

**Building more performant large scale networks for the Internet of Things**

A dissertation submitted to the Graduate College

of the University of Cincinnati in partial fulﬁllment of the requirements for the degree of

Doctor of Philosophy

in the Department of Computer Science

of the College of Engineering and Applied Science by

Saibal K Ghosh November, 2022

Committee Chair: Marc Cahay, Ph.D.

Abstract

Inspired by the rise of smaller computing devices and ubiquitous network connec- tivity, this dissertation focuses on the dynamics of extremely large scale computer networks and mechanisms to improve their performance. Traditional networks were not designed to support the billions (and possibly more than a trillion) of de- vices that are expected to be part of the Internet of Things (IoT) and eventually the Internet of Everything (IoE). We look at mechanisms to eﬃciently allocate resources in large networks and show how they can scale with the network size. We tackle the hard problem of power in energy limited networks and show how setting priorities can improve latencies even in huge networks. We devise ways to enhance the mobile computing experience through collaboration and moving the computation away to the network edge. Finally we devise a mechanism to improve caching at the network edge and show that improved caching at the edge can support a vast number of users without sacriﬁcing on the Quality of Service. We propose to extend this work by incorporating mobility for the vast number of smaller devices connecting to the network. Mobility modeling on extremely large scales in computationally very expensive. We propose to use heuristics and machine learning models to improve computation times. Current networks take heavy advantage of the centralized nature of the cloud to compute network traﬃc ﬂows and take routing decisions. As networks scale, this centralized approach would become infeasible. We propose to take a hybrid approach wherein the cloud only acts as an exchange for these decisions and network management is performed by devices in the network itself. Furthermore, since decisions need not be coordinated centrally, we propose to show that this approach can lead to optimum network performance on a massive scale.

i

I, Saibal Ghosh, declare that this dissertation titled, ‘BUILDING MORE PERFORMANT LARGE SCALE NETWORKS FOR THE INTERNET

OF THINGS’ and the work presented in it are my own. I conﬁrm that:

* This work was done wholly or mainly while in candidature for a research degree at this University.
* Where any part of this dissertation proposal has previously been submit- ted for a degree or any other qualiﬁcation at this University or any other institution, this has been clearly stated.
* Where I have consulted the published work of others, this is always clearly attributed.
* Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this dissertation proposal is entirely my own work.
* I have acknowledged all main sources of help.
* Where the dissertation proposal is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.



November 15th, 2O22

*To my parents for everything*

### Acknowledgments

This dissertation has been a long time in the making and there are so many people who have inspired and guided me. My dissertation advisors Dr. Dharma Agrawal and Dr. Marc Cahay are the best that a student could have. They have always been there for me with suggestions and guidance to take on tough problems. My committee members Dr. Raj Bhatnagar, Dr. Yizong Cheng, Dr. Fred Annexstein, Dr. Rui Dai and Dr. Emily Kang have also helped me immensely with their suggestions throughout my research at UC.

A special thank you to my parents Mr. Shibnath Ghosh and Mrs. Rama Ghosh and my family for all their love and support and letting me work on my research during odd hours. While it has been a long and eventful journey, they have been there for me every step of the way. I would also like to thank my high school English teacher Ms. Chanda Chatterjee and one of my ﬁrst colleagues and best friends Mr. Sourav Mukerji for inspiring me to chase my dreams.

My best friends Sushobhan and Nandini who believed in me when I set out on my graduate school journey. They have always been ready with a joke or a witticism when I needed to lift my spirits. I met Sounak, Souvik, Anirban and Andrew in graduate school and while we studied and had fun, they all made me see the world diﬀerently in their own way. I could not have survived graduate school without them.

iii

# Contents

[Acknowledgments iii](#_TOC_250057)

Abstract iv

[List of Figures ix](#_TOC_250056)

[List of Tables xi](#_TOC_250055)

1. Introduction 1
   1. [Organization of this Dissertation 3](#_TOC_250054)
2. Existing research 6
   1. [Resource allocation and utilization in the Internet of Things 7](#_TOC_250053)
   2. [The problem of energy in the Internet of Things 10](#_TOC_250052)
   3. [Improving responsiveness and prioritizing transmissions in the In- ternet of Things 11](#_TOC_250051)
   4. [Reducing network congestion to improve user experiences 13](#_TOC_250050)
   5. A Game theoretic approach to optimize mobility and computing

at the Edge 16

* 1. [Cooperative computing at the Network Edge 19](#_TOC_250049)
  2. [Caching in Mobile Edge Computing Environments 20](#_TOC_250048)
  3. [Cooperative Caching at the Network Edge 23](#_TOC_250047)

1. Allocating and utilizing resources more eﬃciently in the Inter- net of Things 25
   1. [Modeling the System 26](#_TOC_250046)
      1. [CPU workload 27](#_TOC_250045)
      2. [Channel Traﬃc and Bandwidth 28](#_TOC_250044)
      3. [Modeling the access node 28](#_TOC_250043)
      4. [Modeling the computing node 29](#_TOC_250042)
      5. [Energy expenditure 30](#_TOC_250041)
   2. [Formulating the problem 30](#_TOC_250040)
   3. [Simulation 31](#_TOC_250039)
   4. [Results and Analysis 32](#_TOC_250038)

vi

1. Tackling the energy problem in the Internet of Things 39
   1. [Scoping the model to the energy problem 40](#_TOC_250037)
   2. [Simulation 43](#_TOC_250036)
   3. [Observations and Analysis 44](#_TOC_250035)
2. Reducing network congestion by optimizing latencies and pri- orities 51
   1. [Modeling the system 52](#_TOC_250034)
      1. [Latency 52](#_TOC_250033)
         1. [Local computation 52](#_TOC_250032)
         2. [Oﬄoaded computation 52](#_TOC_250031)
      2. [Energy consumption 53](#_TOC_250030)
         1. [Local computation 53](#_TOC_250029)
         2. [Oﬄoaded computation 53](#_TOC_250028)
   2. [Modeling for congestion control 54](#_TOC_250027)
   3. [Simulation and Analysis 55](#_TOC_250026)
3. Optimizing mobility and cooperative computing at the Edge 68
   1. [The Computation Oﬄoading Game 71](#_TOC_250025)
   2. [The computation oﬄoading mechanism 71](#_TOC_250024)
   3. [Simulation results and analysis 72](#_TOC_250023)
4. A foray into caching for mobile edge computing environments 82
   1. [Caching in the Modern Web 83](#_TOC_250022)
   2. [The Hierarchical Caching Mechanism 84](#_TOC_250021)
      1. [Formalizing the caching system 84](#_TOC_250020)
      2. [Hierarchical Ranking 85](#_TOC_250019)
      3. [Convergence 87](#_TOC_250018)
   3. [Modeling the Cooperative Caching Mechanism 87](#_TOC_250017)
      1. [Creating a cooperative caching unit 94](#_TOC_250016)
      2. [Determining edge server capacity 95](#_TOC_250015)
      3. Cache allocation in cooperating edge servers 98
      4. Eﬀective cooperative caching 102
      5. Analyzing cooperation performance 105
   4. Simulation and Analysis 106
   5. Results 106
5. Summary and Future Work 119
   1. [Summary of Results 120](#_TOC_250014)
   2. [Future Work 123](#_TOC_250013)

[Bibliography 126](#_TOC_250012)

[Appendix A: Lyapunov optimization and drift 174](#_TOC_250011)

Appendix B:

Estimating the optimal oﬄoading decision 178

[Appendix C: Cubic Splines 182](#_TOC_250010)

Appendix D:

Suﬃciency for global optimality 186

[Appendix E:](#_TOC_250009)

[Proof of global optimality 190](#_TOC_250008)

[Appendix F:](#_TOC_250007)

[The optimal cooperative strategy is NP-hard 195](#_TOC_250006)

[Appendix G:](#_TOC_250005)

[The relationship between processing delay and the cooperation block 197](#_TOC_250004)

[Appendix H:](#_TOC_250003)

[Tuning for optimal cooperation 199](#_TOC_250002)

[Appendix I:](#_TOC_250001)

[Bounds on allowed cache access latencies 200](#_TOC_250000)

**List of Figures**

* 1. The IoT network with users, edge nodes and the cloud 27
  2. Input data size vs Energy consumption. 33
  3. Channel bandwidth vs Energy consumption 34
  4. CPU utilization vs Energy consumption. 35
  5. Latency threshold vs Energy consumption 35
  6. Input data size vs Execution time. 36
  7. Channel bandwidth vs Execution time 37
  8. CPU utilization vs Execution time. 37
  9. Latency threshold vs Execution time 38
  10. Average task execution delay variation with change of *τ* 45
  11. Average task execution accuracy variation with change of *τ* 46
  12. Average gain in task execution delay with variation in *λ* 48
  13. Average gain in task execution delay with variation in *λ* 48
  14. Average gain in task execution delay with variation in *λ* 49
  15. Network topology used in simulations 56
  16. Variation of total energy consumption with number of devices. 58
  17. Variation of total energy consumption with transmission power. 58
  18. Variation of latency and total energy consumption 59
  19. Variation of total energy consumption with device CPU frequency. 60
  20. Variation of total energy consumption with available channels. 60
  21. Variation of total energy consumption with number of users. 61
  22. Variation of LSTM predictions and actual values 62
  23. Variation of MLP predictions and actual values 63
  24. Diﬀerence between LSTM predictions and actual values 63
  25. Diﬀerence between predictions and actual values 64
  26. Variation of computing cost with required CPU cycles 73
  27. Variation of computing cost with input data sizes. 73
  28. Averaged computing cost 74
  29. Convergence of the oﬄoading mechanism 75
  30. Variation of normalized cost for the routing algorithms under the balanced tree topology 78
  31. Variation of normalized cost for the routing algorithms under the

Fog network topology 78

ix

* 1. Variation of normalized cost for the routing algorithms under the Connected-ER topology 79
  2. Variation of normalized cost for the routing algorithms under the Small World topology without queueing 79
  3. Variation of normalized cost for the routing algorithms under the Small World topology with queueing delay 80
  4. The cooperative caching network 88
  5. Achieving maximal cooperation 103
  6. Read ratios in Local and MEC caches with misses. 107
  7. Write ratios in Local and MEC caches with misses 108
  8. Mean request sizes (KB) in Local and MEC caches 109
  9. Comparison of online and oﬄine policies for reads in Local and

MEC caches with misses. 110

* 1. Comparison of online and oﬄine policies for writes in Local and

MEC caches with misses. 110

* 1. Variation of network performance with available edge servers 112
  2. Variation of latency with available edge servers 113
  3. Variation of network throughput with changes in requested cache

sizes 114

* 1. Variation of latency with changes in requested in cache sizes 114
  2. Variation of network throughput with cache request rate 115
  3. Variation of latency with changes with cache request rate 116
  4. Variation of network throughput with CPU frequencies 117
  5. Variation of latency with CPU frequencies 117

1. Non-monotonic and monotonic splines 182
2. A single cubic polynomial segment 184
3. A set of ﬁve interpolating cubic polynomials 185

# List of Tables

* 1. Simulation Parameters 32
  2. Simulation parameters 57
  3. Simulation Parameters 57

6.1 Simulation parameters 77

* 1. Symbols and Notations 83
  2. Simulation Parameters 106

xi

**Chapter 1**

# Introduction

The last couple of decades have seen unprecedented progress in the ﬁelds of mi- croprocessors and computer networks. It is almost impossible to believe that smartphone microprocessors today are almost as powerful as general purpose processors were at the turn of the century. Highly eﬃcient mobile devices now contribute to the vast majority of consumer computing while datacenter proces- sors have achieved enormous levels of hyperscaling from the density of computing nodes that are present in a modern datacenter. While reduction in transistor gate sizes have slowed, the number of processing cores present in a single pro- cessor has steadily increased. Multi-core processors are now common in even in the most basic smartphones. Recent processor architectures even include hybrid processing cores wherein a fraction of the available cores are focused on perfor- mance while the rest are highly eﬃcient. This hybrid architecture meshes nicely with most consumer computing workloads making these processors even more

1

*Introduction* 2

energy eﬃcient. On the other end of the spectrum, servers now have hundreds of cores supporting a myriad users and applications on the same processor.

Computer networks have also enjoyed considerable improvements in the last couple of decades. Corporate wired networks have given way to high speed wireless connectivity in most work and public places. Ubiquitous connectivity has been steadily improving over the last few years and it is now entirely possible to stream in high deﬁnition over a mobile ﬁfth generation network. It is now inconceivable to have an electronic device without some form of connectivity. Previously benign household devices such as toasters and refrigerators are now connected to the internet and can automatically take limited actions based on certain parameters. Coupled with advances in miniaturization of sensors, it is now more easier than ever to sense the world and our surroundings.

Considering recent improvements in processors and networks, advances in pro- cessing power and ubiquitous connectivity are only going to scale exponentially over the next decades. Current estimates suggest we have billions of devices con- nected to the internet and this is expected to reach more than a trillion by the end of the decade. The term *Internet of Things* and its updated moniker, the *Internet of Everything* was coined to address this unprecedented volume of con- nected entities. While this ubiquitous connectivity has made the world smaller, it has created its own share of problems that traditional computer networks weren’t designed to handle.

The *Internet* and its precursor the *ARPANET* were originally designed to con- nect computers together for collaboration. Reliability of links was prioritized

over network throughput. Links were costly and re-transmissions were expen- sive. The *seven-layered* OSI model and the *TCP/IP* transport layer protocol were designed for reliable communications on the Internet. While this model worked perfectly for decades, we now face a world that has changed dramatically. Computers have become exponentially faster, network speeds have increased dra- matically and the overwhelming majority of traﬃc in the Internet is streaming video and image data. This is very diﬀerent than what the original Internet was designed to handle.

However, the biggest change has been the rise of mobile computing and the sheer number of devices that can now connect to the internet. The original design of the Internet did not envision billions and possibly more than a trillion devices to be connected to the internet. This is evident in the design of the 32-bit IPv4 addressing system on the Internet which allows for a maximum of 232 addresses. While the addressing problem has been almost solved through *NAT* and the expanded IPv6 addressing scheme, we need better mechanisms to support the vast number of devices and rich media experiences.

#### 1.1 Organization of this Dissertation

The reader is introduced to the paradigms of computation oﬄoading, edge com- puting, network congestion and caching and the existing research with the state of the art in these ﬁelds. I have also mentioned how my work diﬀers from the re- search already carried out in this direction and introduced my published research to the reader.

Chapter 3 introduces the reader to a more eﬃcient resource allocation mecha- nism for the Internet of Things. Modern smartphones and devices have greatly increased their computing capabilities. However, they are still limited by their limited energy capacities. We show how intelligently oﬄoading computation to other devices and the cloud can reduce computing times and greatly improve the battery lifetimes.

We look at the energy problem in depth in Chapter 4. Energy limitation is the primary stumbling block to building large mobile networks. We take inspiration from steam engine governors widely used in the 19*th* century and show how the same interpolation principles of cubic splines can be used to optimize the energy envelopes of networks.

Chapter 5 deals with the bane of modern mobile networks - high latency. We show how a hybrid approach of partially oﬄoading certain computations from the devices can result in faster processing times and reduced perceived latencies. Setting priorities determines which tasks are processed at the locally or oﬄoaded to the cloud and helps to improve computation. We also introduce the reader to network congestion at the edge and how it can be mitigated by proactively rerouting traﬃc away from aﬀected nodes. Machine learning models were applied to predict nodes with heavy traﬃc in the future and appropriate routing changes were applied to improve the throughput of the network.

Chapter 6 extends the ideas of Chapter 5 and shows how user experiences in mobile devices can be improved through computation oﬄoading at the network edge. Oﬄoading to the edge ensures even faster processing as compared to the cloud and can be used to support highly interactive mobile applications such as

interactive gaming. Our simulations also show that the mechanism scales well with the size of the network. We have then explored the problem of cooperative computation oﬄoading in multi hop networks. We show that the problem is non-trivial and non-convex and evaluate methods to determine the performance.

Chapter 7 takes a deep dive into the problem of caching at the mobile net- work edge. We propose a scalable *key-value* store and a novel hierarchical *rank- threshold* product caching mechanism and compare its performance with estab- lished caching mechanisms. We show that the unique nature of the cache at the mobile edge works well with our mechanism, minimizing remote writes and converging quickly to an optimal level of caching. This idea is then extended to create a cooperative caching mechanism at the network edge. We formulate a cooperative caching strategy wherein edge servers share their caches and make them available to the mobile devices. We compare the performance of this dis- tributed caching mechanism with established caching mechanisms and show that it performs better.

We ﬁnally conclude this dissertation in Chapter 8 wherein we summarize our results and discuss our future work.

**Chapter 2**

# Existing research

As the number of devices connected to the Internet has grown, the infrastructure that powers its backbone has started to show its age. The Internet was conceived at a time when network speeds were measured in bits per second and the only entities running networked nodes were research institutions. Network speeds have now exceeded terabits per second and the number of devices connected to the Internet range in billions or possibly even a trillion. While the archaic infrastructure was not designed to support this large volume of devices and data rates, it is also infeasible to rebuild the whole infrastructure from scratch. Therefore, a lot of research and eﬀort has been expended to make the Internet of Things work with the existing infrastructure. My dissertation explores several areas wherein we can extend and improve the performance of these devices. This chapter highlights the current research done in this direction and how my work diﬀers from the existing research.

6

#### Resource allocation and utilization in the Inter- net of Things

Analogous to the multi level caches in modern CPUs that balance capacity and access times, current deployments of IoT devices include a number of special computing nodes to accelerate computation. These nodes are present closer to the computing devices and are therefore termed as *edge* nodes to distinguish them from the cloud. These edge nodes are usually more powerful than indi- vidual IoT devices but less than a cloud node. For eﬃciency, edge nodes can dynamically scale their resources based on their perceived computational load [11]. A hierarchy of nodes are generally employed based on their capabilities to further improve the eﬃciency of the whole system. As the computational load increases in individual IoT devices, they can decide to oﬄoad computation to these edge devices. Each edge device can in turn oﬄoad their computation to the next higher layer depending on their current load and the process continues until it reaches the cloud. The crux of this mechanism ensures that an indi- vidual node is never inundated with heavy computation which results in longer wait times and faster depletion of their energy. However, as studies have shown, this oﬄoading comes with its own added latencies and energy trade oﬀs. In order to ensure optimal utilization of system resources, it is important to decide thresholds and allocation of resources before an actual oﬄoad is triggered.

Traﬃc patterns between IoT devices and the various levels of the edge is inher- ently complex and depends on a variety of factors including the current system load, the wireless channel capacity and overall energy envelope of the system.

If IoT devices merely oﬄoad all computation to the edge, the overall energy consumption would increase exponentially and cause huge latencies on the ﬁnite capacity wireless channel from the spurt in traﬃc. However, if devices all perform their allotted computation locally, the total time to complete the computation would increase exponentially as most devices only have modest computing ca- pacities. Modeling the system therefore should involve intelligent handling of local and oﬄoaded computation. The model should also account for the minor but ﬁnite computing overhead of the oﬄoading decision process. As the size of real world deployments grow, this overhead aﬀects the overall eﬃciency of the system. The decision to oﬄoad is based on a continuous evaluation of the prevalent network traﬃc conditions, the current system load and the nature of the computation.

Traditionally, power saving mechanisms in IoT networks simply oﬄoad compu- tation to the cloud. Devices act as portals and computation is performed in the cloud [21, 22]. Improving performance of IoT and edge computing systems have focused on the design of eﬀective computation oﬄoading strategies. Xiao et. al. and Wang et. al minimized the average task duration and service response times under energy constraints by utilizing the alternating direction method of multipliers [12, 13]. Misra et. al. proposed a greedy heuristic scheme for multi hop computation oﬄoading with oﬄoading path selection. [15]. Liu et. al did a study on the social relationships in IoT users to develop a socially aware of- ﬂoading scheme through a game theoretic approach [14]. Lei et. al performed an extensive study on the joint minimization of latency and energy consumption on the overall IoT network as a continuous time Markov decision process and a

solution was developed using dynamic programming [16].

Stochastic optimization techniques to determine oﬄoading strategies have also been employed for this problem. Mao et. al proposed a dynamic oﬄoading scheme for energy harvesting IoT devices [17]. Chen et. al. developed an adap- tive oﬄoading scheme that minimizes the energy consumed in transmission with a guaranteed queuing latency [18]. Gao et. al proposed a socially aware network resource allocation scheme for device to device communications [19]. Zhang et. al proposed an optimal strategy for oﬄoading in energy harvesting devices based on Lyapunov optimization and Vickrey-Clarke-Groves auction to determine re- wards and improve the system [20].

All of these works assume that statistical information on computation tasks is readily available. However, studies have shown that this is quite diﬃcult to achieve in real world deployments [23]. Moreover, these works assume a static two level network wherein IoT devices oﬄoad their computation to the cloud and receive the computed results. A static hierarchy is simple to implement but cannot be tuned for the diﬀerent kinds of computations in the real world.

I have modeled the problems of computation oﬄoading and resource allocation as an optimization problem with the goal of minimizing power consumption across all levels at the edge while keeping the extra latencies incurred as a result of oﬄoading to a minimum. I have shown that the eﬃciency can be increased by increasing the hierarchy in the edge and use time series analysis to predict resource usage. My work on improving the resource allocation and utilization was published in the 2020 IEEE Latin-American Conference on Communications (LATINCOM) [131].

#### The problem of energy in the Internet of Things

Cubic splines [29] are generally used in interpolation, wherein a smooth con- tinuous function can ﬁt discrete data points. This is achieved by interpolating the data with piecewise cubic polynomials. Lower order polynomials reduce the computing complexity and numerical instability that are often associated with higher degree polynomials. This characteristic can be exploited to manage lim- ited resources in an IoT network to improve throughput and extend its lifetime.

Cubic splines have been used to design stable governors for steam engines by Chebyshev back in the 19th century [30, 31]. A variety of industrial real world problems in the ﬁeld of VLSI, CAD/CAM and digital signal processing also make use of splines for approximation. Splines are very useful in computing stress distribution in load bearing structures such as building basements and airplane wings. Insights into load distribution can be utilized to modify the shape of the structure through exponential and cubic spline interpolation [29, 32–36]. Splines can also exploit the properties of Bernstein polynomials to generate shape preserving curves based on the input data [37]. Piecewise rational interpolation can also be used on strictly monotonic and convex data [38, 39]. The reader is encouraged to refer to Appendix C for more details on cubic splines.

In this dissertation, I have investigated various tradeoﬀs between accuracy and delay for computations performed locally and in collaboration with other devices. We form a multi objective function to compute the average energy consumed by the network under various scenarios. We extend this to create an energy optimization framework that can be used to extend the service lifetimes of IoT

devices. My work on energy optimization using predictive cubic splines was pub- lished in the 2020 IEEE International Conference on Sensing, Communication and Networking (SECON) [129]

#### Improving responsiveness and prioritizing trans- missions in the Internet of Things

A lot of research has been undertaken in improving the responsiveness and set- ting priorities for certain packets such that those are delivered faster and do not get lost in the deluge of network packets in the IoT network. Chen et al. modeled the task oﬄoading problem in a MEC system as a stochastic optimiza- tion problem, reducing it to a deterministic optimization problem and proposed a dynamic oﬄoading mechanism to minimize energy consumption [18]. Dai et al. attempted to minimize user associations as a scheme to minimize energy [40]. Cheng et al. minimized wireless resource allocation to minimize energy consumption [41]. Liu et al. combined the two mechanisms and attempted to minimize power consumption at the user level in lieu of utilizing resources for local computation and oﬄoading [42]. Ren et al. minimized the weighted sum la- tency of all mobile devices through a collaboration of the cloud and the edge [43]. Cui et al. investigated the energy consumption and latency minimization prob- lem and formalized it into a constrained multi-objective optimization problem [44]. Kuang et al. took a diﬀerent approach and minimized the weighted sum of the execution delay and energy consumption while maintaining the transmission power envelope through partial oﬄoading [45].

