different delay bounds.

As one goal of EDAL is to connect source nodes with the minimal number of relay nodes, we collect the number of nodes used to form routes under different delay bounds, as shown in Figure 5.6. Observe that the average number of nodes used by MST and LRR remains almost the same. For MST, this is because the routing tree is fixed, and the source nodes are randomly selected with a constant probability. For LRR, this is because the routes are randomly constructed based on predefined rules; therefore, the ending result does not have considerable fluctuations. For C-EDAL and D-EDAL algorithms, on the other hand, the number of participating nodes decreases with the increased delay constraints. This

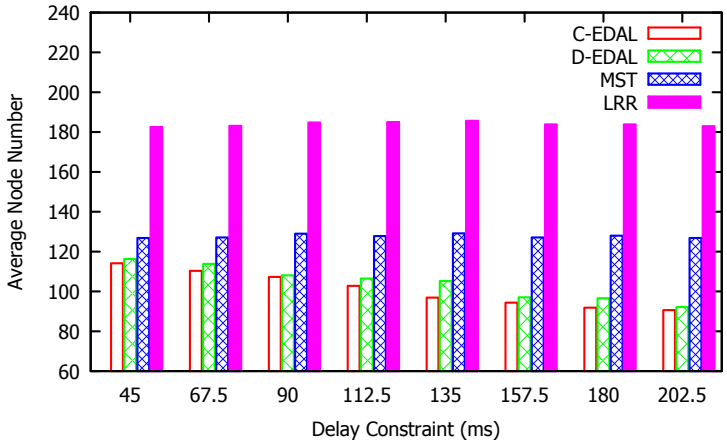


Figure 5.6: The average node number used in each period by different algorithms with different delay requirements

is because as the delay requirement is relaxed, more source nodes can be added into the same route, a feature that is made possible by the tabu search and ant-colony algorithms adopted by EDAL. As a result of that, fewer and fewer nodes are selected due to the triangle inequality (eg. there are two source nodes A, and B. The routes for A or B send packet to the sink node individually takes more nodes than the route for A send packet to B then B send packets to the sink node, in most cases), where relay nodes can serve more source nodes simultaneously.

To evaluate the energy efficiency performance of our design, we measure the average energy consumption of the whole network in each period, as shown in Figure 5.7. We can observe that C-EDAL and D-EDAL consume less energy on average compared to the two baselines. This is because on average they are using fewer nodes to transmit packets in each period.

As the average number of nodes used by MST stays constant under different delay bounds, the network lifetime of MST remains the same as well. As a result, we take the network lifetime of MST as the standard unit, and for each routing algorithm, we compute the ratio of its network lifetime to the standard MST network lifetime. The larger this ratio, the longer the lifetime is. As shown in Figure 5.8, the network lifetimes under C-EDAL and D-EDAL are increasing considerably with the delay bounds, while the network lifetime

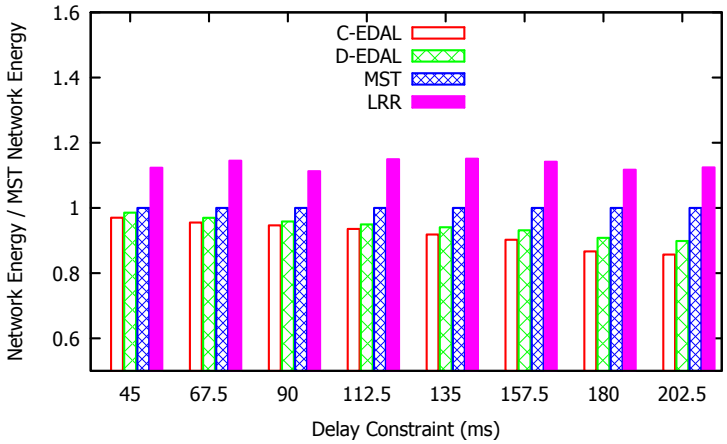


Figure 5.7: The average energy consumption of the network running different routing algorithms

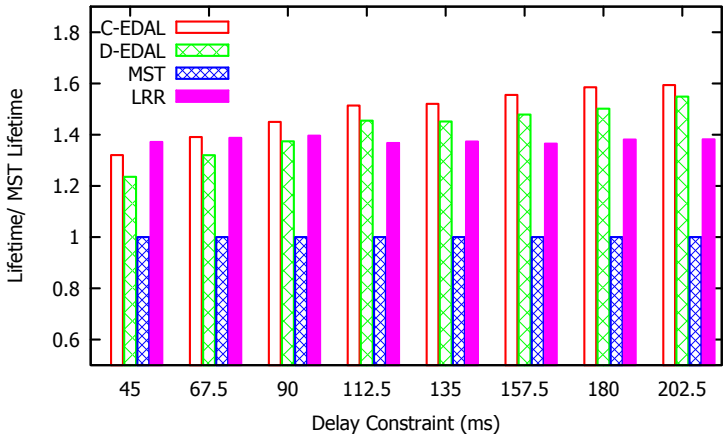


Figure 5.8: The network lifetime while running different routing algorithms with different delay requirements

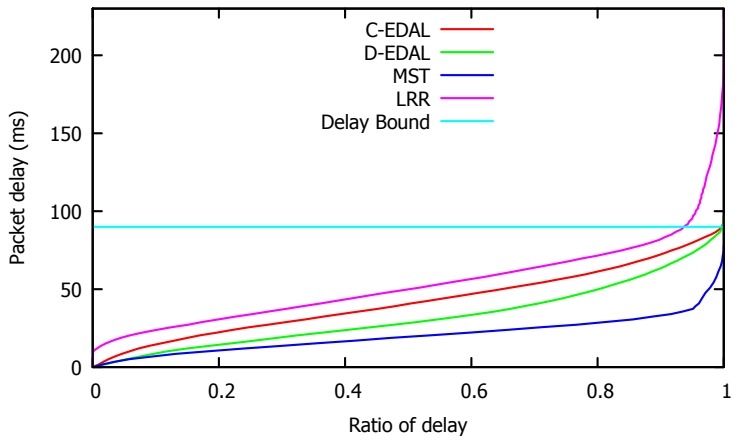


Figure 5.9: The CDF of delay of packets generated in the whole network duration from different routing algorithms

of LRR remains almost constant. This is expected, because as the delay requirements are relaxed, less number of nodes is selected for routing as shown in Figure 5.6. In that case, the total energy consumption also decreases accordingly. In general, compared to MST, C-EDAL increases the overall system lifetime by up to 59.4%, and D-EDAL increases the overall lifetime by up to 54.8%. On the other hand, compared to LRR, C-EDAL increases the lifetime by up to 15.4%, and D-EDAL increases the lifetime by up to 12.1%. The lifetime gains of EDAL over LRR are not very large because in LRR, preliminary optimization has already been applied to filter out those inefficient routes. On the other hand, EDAL takes the delay bound into consideration, so that it is likely to choose the shortest paths especially when the delay bound is tight. That is why the network lifetime is comparably shorter with tighter delay bounds.

To show that EDAL also meets delay constraints, we set the delay bound of data collection tasks to be 90 ms, and measure the overall packet delay for C-EDAL, D-EDAL and the two baselines. As shown in Figure 5.9, the packet delay of MST is the shortest, as the packets take the fewest number of hops to the sink along the routing tree. However, as LRR does not take delay as a constraint when generating random routes, 6.4% of packets violate their delay requirements, and sometimes, the actual delay could be considerably higher. On the other hand, only 0.3% of packets violate delay constraints in C-EDAL

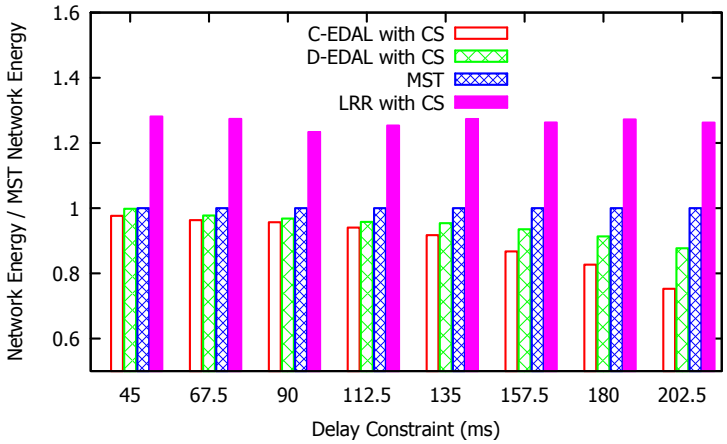


Figure 5.10: The average energy consumption of the network with CS while running different routing algorithms.

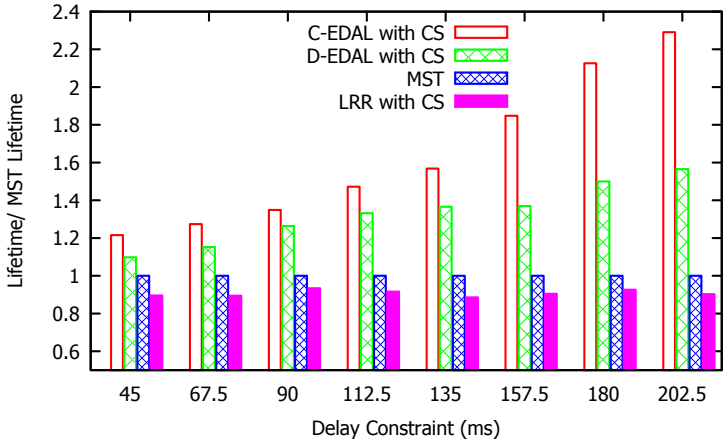


Figure 5.11: The network lifetime with CS implemented on different routing algorithms.

because of the unreliable links. D-EDAL performs a little worse than C-EDAL, having 0.5% of packets violating the delay constraints because of its limited gossiping ranges.

EDAL with CS

Our implementation of EDAL also integrates compressive sensing since this will provide us with better energy efficiency and lifetime balancing. In compressive sensing, sparse data is compressed into a small number of packets. Therefore, we are not only interested in the new energy efficiency of EDAL, but also the reconstruction rate of data.

First, assume that the sparsity of data is known ahead, we evaluate the network lifetime of EDAL and two baselines while changing the packet delay constraints. To measure the energy efficiency performance of EDAL with CS, we also measure the average energy consumption of the whole network in each period. Compare Figure 5.10 and Figure 5.7, we can observe that EDAL with CS consumes less energy during each period than pure EDAL with loose delay bounds. This is because CS enables the network to transmit fewer packets. Figure 5.11 shows the network lifetime of C-EDAL with CS, D-EDAL with CS, MST, and LRR with CS. The lifetime of MST is still used as the standard unit. In general, the network lifetime increases with more relaxed packet delay constraints. In fact, compared to MST, the network lifetime is increased by up to 129.1% for C-EDAL with CS, and up to 56.5% for D-EDAL with CS. This is because EDAL takes into account the number of routes, which it aims to minimize during its optimization process. On the other hand, when the network delay increases, more source nodes can be added into the same path, which results in a further decrease of the number of routes. As a result, the overall relay nodes involved decrease in number, so does the number of packets transmitted by the shared nodes. In contrast, the lifetime for LRR with CS is decreased by 47.1% on average compared to MST. This is because each source node tries to independently generate a random path towards the sink in LRR, causing the nodes near the sink to be shared by many routes. This phenomenon causes such nodes to transmit more packets in total than LRR without CS, which also explains why the lifetime of EDAL with CS is shorter in tight delay bounds compared to EDAL without CS implemented.

To measure the load balancing feature of EDAL, we fix the delay bound to be 180 ms, and collect the CDF of remaining energy on each node at the end of the simulation. As shown in Figure 5.12, as enabled by the load balancing performance of CS, we can observe that the curves of EDAL are less steep compared to the curves of the two baselines, meaning that it achieves a better load balancing result. Also, in this figure, EDAL has less remaining energy because it executes the routing tasks for more periods.

Finally, we consider the case where the sparsity of data is unknown in advance. To achieve the most power-efficient compression ratio, and to achieve low reconstruction

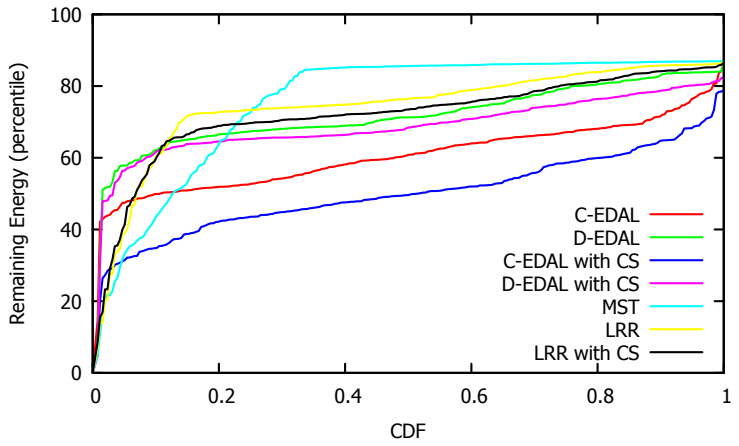


Figure 5.12: The CDF of remaining energy of each node while running different routing algorithms

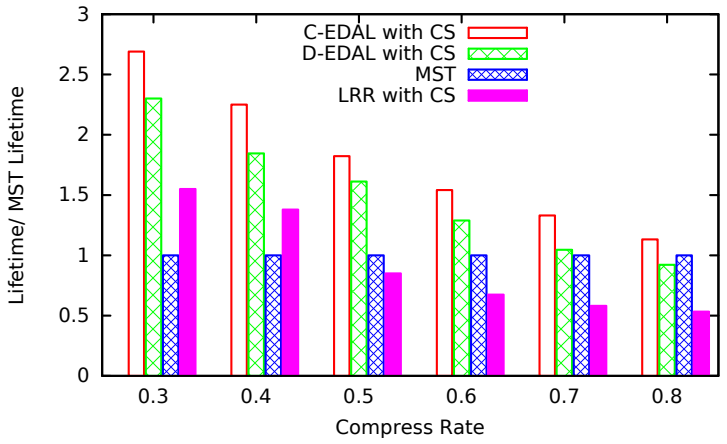
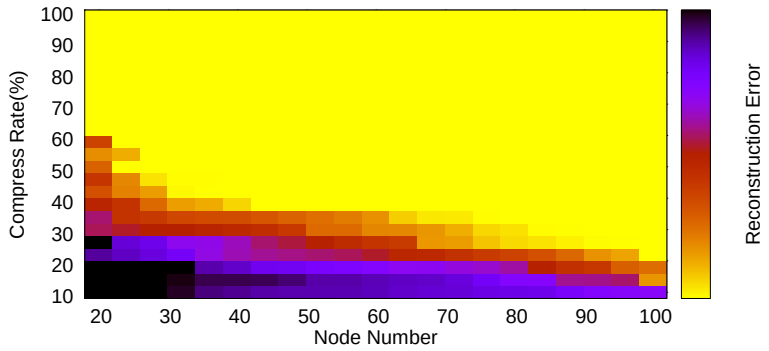
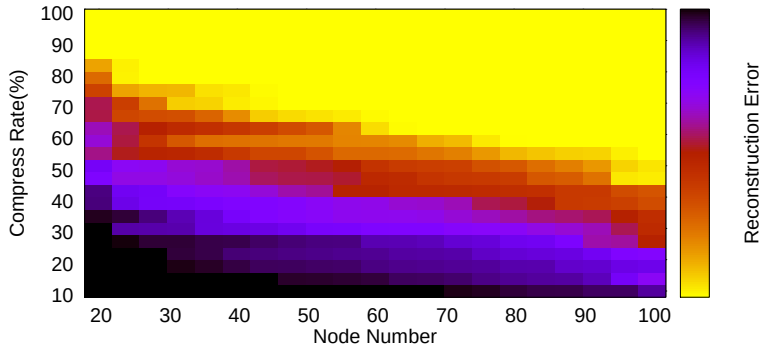