Eﬀects of setting priorities as a potential scheme to solve this problem have also been researched. Paymard et al. proposed a priority based task scheduling policy and jointly optimized the computation and transmission resources [46]. Wu et al. investigated the weighted sum computation eﬃciency optimization problem for partial and binary oﬄoading schemes [47]. Li. et al. developed a heuristic algorithm to jointly optimize the power allocation through a partial oﬄoading scheme [48]. Yang et al. and You et. al. proposed an energy eﬃcient joint oﬄoading and resource allocation optimization method to minimize the energy consumption at the MEC and devices respectively [49, 50].

Existing works have considered the oﬄoading decision as binary. Once an of- ﬂoading decision has been made, the entire computation is oﬄoaded to the edge. Devices run elaborate algorithms to determine if they should oﬄoad. With high computation demand, too many devices choose to oﬄoad and edge servers are inundated reducing overall eﬃciency. However, if most devices perform local computation, computing capacity at the edge is under utilized and the overall eﬃciency of the system reduces.

In this dissertation, I have attempted to address these issues and build a prior- ity oriented oﬄoading scheme. Traditional computation oﬄoading strategies are binary as devices either choose to oﬄoad or perform their computation locally. However, in order to support interactivity and cater to low latency requirements, we relax these strategies and allow tasks to be processed locally and through par- tial oﬄoading in the edge servers. We realize that the optimization variables for oﬄoading decisions and resource allocation are closely related to each other. I

have also considered partial computation oﬄoading wherein parts of the com- putation can be performed both locally and at the edge. I have attempted to minimize the total energy consumption of the system while maintaining the low latency requirements for mobile networks. My work on prioritizing computation oﬄoading and resource utilization was published in the 2021 IEEE 7*th* World Forum on Internet of Things (WF-IoT) [133].

#### Reducing network congestion to improve user ex- periences

Network congestion control is one of the most important facets of traﬃc en- gineering. While many solutions have been proposed both at the application layer and the network protocol level, the usage of machine learning algorithms to reduce network congestion have recently begun to gain prevalence. Machine learning models can be used to predict network congestion from past network traﬃc data. Therefore these models can be used proactively and generally per- form better than classical congestion detection and avoidance mechanisms which are only reactive.

Neural networks have been used to detect congestion in wireless sensor networks [51]. The authors used random traﬃc and used the well known network simulator *ns-2* to create their dataset [52, 53]. The following features were used as inputs to the machine learning algorithm:

* + - Number of network nodes.
    - Traﬃc data rate.
    - Buﬀer utilization.

The neural network had a 3 − 10 − 10 − 1 structure and was able to predict low, medium or high levels of network congestion.

Similar to the work described above, a pair of congestion detection classiﬁers were also used to predict levels of network congestion [54]. The network simulator *ns-*

*2* was also used to generate the dataset and predict three levels of network congestion. The authors compared the performance of a multi layer perceptron and a decision tree for regression tasks (M5 model tree) [55]. They showed that the decision tree trained faster and outperformed the multi layer perceptron with respect to accuracy and positivity rates.

Support Vector Machines have also been used to predict network congestion levels [56]. Lib-SVM and Sequential Minimal Optimization were used for the predictions with Radial Basis Function as the kernel. In order to improve the classiﬁcation accuracy the authors tweaked various parameters to ﬁnd the best values. However, the model failed to perform better than the M5 decision tree mechanism mentioned before.

Time-series analysis have also been used in network congestion control that uses clustering and classiﬁcation techniques [57]. A number of network traﬃc patterns were generated and the relevant data was selected and preprocessed and saved in a database. Various machine learning classiﬁcation algorithms were then used with the data generated earlier as the training set to predict traﬃc ﬂow rates in the future. Network throughput was improved by applying countermeasures.

Low-rate denial of service attacks are also a type of network congestion and work has been done to improve the detection criteria and mitigate such attacks [58]. Since these types of attacks exploit the TCP congestion-control mechanism to bottleneck the target resources, the authors have used the power spectral den- sity entropy function to reduce the calculations required for classifying network traﬃc ﬂows. Network traﬃc ﬂows are classiﬁed as an attack if they are above a certain threshold determined through empirical analysis of the traﬃc ﬂow in the particular network. A support vector machine was used to classify uncertain connections between the deﬁned thresholds using eight classifying features. A very high level of accuracy in detecting these attacks in the dataset was achieved from the experimental results.

These works have utilized machine learning on smaller regular networks and the results have been very good. I have used machine learning algorithms to deter- mine network congestion at the network edge for larger networks with millions of devices. I have compared the performance of three machine learning algorithms namely Autoregressive Integrated Moving Average (ARIMA), Long Short-Term Memory (LSTM) and Multi Layer Perceptrons (MLP) and have shown that LSTM outperforms the others. My work on improving user experiences have been published in the 2020 IEEE Latin-American Conference on Communica- tions (LATINCOM) [130].

#### A Game theoretic approach to optimize mobility and computing at the Edge

Mobile Edge Computing has become prevalent in the last few years as a mecha- nism to provide substantial computing power with ultra low latencies to support full-on interactivity in mobile apps [59, 60]. This involves the deployment of a cloud computing platform at the edge of the radio access network typically in close proximity to the users and their devices thereby eliminating the time required for a round trip to the datacenter. Datacenters are only located in certain specialized places depending on power and network availability and this puts them at a huge disadvantage when latencies are critical. The edge is a scaled down version of the services and resources available at the cloud that is part of a mobile base station or in an access point in close proximity to the base station. Computing demands from users are served by these edge servers if pos- sible and reduces the perceived latency signiﬁcantly [61]. An extreme example of the sudden surges in computation can be gleaned from an urban mass transit system wherein thousands of commuters use their mobile devices to catch up on the news and other apps while on the go. As the conveyance moves, the whole set of mobile users on board that conveyance moves from one base station to another putting enormous pressure to ensure successful handovers for all these devices and ensuring that the latencies do not drop signiﬁcantly [62].

The overall latency as perceived by a user is a combination of the computing and the communication latencies [63, 64]. This suggests ensuring that the edge nodes handling a particular user should always be the ones nearest and are expected to

provide the lowest latency. However, in a dense urban environment, this would also mean overloading some nodes that are in proximity to most users while the rest of the edge computing capability is under utilized. Again, if users are highly mobile, frequent migration of the servicing edge node would lead to higher costs arising from increased bandwidth usage and energy expended in moving the user proﬁle between edge nodes [65]. Therefore, the goal of service localization in cloud computing should involve improving resource utilization and reducing costs through consolidation of resources, utilizing service speciﬁc hardware if available to accelerate computation and eﬃciently distributing services on the edge nodes to balance the overall network load [66–68].

An useful mechanism to facilitate mobile device handoﬀs would be to predict movements for users. Capacity can then be preallocated at the edge nodes that would be part of that path. Nadembega et al. proposed handling the trade-oﬀ between computation overhead and latency, with a mobility based prediction scheme estimating the data throughput, handover time and edge node migra- tion by computing the paths in advance [69]. Wang et al. improved the service migration times by predicting the future cost incurred in communication, com- putation and handovers [70]. Aissioui et al. proposed a framework to capture the trade-oﬀ between reducing service migration and maintaining an acceptable latency based on autonomous vehicle mobility patterns [71].

A lot of work has been carried out for eﬃcient computation oﬄoading designs in mobile users. Rudenko et al. showed signiﬁcant energy savings form compu- tation oﬄoading [72]. Huertacanepa developed an adaptive oﬄoading algorithm

for oﬄoading based on historical application execution data and the current sys- tem state [73]. Xian et al. increased the energy eﬃciency of mobile devices by introducing an eﬃcient timeout scheme for computation oﬄoading [74]. Rahimi et al. proposed a two tier architecture to improve performance and scalability in mobile edge computing [75]. Huang et al. proposed a Lyapunov optimization based dynamic oﬄoading algorithm to strike a balance between performance and latency [76]. Wolski et al. proposed a prediction based decision making framework to determine the cutoﬀ for local and oﬄoaded execution based on performance [77]. Wen et al. proposed an eﬃcient oﬄoading policy by conﬁg- uring the clock frequency of the mobile device to the transmission schedules to minimize energy consumption [78].

A few works have addressed the problem of computation oﬄoading for a multi user scenario. Yang et al. considered a scenario wherein multiple users shared the wireless network bandwidth and employed a centralized heuristic genetic algorithm to maximize performance [79]. Rahimi et al. employed a central- ized greedy scheme to solve the computation oﬄoading problem with multiple users considering user mobility information [80]. Barbarossa et al. proposed a centralized scheduling algorithm to jointly optimize the communication and com- putation resource allocations between multiple users with an acceptable latency [81].

However, all these works have either considered the problem from a single user’s perspective which is hardly the case in a real world scenario or require that all participating mobile devices transmit their capabilities in advance to the cloud and let the cloud determine the oﬄoading schedule accordingly. If these

limitations are all taken into account, predicting user mobility in the future for real world environments from a central vantage point is NP-hard. Therefore, in this work, we let each mobile device make the oﬄoading decision locally using a game theoretic approach. Moreover, removing the necessity that the cloud needs to take part in the oﬄoading decision process frees up valuable resources further improving the eﬃciency. My work on mobility and edge computing was published in the 2021 IEEE 7*th* World Forum on Internet of Things (WF-IoT) [134].

#### Cooperative computing at the Network Edge

The availability of commodity hardware in large volumes have made computing nearly ubiquitous. A number of these modestly powered computing devices can tackle larger problems while working in tandem. Recently a lot of work has been done on multi-hop computation oﬄoading. A possible solution to the energy minimization problem was proposed by the authors for full task oﬄoading [82]. An iterative algorithm for task assignment to devices in a multi hop network has also been studied although it does not consider partial oﬄoading [83]. A joint formulation for computation oﬄoading and routing for both divisible and indivisible tasks was solved using a game theoretic approach by showing the existence of a Nash equilibrium [84]. A similar work also considered a game- theoretic solution for the multi-hop computation oﬄoading problem considering local, edge and cloud computing for computation tasks [85].

Multi-user partial computation oﬄoading for a single hop was studied consider- ing the edge and the cloud to design an iterative heuristic algorithm and make dynamic oﬄoading decisions [86]. A study was also performed on the partial com- putation oﬄoading problem wherein mobile devices can partially oﬄoad tasks to single or multiple cloud servers to optimize the latency and energy consumption of applications [87]. The task allocation problem for multi-hop Wireless Sensor Networks have also been considered [88, 89]. More work has been performed on commercial cellular networks to solve the computational peer oﬄoading prob- lem between small base stations for scheduling tasks [90]. A task scheduling framework was proposed that utilizes the underlying network scheduler to make task placement decisions [91]. Another work has provided a solution to the joint reducer placement and the bandwidth scheduling problem [92]. The works on task scheduling under multi-hop computation oﬄoading have considered the underlying network conditions to make scheduling decisions [93, 94].

#### Caching in Mobile Edge Computing Environments

Caching at the edge has been an interesting topic of research mainly because optimized caching strategies can dramatically reduced network operating costs, improve latency and the Quality of Service. Bastug et al. in an early work proposed a big data framework for mobile network optimization incorporating features from network and users [95]. However, given the computing capacities and storage availability at the edge, the amount of cached data is quite insignif- icant resulting in a low cache hit ratio. Tran et al. highlighted this problem in their work and proposed a cooperative caching method resulting in low latency

content delivery [96]. Zeydan et al. studied the correlation between data and

caching in wireless *fifth* − *generation* networks and applied statistical machine learning to estimate content popularity [97].

Research has also been carried out towards jointly optimizing caching and com- putation. Fan et al. combined caching and computation at the base stations for improving latencies between devices and the cloud [98]. Their resource man- agement helps base stations to jointly schedule computation oﬄoading and data caching. Lee et al. used the idea of the online secretary framework to distribute computing tasks between the edge and the cloud [99]. Tran et al. formulated the joint caching and processing problem as an optimization problem that aims to minimize the backhaul network cost subject to the cache and processing capacity [100].

In order to signiﬁcantly reduce redundant data transmissions and improve con- tent delivery Jiang et al. proposed an optimal cooperative content caching and delivery policy in which both the Femtocell base stations and the user devices cache the content [101]. Hsu et al. proposed a collaborative framework that leverages device to device communications for content caching [102]. Tan et al. formulated a joint optimization problem that maximizes the system capacity with emphasis on bandwidth allocation [103].

Zhou et al. proposed a system involving all virtualized resources wherein com- putation, communication and caching are shared between all users irrespective of their service providers. Wang et al. proposed a computation oﬄoading deci- sion, resource allocation and data caching framework as an optimization problem

wherein the total revenue of the network is considered [104]. Huo et al. pro- posed an energy-eﬃcient framework that jointly optimizes networking, caching and computing for next generation green wireless networks [105]. Chakareski et al. explored the fundamental tradeoﬀs between caching, computing and commu- nication for virtual and augmented reality applications [106]. Cui et al. proposed a joint caching and oﬄoading mechanism that considers task upload and execu- tion, computation output, multi-user diversity and multi-casting [107].

From our extensive study of the current state of the art in caching in MEC, most works consider some format or level of edge caching that is reminiscent of traditional caching mechanisms in computer systems. However, edge nodes are almost always limited in resources when compared to the capabilities of the cloud. This can often result in a low cache hit ratio unless some mechanism is formulated for cooperation amongst multiple edge nodes. While some works have proposed the idea of edge node cooperation as outlined earlier, they do not provide any rigorous framework for analyzing the formation and performance gains achieved in forming and maintaining such collaboration. Moreover, most mobile applications are highly sensitive to latency as most users are turned oﬀ by slow and poorly performing apps. Most works also do not consider compu- tation deadlines for caching strategies and assume the total storage capacity of the system as uniformly available. This is very important in real world net- works as data access times can vary widely depending on the location. In my

dissertation I have modeled a *key* − *value* store for caching and formulated a

hierarchical, size and frequency aware content placement/eviction policy for our cache architecture. I have performed extensive simulations with real world 5*G*

workload traces and prove that our mechanism is scalable and can be tweaked for performance. My work on improving the caching performance in Mobile Edge Computing environments was published in the 2021 IEEE Wireless Communi- cations and Networking Conference (WCNC) [132].

#### Cooperative Caching at the Network Edge

Caching has been at the forefront of research in computer science since the beginning and still remains one of the most important areas even today [108, 109]. While caching was ﬁrst used to optimize storage in the computer’s primary memory, the same principles are used nowadays to improve the performance of the world wide web, content distribution networks and edge caching [110–113].

Studies have shown that traditional caching mechanisms such as LFU and LRU suﬀer from degraded performance when cache requests follow a *time-varying* probability distribution [114–116]. A way to alleviate this problem was to model non-stationary cache request patterns and optimize the caching decisions accord- ingly [117–120].

In order to improve the estimate of cache request probabilities, reinforcement learning has been used [121, 122]. However, owing to the nature of this technique, neither do they scale with the size of the available cache size nor do they provide any bounds on optimality. Online gradient descent has also been used as the caching policy for a single-cache system along with sub-modular policies such as online mirror-descent [123–128]. A comparison of these online learning-based

caching policies have shown that they can perform well with minimal training data and scale well to extremely large input data sets.

**Chapter 3**

# Allocating and utilizing resources more eﬃciently in the Internet of Things

One of the greatest drivers behind the stellar growth of Internet of Things (IoT) in the last couple of decades have been their minimal energy requirements. While this low energy requirements have made them near ubiquitous, it is also a huge bottleneck in real world deployments. A ﬁne balance needs to be achieved such that these devices can perform enough meaningful computation in a reason- able amount of time to justify their deployment while consuming a reasonable amount of power to keep them running for extended periods. A common method to achieve this is through computational oﬄoading wherein these devices move heavy computation away to the edge and the cloud. This drains less power in the

devices and increases the overall network lifetime. Oﬄoading computation from 25

these devices also improves their performance as the computations are performed by powerful hardware at the edge and the cloud. However, this also increases power consumption and network latencies from the need to transmit and re- ceive extra synchronization data in addition to the compute data. Optimum performance can be achieved by intelligently deciding if the computation should be oﬄoaded or performed locally - a task that is inherently diﬃcult given the transient nature of the network and the systems. In this chapter, we formulate a distributed network optimization problem to achieve a balance between per- formance and energy consumption. Simulation results show substantial network latency reduction and reduced overall power consumption even with the added overhead of a multi-tier edge network.

#### Modeling the System

We consider an IoT network consisting of a number of smaller computing devices, a number of edge computing nodes and the cloud as shown in Figure 3.1. The edge consists of nodes deployed at the network boundary oﬀering high speed wireless access links to the IoT devices and more powerful nodes. The computing capacity of these nodes generally exceed the capacity of the nodes connected to the IoT devices. Each IoT device can connect to at least one of these access nodes. The node that the device chooses to oﬄoad is evaluated based on the network load and latencies. Each access node can connect to at least of the more powerful nodes and if the computation exceeds the capacity of these computing nodes, they can oﬄoad to the cloud. Computing nodes access the cloud over high capacity wired links. For our simulations, we model the energy consumption and

perceived latencies in the access and computing nodes. The energy consumption consists of the power consumed in processing and transmission. The perceived latency is a combination of the queueing latency, processing latency and the transmission latency. In our simulations we consider each work unit to complete processing within its allotted time slot.

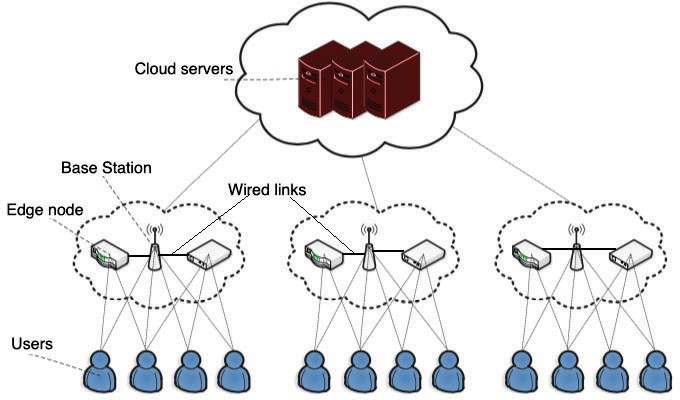


Figure 3.1: The IoT network with users, edge nodes and the cloud.

We now provide a brief description of the factors involved in modeling the system in the following sections.

##### CPU workload

We use the exponential moving average to compute the average CPU utilization [24]. For a given time period *t*, the CPU utilization *Ct* is given by:

*Ct* = *α* · *ct* + (1 − *α*) · *Ct*−1 (3.1)

where, *α* = 2 and *ct* is the instantaneous CPU utilization at time *t*, *α* is the weighted decreasing coeﬃcient and *N* is the time period.

*N* +1

##### Channel Traﬃc and Bandwidth

Similar to the CPU workload computation, we use the exponential moving av- erage metric to compute the traﬃc before making an computation oﬄoading decision. For a given time period *t*, the bandwidth *Bt* is given by:

*Bt* = *β* · *bt* + (1 − *β*) · *Bt*−1 (3.2)

where, *β* = 2

*N* +1

and *bt* is the instantaneous bandwidth at time *t*, *β* is the

weighted decreasing coeﬃcient and *N* denotes the total number of time periods in the simulation.

##### Modeling the access node

We consider that during time slot *t*, there is a ﬁnite amount of computation *C*)*i*(*t*) generated from IoT devices and need to be processed at the access node *i*. The arrival rate of these workloads are diﬀerent for diﬀerent access nodes. Each access node employs time series forecasting methods to estimate the arrival rate of computation based on the previous work load and generate an estimate window *Wi*.

We consider each access node to maintain four memory buﬀers:

* + - 1. Input buﬀer for incoming computations *Ci,*−1(*t*)
      2. Computation estimation buﬀer *Ci,*0(*t*)*, ..., Ci,Wi*−1(*t*)
      3. Processing buﬀer for local computations *P* (*a,p*)(*t*) and

*i*

* + - 1. Computation oﬄoading buﬀer *O*(*a,o*)(*t*)

*i*

##### Modeling the computing node

We consider each computing node *j* ∈ *M* to maintain three memory buﬀers:

* + - 1. Input buﬀer for incoming computations *I*(*c,i*)(*t*)

*j*

* + - 1. Processing buﬀer for local computation *P* (*c,a*)(*t*) and

*j*

* + - 1. Oﬄoading buﬀer for computations for the cloud *Oc,o*(*t*)

*j*

Computational workloads arriving at computing node *j* consist of workloads that have been oﬄoaded from the set of access nodes *Nj*. We denote the amount of computation assigned to the processing buﬀer as *ρ*(*c,i*)(*t*) and that to the

*j*

oﬄoading buﬀer *ρ*(*c,o*)*t* such that:

*j*

0 ≤ *ρc,β*(*t*) ≤ *ρ*(*c,β*) *,* ∀*β* ∈ {*p, o*} (3.3)

*j*

*j,max*

where *ρ*(*c,β*)

*j,max*

≥ 0. Therefore, we have:

*Oc,i*(*t* + 1) ≤ (*Oc,i*(*t*) − (*ρc,p*(*t*) + *ρc,o*(*t*)) + Σ *Ui,j*(*t*)) (3.4)

*j*

*i*

*j*

*j*

*j*∈*Nj*

##### Energy expenditure

The total energy *E*(*t*) consumed by the system in time slot *t* is a combination of the energy expended in processing and transmission of the computational workloads. For a processor running at frequency *f* , the power consumption is denoted as:

*f* = *τ*0*σf* 3

where, *σ* depends on the actual physical hardware of the processor [26]. Therefore, we have:

*E*(*t*) *E*ˆ(*f* (*t*)*, p*(*t*)) =

Σ *τ*0*σ*(*f* (*e*)(*t*))3 + Σ *τ*0*σ*(*f* (*c*)(*t*))3 + Σ Σ *τ*0*pi,j*(*t*)

*i*

*j*

*i*∈*N*

*j*∈*M*

*i*∈*N j*∈*Mi*

(3.5)

where, *f* (*t*) ((*f* (*a*))(*t*))*i*∈*N ,* ((*f* (*c*))(*t*))*j*∈*M* is the set of all processor frequencies

*i*

*j*

in the system, and *p*(*t*) (*p*)*i*(*t*))*i*∈*N* .

#### Formulating the problem

The overall power consumption *P*ˆ is deﬁned as:

*P*ˆ lim

*T* −1

*sup E*{*P* (*t*)} (3.6)

Σ1

*T* →∞

*T*

*t*=0

and the overall buﬀer backlog as:

*T* −1

Σ1

Σ Σ

(*a,β*)

*B*ˆ lim

*T* →∞

*sup*

*T*

*t*=0 *β*∈{*a,p,o*}

*i*∈*N*

*E*{*Bi* (*t*)}+

(3.7)

Σ *E*{*B*(*c,β*)(*t*)}

*j*

*j*∈*M*

Our objective is to minimize the overall power consumption *P*ˆ while ensuring that all computations proceed to completion.