Figure 5.13: The network lifetime with CS implemented on different routing algorithms with delay constraint as 135 ms



(a) 5-sparse



(b) 20-sparse

Figure 5.14: The reconstruction error under different compression rate, data sparsity, and source node number.

errors, we implement a centralized feedback system to adjust the compression ratio for the next period based on the current reconstruction error for all the packets collected in the ongoing period. We define the compression ratio as $\frac{M'}{M}$, where M' is the number of packets collected with compressive sensing, and M is the number of packets that can be used to represent the original data. It is straightforward to understand that the lower the compression ratio is, the fewer the number of packets needs to be transmitted. Therefore, more power can be saved, as shown in Figure 5.13. To make the system work properly, we compute and plot the relationship between reconstruction error and compression ratio, data sparsity, and source node number as shown in Figure 5.14, where the reconstruction error is computed as the average of 100 runs of reconstruction rounds. We predict the data sparsity in next period based on the current reconstruction error and the number of source nodes,

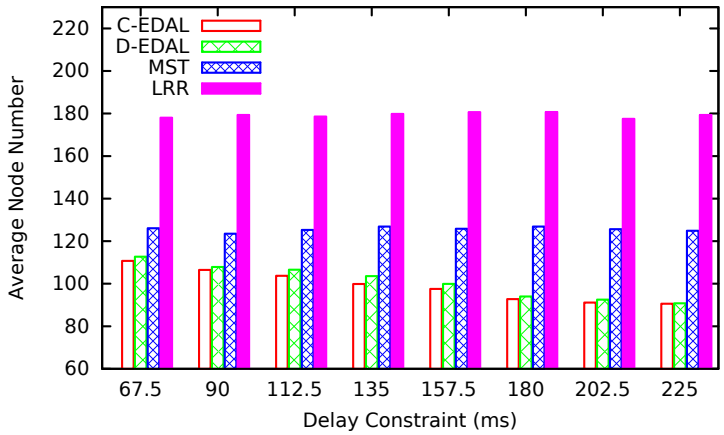


Figure 5.15: The average node number used in each period by different algorithms in the heterogeneous network

so that we can then choose the best compression ratio based on the predicted results. We observe from Figure 5.14 that the compression ratio with acceptable reconstruction error decreases with the decrease of data sparsity and increase of source node number.

5.5.4 Experiment Results for Network with Heterogeneous Nodes

As described in section 5.4, EDAL is designed for heterogeneous networks with heterogeneous nodes and different types of packets. We assume that heterogeneous nodes consume different amount of energy for packet transmissions, and different types of packets have heterogeneous delay bounds. To measure the performance of such networks, we present simulation results on three types of sensor nodes and five types of packets.

EDAL without CS

We first investigate EDAL performance without compressive sensing. We set the average delay bounds of packets to be 67.5 ms, 90 ms, 112.5 ms, 135 ms, 157.5 ms, 180 ms, 202.5 ms, and 225 ms separately for each run of experiments. In each experiment, the number of different types of packets are evenly distributed, and the corresponding delay bounds are set to be $a - 20$, $a - 10$, a, $a + 10$, and $a + 20$ ms, where a is the average delay bound.

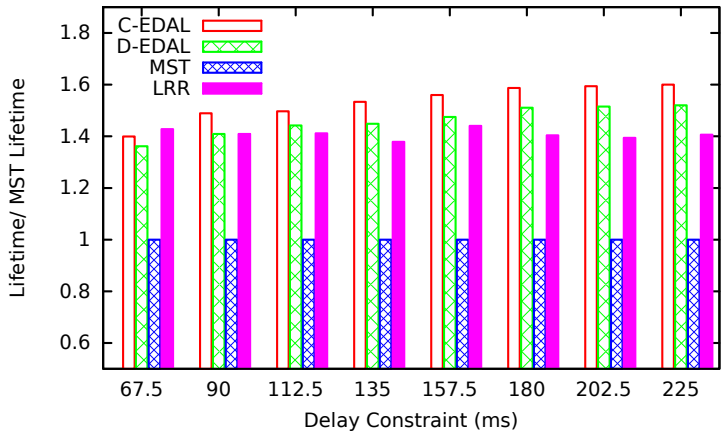


Figure 5.16: The network lifetime while running different routing algorithms in the heterogeneous network

Similar to the earlier experiments, in this section, we first compare the number of nodes used to form routes under different delay bounds, as shown in Figure 5.15. As there is no modification to MST and LRR, the average number of nodes used by these two algorithms is still almost constant, for the same reason explained in the previous section. For C-EDAL and D-EDAL algorithms, as a result of triangle inequality discussed previously, the number of participating nodes is decreasing with the increased delay constraints. However, compared with Figure 5.6, more nodes are used to construct routes for the same average delay bound, as in such heterogeneous networks, the algorithm needs to serve tighter minimum delay requirements. In that case, each route will consist of fewer source nodes.

Secondly, we compare the network lifetime. Similar to the previous section, we take the network lifetime of MST as the standard unit. For each routing algorithm, we compute the ratio of its network lifetime to the standard MST network lifetime as the average number of nodes used by MST stays constant under different delay bounds. As shown in Figure 5.16, similar to the trend of Figure 5.8, the network lifetimes under C-EDAL and D-EDAL are increasing considerably with the delay bounds for the same reason described in previous section. More specifically, compared to MST, C-EDAL increases the overall system lifetime by up to 60%, and D-EDAL increases the overall lifetime by up to 52%. On the other hand, compared to LRR, C-EDAL increases the lifetime by up to 13.79%, and

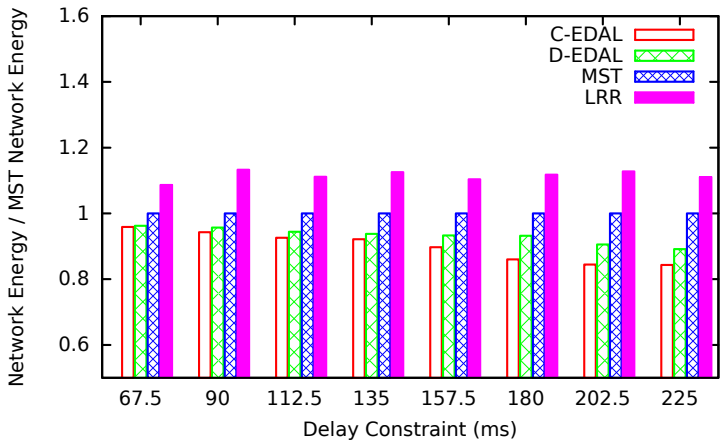


Figure 5.17: The average energy consumption of the heterogeneous network while running different routing algorithms.