In each time slot *t*, each node *k* ∈ *N* ∪ *M* determines the amount of computation

to be performed locally and oﬄoaded denoted by *ρ*(*α,a*)(*t*) and *ρ*(*α,o*)(*t*) respec-

*k k*

tively wherein *α* = {*a, c*} is the type of the node (access or computing) in the system. The decision to oﬄoad is determined by:

*min*

(*α,β*)

(*α,β*)

0≤*ρk* ≤*ρk,max*

(*O*(*α,β*)(*t*) − *O*(*α,a*)(*t*))*ρ*(*α,β*)

wherein, *β* = {*a, o*} and the optimal solution is given as:

*k*

*k*

*k*

⎧⎪ (*α,β*) (*α,β*) (*α,a*)

*k*

*ρ*(*α,β*)(*t*) =

#### Simulation

⎨*ρk,max if Ok* (*t*) *< Ok* (*t*)

⎪⎩0 *otherwise*

(3.8)

We have used a WSNet-based simulator written in C/Modern C++ [265]. WS- Net implements the Physical and MAC layers and can simulate large networks

with high accuracy. We have extended the simulator in the the spectrum us- age and orthogonal models to reduce wireless interference. The *spectrum usage* model represents the time-frequency dependent aspects of communication. It can be used to exploit the available spectrum in terms of spectral resources rather than the traditional logical channel approach. This feature in WSNet allows us to support the heterogeneity in the Physical layer arising from the various makes and models of devices that are more representative of a real world wireless network.

A list of the simulation parameters are shown in 3.1. We have varied the simu- lation parameters to determine if our mechanism undertook the correct decision to oﬄoad the computation. During our simulation, the system had to determine if further oﬄoading should provide a tangible beneﬁt or the computation should proceed locally. We have analysed the eﬀects of these parameters to determine the viability of our scheme. An explanation and discussion of our results follows

in the next section.

Table 3.1: Simulation Parameters

|  |  |
| --- | --- |
| **Parameter** | ***Value*** |
| Input data size | 0 - 1500KB |
| Channel bandwidth | 0 - 500 KB/sec |
| CPU utilization | 10 - 70% |
| Latency threshold | 10 - 30 secs |

#### Results and Analysis

We have compared the variation of the the input data size, channel bandwidth, CPU utilization and the latency threshold with respect to the overall energy

consumption and overall execution time. Each simulation was run for a mil- lion iterations and the results were averaged to remove any transient noise and wireless channel eﬀects.

Figure 3.2 shows that it costs more energy to perform the computations locally, as compared to oﬄoading to the cloud and the cost keeps on increasing as the size of the input data set grows. This is because the energy expended in transmitting the data is less than computation and the local energy consumption increases faster than that on the cloud and our mechanism oﬄoaded the computation.

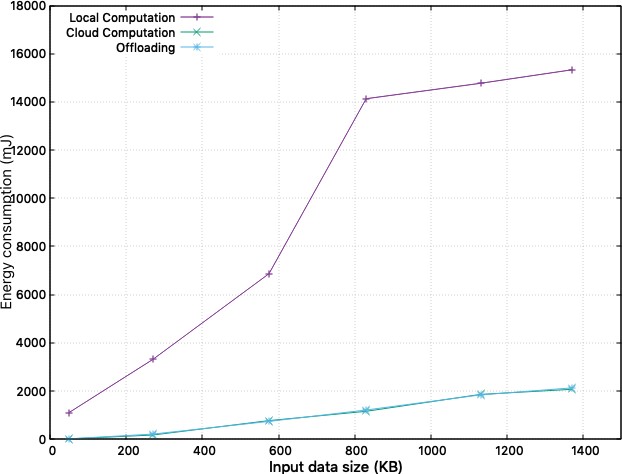


Figure 3.2: Input data size vs Energy consumption.

Figure 3.3 shows that computations use less energy locally when the bandwidth is less but changes as the bandwidth requirements grow larger. This is because, while the power used for transmission remains constant, the eﬀect is much more

pronounced in lower bandwidth ranges. Our mechanism oﬄoads the computation as the bandwidth requirements grow larger.



Figure 3.3: Channel bandwidth vs Energy consumption.

Figure 3.4 shows that the utilization of CPU is less on the cloud as compared to local execution since the weaker IoT devices expend more energy to compute the same amount of work as compared to the cloud and our mechanism oﬄoads the computation.

The latency threshold data in Figure 3.5 shows that it is more economical to run the computations locally when the latency threshold is low since the extra delays incurred in transmitting the data adds up to the overall latency. Our mechanism correctly oﬄoaded the computation once it reached the threshold.

Figure 3.6 shows that the execution time grows much faster with increasing data sizes when the computation is performed locally. This is due to the weaker CPUs

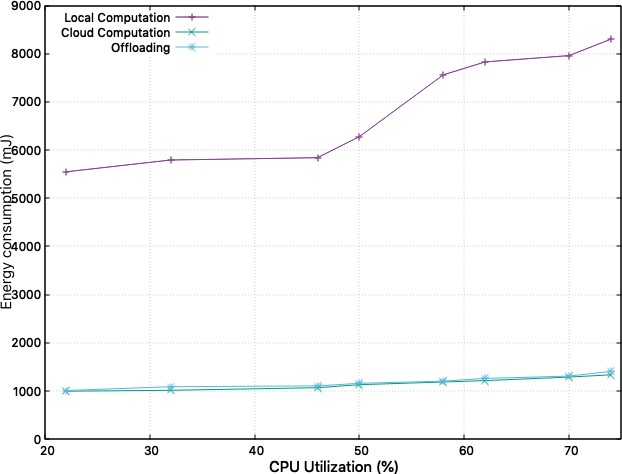


Figure 3.4: CPU utilization vs Energy consumption.



Figure 3.5: Latency threshold vs Energy consumption.

in the IoT nodes and our mechanism oﬄoads the computation.

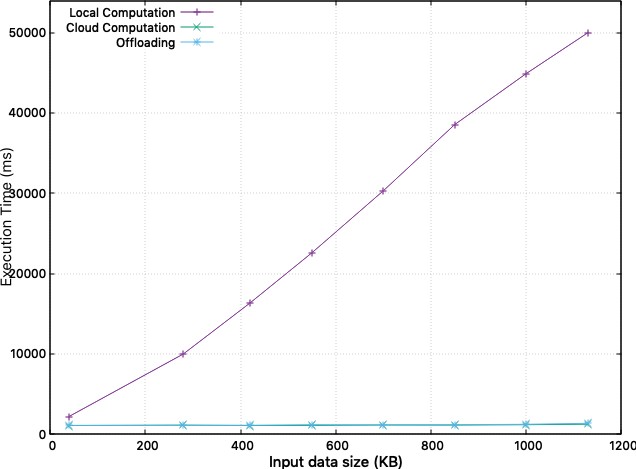


Figure 3.6: Input data size vs Execution time.

Figure 3.7 shows that as the bandwidth increases the execution time decreases for oﬄoaded computations. This is because the transmission overhead is higher in lower bandwidth scenarios. Our mechanism correctly decides to oﬄoad the computation.

The processor utilization has little impact on the overall execution time when the computation is performed on the cloud. This is because of more powerful hardware on the cloud. This scenario is shown in Figure 3.8 and our mechanism oﬄoads the computation.

The latency threshold data in Figure 3.9 is interesting because it shows that oﬄoading the computation reduces the execution time once the threshold exceeds a speciﬁed value and we validate this with our simulations.

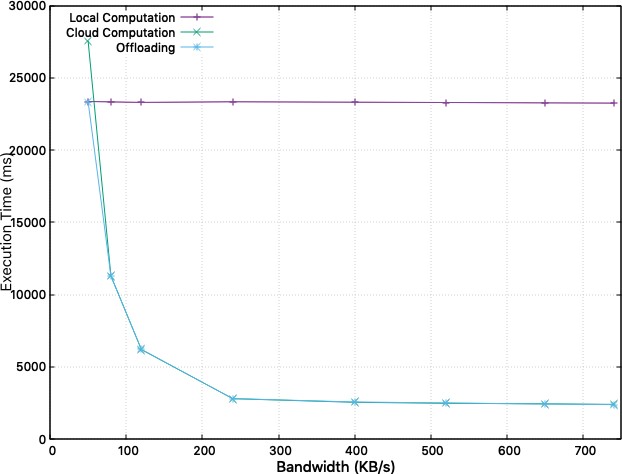


Figure 3.7: Channel bandwidth vs Execution time.

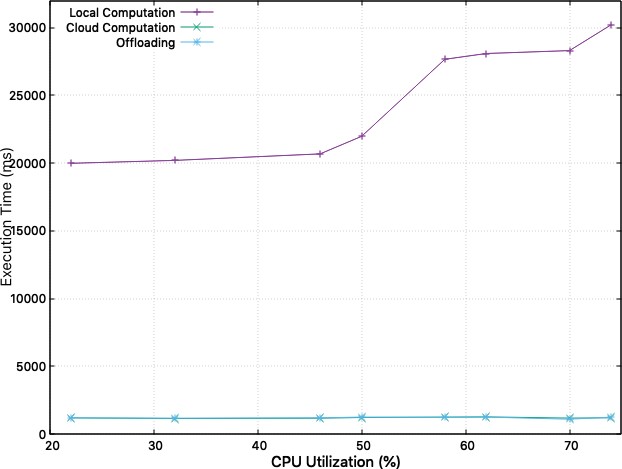


Figure 3.8: CPU utilization vs Execution time.

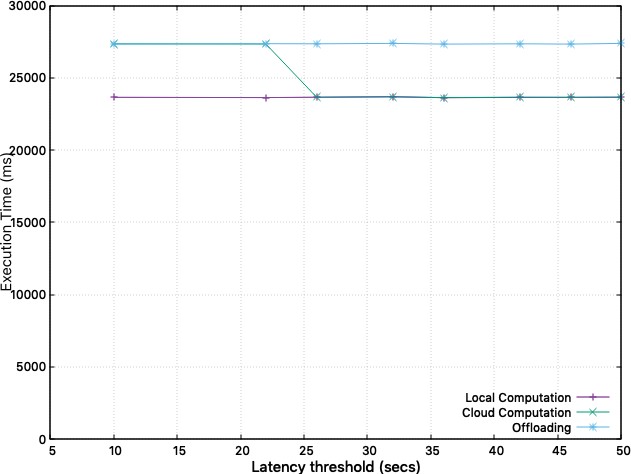


Figure 3.9: Latency threshold vs Execution time.

In this chapter we have explored the problem of computation oﬄoading and formulated an optimization problem and showed that by estimating requirements we can achieve reduced power consumption and latency. In the next chapter we would take another look at optimizing energy consumption in IoT networks using cubic splines - a mechanism that was originally developed to control governors in steam engines in Victorian England!

**Chapter 4**

# Tackling the energy problem in the Internet of Things

One of the major bottlenecks in the widespread deployment of IoT devices is their limited energy supply. With the exception of a very limited type of devices that can use renewable sources of energy such as solar and ambient vibrations, most IoT networks are battery powered which puts an absolute limit to their expected lifetime. Due to the limited computing and communication capabil- ities of these devices, they must often work in tandem with others to provide meaningful information in a reasonable time frame. During the course of normal operations, computing and communication workloads are generally within pre- determined energy envelopes. However, random events occurring in the network cause devices to react and suddenly start performing heavier computations and increase in network traﬃc pushing them beyond these limits. This is a very com- mon phenomenon known as the *Thundering Herd* that causes the well known

39

energy hole problem in networks with limited energy[150].

Current cloud workload distribution frameworks are capable of splitting tasks and oﬄoading the same back to the cloud [151, 152]. However, oﬄoading such computation introduces latencies, making the method unsuitable for real-time applications. This problem can be alleviated to an extent through the use of cloudlets that bring the server infrastructure to the network edge reducing the delay between IoT devices [153–155]. However, in order for the cloudlet infras- tructure to work properly, the system has to be carefully conﬁgured and deployed for a speciﬁc task. Other frameworks such as Misco and CWC have implemented parallel computation on mobile devices on the Map-Reduce paradigm while Mo- biStreams and Swing use the data stream paradigm [156–160]. But none of these systems have any mechanism for optimizing the distribution of tasks and thus do not take into account the energy consumption and execution accuracy.

In this chapter, we investigate various tradeoﬀs between accuracy and delay for computations performed locally and in collaboration with other devices. We form a multi objective function to compute the average energy consumed by the network under the scenarios. We extend this to create an energy optimization framework that can be used to extend the service lifetimes of IoT devices.

#### Scoping the model to the energy problem

In this chapter, we have attempted to extend the lifetime of the IoT network by optimizing the energy consumption of each individual node. A combination

of techniques such as resource coalescing to reduce the number of active origi- nating requests and predictive cubic splines to predict the surge in bandwidth requirements are used to minimize the impact on power consumption. Further- more, we have attempted to minimize the overall execution delay and optimize the execution accuracy of each node performing the computations.

The heterogeneous property of IoT networks makes it prudent to redistribute computing tasks over a number of devices in order to improve the overall through- put of the task. A task assigned to a particular node may be overwhelming for that node based on its computing capacity and power budget. Furthermore, this node may have knowledge of its neighbors that are more computationally capa- ble and therefore better suited for the particular task. Therefore, moving the computation away to another neighbor may prove useful to improve the accuracy and throughput for that particular task. We have to take into account additional delay involved in moving tasks from the originally assigned node to one of its neighbors. However, as our results show, an increased processing capability of the neighbor node makes up for the delays incurred in determining a neighbor and moving the task to that particular node.

For tasks that require a lot of intermediate communication, moving these tasks to nodes that have wide bandwidth channels might seem to be a good idea. However, if those nodes are not computationally capable, then it would have been better to let the original nodes perform the tasks. Therefore, it is sometimes impossible to improve one metric without sacriﬁcing the other. However, for the vast majority of calculations, we can use statistical methods to reduce the amount of computation and extrapolate results. In such circumstances, another

metric that measures the importance of the delay versus the accuracy can be used to determine the execution ﬂow of tasks. Each node learns about the nature of computation as a task progresses and can then route tasks appropriately to nodes that are best suited for a particular computation. Certain computations require a number of nodes to collaborate for completing the task. We use predictive cubic splines to determine such sudden changes in bandwidth requirements and select the optimum group of nodes to perform the tasks that minimizes consumed energy and maximizes the overall IoT network lifetime.

The task execution delay *dc* is denoted by:

*j*

*dc* =

*j*

*h*

*j*

*pj*

1

Σ*k*∈*Vj xk,jλk*

−

*,* ∀*j* ∈ *V* (4.1)

Therefore, the average computational accuracy of each node is denoted by:

*Wi*(*xi*) = *xi,jwi,j,* ∀*i* ∈ *V* (4.2)

Σ

*j*∈*Vi*

In order to solve for this multi objective problem, we use a convex utility ap- proach [166]. The crux of the idea lies in the fact that in order for the system to achieve a meaningful objective, we need to capture the diminishing returns eﬀect inherent in optimizing multi objective systems. In a similar manner, if the average delay in transmission is already high, then minimizing the delay is more important since a minute increase in the current delay would have a higher negative impact than if the system was experiencing negligible to no delays.

#### Simulation

We have used a custom simulation environment based on the ns-3 simulator for running our simulations [167]. The communication power consumption has been calculated by exchanging standard *iperf* UDP traﬃc at varying data rates suitable for IoT networks [168]. These power measurements are used to determine

, *e*

*j*

the values of the *et*

*i,j*

*r i,j*

and *ec* parameters for our simulation.

We setup the simulation nodes to run an optical character recognition applica- tion based on the open source Tesseract OCR engine to convert scanned images containing text into plain text ﬁles [169]. Tesseract is an neural net based OCR engine capable of recognizing lines and character patterns with support for more than 100 languages. The application scans in images as .png ﬁles from a TWAIN compatible scanner feed, detects characters and compares them to training data ﬁles to improve the conﬁdence of matched characters. Each node ran the appli- cation locally to setup a baseline for the rest of the simulation. We used four image resolutions *Si* with approximate image ﬁle sizes of 8KB, 16KB, 24KB and 32KB to measure the throughput and accuracy. Increasing the image resolution increases the accuracy of the character recognition but also increases the amount of computation and energy consumed by each node.

In order to obtain an optimized solution to the multi objective problem, nodes are randomly placed in a square area of side 500 meters. We assume an upper bound of 30 meters for Wi-Fi communication range and the link capacities are calculated taking into account standard path losses in wireless networks and Rayleigh Fading. We ran each simulation for a million iterations and the results

presented here have been averaged taking into account the conﬁdence levels for each metric.

#### Observations and Analysis

Prioritizing the task execution delay over the increased task execution accuracy is shown in Figures 4.1 and 4.2 with averaged delay and accuracy results. From Figure 4.2, we can observe that as the value of *β* increases to 0.3, the incurred task execution delay falls to less than 0.4 seconds. This is a large improvement as compared to when *β* = 0. As the value of *β* is further increased, the task execution delay is further reduced although the rate of reduction deteriorates. This reduction follows a monotonic function that makes it suitable for our pre- dictive cubic spline energy approximations. The graph also shows the constant reduction of accuracy with the conﬁdence values reaching only a little above 70% with *β* = 1. However, we need to account for the fact that values in the graph in Figures 4.1 and 4.2 are also dependent on other system parameters, notably the structure of the overall network and the capabilities of neighboring nodes.

In order to determine the nature of trade oﬀs between accuracy and latency within a given power envelope for a particular network, we use the Kendall rank correlation coeﬃcient [170]. *Vi*(*Vi*−1) is the number of all possible pairs of nodes

2

and thus −1 ≤ *τi* ≤ 1. We averaged for all nodes *i* ∈ *VT* in the network to obtain

the similarity index *τ* . We notice that when the value of *τ* is -1, the value of *β* is very signiﬁcant. For *β* = 0, we observe the best possible output from the system wherein, the delay and the accuracy are the highest. However, this also

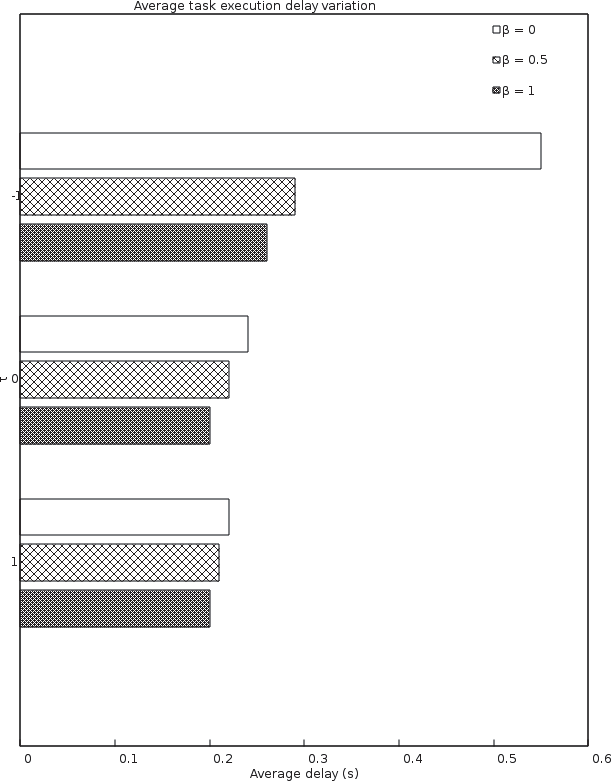


Figure 4.1: Average task execution delay variation with change of *τ*

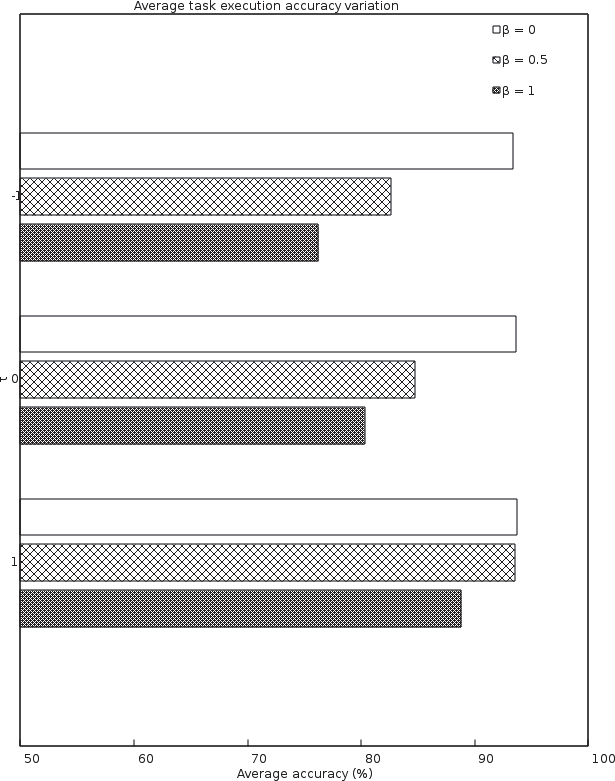


Figure 4.2: Average task execution accuracy variation with change of *τ*

consumes the maximum energy. This is because when *τ* = −1, the objectives in

our equation are in maximum conﬂict. As we increase the level of correlation, the signiﬁcance of *β* reduces. The optimization is maximum when *β* = 1.

We observe that the net gain is positive. This is due to the fact that some nodes can speed up the tasks originally allocated to them by delegating them to neighboring nodes with more capable processors and better connectivity. This can result in the thundering herd problem wherein a node having proven its capacity to outperform its neighboring nodes is ﬂooded by tasks arriving from its neighbor nodes. This would cause a huge bottleneck in the network and also deplete the energy resources of the node. We use cubic splines to predict such situations in the network while the simulation is running and actively redistribute tasks such that the overall power dissipation is optimized.

We also observe that for increasing values of *λ*, the queuing delays increase due to increasing congestion leading to reduced performance in order to operate the network within the predeﬁned power budget. Figure 4.4 shows the cooperative accuracy gain which is deﬁned as the diﬀerence between the averaged task accu- racy when the nodes are cooperating and when they execute their tasks locally. As expected we ﬁnd the gains diminish with increasing load values. The overall understanding from this behavior is that a cooperating IoT network can improve the accuracy of performed tasks while reducing latency even when its operating within a precomputed power envelope. However, as the amount of computing tasks increase, delegating tasks to neighbor nodes increases the costs in energy and bandwidth, thereby reducing overall gains.

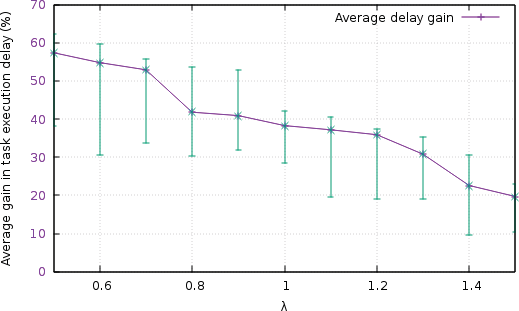


Figure 4.3: Average gain in task execution delay with variation in *λ*

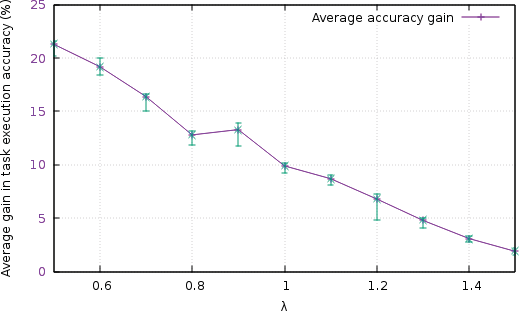


Figure 4.4: Average gain in task execution delay with variation in *λ*

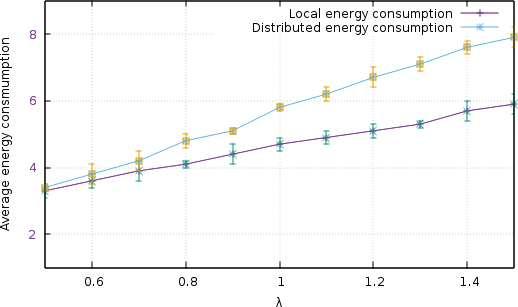
Figure 4.5 shows the averaged network power consumption for local and dis- tributed task executions for a million network instances. For lower values of *λ* (*<* 0.75), we observe that both local and distributed executions are within the power envelope. However, as the computing load increases, this can exceed the individual power budget for the nodes. Moreover, if delegation is performed, the overall power budget increases which in turn allows more tasks to be performed since the consumed power is still within the overall power budget of the whole network.