D-EDAL increases the lifetime by up to 8.1%. However, as the minimum delay bound of the heterogeneous network is tighter compared to the homogeneous network, for the same average delay bound, the lifetime of the heterogeneous network is shorter.

Thirdly, the average energy consumption for each period of the whole network is measured to explicitly show the energy efficiency performance of our design in heterogeneous network. As shown in Figure 5.17, as fewer nodes are used to transmit packets in each period, C-EDAL and D-EDAL consumes less energy on average compared to the two baselines. However, compared to Figure 5.7, more energy is consumed for serving a tighter minimum delay requirement.

Finally, we set the average delay bound of the heterogeneous network to be 90 ms, and measure the overall packet delay for C-EDAL, D-EDAL and the two baselines. As shown in Figure 5.18, almost all packets of EDAL meet their deadline requirements.

EDAL with CS

In this section, we will compare EDAL with the two baselines, under the condition that the data sparsity is known ahead and the compressive sensing is integrated.

We first evaluate the network lifetime, while the average packet delay constraints are set to different values. The network lifetime of C-EDAL with CS, D-EDAL with CS, MST,

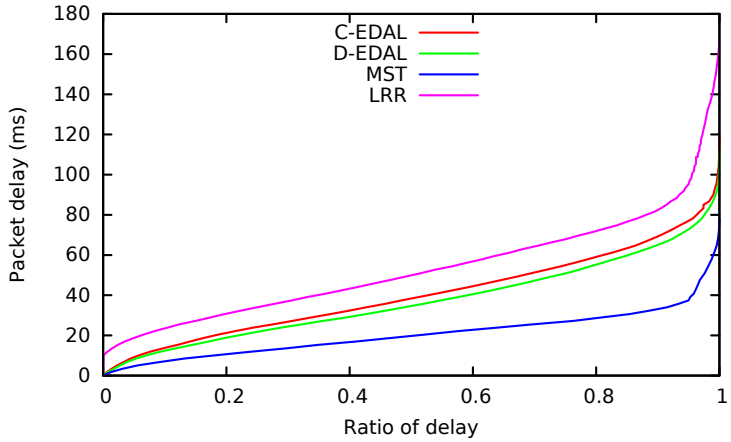


Figure 5.18: The CDF of delay of packets generated in the heterogeneous network from different routing algorithms.

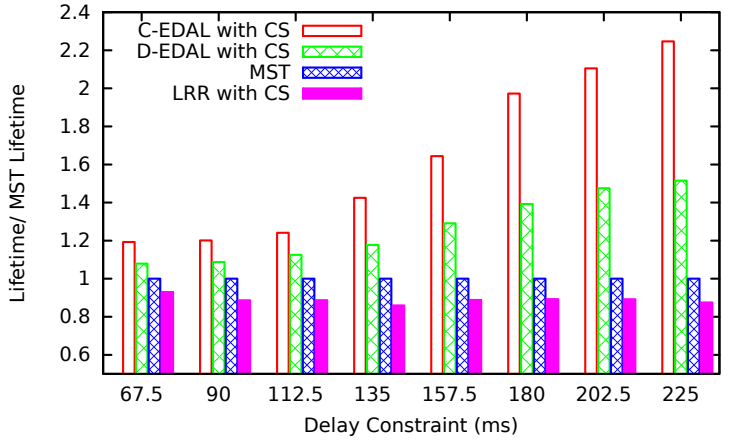


Figure 5.19: The lifetime of the heterogeneous network with CS implemented on different routing algorithms

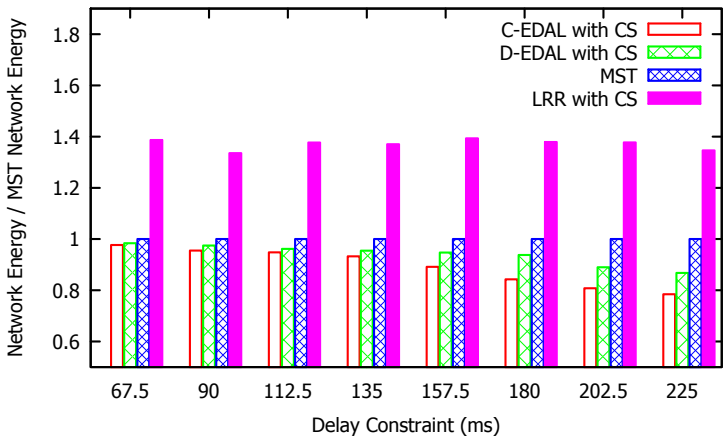


Figure 5.20: The average energy consumption of the heterogeneous network with CS while running different routing algorithms.

and LRR with CS are shown in Figure 5.19. The lifetime of MST is used as the standard unit. Similar to the results shown in Figure 5.16, the network lifetime increases with more relaxed packet delay constraints. As compressive sensing enables fewer number of packets to be transmitted, the lifetime of EDAL with CS is generally longer than the lifetime of EDAL without CS. On the other hand, compared to MST, as EDAL considers generating the minimal number of routes as one of its primary optimization goals, the network lifetime increases by up to 124% for C-EDAL with CS, and up to 51% for D-EDAL with CS. Finally, also observe that the lifetime for LRR with CS is decreased by 50.5% on average compared to MST, and the lifetime of LRR with CS is shorter compared to LRR without CS, which is consistent with our results in the previous section.

Secondly, the average energy consumption of the whole network in each period is measured to show the energy efficiency performance of EDAL. Compare Figure 5.20 and Figure 5.17, we can observe that EDAL with CS consumes less energy for each period than EDAL only, as CS enables the network to transmit fewer number of packets.

Finally, we measure the load balancing feature of EDAL, by fixing the delay bound to be 180 ms, and collecting the CDF of remaining energy on each node at the end of the simulation. As shown in Figure 5.21, the curves of EDAL are smoother than the curves

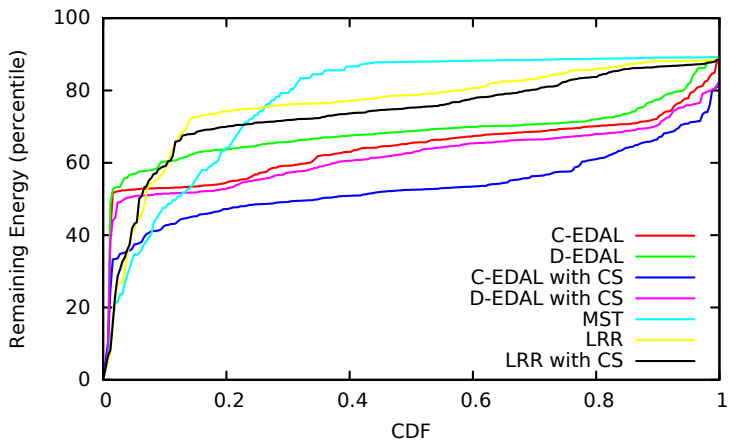


Figure 5.21: The CDF of remaining energy of each node while running different routing algorithms in the heterogeneous network

of the two baselines, as EDAL performs better on load balancing than baselines. The less remaining energy in EDAL is because it runs for more periods.

5.5.5 EDAL Application for Sparse Event Detection

As EDAL tries to select the minimal number of extra nodes, we can use it in sparse event detection to provide improved system lifetime. In such scenarios, we suppose that K of S total events are randomly generated to be measured in each time period, where $K \ll S$. To detect such events, we deploy a sensor network of N nodes so that each node can detect the event with a probability as Pr. Assume there are M sensors that are located in the vicinity of the events, where $K < M \ll S$, and the remaining sensor nodes are put into sleep state. The received signals of sensors are the mix of signals from simultaneously happening events and the thermal noise. The event signals can be reconstructed from sensor readings of no fewer than required number of sensor nodes.

When EDAL collects data on those M events, it selects the minimum number of extra nodes. In that case, if the relay nodes are employed to sense the events besides the source nodes, under the same Pr, fewer nodes are selected compared to other routing algorithms. Assume the total number of selected nodes is n, if n is larger than m, where m is the lower bound of nodes to provide acceptably accurate estimation of X, Pr can be decreased to save

power by selecting fewer than n nodes. To achieve this, we build up a feedback system in C-EDAL to choose the most power efficient Pr based on the signal reconstruction error. If the signal sparsity is known ahead, C-EDAL adjusts Pr simply based on the total nodes used to form the data collection routes. However, in most scenarios, the data sparsity is unknown. Luckily, as shown in Table 5.2, which is a subset of average mapping between Pr and different data sparsity of 100 runs in our simulation network, there is an obvious trend that the reconstruction error increases with the increase of data sparsity under the same Pr, and decreases with the increase of Pr under the same data sparsity. In that case, we can estimate the data sparsity based on the given Pr and reconstruction error, and then choose the minimum Pr that can provide acceptable reconstruction accuracy for the next period.

Table 5.2: Reconstruction error with different node selection probabilities and event sparsity for $N = 256$, $S = 256$.

Pr	5	10	15	20	25	30
10%	0.057	0.215	0.374	0.461	0.571	0.614
15%	2.9E-3	9.8E-3	0.149	0.219	0.317	0.399
20%	1.7E-3	6.6E-3	0.042	0.063	0.165	0.218
25%	2.9E-4	5.4E-3	0.013	0.023	0.061	0.113
30%	2.4E-6	2.1E-3	2.9E-3	3.1E-3	0.026	0.038
35%	5.2E-7	3.9E-5	5.5E-4	2.4E-3	7.1E-3	0.012

5.6 Hardware Evaluation

In this section, we show experimental results for running both the centralized heuristic (C-EDAL) and the distributed heuristic (D-EDAL) described in Section 5.4, on a small scale hardware testbed. We compare their performance in terms of network lifetime, energy consumption, and packet delay, with and without the integration of compressive sensing, to two selected baselines. The details are shown in the following parts.

5.6.1 Experimental Settings

Figure 5.22: Hardware testbed topology.

The hardware testbed consists of 26 IRIS MEMSIC (2012) motes, and a DELL Precision workstation as the central controller. Each IRIS mote is equipped with Atmel ATmega1281 microcontroller, Atmel AT86RF23 radio, and 512K flash space. We deploy the 25 motes in a uniform network topology on a wood board with size 2.3m$\times$2.3m, and use the extra mote as the sink node, as shown in Figure 5.22. To deal with the short mote distances, we set IRIS mote to use the smallest communication power to reduce communication range and interference between motes. The sink node is connected to the DELL workstation, which performs two tasks: first, it collects the traffic information from the sink mote; second, it runs C-DEAL periodically and send result routes to sink mote. The workstation and the sink mote communicate through serial port.

In this hardware testbed, the communication links are not reliable. We collect the number of packets transmitted n_{tr}, and the number of packets received n_r during

experiments between each pair of adjacent nodes, and compute link quality as $\frac{n_r}{n_{tr}}$. The routing tree of MST algorithm is constructed with link cost computed as $\frac{n_{tr}}{n_r}$.

5.6.2 Hardware Experiment Results

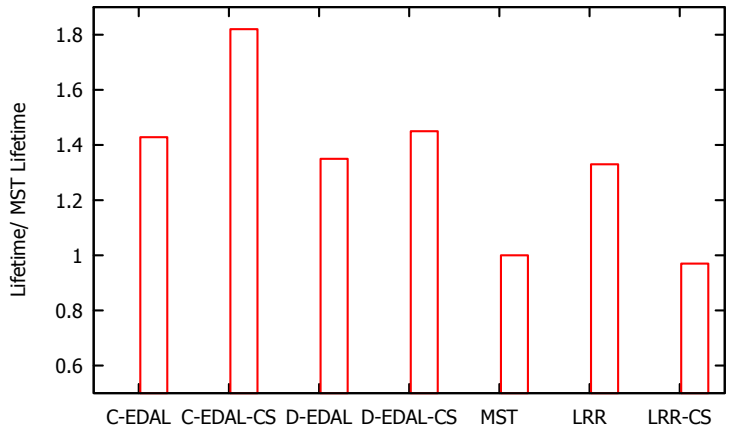


Figure 5.23: The network lifetime while running different routing algorithms in the small scale testbed network

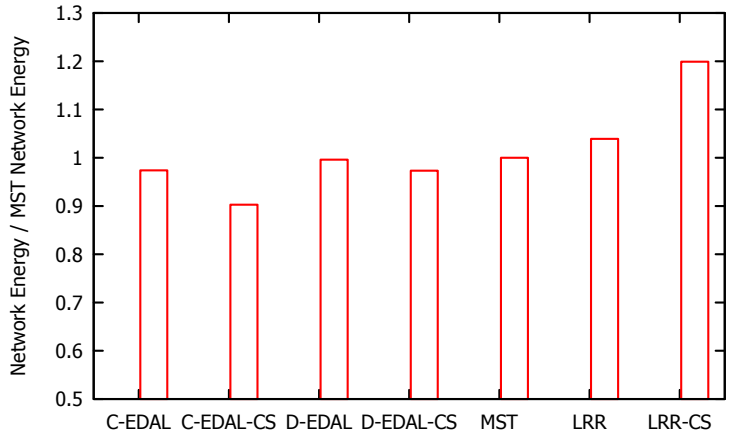


Figure 5.24: The average energy consumption while running different routing algorithms in the small scale testbed network

To evaluate the energy efficiency and lifetime balancing effects of EDAL, we first run a set of experiments with EDAL running alone under fixed delay requirements as 250 ms. In these experiments, similar to what we did in simulations, we take the network lifetime of

MST as the standard unit. For each routing algorithm, we compute the ratio of its network lifetime to the standard MST network lifetime. As shown in Figure 5.23, compared to MST, C-EDAL increases the overall system lifetime by 42.8%, and D-EDAL increases the overall lifetime by up to 34.8%. On the other hand, if we enable CS, these values are increased to 81.9% and 44.9%. Finally, we observe that LRR increases the lifetime by up to 32.9%. However, if CS is enabled, LRR decreases the lifetime by 3.1%.