Figure 4.5: Average gain in task execution delay with variation in *λ*

In this chapter we have simulated a distributed optical character recognition system based on Tesseract and have compared the energy consumption with the tasks executed locally. Our results show that redistribution and collaboration of computing tasks in an IoT network can achieve enhanced performance by utilizing the inherent heterogeneity of devices that comprise the IoT network. We have formulated a multi objective optimization problem that strives to minimize

the latency and maximize the accuracy of the computation while keeping the IoT network within its established power envelope.

Our simulations show that the mechanism scales well for extremely large net- works with support for distributed computations that might overwhelm individ- ual devices. However, careful network design and engineering is required in order to achieve the right balance between latency, accuracy and power consumption. In the next chapter we will see how latencies are vital to the quality of service in mobile edge computing networks and how prioritizing network traﬃc helps to provide us with a richer mobile computing experience that we are all used to today.

**Chapter 5**

# Reducing network congestion by optimizing latencies and priorities

Improvements in computing and networking in the last decades have led to an explosive growth in mobile computing. Users increasingly demand richer and more immersive experiences from their mobile devices that were only previously possible with powerful computing devices. New categories of apps such as vir- tual/augmented reality, image and facial recognition and massively multiplayer online gaming have pushed the limits of existing computing infrastructure. In order to keep up with computing demands, mobile and other low powered de- vices have traditionally oﬄoaded computation to the cloud [192]. While this has solved the problem of computing to some extent, it has added latency resulting in poor user experiences.

51

In this chapter we have attempted to address these issues and build a priority oriented oﬄoading scheme. Traditional computation oﬄoading strategies are binary as devices either choose to oﬄoad or perform their computation locally. However, in order to support interactivity and cater to low latency requirements, we relax these strategies and allow tasks to be processed locally and through partial oﬄoading in the edge servers. We realize that the optimization variables for oﬄoading decisions and resource allocation are closely related to each other. Computation is allowed to be performed both locally and at the edge.

#### Modeling the system

##### Latency

###### Local computation

The time taken *tk,l* for local computation for the remaining (1 − *λk*)*dk* input data at user *k*:

*tk,l*

= *ck*(1 − *λk*)*dk*

*fk*

(5.1)

###### Oﬄoaded computation

The oﬄoading time *tk,off* for user *k* is given as:

*tk,off* = *tk,u* + *tk,m*

= *λkdk* + *λkdkck ,*

(5.2)

*rk fk,m*

where, *tk,u* is the transmission time for the user *k* to the edge and *tk,m* is the computing time at the edge. As a number of users execute their tasks in parallel at the edge, the overall latency for user *k* is *tk* = *max*{*tk,l, tk,off* }.

##### Energy consumption

The overall energy consumption in the model is a combination of the energy ex- pended in local computation and oﬄoaded computation. The following sections describe the contributions of each.

###### Local computation

Power consumption for user *k* is K*kf* 3, where *fk* is the processor speed and K*k* is the computation energy eﬃciency coeﬃcient which is governed by the intrinsic properties of the silicon in the processor [193]. Therefore, from Equation 5.1 we

*k*

have:

*Ek,l* = K*kf* 3*tk,l* = K*kck*(1 − *λk*)*dkf* 2

*k*

*k*

(5.3)

###### Oﬄoaded computation

This total cost of oﬄoaded computation includes the cost of transmission and the actual computation:

*Ek,off* = *Ek,u* + *Ek,m*

(5.4)

= Σ *x p*

*k,n*

*k,n*

*rk*

*m*

*k*

*k*

*k*

*k,m*

*n*∈N

*λkdk* + K

*λ c d f* 2 *,*

#### Modeling for congestion control

In order to simulate the various machine learning algorithms and evaluate their performance, a network testbed was created using the WSNet network simulator with some custom bindings for its packet based network simulation capabilities. Standard TCP and UDP traﬃc was generated with the links being put under various levels of network traﬃc load. Network links in the testbed were conﬁg- ured to support a maximum data rate of 1 Gbps with 1ms of maximum latency between the nodes in the network. A controller node is dedicated for house- keeping duties during simulation. This arrangement most accurately mimics a real world network with monitoring nodes sampling the network throughput at speciﬁed intervals.

The network traﬃc was generated using the standard *iPerf3* tool. It can generate TCP and UDP traﬃc with deﬁned ﬂow and latency parameters. A custom Python script was created to modify the default behavior of iPerf3 and create more realistic network traﬃc data. The network traﬃc parameters are shown in Table 1. Network bottlenecks were adequately simulated with the help of the Python script. Another Python script running on the controller node was responsible for data collection. Data collection was also performed using iPerf3. The simulation generated 100,000 data samples from each monitored network interface in the testbed.

In order to generate information on the network links and interfaces, the Net- work Measuring and Accounting Meter (Netmate) was used [240]. Netmate was conﬁgured to run on the router nodes to capture and log the network statistics.

Other system resource utilization statistics were captured using the DStat tool available in Linux as a system utility. System resources collected included CPU, RAM network interface utilization and other interfaces [241].

The download and upload bitrates for each network interface are obtained from Dstat. The *maxBitrate* feature is generated from the available information. This new feature can also be used to determine the network link usage ratio for the upload and download channels. Now, the *maxBitrate* feature can be used to compute the *futureBitrate* feature by shifting the *maxBitrate* value. The fre- quency of the collected data and the oﬀset value determines the depth of the future predictions. The dataset is ﬁnally normalized to have values between 0 and 1 in order to reduce the diﬀerences in the feature scales.

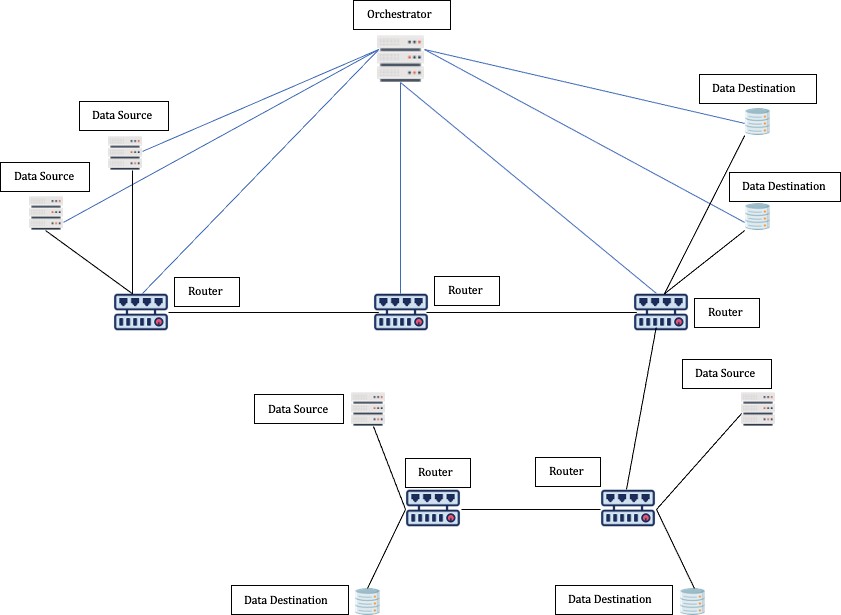
#### Simulation and Analysis

For performance analysis, we have compared our scheme with local computing and the ﬁxed oﬄoading ratio mechanisms. With local computing, all devices process tasks onboard their processors while in the ﬁxed oﬄoading ratio scheme, oﬄoading ratios are assigned to each user based on the time constraints in *C*1. We consider the Rayleigh distributed fading and path loss models for our sim- ulation denoted as |*β*| , where *β* is the short-term channel fading and *d* is the distance between two nodes. Simulation parameters are listed in Table 5.2.

*d*2

In order to simulate the various machine learning algorithms and evaluate their performance, a network testbed was created using the WSNet network simulator with some custom bindings for its packet based network simulation capabilities.

Figure 5.1: Network topology used in simulations



Standard TCP and UDP traﬃc was generated with the links being put under various levels of network traﬃc load. Network links in the testbed were conﬁg- ured to support a maximum data rate of 1 Gbps with 1ms of maximum latency between the nodes in the network. A controller node is dedicated for house- keeping duties during simulation. This arrangement most accurately mimics a real world network with monitoring nodes sampling the network throughput at speciﬁed intervals. A schematic of the network topology is shown in Figure 5.1.

The network traﬃc parameters are shown in Table 5.1.

Figure 5.2 shows the variation of the total consumed energy for all the three schemes with respect to the number of users considering *N* = 200. As the number of users increase, energy expenditures in all three schemes increase with increase in computation. However, our mechanism can reduce the energy expenditure by

Table 5.1: Simulation parameters

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Value** | **Distribution** |
| Protocol | TCP and UDP | Uniform |
| Traﬃc duration | [20, 1000] seconds | Uniform |
| Packet size | [32, 1472] bytes | Uniform |
| Bitrate | 5 Mbps Mean  512 Kbps nominal | Normal *i* |
| Flow Inter-arrival duration | [10, 1000] seconds | Uniform |
| Source/Destination Endpoints | Host IP | Uniform |
| iPerf3 port | [6000 - 6009] | Uniform |

Table 5.2: Simulation Parameters

|  |  |  |
| --- | --- | --- |
| **Symbol** | **Parameter** | ***Value*** |
| *T* | Maximum allowed latency | 50 ms |
| *ck* | Computing workload/intensity | 1000 cycles / bit |
| *σ*2 | Noise | 10−13 watts |
| *B* | Bandwidth | 12 KHz |
| *F* | CPU frequency of Edge server | 5*.*0 GHz |
| *fk* | CPU frequency for device *k* | 0*.*5 − 0*.*8 GHz |
| *pk* | Transmission power for user *k* | 30 dBm |
| *dk* | Input data size for user *k* | 103 − 2*x*103 bits |
| *N* | Orthogonal Wireless Channels | 40 − 200 |

25% − 50% over the ﬁxed oﬄoading ratio mechanism and 40% − 60% over the local computation mechanism.

From Figure 5.3 we observe that the energy consumption of the network decreases as the devices increase their transmission power. This is explained from the fact that as transmission power increases, the oﬄoad rate increases which reduces the transmission time *tk,u*. Enabling devices to oﬄoad more data increases system eﬃciency. However, the rate of decrease slows down as the expended transmission power goes beyond 35 dBm, since the eﬀect of the transmission time on energy consumption is small as compared to the actual execution time.

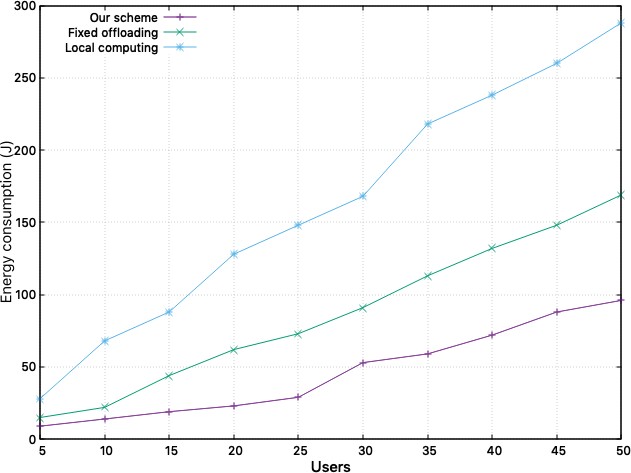


Figure 5.2: Variation of total energy consumption with number of devices.

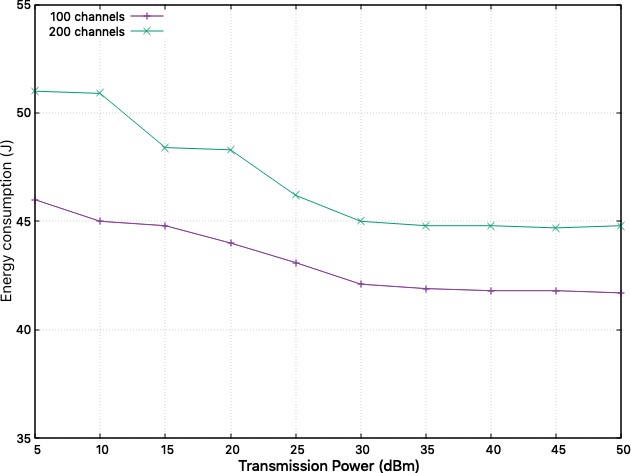


Figure 5.3: Variation of total energy consumption with transmission power.

From Figure 5.4 we observe that the total energy consumption reduces as we relax the required latency of devices. This is because a larger allowance of latency allows devices to oﬄoad more computation reducing overall energy consumption. However, latency relaxation has a diminishing inﬂuence over the total energy consumption as the allowable latency approaches a threshold.

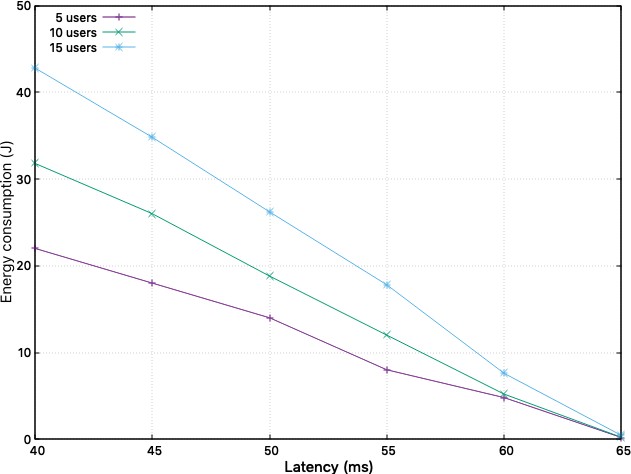
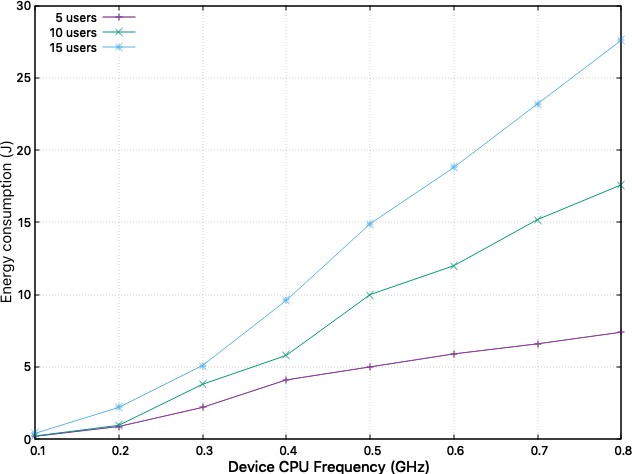


Figure 5.4: Variation of latency and total energy consumption.

Figure 5.5 shows that energy consumption grows with increase in the CPU fre- quency since local computation results in reduced oﬄoading and increasing en- ergy consumption. Partial oﬄoading works better in such circumstances.

Figure 5.6 shows that energy consumption reduces with increase in wireless chan- nels since users have more options to choose a channel with a preferable channel gain. This helps the device to oﬄoad more computation while expending the same power and reducing the overall energy consumption.



Figure

5.5: Variation of total energy consumption with device CPU fre-

quency.

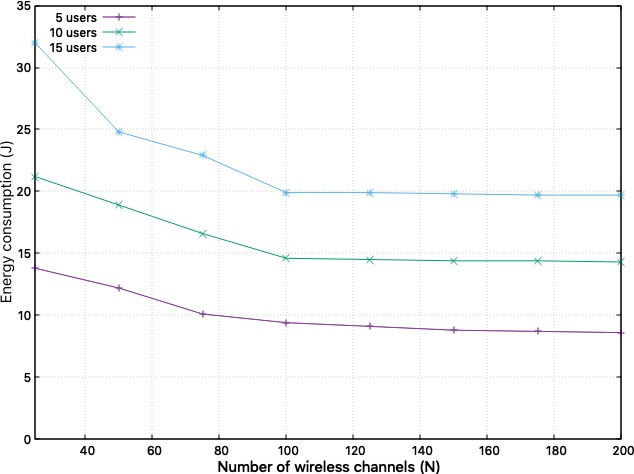


Figure 5.6: Variation of total energy consumption with available channels.

Figure 5.7 shows that the total energy consumption of the system decreases with increase in the CPU frequency at the edge. As more processor cycles become available, more devices oﬄoad, reducing the overall energy consumption as the edge servers are more computationally eﬃcient than individual devices.

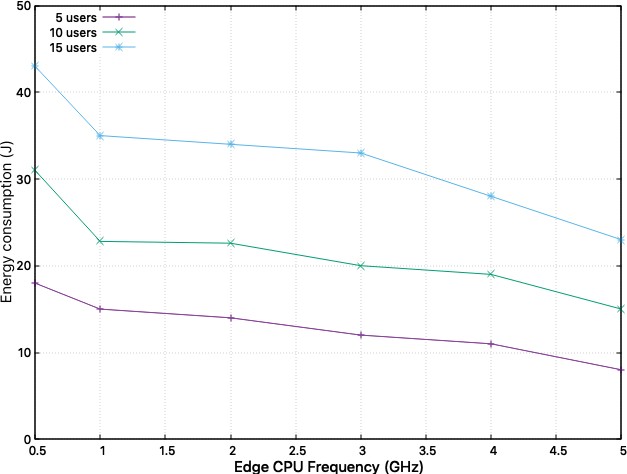
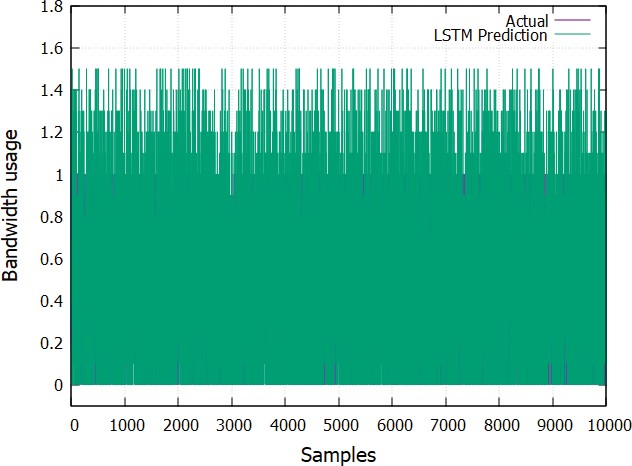


Figure 5.7: Variation of total energy consumption with number of users.

We have also compared the performance of Multi Layer Perceptrons, Long Short- Term Memory network and the Autoregressive Integrated Moving Average. Su- pervised learning was used to train these algorithms to predict network conges- tions based on future bandwidth utilization. The *futureBitrate* feature is used by the ARIMA algorithm while a much larger set of 115 features was used by LSTM MLP. Each model was trained on a number of network interfaces in order to reduce bias and improve their prediction capabilities.

All of these models have their own strengths. While ARIMA can model periodic

Figure 5.8: Variation of LSTM predictions and actual values



trends and behaviors it does so with a singular feature set. However, studies have shown that network traﬃc behavior is based on a number of factors that require multiple feature sets to accurately model. Both MLP and LSTM can eﬀectively utilize multiple feature sets. LSTM keeps a record of past events while leaning new behaviors - a useful feature for predicting future bandwidth requirements. MLP can process the data substantially faster as it does not need to keep historical records.

The predictions from the LSTM and MLP models are shown in Figures 5.8 and

5.9 respectively. Figures 5.10 and 5.11 show the diﬀerences between the actual and predicted values for the LSTM and MLP models.

The results from the ARIMA models show a high degree of bias and poor accu- racy as compared to the other models. As such these errors are quite signiﬁcant

Figure 5.9: Variation of MLP predictions and actual values

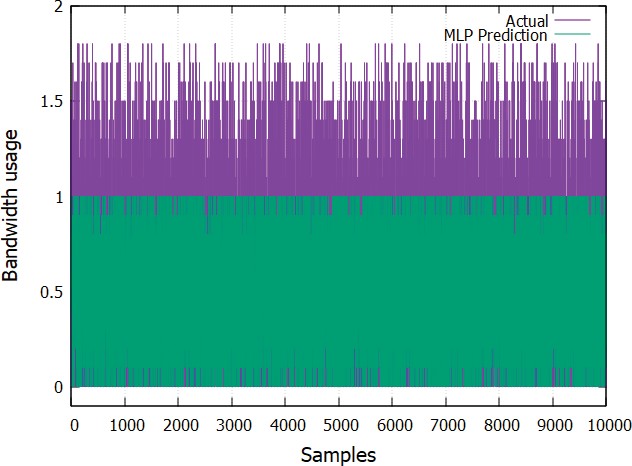


Figure 5.10: Diﬀerence between LSTM predictions and actual values

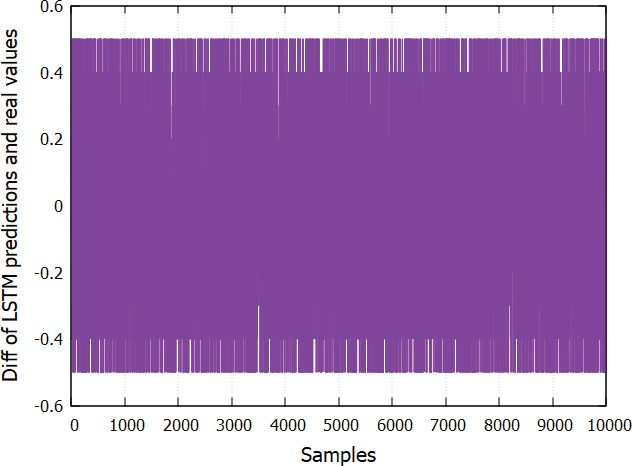
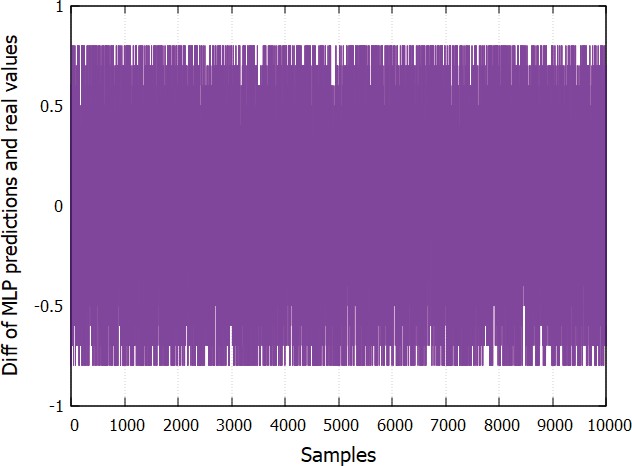


Figure 5.11: Diﬀerence between predictions and actual values



thereby rendering this model unsuitable in a real world scenario. The inaccu- racies are related to the single feature set that ARIMA uses for model ﬁtting. Multiple feature sets tend to improve the quality of predictions.