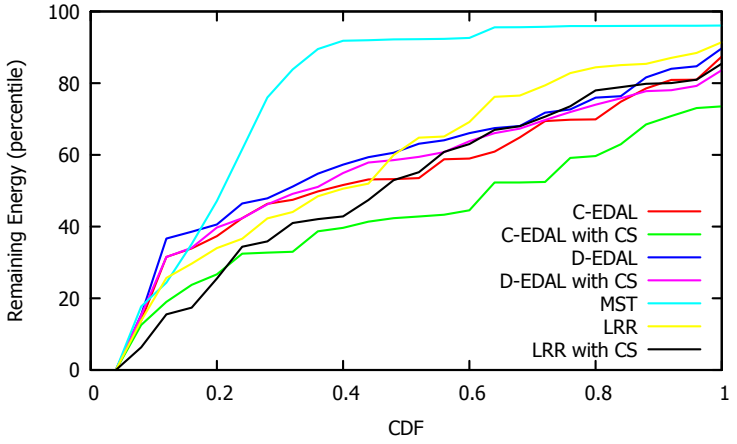


Figure 5.25: The CDF of remaining energy of each node while running different routing algorithms in the small scale testbed network

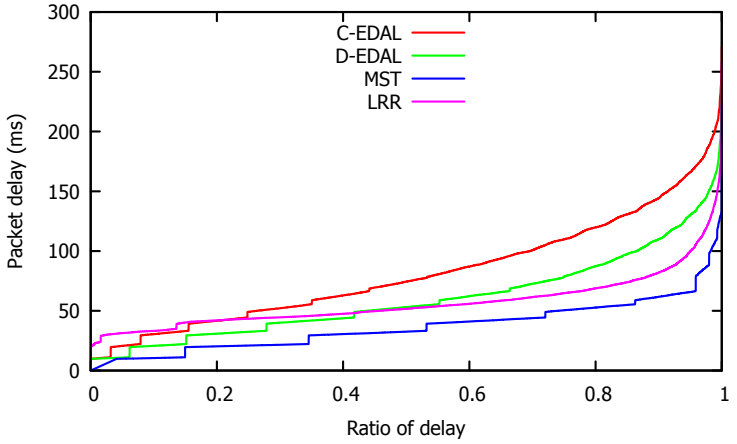


Figure 5.26: The CDF of delay of packets generated in the whole testbed network duration from different routing algorithms

Next, to evaluate the energy efficiency of our design, we measure the average energy consumption of the whole network in each period, as shown in Figure 5.24. Consistent with the simulation results, C-EDAL and D-EDAL consume less energy on average compared to two baselines. This is because on average, they are using fewer nodes to transmit packets in each period. As shown in Figure 5.25, enabled by the load balancing performance of CS, we can observe that the curves of EDAL are smoother than the curves of the two baselines. This shows that EDAL performs better on load balancing than baselines. Finally, to demonstrate that EDAL also meets delay constraints, we measure the overall packet delay for C-EDAL, D-EDAL and the two baselines. As shown in Figure 5.26, we can observe that only less than 0.3% of packets violate their delay bound.

5.7 Conclusion

In this paper, we propose EDAL, an Energy-efficient Delay-Aware Lifetime-balancing protocol for data collection in wireless sensor networks, which is inspired by recent techniques developed for open vehicle routing problems with time deadlines (OVRP-TD) in operational research. The goal of EDAL is to generate routes that connect all source nodes with minimal total path cost, under the constraints of packet delay requirements and load balancing needs. The lifetime of the deployed sensor network is also balanced by assigning weights to links based on the remaining power level of individual nodes. We prove that the problem formulated by EDAL is NP-hard, therefore, we develop a centralized heuristic to reduce its computational complexity. Furthermore, a distributed heuristic is also developed to further decrease computation overhead for large scale network operations. Based on both simulation and hardware testbed evaluation results, we observe that compared to baseline protocols, EDAL achieves a significant increase on network lifetime without violating the packet delay constraints. Finally, we demonstrate that by integrating compressive sensing with EDAL, additional lifetime gains can be achieved.

Chapter 6

Conclusions

Developing power efficient algorithms on networking systems is a very important topic. In this dissertation, we concentrate on achieving energy efficiency on networking systems, such as DCN, WSN and Internet Routers, through three methodologies.

My first work is based on the first method, NP-complete problems and heuristics. The goal of this work is to produce an energy efficient routing protocol for DCNs. Through analyzing the real DCN traces, we found that as long as we avoid consolidating traffic flows that were positively correlated based on 90-percentile demands instead of maximum demands, we could further improve the energy efficiency of traffic consolidation. As a result of that we propose CARPO, a correlation-aware power optimization algorithm that consolidated traffic flows based on the correlation analysis among flows in a DCN, and integrated the traffic consolidation with link rate adaptation for maximized power savings. The integration was formulated as an optimal flow assignment problem, which was known to be NP-Complete. The optimal consolidation solution and data rate of each link in the DCN were computed using a linear programming approach. To reduce the computation complexity, we then proposed a heuristic algorithm to find a consolidation and rate configuration solution with acceptable overheads.

The second method, compressed data structures, is implemented in our second work to save energy consumption on Internet routers. The problem of the second work is

formulated in network measurement, where we addressed the problem on how to quickly identify those heavy-hitters with limited memory space on Internet routers. As many applications could benefit from approximate identifications and measurements of those large flows, we presented a probabilistic version of the Bloom Filter (BF) and its operations, called Probabilistic Bloom Filter (PBF), such that we could identify large flows using a small amount of states. In that case, we could reduce the memory usage to achieve energy efficiency.

The third work mentioned above adopts the third method, which is the combination of the first two methods. In this work, we find that recent research efforts on open vehicle routing (OVR) problems are usually based on similar assumptions and constraints compared to wireless sensor networks. Motivated by this observation, we developed EDAL, an Energy-efficient Delay-Aware Lifetime-balancing data collection protocol. As EDAL addressed an NP-hard problem, we introduced both a centralized meta-heuristic based on the tabu search method, and a distributed heuristic based on ant colony gossiping, to obtain approximate solutions and reduce its computational overhead. Finally, we integrated our algorithm with Compressive Sensing (CS), which helped reduce the amount of traffic generated in the network to save more energy.

Motivated by my current and past research efforts, my future work will expand to the more general domain of energy efficiency related research on networking systems and mobile, ubiquitous systems. Current related works are focusing on designing better network architecture, management algorithms, and performance optimizations for such systems, and among them, energy efficiency is a critical concern that relates to all layers of system and protocol designs. Such a need motivates the development of algorithms to address energy efficiency on different aspects of ubiquitous networks, such as network routing, information sharing, memory usage, CPU utility, among others.

Bibliography

Abts, D., Marty, M., Wells, P., Klausler, P., and Liu, H. (2010). Energy proportional datacenter networks. In *ISCA*. 2, 4, 5, 10, 11, 14, 15, 27, 28, 39

Agency, U. S. E. P. (2007). Report to congress on server and data center energy efficiency. 1, 10

Al-Fares, M., Loukissas, A., and Vahdat, A. (2008). A scalable, commodity data center network architecture. In *SIGCOMM*. 14

Ananthanarayanan, G. and Katz, R. H. (2008). Greening the switch. In *PACS*. 4, 14

AOL (2006). 500k user session collection. xiii, 74

ATLAS collaboration (2012). Observation of a New Particle in the Search for the Standard Model Higgs boson with the {ATLAS} Detector at the {LHC}. *Physics Letters B*, 716(1):1 – 29. 44

Benson, T., Anand, A., Akella, A., and Zhang, M. (2009). Understanding data center traffic characteristics. In *WREN*. 15

Bloom, B. H. (1970). Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426. 7, 44

Braysy, O. and Gendreau, M. (2005). Vehicle routing problem with time windows, part i: Route construction and local search algorithms. *Transportation Science*, 39(1):104–118. 5, 92, 95, 99

Broder, A. and Mitzenmacher, M. (2003). Network Applications of Bloom Filters: A Survey. *Internet Mathematics*, 1(4):485–509. 7, 42, 44, 45

CAIDA (2013). Internet trace data from caida. xiv, 82

Caione, C., Brunelli, D., and Benini, L. (2012). Distributed compressive sampling for lifetime optimization in dense wireless sensor networks. *Industrial Informatics, IEEE Transactions on*, 8(1):30 –40. 6, 95