LSTM models show better performance than MLP models. In our simulations, the diﬀerence between actual and the predicted values in the LSTM were minimal while those in the MLP were quite signiﬁcant. This diﬀerence in the predicted and actual values would cause signiﬁcant overestimation and underestimation of the network capacity in the future. Since LSTM models utilize past variations in the network traﬃc to estimate future traﬃc, the quality of the estimations is signiﬁcantly better. The simulations revealed that the estimations for some network interfaces were signiﬁcantly better than other interfaces. Further analy- sis revealed that the predictions performed poorly during higher variance of the

network traﬃc in the interfaces.

Simulations were conducted on real-time bandwidth usage prediction after the LSTM model was trained and evaluated. Each interface ran the Python script for the prediction mechanism to generate data about network traﬃc in the future. Using out-of-band signalling the data was generated 20 times a minute and nodes were able to look ahead 15 seconds into each interface’s traﬃc bandwidth. Once the network traﬃc data for the future was made available, the computing nodes in the network could use this data to reduce congestion. Simulation for congestion avoidance and mitigation were carried out under the following scenarios:

* Congestion avoidance: Congestion avoidance is a proactive approach un- dertaken by the nodes when their network interfaces are predicted to have a high rate of traﬃc in the future. Once a predeﬁned network threshold is predicted to be reached in the future, the network controllers would start routing traﬃc on diﬀerent paths in order to ease the congestion on the path that was predicted to become congested. In the simulations, a threshold of 65% link utilization was used to start diﬀerential routing of packets at the network interface.
* Congestion mitigation: This is a much more drastic approach to network congestion. If the predicted traﬃc on a network interface is predicted to exceed a deﬁned threshold in the future, the interface would start dropping new incoming packets as a mechanism to avoid a traﬃc overload. Network packets would continue to be dropped until the models predict that the future network utilization would drop below the deﬁned threshold. In the

simulations, a threshold of 85% was used to drop new incoming packets at the interface.

In this chapter we have explored the growth of mobile computing and how the limitations of these devices can be overcome through a combination of resource allocation and computation oﬄoading. We have also studied the eﬀects of a dif- ferent computation oﬄoading strategies and shown how a combined resource allo- cation and oﬄoading strategy can yield the best results. Our optimization mod- eling method to determine the best possible approach in networks with thousands of devices have shown how these computationally constrained devices can pro- vide richer experiences to users without losing their appeal as small, lightweight consumer devices. Our combined optimization mechanism can reduce the overall system cost over extended computations while minimizing the penalties incurred from the extra network traﬃc and determining oﬄoading decisions.

Our work in this and the previous chapters have focused on optimizing workloads in IoT devices. In the next chapter, we look at optimizing computing at the edge nodes. At any time, a particular edge node is connected to multiple devices. As devices move in and out from the service area of a particular edge node, it has to take care of the handoﬀs and its associated housekeeping. The edge node has to intelligently decide the next edge node for handoﬀs and the data that needs to be cached and/or forwarded. Given the nature of computations and the way the edge is designed makes this a particularly hard problem. We model this scenario as a game to arrive at an acceptable solution.

In this chapter we have evaluated a mobile edge computing network and de- signed an algorithm to minimize the total energy consumption with strict la- tency requirements. Owing to the nature of the problem we have divided it into three problems that optimize the oﬄoad ratio, transmission power and resource allocation and solved them sequentially. Our simulation results show that the proposed algorithm outperforms traditional local computing and ﬁxed oﬄoading mechanisms. A hybrid approach to computation is an eﬀective way to improve performance in mobile devices without putting too much strain on the devices themselves. In the next chapter we will build on this idea and look at mechanisms to improve mobile computing experiences.

**Chapter 6**

# Optimizing mobility and cooperative computing at the Edge

As more users turn to mobile computing as their primary means of consuming services and depend on them for their daily computing needs, mobile application developers are creating increasingly sophisticated applications that rely on com- putationally intensive tasks such as image recognition, augmented reality and massively online interactive gaming [196–198]. Most mobile devices have lim- ited computational and energy capacities and is considered to be a prime factor holding back the growth of ubiquitous mobile computing [199]. While recent advances in network capacity from widely deployed fourth and ﬁfth generation networks have solved the problem of mobile bandwidth, work needs to be done

68

*Optimizing mobility and cooperative computing at the Edge* 69

to improve the computing capacity and energy requirements of mobile device themselves [200].

Most popular mobile applications today are hosted on commercial cloud providers such as Amazon Web Services, Microsoft Azure and Google Cloud and perform demanding computations while users interact with these services from their mo- bile devices [201, 202]. While, this seems a reasonable approach to move the computation away from user’s devices, this does little to address the low latency requirements that most user’s have come to expect from their devices. Moreover, in crowded places when too many users interact with the same services, it results in substantially high loads on the cloud services and in the radio access spectrum reducing wireless access eﬃciency and increased latency [203].

With the recent prevalence of mobile devices, the amount of computing that each device is expected to perform has grown substantially. In order to provide better user experiences, mobile devices are expected to perform complex com- putations such as image processing or model training. Given that these devices are energy constrained, computation is oﬄoaded to more powerful servers at the edge. As nodes are mobile, a multi-hop routing model can eﬀectively model the network. Each node in the network can choose to compute on the received data forward the resulting data onto other nodes or to the destination. A network wide cost minimization problem that optimizes the routing and task oﬄoading is considered here.

A multi hop network with an ad-hoc topology most accurately mimics real world networks. Nodes in the network collaborate to perform computations and results are obtained sooner than regular on device computations. Nodes are considered

to be heterogeneous in their computing and storage capabilities. Since each node can also be the originator of the dataset, a task is considered completed only when the results of the computation have been merged, possibly from various computing nodes and sent back to the originating node. This is a non-trivial and non-convex problem and this work establishes the necessary and suﬃcient conditions for a globally optimized solution.

In order to achieve a middle ground, the computation handover strategy should ﬁnd a balance between the computing and the communication latencies. Further, it should also compute the performance and cost metrics to decide on an overall oﬄoading strategy. The main contributions in this chapter are:

* Formulate the computation oﬄoading problem for multiple devices without the cloud playing a role in the process.
* Perform extensive analysis of the oﬄoading game for both homogeneous and wireless access.
* Achieving Nash equilibrium guarantees that the system has achieved an optimum level of task distribution.
* Compare our mechanism with against the cloud making all the decisions on behalf of the devices. We demonstrate that our mechanism achieves eﬃcient oﬄoading and scales well with the system size.

#### The Computation Oﬄoading Game

Mobility while being the raison d’ˆetre of smartphones also lends to the diﬃculty of optimizing operating costs over longer time periods. This is because it is virtually impossible to predict and track movements of mobile devices and this data cannot be used to allocate resources at the edge. Resource allocation at the edge needs to be carried out on the ﬂy while causing minimum impact the the perceived QoS. We employ game theory to get around this limitation since it allows us to analyze interactions between multiple mobile devices that do not follow any deﬁned mobility patterns and create a computation oﬄoading mechanism that can work independent of the cloud. Employing the intelligence gleaned from these devices, we can devise solutions that are less complex and converge satisfactorily within a reasonable time.

Moving the oﬄoading decision away from the cloud would greatly reduce the ex- tra traﬃc overhead from synchronization and housekeeping. Considering *a*−*n* = *a*1*, a*2*, an*−1*, an*+1*, ..., aN* as the set of oﬄoading decisions for all users except

user *n*, user *n* would want to select an oﬄoading decision between cloud or lo- cal *an* ∈ {0*,* 1} such that it can minimize the processing time and the energy

consumption.

#### The computation oﬄoading mechanism

There have been extensive work on computation oﬄoading mechanisms as out- lined in previous chapters. While they depend on a rigorous centralized decision

to be made before the start of the computation, our mechanism hinges on co- ordination between devices to achieve a solution that is mutually beneﬁcial to all. We divide the time into discrete time slots and in each slot we measure the interference and the devices contend on updating their oﬄoading decision.

Each device that chooses to oﬄoad at a particular time slot transmits a beacon to the base station which allows other devices to measure the perceived interference by enquiring the base station.

#### Simulation results and analysis

We have compared the impact of the computation size on our mechanism as compared to local computation. Under the local computation scheme all mobile devices perform their computation locally while they undertake a distributed oﬄoading strategy with our mechanism. The results are shown in Figure 6.1.

We see that the computing cost for the system increases as the number of required processor cycles increase for both local computing and distributed oﬄoading schemes. However, the increase in cost from the oﬄoading mechanism is much slower as compared to the local computation. This is because as the required number of processor cycles increase, devices choose to oﬄoad their computation in order to reduce the heavy cost of local computing.

We have also compared the impact of the data size on our mechanism as com- pared to local computation. We observe that as the data size increases, the system cost for computation oﬄoading also increases. This is because, the over- head for computation increases as the data size increases. However, when the

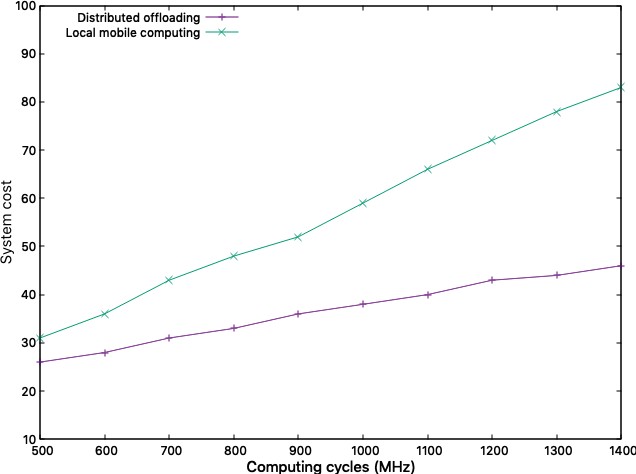


Figure 6.1: Variation of computing cost with required CPU cycles.

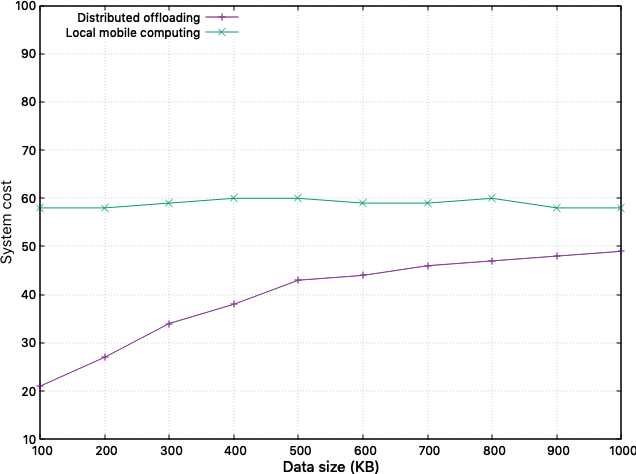


Figure 6.2: Variation of computing cost with input data sizes.

data size is large, the rate of increase in the computing cost is slow. This is because when the data size is large, more devices would choose to perform local computation and avoid the heavy overhead of oﬄoading. This behavior is shown in Figure 6.2

In order to compute the averaged system cost with respect to the number of devices taking part in the computation, we have repeated our experiments for both local and oﬄoaded computation starting with ten devices and going up to ﬁfty devices. Each of these experiments were repeated at least a million times in order to take into consideration any transient eﬀects. The results are shown in Figure 6.3.

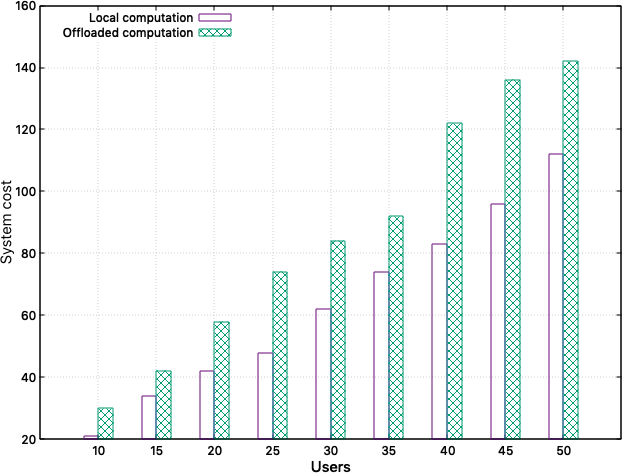


Figure 6.3: Averaged computing cost.

We observe that our mechanism improves performance by 35% to 40% over local computing and the oﬄoading overhead is generally within 10% for all devices.

This shows that substantial improvements in computing can be achieved without an excessive overhead.

In order to determine the feasibility of our mechanism in commercial mobile ser- vice deployments we have computed the time required for the system to stabilize as the number of devices taking part in the computation increases. The results for this experiment are shown in Figure 6.4.

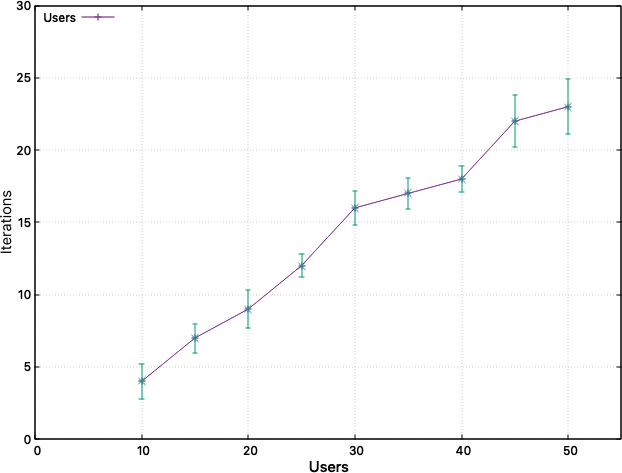


Figure 6.4: Convergence of the oﬄoading mechanism.

We observe that the average convergence time increases linearly with the number of devices. This proves that our mechanism scales well with the number of devices. Generally, the number of devices in a network increases over time. However, our mechanism still ensures a linear bound on the convergence time that bodes well for the performance of our mechanism.

We also simulated a number of network topologies using the WSNet discrete event network simulator. The following topologies were simulated:

* Balanced Tree
* Fog
* Connected ER
* Small World

In order to evaluate the performance of the proposed mechanism, it was com- pared with some well-known baseline algorithms. These are brieﬂy described here:

* Shortest Path Optimal Routing: This computes the shortest path between nodes. If the source and destination nodes are the same, this eﬀectively is the same as local computation [284].
* Local Computation Optimal Routing: This tends to perform computations entirely locally at the source. If pure local computation is not feasible, it performs minimal oﬄoading [285].
* Gradient Projection: This is a *gradient-type* search procedure designed to handle constrained network routing optimizations and is well suited for networks with a limited number of *source-destination* pairs.
* Linear Program Rounded: This is a combined path routing and oﬄoad- ing method that does not consider partial oﬄoading, congested links and network ﬂows [286].

All simulations were performed by setting *b* = 0. The value of *at* was set to be exponential with a mean value of 0*.*5 in the interval [0*.*1*,* 5]. Empirical studies show that most computational tasks have a value of *at <* 1. However, certain computationally intensive tasks such as optical character recognition and video rendering have larger values of *at*. Given that most devices participating in the computation would have modest computing power and oﬄoad most of their

computation, higher values of *at* would be rare.

Table 6.1: Simulation parameters

|  |  |
| --- | --- |
| **Parameter** | **Value** |
| *tmax* | 40 |
| *tmin* | 20 |
| *smax* | 40 |
| *smin* | 4 *i* |
| *M* | 5 |
| *rmin* | 0.5 |
| *rmax* | 1.5 |

We have compared the cost of our mechanism with the other algorithms based on the parameters as noted in Table 6.1. The results are shown in Figures 6.5 through 6.9. Our algorithm signiﬁcantly outperforms other algorithms in all sim- ulated network topologies. The most signiﬁcant performance gain was over the Linear Program Rounded algorithm with over 25% performance improvement. From the performance gains observed in the Small World networks for linear and queue, the proposed algorithm works exceptionally well in congested networks.

In order to simulate real world network failures and determine the resiliency of the algorithm, we simulated node failures during our simulations. The proposed algorithm converged much faster than others adapting to the new topology. This speed can be attributed to the ﬁne tuning of the scaling matrices. Moreover as

Figure 6.5: Variation of normalized cost for the routing algorithms under the balanced tree topology.

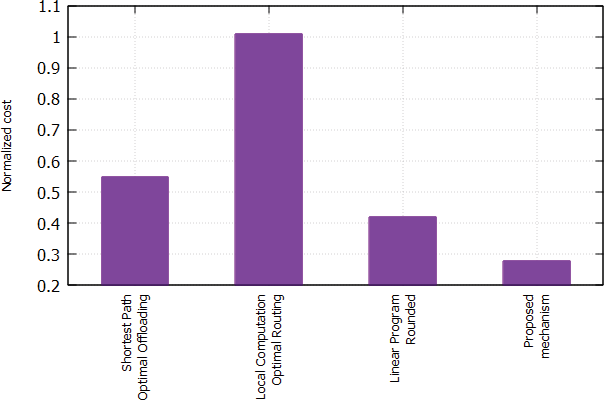


Figure 6.6: Variation of normalized cost for the routing algorithms under the Fog network topology.

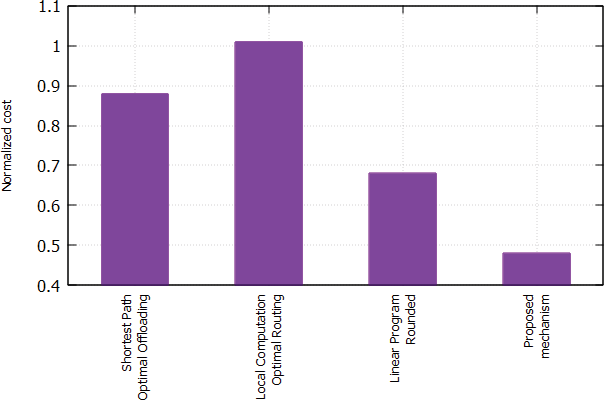


Figure 6.7: Variation of normalized cost for the routing algorithms under the Connected-ER topology.

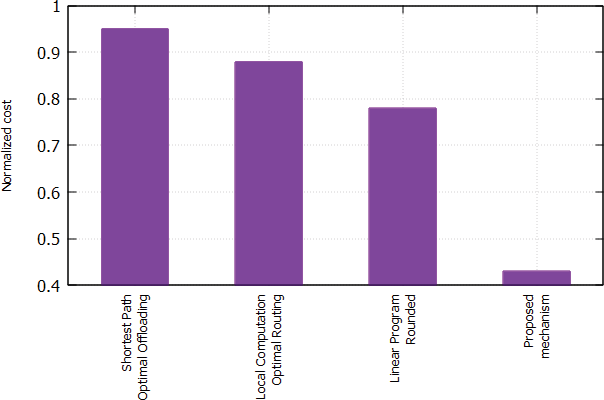


Figure 6.8: Variation of normalized cost for the routing algorithms under the Small World topology without queueing.

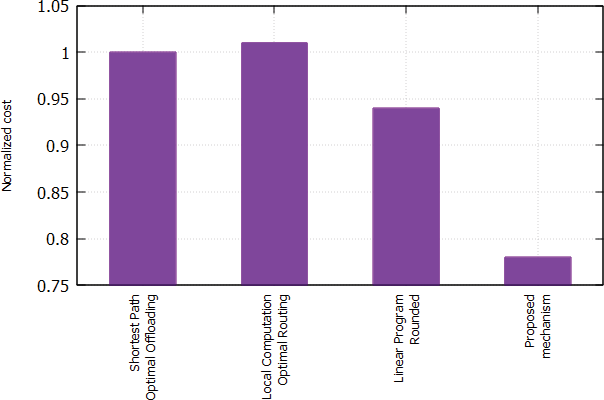
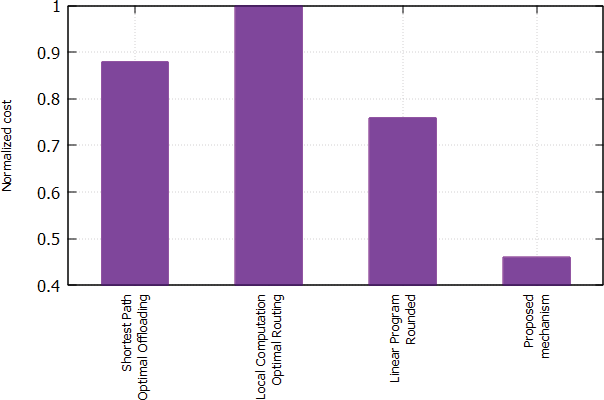


Figure 6.9: Variation of normalized cost for the routing algorithms under the Small World topology with queueing delay.



the scaled input rates *ri*(*d, t*) was increased while keeping the other parameters constant, the proposed algorithm ensures that the network can scale up faster than the other algorithms to handle the additional traﬃc.

Considering *Hd* and *Hr* as the average hop distances for data packets from the source to the computation node and the result packets from the computation node to the destination respectively, *Hd* and *Hr* was compared by varying the values of *at*.

Our experiments in this chapter demonstrate the feasibility of the mechanism and shows that it scales well as the size of the system grows. As part of these simulations we realized that the major bottlenecks arise during service handoﬀs as user proﬁle and computation data needs to be replicated between edge nodes.

We also saw how computation oﬄoading can have an eﬀect on distributed com- putation on large scale networks. We also saw how the topology of a network determines its eﬃcacy and how our proposed algorithm is able to take into ac- count changing input and result data packets and network topologies and move computation closer to the destination to reduce the strain on the network. In the next chapter, we would look at caching mechanisms to improve performance at the edge.

**Chapter 7**

# A foray into caching for mobile edge computing environments

Mobile Edge Computing (MEC) has been in the forefront recently for its ability to supplement cloud computing. MEC servers are deployed closer to the users at the network edge in order to reduce the delay and computation load on the cloud. A common method of reducing latency and content delivery charges is through the use of edge caching. Popular content is cached in the MEC nodes that are closer to the users thereby reducing access latencies. However, eﬃcient caching mechanisms are required to make optimum utilization of the limited storage capacity in MEC nodes. In this chapter, we propose a scalable and hierarchical caching mechanism that intelligently computes content popularity and caches the content at the optimum location to improve throughput. We compare our mechanism with established mechanisms and prove its superiority in real world workloads.

82

#### Caching in the Modern Web

Studies have shown that web traﬃc follows a speciﬁc pattern. Popular content is consumed by more and more users causing traﬃc spikes while unpopular content hardly creates much traﬃc over time. Moreover, diﬀerent users connect to the internet over a variety of connections. With the growth of mobile computing, a large fraction of users connect over slower and/or congested mobile networks. A larger ﬁle size turns oﬀ users with slower connections while slower content delivery causes the content to lose popularity over time.

Table 7.1: Symbols and Notations

|  |  |
| --- | --- |
| **Notation** | **Description** |
| N | Total possible set of content requests |
| *ci* | File contents of the *ith* ﬁle |
| *pi* | Content popularity for *ci* |
| *wi* | Number of write requests for content *i* |
| *si* | File size of content *i* |
| *cloc* | Size of local cache |
| *cmec* | Size of MEC cache |
| *sloc* | Size of local cache |
| *smec* | Size of MEC cache |
| *rk*  *loc* | Rank for content *k* in local cache |
| *rk*  *mec* | Rank for content *k* in MEC cache |
| *rloc* | Current minimum rank in local cache |
| *rmec* | Current minimum rank in MEC cache |
| *thrmem*  *loc* | Current memory threshold for local cache |
| *thrmem*  *mec* | Current memory threshold for MEC cache |

Caching policy is a linear utility maximization problem in that the primary design decisions are to maximize the cache hit ratio [248, 249]. Formally, this

can be stated as:

*max*

M⊆N

*pici*

*i*∈M

Σ

Σ

(7.1)

*s.t. si* ≤ *φ*

*i*∈M

where, N is the universe of ﬁle contents in the system, M is the set of ﬁles that needs to be cached, *ci* is the ﬁle contents of the *ith* ﬁle, *si* is the ﬁle size, *pi* is the estimated popularity of the ﬁle with contents *ci* and *φ* is the cache size [249].

Therefore, the optimal eviction policy *Pv* to evict a ﬁle is given by:

*P* = *argmin* *pici*  *,* ∀*i* ∈N (7.2)

*si*

*v*

A summary of all notations used in this work is given in Table 1.

#### The Hierarchical Caching Mechanism

##### Formalizing the caching system

For an eﬀective caching mechanism in MEC environments, we consider the local storage on-board the devices and the storage present in the MEC nodes while the cloud is considered to be the eventual storage. Let us consider *Cloc* and *Cmec* to be the content present in the local storage and the MEC server respectively.

Then we have:

*max*

*Cloc*+*Cmec*⊆*N*

*i* ∈ *Cloc,*

*j* ∈ *Cmec* + Σ *pici* + *pjcj*

*s.t. C*1 : *i* ∈ *Cloc*

*C*2 : Σ *si* ≤ *φ C*3 : *j* ∈ *Cmec C*4 : Σ *sj* ≤ **Φ**

(7.3)

where, constraint *C*1 speciﬁes that content *i* is present in the local cache, *C*2 ensures that the size of content *i* is smaller that the local cache size, *C*3 speciﬁes that content *j* is present in the MEC cache and *C*4 ensures that the size of content *j* is smaller than the MEC cache.

##### Hierarchical Ranking

Based on the formalization of the caching mechanism in the section above, we propose a variation of a greedy algorithm for caching contents and moving them dynamically between the local system and the MEC. The algorithm is presented in Listing 1.

In order to formulate the hierarchy, we designate each content object with two

ranks - one for storage in the local device storage and the other for MEC storage.

The rank for local storage *rloc* is denoted as

*pkwh*

*k* . We remove contents with the

*s*

*k*

minimum rank. The rank for the content in the MEC *rmec* is designated as *pk* .

*skwh*

*k*

This designation ensures that content with low popularity and a higher ﬁle size will be eventually evicted from the MEC memory and onto the cloud.

The algorithms for the storage allocation is in Listing 2.

**Algorithm 1:** Modiﬁed Greedy

Function: mGreedy

*Modiﬁed Greedy algorithm for caching policy*

**for *each*** *iteration* **do**

**if** *objecti Cloc + Cmec* **then**

∈

**Add** *object* access request to queue

**Update** *objecti* statistics

**else**

*object* ← Run *allocStorage*

###### Adjust local and MEC storage thresholds if *η <* 0*.*5 then

*thrm*† *em* ← 0

###### else

*thrm*† *em* ← *η*

Run memory usage predictions

*Run evictObjects(UpdateObject)*

###### end

† = memory threshold

**Algorithm 2:** Storage Allocation

***Function: allocStorage*** *Allocate storage for objects rloc, rmec* ← Run *getRank(key)*

**if** *rloc > α* · *rmin and memfree >* 0 **then**

*loc*

**if** *rmin > rloc* **then**

*loc*

*rmin* = *local*

*loc*

return **local** cache

*loc*

*free*

**if** *r > β* · *rmin and mem >* 0 **then**

*mec*

*mec*

*mec*

**if** *rmin > rmec* **then** *rmin* = *MEC* return **MEC** cache

*mec*

*mec*

return **cloud**

##### Convergence

In order to ensure fast convergence, we include two parameters (*α, β*) to deﬁne the threshold number for the local and the MEC storage respectively. These parameters are dependent on the utilization *η* of the storage present in the devices and the MEC server.

We also keep a track on the lowest ranked object currently stored in the local

memory *rmin* and the MEC *rmin* server respectively. The algorithm for deter-

*loc mec*

mining the rank is in Listing 3.

**Algorithm 3:** Determine Object Ranks

***Function: getRank*** *Determine rank for objects r , pk*

←*loc*

*wh*·*sk*

*k*

*h*

*,* ← *pk*·*w*

*rmec k*

*s*

*k*

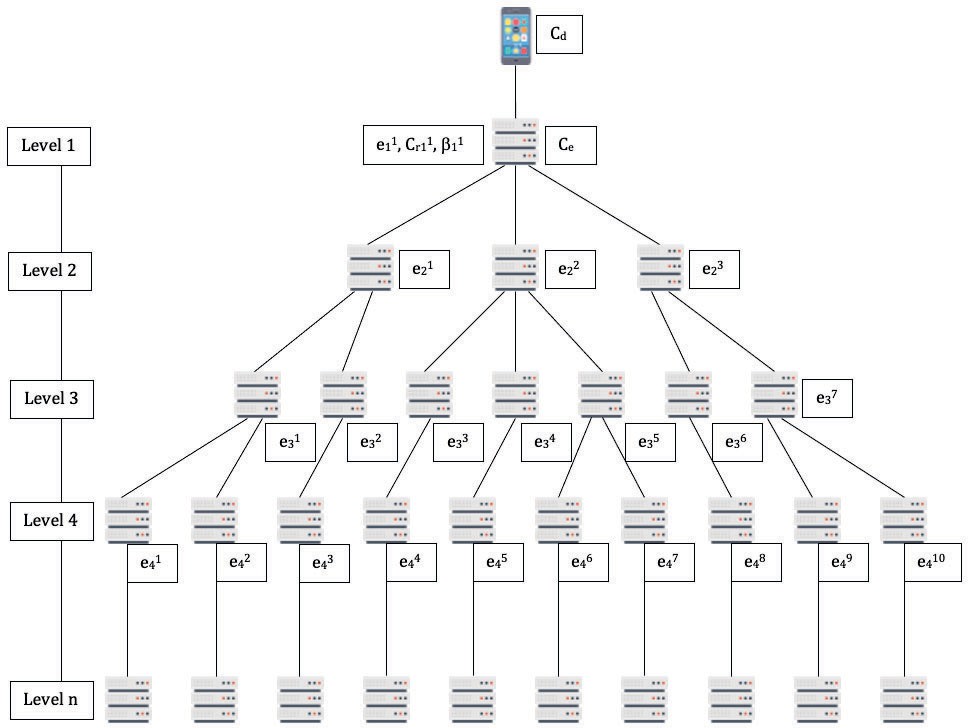
return *rloc, rmec*

#### Modeling the Cooperative Caching Mechanism

we consider a scenario wherein mobile devices take advantage of the storage available in the edge servers to cache data. We denote *β* ∈ [0*,* 1] as the edge caching ratio. Since each edge server is connected to multiple other edge servers, it can choose to cooperate with other edge servers and extend the cache available to the users. This cooperation between edge servers would be dynamic depending on the performance requirements and prevalent network conditions.

During the course of computation a mobile device *m* might need to cache data required for the computation. Depending on the amount of locally available

Figure 7.1: The cooperative caching network



cache and the number of edge servers available the mobile device might choose to store it locally or move it to one of the edge servers. The amount of cache in the edge server determines the edge caching ratio *β*

The cooperation between edge servers is shown in Figure 7.1. The root server is the one that is currently serving a particular mobile device. The edge servers in the other levels are determined based on their connectivity with the root server and latencies. Each server in the hierarchy is represented by the tuple (*e, Cr, β*) wherein, *e* refers to a particular edge server, *Cr* is the amount of available cache in the server and *β* denotes the edge caching ratio for the particular server.

As an example, the root server can be represented as (*e*1*,C*1 *, β*1). The cache

1 *r*1 1

can be shared between other servers as long as they can communicate and have reasonable latencies. The total cache available to a devices is the cumulative

amount of cache in the device and the edge server:

*C* = *Cd* + *Ce* (7.4)

We denote the servers that can split and send cache data to other servers as anchor servers. All servers in the hierarchy except the last level are therefore

anchor servers. In Figure 7.1, edge servers *e*1, *e*1, *e*2, *e*3

are anchor servers.

1 2 2 2

Edge servers that can receive receive and allocate cache requests are denoted as reciprocal servers. All servers in the hierarchy except the root server can be a

reciprocal server. In Figure 7.1, edge servers *e*1, *e*2 and *e*3 are the reciprocal

2 2 2

servers for edge server *e*1. Any server that is not the root server or does not belong to last level of the hierarchy can be both an anchor and a reciprocal server. A combination of the anchor and reciprocal edge servers forms the cooperative

1

cache. In Figure 1, the edge servers, *e*1, *e*1, *e*2 and *e*3 form the cooperative cache

1 2 2 2

wherein, the edge server *e*1 as the anchor server and the edge servers *e*1, *e*2 and

1 2 2

3 *e*1 1 2 3

*e*2 are the reciprocal servers denoted as *C* 1 (*e*2*, e*2*, e*2).

The actual cache capacity of an edge server *e* is therefore the capacity that is made available to the other edge servers and mobile devices without the recip- rocal edge servers *ce*. Therefore, the actual available cache is the combination of the cache in the server and the cache available from the reciprocal servers. For

*e*2 3 4 5

the cooperative cache *C* 2 (*e*3*, e*3*, e*3), the cache capacities for the reciprocal edge

servers are *ce*3 , *ce*4 and *ce*5 respectively, while the total cache capacity of *e*2 is

3 3 3 2

*Ct*2 . Therefore, the capacity for edge server *e*2 is denoted as:

2 2

*ce*2 = *Ct*2 + *ce*3 + *ce*4 + *ce*5

(7.5)

2 2 3 3 3

When a mobile device needs to store data in the cache, it splits its cache require- ments *C* into the on device cache *Cd* and the edge server cache *Ce* as shown in Equation 7.4. The amount of data oﬄoaded to the edge cache depends on the cache capacity made available to the devices by the edge servers which in turn is a combination of their own capacity and the cooperative cache as shown in Equation 7.5. Edge servers that make up the cooperative cache in turn repeat this process recursively until the total requested cache capacity can be satisﬁed. Therefore, if the requested cache size is substantial, the cache may be spread over edge servers that are more than one hop away. The spread of the cache is determined by size of the requested cache and the availability of caches in the edge servers.

The latencies in the cooperative caching system can be divided into the following:

* *Cache computation latency*: This is the latency incurred in the edge servers for computing the cache distribution
* *Remote cache request latency*: This is the latency incurred in forwarding cache access requests between cooperating cache servers.
* *Remote cache response latency* : This is the latency incurred from retrieving the data from the remote cache.

The processing capacity of a mobile device is the number of processing cycles required for processing a 1-bit task and is denoted as:

1

*Cm* =

*f*

*m*

*,* (7.6)

where, *fm* is the frequency of the mobile device *m*. Similarly, the processing cycles required for the edge server is given as:

1

*Ce* =

*f*

*e*

*,* (7.7)

where, *fs* is the frequency of the of the edge server *e*.

Each edge server *e* maintains a queue of cache access requests *qe*. Hence, the cache computation latency *De* denoting the delay at the edge server before a particular cache access request *k* can be processed is:

*De* = *qe* · *Ce*

= *qe , fe*

(7.8)

and for a cache for size *lk*, the latency would be:

*Dk* = (*qe* + *lk*) · *Ce*

*e*

= *qk* + *lk*

*fe*

(7.9)

The edge servers maintain a cache request queue *qreq* for oﬄoading cache service requests to other cooperating edge servers. The latency incurred for remote

*e*

cache requests *Dreq* can be denoted as:

*e*

*Dreq* = *qreq* + *lreq creq,* (7.10)

*e*

*e*

*e*

*e*

where, *qreq* is the cache request queue length, *lreq* is the size of the cache that is

*e e*

to be oﬄoaded by the edge server *e* to its cooperating edge servers, *creq* is the number of processing cycles required to oﬄoad the cache.

*e*

The edge server also maintains a cache response queue that contains cache re- quest responses from other cooperating edge servers. The latency incurred for remote cache request responses *Dres* can be denoted as:

*e*

*Dres* = *qres* + *γlres cres,* (7.11)

*e*

*e*

*e*

*e*

where, *qres* is the size of the cache request response queue, *lres* is the size of the

*e e*

cache that is to be sent in response to a request by edge server *e* to its cooperating edge servers, *cres* is the number of processing cycles required to send the cache request response back. In order to keep the model simple, we do not consider the latencies incurred in the communication links since they are negligible to the latencies described earlier.

*e*

Considering, the total available cache in the system to be *C*, the cache oﬄoading ratio is denoted as *β* = *Ce* . We denote the set *S* = *s*1*, s*2*, s*3*, ..., sn* as the set of edge servers that are cooperating for the cache and *n* is the total number of

*C*

edge servers in a cooperation block. Therefore, the set of task oﬄoading ratios

are given as *B* = *β*1*, β*2*, β*3*, ..., βn*, where, Σ*n*

*i*=1

*βi* = 1.

In order to use the available cache in the system most eﬀectively, the cache accesses are processed in multiple edge servers simultaneously contributing to multiple latencies for cache access. Therefore, in order to minimize the cache allocation and access latencies, we need to minimize the maximum delay of the mobile devices and the edge servers as a min max optimization problem:

*min max Cm*(*βmC*)*, max*(*Cei* (*βei C*) + *Cei* (*βei* )*C*)

*s.t.* 1 ≤ *i* ≤ *N* (*C*1)

*N*

Σ

*βm* + *βei* = 1 (*C*2)

*i*=1

0 ≤ *βm* ≤ 1 (*C*3)

0 ≤ *βei* ≤ 1 (*C*4)

0 *< γ <* 1 (*C*5)

(7.12)

*qei*

≤ *q*

*max ei*

(*C*6)

*qreq*

≤ *q*

*qeei*

*i*

*qres*

*, qeei*

*i*

*max f*

(*C*7)*,*

where, constraint *C*1 ensures that all cache allocation and accesses should only be considered between cooperating edge servers, constraints *C*2*,C*3 and *C*4 ensure that the total available cache is to be allocated from the available cache capacities of the mobile devices and the cooperating servers, constraint *C*5 ensures that the size of the requested cache is less than the available cache and constraint *C*7 that the cache request and response queue sizes in the edge servers do not exceed the maximum speciﬁed value. Since, this min max problem requires a decentralized solution it is NP-hard.

##### Creating a cooperative caching unit

The MEC network environment is always changing depending on the number of mobile devices connected, their capabilities and the eﬀective data throughput. The current network condition would inﬂuence the choice of cache allocations and distributions for optimum latency. Therefore, each edge server periodically exchanges information with its one-hop neighbors and uses this information to determine the cooperative caching unit. If multiple servers are one-hop away from an edge server requesting the cache, they need a mechanism to determine the cooperation unit.

In order to determine the cooperation unit, the values of the processing queue

*qe*, the cache capacity *Ce*

and the cache request and response queue sizes *qreq* +

*qres*. In order to maximize performance, these parameters need to be minimized resulting in reduced processing latencies and balanced networking performance.

*e*

*e*

Each edge server in the network is completely deﬁned by a matrix of parameters

*Ce , qe , qe* , wherein *Ce* is the edge server capacity, *qe* is the size of the pro-

*i*

*i*

*f*

*i*

*i*

cessing queue and *qei* = *qrec* + *qres* is the size of the forwarding queue for cache

*f ei ei*

requests and responses. Now, the usability of the edge server is given by:

*Ui* = *α Cei*

+ *δ qei*

+ *ρ qei,* (7.13)

*ei*

*f*

wherein, *σ*, *δ* and *ρ* are the weights of *Cei* , *qei* and *q* respectively. The weights

*f*

of these parameters are determined based on their variances *νCe*, *νq* and *νqf* .

Therefore, we have:

*γ* = *νCe*

*νCe* + *νqe* + *νqf*

*,* (7.14)

*δ* = *νqe ,* (7.15)

*νCe* + *νqe* + *νqf*

*νqf*

*ρ* = *e*

*νCe* + *νqe* + *νqf*

*,* (7.16)

wherein *α* + *δ* + *ρ* = 1.

The usabilty of each edge server is therefore given by:

*Ui* = min{*Ui,j* ∈ [1*, n*]}*,* (7.17)

*j*

wherein *Ui* is the usability of the *ith* server, forming a cooperation unit with a particular edge server. This mechanism ensures that both the cache allocation and the forwarding latencies are taken into account when selecting the master edge server. Therefore, selecting an edge server with the least capacity would help to reduce the processing delay as shown in Appendix B.

##### Determining edge server capacity

The capacity of a particular edge server *Ce* is deﬁned as the cache that can be utilized within a deﬁned unit latency *Lu*. This calculated capacity is ephemeral and is based on the capacity of the edge server and its cooperating servers, the communication delay between the edge server and the cooperating servers and the communication delay between the edge server and the mobile devices.

Considering an edge server denoted as *e*1 and its cooperation servers as *e*1, *e*2

1 2 2

and *e*3, the latency at the edge server *e*1 can be computed as:

2 1

*T* + *q*

*e*1 *e*1

*L* 1 = 1 1 + *Ltf*

*e*

*e*

1 *f* 1 1

*e*

1

1

*r*

+ *Le*1−*m*

1

*f*

*,* (7.18)

where, *Ts*1 is the size of the cache allocated to the edge server *e*1, *fe*1 is the pro-

1 1 1

cessor frequency for edge server *e*1, *Lrf*

is the latency incurred in transferring

* + - 1. *e*1−*m*

1

the contents of the cache in *e*1 to the mobile device, *Ltf*

is the latency incurred

1 1

*e*

1

in *e*1 for splitting and transferring the cache requests to the cooperation servers

1

*e*1*, e*2 and *e*3.

2 2 2

The latencies for the other edge servers in the cooperation block are:

*T*

*e*1

*L* 1 = 2 + *Lrf*

*e*

*,* (7.19a)

* + - 1. *ci* 1

2

*e*2

*Te*2

*e*1−*m*

*rf*

*L* 2 = 2 + *L*

2

*e*

*,* (7.19b)

2 *ci* 2

*e*2

*Te*3

*e*2−*m*

*rf*

*L* 3 = 2 + *L*

2

*e*

*,* (7.19c)

2 *ci* 3

*e*2

*e*3−*m*

wherein, *Lrf*

*s*1−*m*

2

is the latency incurred in moving the result of edge server *e*2

from edge server *e*1 to the mobile device, *ci*1 the is capacity of the edge server *s*1

2

1 *s*2 2

at the *ith* interval. The latency incurred in moving the cached data of edge server

*e*2 from *e*2 to *e*1 is used in computing the capacity of the cooperation block.

2 2 1

The capacities for the cooperation block can be calculated as:

*C* = *c*0 *L*

*tf rf*

*e*

*e*

*l* 1

1

1

1

*u*

— *L* − *L*

− *q* 1 *,* (7.20a)

*Cli* 1

*e*2

*Cli* 2

*e*2

*Cli* 3

*e*2

= *li*1

*e*2

= *li*2

*e*2

= *li*3

*e*2

*Lu*

*Lu*

*Lu*

*rf ,* (7.20b)

2

*s*

1

— *Ls* −*m*

1

1

*e*1−*m*

*e*1

1

*rf ,* (7.20c)

— *Ls* −*m*

2

2

*rf ,* (7.20d)

— *Ls* −*m*

3

2

wherein, *Le*1 = *Lu*, *Le*1 = *Lu*, *Le*2 = *Lu* and *Le*3 = *Lu*. Therefore, the actual

1 2 2 2

capacity of *e*1 is:

1

3

Σ *j*

*ci*1 = *Cl*

*e*

*s*

1

1 1

+ *C li ,* (7.21)

*s*2

*j*=1

wherein, *le*1 is the capacity of *e*1 and the cooperation block composed of *e*1, *e*2

1 1 2 2

and *e*3.

2

Now, the capacities of the edge servers can also be expressed as:

*c*0 *L*

1

— *L* − *L*

*tf rf*

*e*

1

+ *q ,*

1

*e*

1

*rf* + *Ltf* + *qe*1

≤ *L*

1

*e*

*c*

*e*

*Cl* 1 =

*e*1

⎨ *e*1 *u*

⎪

0*, if L*

1 *e*1−*m*

1 *if Lu > Le*1−*m*

1

1 0

1 1

1 (7.22a)

*q*

*f e*

1

*t* 1

+ *L* + 1 *,*

*Clij*

*e*2

⎧⎪⎨*lij Lu*

⎪

*s*2

=

*rf , if Lu*

2

— *Ls* −*m*

*j*

*u*

*r*

*f*

*> L j*

*e*2−*m*

*rf*

⎩0*, if Lu* ≤ *Lej* −*m*

2

*e*1−*m*

1 0

1 1

*rf*

*e*

*e*

*c*

1

(7.22b)

Therefore, for the edge server *e*1, cache should only be allocated if the request for- warding latency, the result forwarding latency and the processing queue latency are greater than the unit latency *Lu*.

1

In a cooperative caching block, the sizes of the request forwarding queue and

the allocate cache sizes might be diﬀerent based on the respective edge server’s capacity. Therefore in order to evaluate the degree of the request forwarding delay, the following parameters are deﬁned:

*k j* =

*e*

2

*Lrf*

*ej* −*m*

2

*Lu*

*,* (7.23a)

*Lt* + *L*

*e*

*f*

1

1

*q* 1

1

+

*e*

*rf*

*e*1−*m c*0

1

*e*1

*ke*1 = 1 *,* (7.23b)

1 *Lu*

wherein, *k j*

*e*

2

and *k* 1 are the parameters to determine the degree of the request

1

*e*

forwarding latency. For the cooperation block to be eﬀective the parameters

should be bounded by 0 *< ke*1 *<* 1 and 0 *< ke*2 *<* 1. This guarantees that the

1 *j*

cooperation block would be able to allocate the cache and process requests in

a reasonable time. As the value of *k j*

*e*

*i*

increases, the cache request forwarding

latency *ej* also increases resulting in increased latencies.

*i*

##### Cache allocation in cooperating edge servers

Caches allocated on cooperation blocks are feasible if the latencies between the mobile device that is requesting the cache and the edge servers in the cooperation block are the same. If the latency to a particular edge server in a cooperation block is lower than the others, then all caches would be allocated to that server subject to cache memory availability. Therefore for a cooperation block con-

*e*1 2 2 3

sisting of 4 edge servers *B* 1 (*e*1*, e*2*, e*2), *Le*1 = *Le*1 = *Le*2 = *Le*3 = *Lm*, wherein

1 2 2 2

*Lm* is the latency incurred by the mobile device. Moreover, the cache allocation mechanisms needs to ensure that requested caches from mobile devices should be allocated to the cooperation block. For a cooperation block consisting of 4

*e*1 2 2 3

edge servers *B* 1 (*e*1*, e*2*, e*2), (1 − *α*)*P* = *Pe*1 + *Pe*1 + *Pe*2 + *Pe*3 , wherein, *P* is the

1

2

2

2

total size of the requested cache, *Pe*1 *, Pe*1 *, Pe*2 and *Pe*3 are the caches allocated

to *e*1*, e*1*, e*2 and *e*3 respectively.

1 2 2 2

1 2 2 2

Denoting the capacity of edge server *e*1 as *ci*1, the cache allocation between the

mobile device *m* and *e*1 is:

1

⎧⎪*L*

1 *e*1

= *αP* + (*a* − *α*)*P f*

*m fm cm*

⎪

1

⎪⎨*L* 1

= (1−*α*)*P*

(7.24)

*e*1 *ci*

*e*

1

⎪

1

⎪⎩*Lm* = *Le*1

wherein,

*fm*(1 − *cf ci* )

*α* = 1 *,* (7.25)

*m e*1

1

*m e*1

*ci* 1

*e*1

+ *fm*

(1 − *cf ci* )

and

*P*

*Lm* = *Le*1 = *f*

*e*1

1

(7.26)

*ci* 1

*e*1

+ *fm*(1 − *cmci*1)

where, *fm*

(1 − *cf ci*

) is the cache capacity of the mobile device.