Cao, G., Yu, F., and Zhang, B. (2011). Improving network lifetime for wireless sensor network using compressive sensing. In *High Performance Computing and Communications (HPCC), 2011 IEEE 13th International Conference on*, pages 448 – 454. 6, 95, 96

Chen, G., Guo, T.-D., Yang, W.-G., and Zhao, T. (2006). An improved ant-based routing protocol in wireless sensor networks. In *Collaborative Computing: Networking, Applications and Worksharing, 2006. CollaborateCom 2006. International Conference on*, pages 1 –7. 105

Chen, H., Jin, H., Luo, X., Liu, Y., Gu, T., Chen, K., and Ni, L. M. (2012). BloomCast: Efficient And Effective Full-Text Retrieval In Unstructured P2P Networks. *IEEE Transactions on Parallel and Distributed Systems*, 23(2):232–241. 44

Chen, T., Guo, D., He, Y., Chen, H., Liu, X., and Luo, X. (2013). A Bloom Filters Based Dissemination Protocol In Wireless Sensor Networks. *Journal of Ad Hoc Networks*, 11(4):1359–1371. 44

Cheng, C.-B. and Wang, K.-P. (2009). Solving a vehicle routing problem with time windows by a decomposition technique and a genetic algorithm. *Expert Systems with Applications*, 36(4):7758–7763. 6, 95

Chiang, W.-C. and Russell, R. (1996). Simulated annealing metaheuristics for the vehicle routing problem with time windows. *Annals of Operations Research*, 63:3–27. 6, 95

Cisco (2010). Cisco IOS NetFlow. 41, 73

CMS Collaboration (2012). Observation of a New Boson at a Mass of 125 GeV with the {CMS} Experiment at the {LHC}. *Physics Letters B*, 716(1):30 – 61. 44

Cormen, T. H., Stein, C., Rivest, R. L., and Leiserson, C. E. (2001). *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition. 103

Dautrich, J. and Ravishankar, C. (2013). Inferential Time-decaying Bloom Filters. In *Proceedings of the International Conference on Extending Database Technology: Advances in Database Technology*. 45

Debnath, B. K., Sengupta, S., Li, J., Lilja, D. J., and Du, D. H.-C. (2011). BloomFlash: Bloom Filter On Flash-Based Storage. In *Proceedings of the International Conference on Distributed Computing Systems (ICDCS)*. 44

Donnet, B., Gueye, B., and Kaafar, M. A. (2012). Path Similarity Evaluation Using Bloom Filters. *Journal of Computer Networks*, 56(2):858–869. 44

Donoho, D. and Tsaig, Y. (2006). *Fast solution of l1-norm minimization problems when the solution may be sparse*. Department of Statistics, Stanford University. 96

Du, L. and He, R. (2012). Combining nearest neighbor search with tabu search for large-scale vehicle routing problem. *Physics Procedia*, 25(0):1536 – 1546. 6, 95

Eksioglu, B., Vural, A. V., and Reisman, A. (2009). The vehicle routing problem: A taxonomic review. *Computers and Industrial Engineering*, 57(4):1472 – 1483. 5, 92, 94

Estan, C. and Varghese, G. (2002a). New directions in traffic measurement and accounting. *SIGCOMM Comput. Commun. Rev.*, 32(4):323–336. 7, 45

Estan, C. and Varghese, G. (2002b). New Directions in Traffic Measurement and Accounting. In *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, volume 32, page 75. 41

Estan, C., Varghese, G., and Fisk, M. (2006). Bitmap Algorithms for Counting Active Flows on High-Speed Links. *IEEE/ACM Transactions on Networking*, 14(5):925–937. 41

Fan, L., Cao, P., Almeida, J., and Broder, A. Z. (2000). Summary cache: a scalable wide-area web cache sharing protocol. *IEEE/ACM Trans. Netw.*, 8(3):281–293. 7, 42, 44, 73

Greenberg, A., Hamilton, J., Maltz, D. A., and Patel, P. (2008). The cost of a cloud: research problems in data center networks. In *SIGCOMM*. 2, 10

Gunaratne, C., Christensen, K., Nordman, B., and Suen, S. (2008). Reducing the energy consumption of ethernet with adaptive link rate (alr). *Computers, IEEE Transactions on*. 5, 15

Guo, C., Lu, G., Li, D., Wu, H., Zhang, X., Shi, Y., Tian, C., Zhang, Y., and Lu, S. (2009). Bcube: a high performance, server-centric network architecture for modular data centers. In *SIGCOMM*. 14

Guo, C., Wu, H., Tan, K., Shi, L., Zhang, Y., and Lu, S. (2008). Dcell: a scalable and fault-tolerant network structure for data centers. In *SIGCOMM*. 14

Gupta, M., Grover, S., and Singh, S. (2004). A feasibility study for power management in lan switches. In *ICNP*. 4, 11, 14

Gupta, M. and Singh, S. (2003). Greening of the internet. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '03, pages 19–26, New York, NY, USA. ACM. 1, 4, 14

Heller, B., Seetharaman, S., Mahadevan, P., Yiakoumis, Y., Sharma, P., Banerjee, S., and McKeown, N. (2010). Elastictree: Saving energy in data center networks. In *NSDI*. 2, 4, 10, 11, 14, 28, 39

Holland, J. H. (1992). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, Cambridge, MA, USA. 6, 95

Kaser, O. and Lemire, D. (2012). Strongly universal string hashing is fast. *CoRR-arXiv*, abs/1202.4961. 55

Kumar, A., Xu, J., and Wang, J. (2006). Space-code bloom filter for efficient per-flow traffic measurement. *IEEE J.Sel. A. Commun.*, 24(12):2327–2339. 45, 73, 76

Li, M. and Liu, Y. (2009). Underground coal mine monitoring with wireless sensor networks. *ACM Trans. Sen. Netw.*, 5:10:1–10:29. 92

Li, T., Chen, S., and Ling, Y. (2012). Per-Flow Traffic Measurement Through Randomized Counter Sharing. *IEEE/ACM Transactions on Networking*, 20(5):1622–1634. 41

Ling, Q. and Tian, Z. (2010). Decentralized sparse signal recovery for compressive sleeping wireless sensor networks. *Signal Processing, IEEE Transactions on*, 58(7):3816 –3827. 6, 96

Liu, L., Zhang, X., and Ma, H. (2010). Optimal node selection for target localization in wireless camera sensor networks. *Vehicular Technology, IEEE Transactions on*, 59(7):3562 –3576. 92

Liu, W., Qu, W., Liu, Z., Li, K., and Gong, J. (2012). Identifying elephant flows using a reversible multilayer hashed counting bloom filter. In *High Performance Computing and Communication 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on*, pages 246–253. 7, 45

Luo, C., Sun, J., Wu, F., and Chen, C. W. (2009). Compressive data gathering for large-scale wireless sensor networks. In *in Proc. ACM Mobicom '09*, pages 145–156. 6, 95

Mahadevan, P., Sharma, P., Banerjee, S., , and Ranganathan, P. (2009a). A power benchmarking framework for network devices. In *IFIP Networking*. 15, 16

Mahadevan, P., Sharma, P., Banerjee, S., and Ranganathan, P. (2009b). Energy aware network operations. In *IEEE Global Internet Symposium*. 10, 11

Mehrjoo, S., Shanbehzadeh, J., and Pedram, M. (2010). A novel intelligent energy-efficient delay-aware routing in wsn, based on compressive sensing. In *Telecommunications (IST), 2010 5th International Symposium on*, pages 415 –420. 6, 96

MEMSIC, I. (2012). IRIS sensor nodes. 129

Moreira, M., Laufer, R., Velloso, P., and Duarte, O. (2012). Capacity And Robustness Tradeoffs In Bloom Filters For Distributed Applications. *IEEE Transactions on Parallel and Distributed Systems*, 23(12):2219–2230. 44

Nedevschi, S., Popa, L., Iannaccone, G., Ratnasamy, S., and Wetherall, D. (2008). Reducing network energy consumption via sleeping and rate-adaptation. In *NSDI*. 5, 11, 15

NS-3 (2011). Ns-3. Technical report. 93, 94, 110

OPNET Technologies, I. (2010). 13, 31

Ozyurt, Z., Aksen, D., and Aras, N. (2006). Open vehicle routing problem with time deadlines: Solution methods and an application. In Haasis, H.-D., Kopfer, H., and Schnberger, J., editors, *Operations Research Proceedings 2005*, volume 2005 of *Operations Research Proceedings*, pages 73–78. Springer Berlin Heidelberg. 5, 6, 92, 95, 99

Raghavan, B. and Ma, J. (2011). The energy and emergy of the internet. In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, HotNets-X, pages 9:1–9:6, New York, NY, USA. ACM. 1

Robert, G. (1992). A primer in game theory. *FT Prentice Hall Publisher, London*. 68, 69, 71

Rottenstreich, O. and Keslassy, I. (2012). The Bloom Paradox: When Not To Use A Bloom Filter? In *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM)*. 44

Sarela, M., Rothenberg, C. E., Aura, T., Zahemszky, A., Nikander, P., and Ott, J. (2011). Forwarding Anomalies In Bloom Filter-based Multicast. In *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM)*. 44

Sarrar, N., Uhlig, S., Feldmann, A., Sherwood, R., and Huang, X. (2012). Leveraging Zipf S Law For Traffic Offloading. *ACM SIGCOMM Computer Communication Review*, 42(1):16–22. 41, 45

Shen, H. and Zhang, Y. (2008). Improved approximate detection of duplicates for data streams over sliding windows. *J. Comput. Sci. Technol.*, 23(6):973–987. 7, 45

Skiscim, C. C. and Golden, B. L. (1983). Optimization by simulated annealing: A preliminary computational study for the tsp. In *Proceedings of the 15th conference on Winter Simulation - Volume 2*, WSC '83, pages 523–535, Piscataway, NJ, USA. IEEE Press. 6, 95

Solomon, M. M. (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints. *Oper. Res.*, 35(2):254–265. 5, 95, 102

SPEC (2009). Specweb2009 release 1.20 banking workload design document. 33

Tan, K., Lee, L., Zhu, Q., and Ou, K. (2001). Heuristic methods for vehicle routing problem with time windows. *Artificial Intelligence in Engineering*, 15(3):281–295. 6, 93, 95, 102

Tarkoma, S., Rothenberg, C. E., and Lagerspetz, E. (2012). Theory and Practice of Bloom Filters for Distributed Systems. *IEEE Communications Surveys and Tutorials*, 14(1):131–155. 7, 42, 44, 45

Tolle, G., Polastre, J., Szewczyk, R., Culler, D., Turner, N., Tu, K., Burgess, S., Dawson, T., Buonadonna, P., Gay, D., and Hong, W. (2005). A macroscope in the redwoods. In *Proceedings of the 3rd international conference on Embedded networked sensor systems*, SenSys '05, pages 51–63, New York, NY, USA. ACM. 92

Vasudevan, V., Phanishayee, A., Shah, H., Krevat, E., Andersen, D. G., Ganger, G. R., Gibson, G. A., and Mueller, B. (2009). Safe and effective fine-grained tcp retransmissions for datacenter communication. In *SIGCOMM*. 35

Verma, A., Dasgupta, G., Nayak, T., De, P., and Kothari, R. (2009). Server workload analysis for power minimization using consolidation. In *USENIX*. 11, 12

Vicaire, P., He, T., Cao, Q., Yan, T., Zhou, G., Gu, L., Luo, L., Stoleru, R., Stankovic, J. A., and Abdelzaher, T. F. (2009). Achieving long-term surveillance in vigilnet. *ACM Trans. Sen. Netw.*, 5:9:1–9:39. 92

Wang, X., Chen, M., Lefurgy, C., and Keller, T. (2009). Ship: Scalable hierarchical power control for large-scale data centers. In *PACT*. 33

Wang, X., Zhao, Z., Xia, Y., and Zhang, H. (2010). Compressed sensing for efficient random routing in multi-hop wireless sensor networks. In *GLOBECOM Workshops (GC Wkshps), 2010 IEEE*, pages 266 –271. 6, 96, 110

Werner-Allen, G., Lorincz, K., Johnson, J., Lees, J., and Welsh, M. (2006). Fidelity and yield in a volcano monitoring sensor network. In *Proceedings of the 7th symposium on Operating systems design and implementation*, OSDI '06, pages 381–396, Berkeley, CA, USA. USENIX Association. 92

Wu, J. (2009). Ultra-lowpower compressive wireless sensing for distributed wireless networks. In *Military Communications Conference, 2009. MILCOM 2009. IEEE*, pages 1 –7. 6, 95

Xiang, L., Luo, J., and Vasilakos, A. (2011). 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, pages 46 –54. 6, 96

Xu, N., Rangwala, S., Chintalapudi, K. K., Ganesan, D., Broad, A., Govindan, R., and Estrin, D. (2004). A wireless sensor network for structural monitoring. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, SenSys '04, pages 13–24, New York, NY, USA. ACM. 92

Yahoo! (2008). Yahoo! webscope network flows data version 1.0. 19

Yang Wengi, Wendong Xiao, L. X. (2011). Sensor selection for parameterized random field estimation in wireless sensor networks. *Journal of Control Theory and Applications*, 9:44–50. 92

Yao, Y., Cao, Q., and Vasilakos, A. V. (2013). Edal: An energy-efficient, delay-aware, and lifetime-balancing data collection protocol for wireless sensor networks. In *Mobile Ad-Hoc and Sensor Systems (MASS), 2013 IEEE 10th International Conference on*, pages 182–190. 93

Zhao, Q. G., Ogihara, M., Wang, H., and Xu, J. J. (2006). Finding global icebergs over distributed data sets. In *Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '06, pages 298–307, New York, NY, USA. ACM. 7, 45

Zheng, H., Xiao, S., Wang, X., and Tian, X. (2011). On the capacity and delay of data gathering with compressive sensing in wireless sensor networks. In *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE*, pages 1 –5. 6, 96

Zheng, H., Xiao, S., Wang, X., and Tian, X. (2012). Energy and latency analysis for in-network computation with compressive sensing in wireless sensor networks. In *INFOCOM, 2012 Proceedings IEEE*, pages 2811 –2815. 6, 95

Zhu, Y. and Wang, X. (2010). Multi-session data gathering with compressive sensing for large-scale wireless sensor networks. In *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*, pages 1 –5. 6, 96

Vita

Yanjun Yao received the B.S. degree in Software Engineering from Zhejiang University, Hangzhou, Zhejiang, China, in 2006, and the M.S. degrees in Security and Mobile Computing from Helsinki University of Technology, Espoo, Finland, and University of Tartu, Tartu, Estonia, in 2008, and is currently pursuing the Ph.D. degree in Computer Science at University of Tennessee, Knoxville, TN, USA.

Her research interests include energy-efficient network communication protocols, probabilistic data structures for data-aware networking systems, and mobile system implementations.