1

When the edge server *e*1 receives a request for allocating a cache of size (1 − *α*)*P* ,

*m e*1

1

it splits the cache between *e*1*, e*2*, e*2 and *e*3. Now, for the cooperation block

1 1 2 2

*e*1 2 2 3

*B* 1 (*s*1*, s*2*, s*2), the latencies are:

*Lrf* = *qrf* + *γP* 1 *cf*

*e*

*e*

*e*1−*m*

1

1

*e*1

1

1

1

(7.27a)

*Ltf* = *qtf* + (1 − *α*)*P*  *cf*

*e*

*e*

*e*

(7.27b)

*Lrf*

2

1 1

1 1

= *qtf* + *P*

*e*

*s*

*sj* −*m*

1

1

1

1

(7.27c)

*j*

2

*e*1 2 2 3

and the caches allocated to each server in the cooperation block *B* 1 (*s*1*, s*2*, s*2)

are:

*c*0 *Lm* − *cf* (1 − *α*)*P* − *qrf* + *qtf*  *cf*  − *q* 1

*e*1

*e*1

*e*1

*e*1

*e*1

*e*1

*P* 1 = 1 1 1 1 1 = 1 1

*c*0 *g*(*P* ) − *q*

*e*1

*e*1

(7.28a)

*e*1 *γcf c*0 + 1

*e*

*e*

*γcf c*0 + 1

*ci*  *Lm* − *cf qrf*

*e*1

*e*1

*e*1

*e*2

*f*

1 1 1 1

1 1 1 1

*e*

*e*

*ci L* − *θ*

*e*1

*m*

*e*1

*P* 1 = 2 1 1 = 2 1

(7.28b)

*f*

*γci*1 *c* + 1

*e*1

*e*2 1

*γci*1 *c* + 1

*e*2 1

*e*1

*ci*  *Lm* − *cf qrf*

*e*2

*e*1

*e*1

*e*2

*m*

*e*1

*P* 2 = 2 1 1 = 2 1

(7.28c)

*e*2

*f*

*ci L* − *θ*

*f*

*γci*2 *c* + 1

*e*1

*e*2 1

*γci*2 *c* + 1

*e*2 1

*e*1

*ci*  *Lm* − *cf qrf*

*e*3

*e*1

*e*1

*e*3

*m*

*e*1

*P* 3 = 2 1 1 = 2 1

(7.28d)

*e*2

*f*

*ci L* − *θ*

*f*

*γci*3 *c* + 1

*e*1

*e*2 1

*γci*3 *c* + 1

*e*2 1

*e*1

where, *θ* 1

*rf f*

*e*1

*e*1

*e*1

1

1

1

1

*m* − *cf* (1 − *α*)*P* − *σ* 1 , and *σ* 1 = *qrf*

*tf f* .

1

1

The ratio of the caches allocated to the edge servers comprising the cooperation

*e*1 = *qe*1 *ce*1 , *g*(*P* ) = *L*

*e*1 + *qe*1 *ce*1

*e*1 1 2 3

block *B* 1 (*e*2*, e*2*, e*2) is:

*Pe*1

*βe*1 = 1

(7.29a)

1 *Pe*1 + *Pe*1 + *Pe*2 + *Pe*3

1 2 2 2

*Pe*1

*βe*1 = 2

(7.29b)

2 *Pe*1 + *Pe*1 + *Pe*2 + *Pe*3

1 2 2 2

*Pe*2

*βe*2 = 2

(7.29c)

2 *Pe*1 + *Pe*1 + *Pe*2 + *Pe*3

1 2 2 2

*Pe*3

*βe*3 = 2

(7.29d)

2 *Pe*1 + *Pe*1 + *Pe*2 + *Pe*3

1 2 2 2

wherein, *βe*1 + *βe*1 + *βe*2 + *βe*3 = 1. In order to ensure that (1 − *α*)*P* = *Pe*1 +

1

2

2

2

1

*Pe*1 + *Pe*2 + *Pe*3 , the cache sizes allocated to edge servers in the cooperation block

2 2 2

*e*1 1 2 3

*B* 1 (*e*2*, e*2*, e*2) are:

*Pe*j1 = *βe*1 (1 − *α*)*P*

1

1

*Pe*j1 = *βe*1 (1 − *α*)*P*

2

2

*Pe*j2 = *βe*2 (1 − *α*)*P*

2

2

*Pe*j3 = *βe*3 (1 − *α*)*P*

2

2

Now, *Pe*j1 + *Pe*j1 + *Pe*j2 + *Pe*j3

1

2

2

2

= (1 − *α*)*P* (*βe*1 + *βe*1 + *βe*2 + *βe*3 ) = (1 − *α*)*P* .

However, repeated computation of the cache sizes allocated to the cooperation

1

2

2

2

*e*1 1 2 3

block *B* 1 (*e*2*, e*2*, e*2) can cause the latency in these edge server to diﬀer from

*Lm* by small amounts. Therefore, it is prudent for the mobile device to request cache allocation at the edge server only when the capacity of the edge server *ci*1

*e*1

satisﬁes the following constraints:

*ci* 1

*e*1

*ci* 1

*e*1

*fmcf*

*m*

1

*< f , or,*

*c*

*m*

*fm*

*> f cf* − 1 *, and,*

*m m*

*>* 1

Conversely, if 1

*c*

*m*

*m* 1

*< ci*

*<  fm* , the mobile device should only allocate cache

locally.

*e*

*f*

*f* 1 *mcf* −1

As the size of the allocated cache in the cooperation block *P* increases, the sizes

of the cache in the associated edge servers *Pe*1 *, Pe*1 *, Pe*2 and *Pe*3

increases non

1 2 2 2

uniformly. When the mobile device is able to request cache, for edge cache *Pe*1 ,

1

*fm cf* −*cf*

*e*

if *α >*

*fm*

1

1

*cf* −*cf*

*e*

*m*

+1

, then *P* 1

1

*e*

increases with an increase in *P* . Conversely, if

1 *m*

1

1

*e*

*fm cf* −*cf*

*α <* 1

*fm cf* −*cf*

*e*

*m*

+1

, then *P* 1 decreases with an increase in *P* . However, the values

1

*e*

1 *m*

1

of *Pe*1 *, Pe*2 and *Pe*3 always increases with an increase in *P* . This is because when

2 2 2

*fm cf* −*cf*

*e*

*e*

1

*α >* 1

*fm cf* −*cf*

*m*

+1

, then more cache is allocated locally on the mobile device

1 *m*

1

*fm cf* −*cf*

*e*

and the value of *Lm* increases. Conversely, when *α <*

*fm*

1

1

*e*

*cf* −*cf*

*m*

+1

, most of the

1 *m*

1

cache would be allocated on the edge servers.

As the size of *P* increases, the latency increases in the mobile devices since the

1

1

value of *α* is low increasing the fraction of the cache allocated to the edge servers.

Since *e*1 is also responsible for splitting and allocating the individual caches on

the cooperation block, the latencies on *e*1 also increases reducing the size of the

cache that is available on *e*1 since *Lm* is small. If most of the cache is allocated on the mobile devices themselves, the size of the cooperation block is small since allocating caches on multiple edge servers only serves to increase the cache access latencies. Conversely, if most of the cache is allocated to the edge, most of the latencies would arise from the edge, thus incentivizing *e*1 to distribute the cache to the other edge servers in the cooperation block thereby reducing latencies in the edge servers.

1

1

##### Eﬀective cooperative caching

Cache allocation and distribution at the edge servers is a recursive process. For

the cooperation block *B* 1 (

|  |  |  |
| --- | --- | --- |
| *e*1 | 1 2 3 | 1 |
| *e*1 | *e*2*, e*2*, e*2), all the edge servers other than *e*1  to continue allocating caches and building | |

execute the process at 1

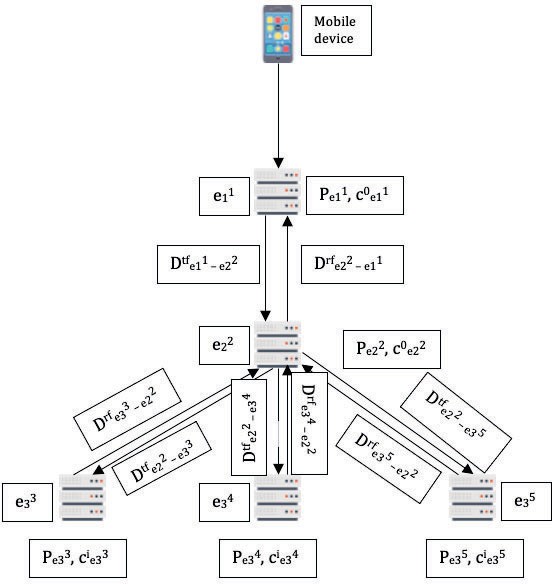
recursively

the coop-

eration blocks. This process would continue until creating further cooperation

blocks cannot improve the latencies anymore.

Figure 7.2: Achieving maximal cooperation



For the cooperation block in Figure 7.2, the mobile device *m* is directly connected

1 *e*1 1 2 3

to the edge server *e*1, which forms the cooperation block *B* 1 (*e*2*, e*2*, e*2) with edge

servers *e*1*, e*2 and *e*3 in the second level. The edge servers in the next level can

2 2 2

also form their own cooperation blocks as shown in Figure 7.2.

*ej j*

For a cooperation block *B i* , the master edge server is *ei* in level *i* and its

cooperating servers are in level (*i* + 1). Considering that the requested cache has been allocated into the cooperating servers the latencies of the edge servers in

level *i* is:

*j*

*Lej*

*Pej*

0

=

*i*

2

+ *Lrf*

Σ

*ej* −*el*

2

+ *Ltf*

Σ

*el*

−*e* + *μej*

(7.32)

*i c j*

*ei*

*n*=*i*

*n n*−1

*n*=*i*

*n*−1 *n i*

where, *ej* refers to the *jth* edge server at the *nth* level, *el*

refers to the *lth* server

*n*

at the (*n* − 1)*th* level, *el*

*n*−1

*ej*

*t*

*f*

*t*

*f*

*n*−1

is the master edge server of *ej* , *Lrf*

*n*

*ej* −*el*

*i*−1

Σ2

*i*

*q j e*

= *qrf* +

*i*

*r*

*f*

*γP j c j* , *L*

= *q* + *γP j c j* and *μ j* = *i* . The values of *L*

*ei ei*

*ej* −*el*

*ej ei ei*

*ei c*0

*e*

*n*=*i*

*ej* −*el*

*i i* 1 *i j*

−

*i*

*n n*−1

and Σ2

*j*

*Ltf*

*el*

*n*−1

−*en*

are computed by sending the values of the request and result

cache forwarding queue sizes from the master to its cooperating edge servers

*n*=*i*

between adjacent levels in the cooperation block. The value of *P j* is computed

*e*

*i*

by recursively performing the computations as performed by the master edge server *e*1. Therefore, the cache cooperation block computation would complete

1

when:

*j*

*Lm* = *μej*

2

+ *Lrf*

Σ

*ej* −*el*

2

+ *Ltf*

Σ

*el* −*e*

(7.33)

*i*

*n*=*i*

*n n*−1

*n*=*i*

*n*−1 *n*

As caches are requested and allocated on the edge cooperation block, the depth of cooperation levels varies for diﬀerent branches in the cooperation network.

2

3

1

2

3

Considering Figure , the branches *e*1

1

→ *e*2

→ *e*3

and *e*1

→ *e*2

→ *e*4

have

associated cache request and response latencies *Lrf tf*

*rf tf ,*

*e*1−*e*1 *, Le*1−*e*1 *, Le*1−*e*1 *, Le*1−*e*1

2

1

1

2

2

1

1

2

*Lrf*

*e*1−*e*1

2

1

and *Ltf*

*e*1−*e*1

1

2

that are in turn related to the sizes of the caches in *e*3 and

*e*4 and the queue sizes in *e*1 and *e*2. However, while the cache sizes in *e*3 and

2

3 1 2 3

*e*4 are diﬀerent, the queue sizes of *e*1 and *e*2 are the same. Now, if *Pe*3 *> Pe*4 ,

3 1 2 3 3

then *L* 3

*e*

3

*> L* 4 . Therefore, no new edge servers in the cooperation branch

3

*e*

*e*1 → *e*2 → *e*3 would be added while the *e*1 → *e*2 → *e*4 cooperation branch would continue to have new edge servers added.

1

2

3

1

2

3

The values of the maximum and minimum allowed latencies for the cache request and response queues are:

*j*

*rf*

*L*

min

= min,*Lrf*

*r*

*f*

*l ,L l*

*, ..., Lrf*

*h*

*k*

1, (7.34)

*Lrf*

*ei* −*ei*−1

= max,*Lrf*

max

*j*

*l*

*ei* −*ei*−1

*ei*−1−*ei*−2

*rf*

*,L l*

*ei*−1−*ei*−2

*h*

*e*2 −*e*1

*, ..., Lrf*

*k*

1

*e*2 −*e*1

, (7.35)

*tf*

*L*

min

= min,*Ltf*

*l*

*t*

*j ,L h*

*f*

*, ..., Ltf*

*l*

1

*k* , (7.36)

*Ltf*

*ei*−1−*si*

= max,*Ltf*

max

*l*

*j ,L h*

*ei*−1−*si*

*ei*−2−*ei*−1

*tf*

*ei*−2−*ei*−1

*l*

*e*1−*e*2

*, ..., Ltf*

1

*k*

*e*1−*e*2

, (7.37)

*μmax* = max *μej , μel , ..., μe*1

(7.38)

*i i*−1 1

*μmin* = min *μej , μel , ..., μe*1

(7.39)

*i i*−1 1

Now, since *μ*

min

+ (*n* − 1) *Lrf*

*tf*

min

+ *L*

*< Lm*

*< μ*max

+ (*n* − 1) *Lrf*

*tf*

max

+ *L*

,

the number of cache cooperation levels is bounded by:

min

max

*Lm* − *μ*max + 1 *< n < Lm* − *μ*min + 1 (7.40)

*rf*

*L*

+ *L*

*L*

+ *L*

max

*tf*

max

*rf*

min

*tf*

min

##### Analyzing cooperation performance

The performance of this cooperative caching caching mechanism was evaluated by comparing to classical centralized caching mechanisms. The approximation ratio is Δ = *Lm* , wherein *L*∗ is the optimal cache allocation latency based on centralized caching mechanisms and:

∗*L*

*Lm*  *Lm*

*<* Δ *< ,*

min

*μmax*

max

+ *L*

+ *L*

+ (*n* − 1) *Lrf*

*tf*

max

*μ*min

+ (*n* − 1) *Lrf*

*tf*

min

which ensures that the mechanism is at most *L*max

*L*

min

— *approximation* as shown

in Appendices G and H.

#### Simulation and Analysis

We have used a WSNet-based simulator written in C/Modern C++ [265]. The structure of requests was simulated using a standard open source web-traﬃc trace generator [266]. The trace generator is a single threaded implementation of the cache admission and eviction policies written in C++ and is an input for WSNet. We consider an object space consisting of 5 million unique objects serving around 20 million user requests. The set of simulation parameters are given in Table 7.2.

Table 7.2: Simulation Parameters

|  |  |
| --- | --- |
| **Parameter** | **Value** |
| Total content requests | 20 million |
| Unique contents | 5 million |
| File sizes | 1 MB, 5 MB and 20 MB |
| Local Cache capacity | 0.05 GB, 0.12 GB, 0.25 GB |
| MEC Cache capacity | 0.2 GB, 0.8 GB and 4 GB |

#### Results

We have run our simulations on the traﬃc traces by varying the parameters as described in the previous section. The abscissa values are the average content size (MB), the local cache capacity, the MEC cache capacity and Zipf’s parameter respectively. Figures 7.3 and 7.4 depict the total number of reads and writes

respectively from the local and MEC caches. We see that as the fraction of writes handled by the MEC cache decreases, the amount of reads handled by the MEC cache increases. As reads are generally faster, this result demonstrates that our mechanism is write access frequency aware.

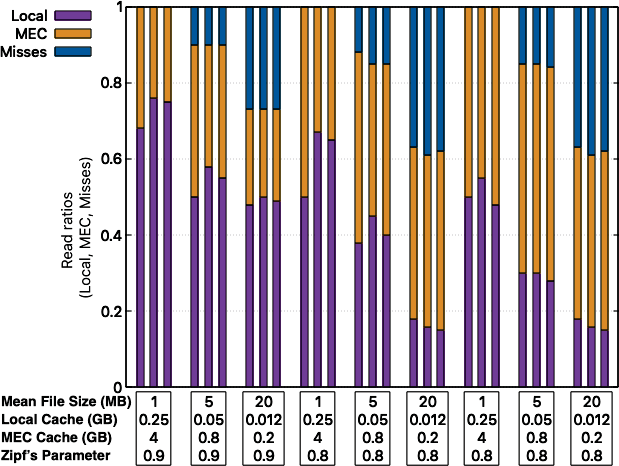


Figure 7.3: Read ratios in Local and MEC caches with misses.

As we increase the average ﬁle size for a ﬁxed Zipf’s parameter, we observe that the size of both local and MEC caches decrease leading to more cache misses. However, most content access requests are handled by a combination of the local and MEC caches even for very small cache capacities. If we keep the average size of the ﬁle constant, increasing the Zipf’s parameter results in more requests being handled by the local cache which reduces the overall content access time. This is because, a skewed distribution allows easier estimation of a particular ﬁle’s popularity. Moreover, the ﬁgures also show that increasing the Zipf’s parameter

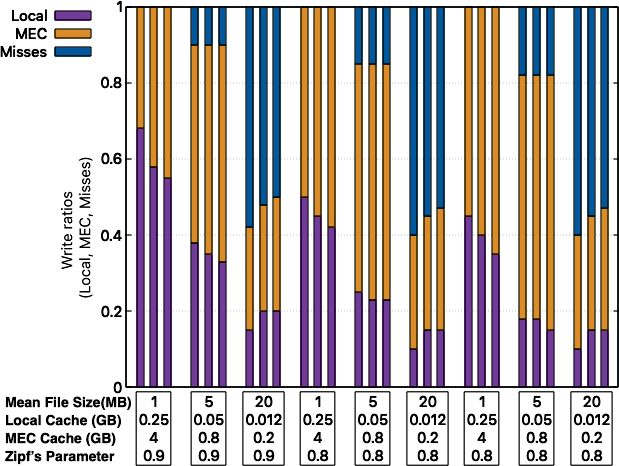


Figure 7.4: Write ratios in Local and MEC caches with misses.

increases the cache hit ratio and that most of the access requests can be handled even with local and MEC caches operating at a small fraction of their required capacity. Empirical evidence suggests that the value of Zipf’s parameter is close to 1 in modern web traﬃc.

Figure 7.5 shows the mean ﬁle size as stored in the local and MEC caches. We observe that the mean ﬁle size stored in the MEC cache is larger than the local cache. This helps with overall access times as the MEC cache can be read as a block device and a larger amount of data can be accessed with the same access penalty. This observation proves that our mechanism is content-size aware.

Figure 7.5 also shows a comparison of the average ﬁle size with respect to Zipf’s parameter. We observe that as the cache capacity decreases, the average ﬁle size

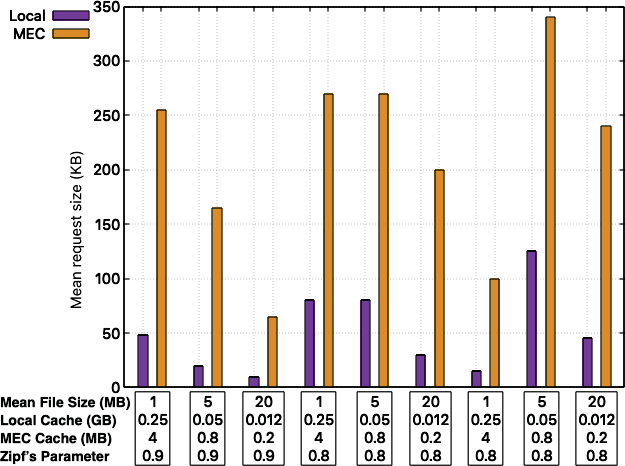


Figure 7.5: Mean request sizes (KB) in Local and MEC caches.

in the cache also decreases. This is because our popularity determination metric prioritizes the smallest and the most popular ﬁles.

A comparison would help us evaluate the eﬀectiveness of our policy in a real- world environment wherein the statistics of data content such as the popularity are determined on the ﬂy and therefore can be noisy. In comparison an oﬄine policy computes the popularity, the number of write and read requests and their sequence for each request in advance. The results from the comparison of the two policies for reads and writes are shown in Figures 7.6 and 7.7 respectively.

Considering average statistics for simplicity, the average number of read requests handled by the local and MEC caches and cache misses are in the ratio 45*.*5 : 36*.*4 : 12*.*6 and 41*.*7 : 42*.*2 : 16*.*2. The average number of write requests handled by the local and MEC caches and cache misses are in the ratio 50*.*31 : 23*.*4 : 24*.*3

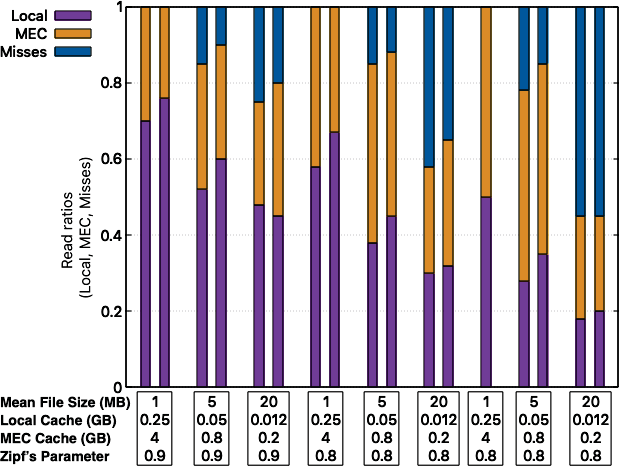


Figure 7.6: Comparison of online and oﬄine policies for reads in Local and MEC caches with misses.

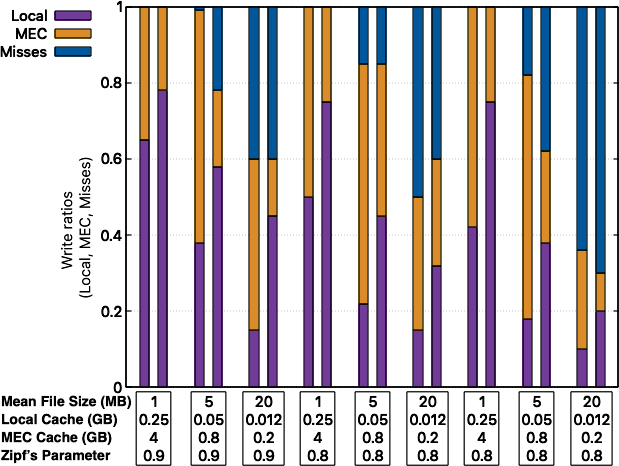


Figure 7.7: Comparison of online and oﬄine policies for writes in Local and MEC caches with misses.

and 50*.*69 : 27*.*65 : 20*.*33. From these ﬁgures, we can observe that our online policy is quite good in handling read requests as compared to write requests. This is because the number of read requests are always higher than writes requests in an real world system. This large number of read requests help in forming a better estimate of the popularity. Heuristics can be used to improve the popularity estimation in write requests as there is not enough data to form an eﬃcient estimation.

In order to evaluate the performance of the cooperative caching mechanism, We have used a custom WS-Net based simulator for simulating the edge computing network with 100 edge servers. Each edge server can connect to a maximum of 5 neighbor servers to share their cache. We consider 5 GB of allocated cache space in each server. Cache requirements are uniformly distributed between 1 and 10 MB. The computing capacity of the edge servers range from 1 to 10 GHz.

In order to evaluate the performance of the mechanism, we have compared with traditional LRU and LFU cache control and replacement mechanisms. For these mechanisms, traditional edge caching is considered wherein a mobile device shares its cache with a singular edge server over the duration of the compu- tation. Tests were also carried out with the entire cache on the mobile device as a control mechanism. We have evaluated the performance by varying the number of available edge servers, varying the requested cache size, varying the cache request rate and under varying processing capabilities. For each simulation we have compared the network throughput and the processing latency as per- formance metrics and have shown that our mechanism outperforms traditional caching in edge computing.

Figure 7.8: Variation of network performance with available edge servers

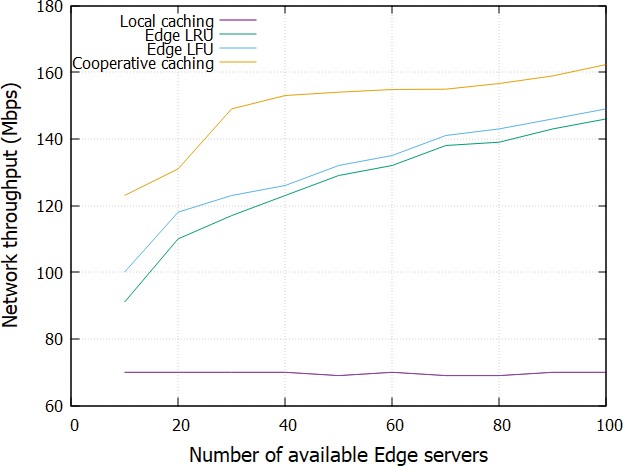
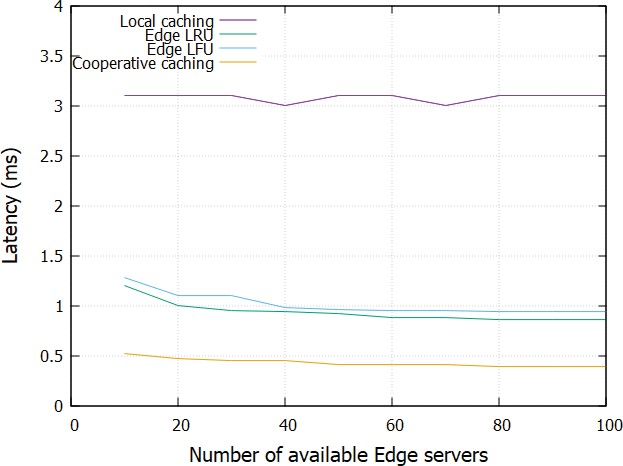


Figure 1 shows that the network throughput increases as the number of available edge servers increase for all algorithms except Local. This is because, the local caching mechanism allocates all cache requests on the mobile device’s cache and therefore does not depend on the availability of the edge servers. The cooperative caching mechanism performs better than traditional LRU and LFU strategies. However, we ﬁnd that the rate of increase in the network throughput tapers oﬀ as the number of available edge servers increase beyond a threshold. This is because the signiﬁcance of the increased availability of edge servers has a lesser impact as the cache requirements are already satisﬁed.

Figure 2 shows the variance of the incurred latencies and the various algorithms. Once again, the latencies reduce with an increase in the number of available edge servers except for the local caching mechanism which does not depend

Figure 7.9: Variation of latency with available edge servers



on the network cache. Cooperative caching has the lowest latencies of all the mechanisms. However, the rate of decrease of the incurred latencies decrease once the number of available edge servers go beyond a threshold signifying that increased number of servers have a diminishing eﬀect over the overall system latency.

Figure 3 shows the variation of the network throughput with changes in the re- quested cache sizes. As the sizes of the requested caches increases, the network throughput decreases for all the algorithms. Once the requested cache size in- creases beyond 4 Mb, there is a drastic reduction in the network throughput. This is because larger sizes of requested caches increases the requested cache queue length in the edge servers, increasing the processing required and reduc- ing the network throughput.

Figure

7.10: Variation of network throughput with changes in requested

cache sizes

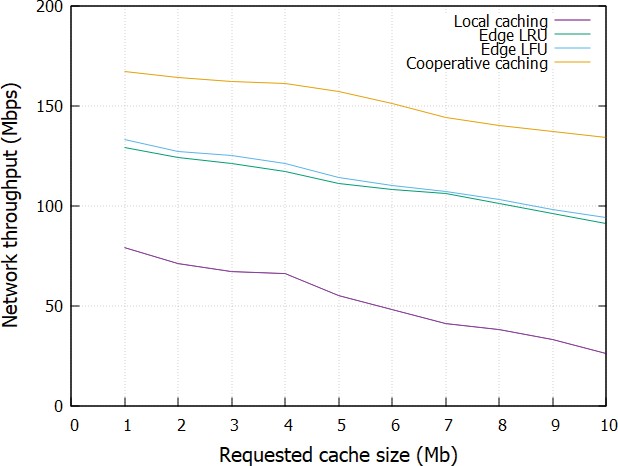


Figure 7.11: Variation of latency with changes in requested in cache sizes

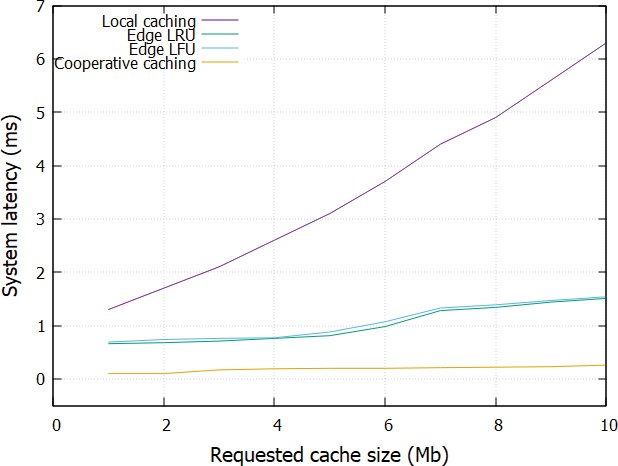
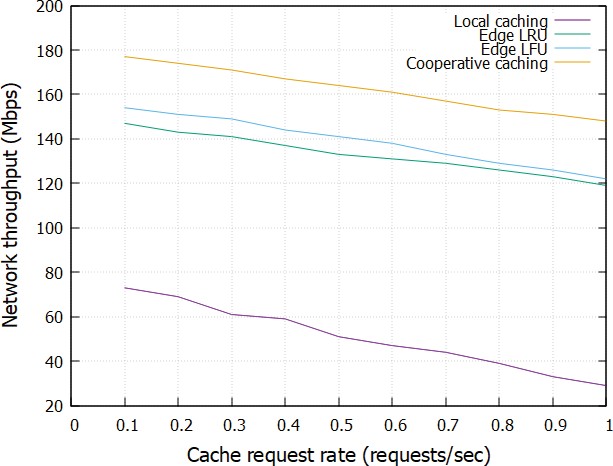


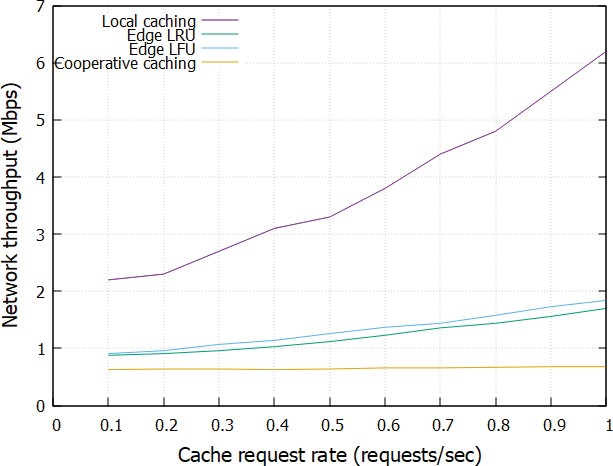
Figure 7.12: Variation of network throughput with cache request rate



A similar observation for the system latency is observed in Figure 4. Increasing sizes of the requested cache put more strain on the system thereby increasing its latency. Once the requested cache sizes go beyond the threshold, it increases the system latency signiﬁcantly as most of the requested caches are now distributed across the cooperating edge servers and require extra processing to retrieve them.

In order to measure the performance of the cooperative caching mechanism with the rate of cache requests, the cache request size is ﬁxed at 5 Mb with 80 edge servers. This conﬁguration is maintained for the other mechanisms to evaluate the performance. It is observed that the network throughput decreases with increases in the rate of cache requests. This is because as the rate of cache requests increases a lot of processing needs to be performed in order to allocate the caches thereby decreasing the throughput. Cooperative caching performs

Figure 7.13: Variation of latency with changes with cache request rate



better than other mechanisms and the rate of decrease with increasing cache requests is also less.

The processing latency increases with increases in the cache request rate. Once again cooperative caching performs better than the other mechanisms and the rate of decrease is better than the other mechanisms.

Figure 7 shows that the network throughput increases with increase in the CPU frequency. The network throughput for the local mechanism does not change since all computations do not make use of the network. The performance of cooperative caching exceeds the performance of traditional caching mechanisms.

In a similar manner the latency for the local mechanism does not change as the network takes no part in purely local computations. However, the cooperative

Figure 7.14: Variation of network throughput with CPU frequencies

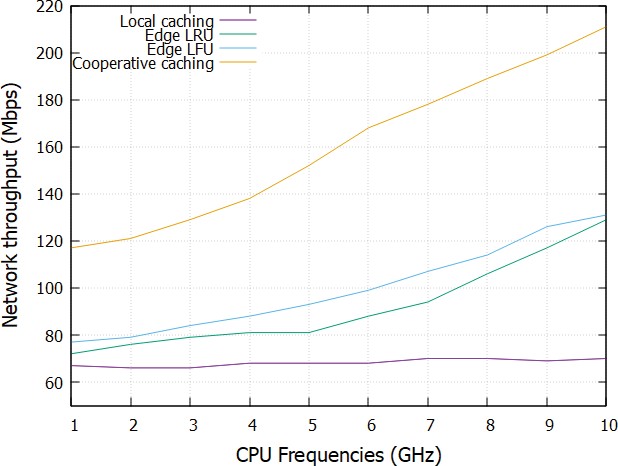
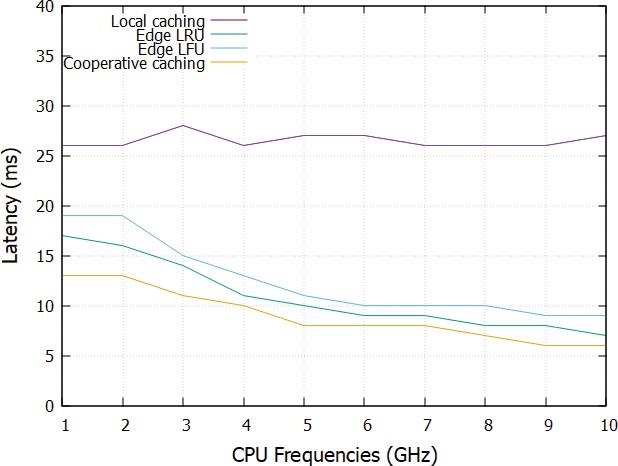


Figure 7.15: Variation of latency with CPU frequencies



caching still outperforms established cache control mechanisms.

In this chapter, we have explored a hierarchical variant of the greedy algorithm for caching policies in MEC environments. Our policy is aware of the penal- ties associated with remote writes on MEC caches and the size and frequency of requested contents. We have optimized the policy by computing the *rank- threshold* product that helps to ﬁnd the optimum convergence. We have com- pared our policy against established policies through simulations and have shown the superiority of our policy.

**Chapter 8**

# Summary and Future Work

The number of devices connected to the Internet is increasing steadily over the last few years and is only expected to go up. Modest estimates put the number at a few hundred billion while some researchers are even looking at numbers surpassing a trillion. While this unprecedented levels of connectivity has made the world smaller, it is putting enormous pressure on existing network infras- tructures. The existing infrastructure of the Internet was not built to support so many devices. However, given the amount resources that have been spent over the last few decades for expansion and connect almost all of humanity, it is infeasible to build an entirely new Internet architecture from scratch. Therefore, it is only prudent to work around these limitations and use existing resources and tools to optimize the existing networks.

119

#### Summary of Results

Our simulations were carried out using the ns-3 and the WSNet simulators. In certain scenarios we had to write custom modules to extend the functionality of the simulators and ﬁt our test cases. Both ns-3 and WSNet are discrete event simulators and are capable of simulating the entire network stack including fading models and path losses in wireless links. Edge nodes were simulated using virtual machines to provide more computational resources representative of actual edge nodes.

In order to explore the problem of dynamic computation oﬄoading and resource allocation in IoT networks, we formulated an multi-variate optimization prob- lem and solved it using a variation of the Lyapunov optimization technique by splitting the problem into sub-problems over discrete time-slots.

In order to tackle the energy problem, we simulated a distributed optical charac- ter recognition system based on Tesseract and compared the performance with local execution. We formulated a multi-objective optimization problem using cubic splines that minimizes latency and maximizes the accuracy of the compu- tation adhering to an established power envelope. Our simulations proved that this mechanism scales well for large networks without overwhelming individual computing devices.

Extending our simulations to include multiple intermediate computing levels showed that performance can drastically improve over the ﬁxed cloud oﬄoading model. Each level consists of more powerful computing hardware until the level reaches the cloud which we assume to always have enough computing capacity.

Having multiple computing levels allowed us to build a hybrid model of computa- tion oﬄoading wherein computationally intensive parts can be further moved up the computing levels while other tasks can be processed locally. This multi-level oﬄoading mechanism allowed us to support extremely low latencies suitable for highly interactive applications.

While optimizing computation at the user device level has been promising, we have also extended our work from the point of view of the edge nodes. Edge nodes are responsible for handling multiple user devices. This makes it an inherently hard problem. We have modeled the computation oﬄoading process as a game to arrive at a solutions. Our work has shown that the process admits a Nash equilibrium proving that the system has achieved and optimum level of oﬄoading and no further changes can improve the process.

With the availability of mobile computing devices, general computing have be- come ubiquitous. While these devices have modest computing resources, they have achieve substantial results when working in tandem. This opens the pos- sibility for cooperative computation with task oﬄoading. The problem becomes more diﬃcult when multi-hop ad-hoc networks are considered as computing nodes need to keep track of the sub-tasks which may be related but might be computed on nodes that are multiple hops away. We have shown that this is a non-trivial and non-convex problem and have established the necessary and suﬃcient conditions for a globally optimized solution.

We also made a brief foray into caching at the network edge. Caching at the edge is important in mobile networks since devices are always moving in and out

of the service area and the edge has to ensure handoﬀs are completed success- fully without a lot of overhead. We explored a hierarchical variant of the greedy algorithm for the caching policy. Our policy is aware of the penalties associ- ated with remote writes on MEC caches and the size and frequency of requested contents. We have optimized the policy by computing the *threshold-rank* prod- uct that helps in optimum convergence. We have compared our policy against established policies through simulations and have shown the superiority of our policy.

Similar to the problem of task oﬄoading and computation in multi hop networks, caching can also perform better and faster in cooperative mode. The very nature of the cache ensures that both mobile devices and edge servers have a limited amount of cache. Therefore, caches must be allocated intelligently to ensure that the limited resource is used optimally. Moreover, the caching strategy should en- sure a high cache hit ratio in order to make moving objects in and out of the cache feasible.We have considered a scenario wherein mobile devices take advan- tage of the storage available in the edge servers to cache data. Since each edge server is connected to multiple other edge servers, it can choose to cooperate with other edge servers and extend the cache available to the users. This coop- eration between edge servers would be dynamic depending on the performance requirements and prevalent network conditions. The devices just see an extended available cache in addition to their local on board cache. The extended cache is seamlessly distributed across multiple edge servers. We compare the perfor- mance of traditional caching mechanisms at the edge that work on singular edge servers and show that this multi hop cooperative caching mechanism performs

better and provides larger caches thereby improving application performance.

#### Future Work

Based on the work described in this dissertation, the interested reader is encour- aged to peruse the following directions of further research.

* + - Our simulations using machine learning to predict network congestion have shown their feasibility by predicting network bandwidth with a very high degree of accuracy. These models can be used in a Software Deﬁned Net- working platform to work in real-time and take proactive measures against network congestion. This mechanism can also improve the performance of network load balancers by maximizing their throughput.
    - The Linux network analysis tool *Dstat* collects a lot of information from network interfaces that generates a lot of feature set data. It should be possible to utilize dimensionality reduction to create smaller feature sets and evaluate their performance. Reducing the number of features should also reduce the computation and bandwidth requirements.
    - Network congestion and reduction in the available bandwidth capacity are also related to the availability of other computing resources in the network notably RAM and the load on the CPU. If those resources are not available, it may also result in reduced network throughput. The machine learning models can be updated to include a holistic approach to availability of computing resources to have better estimates of network congestion.
    - Our studies on cooperative computation oﬄoading have shown that it per-

forms better than traditional ﬁxed oﬄoading. Moreover, we have also shown that performance is also related to the network topology. Since the topology is generated on the ﬂy, studies can be performed to see how the performance varies if we allowed to fully reconﬁgurable topologies based on the current computational load on the network. High performing nodes could be incentivized to keep track of the best performing topologies and reconﬁgure the network for maximum performance. Machine learning al- gorithms can also be used to assist in reconﬁguration decisions.

* + - From our simulations, we have seen that optimal computation oﬄoading is possible when nodes work in batches. This is due to the temporal local-

ity of most computational tasks. Nodes in a cluster are generally located closer together or connected over low latency links. Sometimes, the nature of the computation necessitates clusters to communicate to share data and intermediate computation results. Studies can be undertaken to compute the optimal intra-cluster and inter-cluster distances and analyze the con- vergence over varying network conditions.

* + - This dissertation has focused on large scale networks that form the basis of

the Internet of Things. Network failures can also aﬀect these large networks and given the sheer number of network links, the probability of a link failing increases substantially. Mechanisms to identify and reroute traﬃc from links that can fail is an area that needs to be studied comprehensively. As traﬃc in a particular application may not pass through all network

components, federated measurements from multiple applications can be useful to diagnose link failures.

* + - Most of the studies carried out as part of this dissertation were performed

on discrete event network simulators such WSNet and ns-2/ns-3 with cus- tom Python scripts and C/C++ bindings. This approach has been very useful to quickly prototype new mechanisms and build massively large net- works very quickly and very eﬃciently. However, a real device testbed would more accurately reﬂect real world scenarios and allow ﬁner grained control. While ﬁnances and hardware availability would necessarily com- plicate the problem, the interested reader could possibly create a hybrid testbed with fast prototyping in software and critical algorithms deployed in actual devices to evaluate their performance.

* + - The machine learning models, I have discussed in my dissertation bene-

ﬁt from the orchestrator having access to the devices in the network and updating the models with new parameter data periodically. In my simu- lations, I have experimented with a ﬁxed number of devices that are con- nected to the orchestrator and the update intervals. More research can be undertaken to determine the optimal size of the device cluster connected to the orchestrator and the update interval. Moreover, a dynamic device cluster based on the capabilities of the devices themselves and the prevail- ing network conditions could optimize the orchestration mechanism and improve the overall prediction capability of the network.

# Bibliography

1. D. Rogers, J.A. Adams Mathematical Elements for Computer Graphics, McGraw-Hill, New York (1990).
2. G. Wolberg Digital Image Warping, IEEE Computer Society Press, Los Alamitos, CA (1990).
3. L. Bracciale and P. Loreti, ”Lyapunov Drift-Plus-Penalty Optimization for Queues With Finite Capacity,” in IEEE Communications Letters, vol. 24, no. 11, pp. 2555-2558, Nov. 2020, doi: 10.1109/LCOMM.2020.3013125.
4. J. Nu´n˜ez-Mart´ınez and J. Mangues-Bafalluy, ”Distributed Lyapunov drift- plus-penalty routing for WiFi mesh networks with adaptive penalty weight,” 2012 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2012, pp. 1-6, doi: 10.1109/WoW- MoM.2012.6263779.
5. Y. Qi, L. Pan, S. Liu, ”A Lyapunov optimization-based online scheduling algorithm for service provisioning in cloud computing,” Future Generation Computer Systems, v 134, pp. 40-52, 2022.

126

1. Y. Cui, V. K. N. Lau, R. Wang, H. Huang and S. Zhang, ”A Survey on Delay-Aware Resource Control for Wireless Systems—Large Deviation The- ory, Stochastic Lyapunov Drift, and Distributed Stochastic Learning,” in IEEE Transactions on Information Theory, vol. 58, no. 3, pp. 1677-1701, March 2012, doi: 10.1109/TIT.2011.2178150.
2. I. G. Burova, ”On approximation with the continuous polynomial cubic splines,” 2020 24th International Conference on Circuits, Systems, Com- munications and Computers (CSCC), 2020, pp. 126-129, doi: 10.1109/C- SCC49995.2020.00030.
3. I. Strelkovskaya, I. Solovskaya and A. Makoganiuk, ”Predicting Self-Similar Traﬃc using Cubic B-Splines,” 2019 3rd International Conference on Ad- vanced Information and Communications Technologies (AICT), 2019, pp. 153-156, doi: 10.1109/AIACT.2019.8847761.
4. R. Sayfullin and A. Bochkarev, ”Using of a Hierarchical Algorithm Based on Chebyshev-Hermite Functions for Analytical Device Signals Process- ing,” 2019 XXI International Conference Complex Systems: Control and Modeling Problems (CSCMP), 2019, pp. 612-615, doi: 10.1109/C- SCMP45713.2019.8976622.
5. B. Santhanam and T. S. Santhanam, ”Discrete Gauss-Hermite Functions and Eigenvectors of the Centered Discrete Fourier Transform,” 2007 IEEE International Conference on Acoustics, Speech and Signal Processing - ICASSP ’07, 2007, pp. III-1385-III-1388, doi: 10.1109/ICASSP.2007.367104.
6. S. Yi, Z. Hao, Z. Qin and Q. Li, ”Fog Computing: Platform and Appli- cations,” 2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb), 2015, pp. 73-78, doi: 10.1109/HotWeb.2015.22.
7. Y. Xiao and M. Krunz, ”QoE and power eﬃciency tradeoﬀ for fog comput- ing networks with fog node cooperation,” IEEE INFOCOM 2017 - IEEE Conference on Computer Communications, 2017, pp. 1-9, doi: 10.1109/IN- FOCOM.2017.8057196.
8. Y. Wang, X. Tao, X. Zhang, P. Zhang and Y. T. Hou, ”Cooperative Task Oﬄoading in Three-Tier Mobile Computing Networks: An ADMM Frame- work,” in IEEE Transactions on Vehicular Technology, vol. 68, no. 3, pp. 2763-2776, March 2019, doi: 10.1109/TVT.2019.2892176.
9. L. Liu, Z. Chang and X. Guo, ”Socially Aware Dynamic Computation Of- ﬂoading Scheme for Fog Computing System With Energy Harvesting De- vices,” in IEEE Internet of Things Journal, vol. 5, no. 3, pp. 1869-1879, June 2018, doi: 10.1109/JIOT.2018.2816682.
10. S. Misra and N. Saha, “Detour: Dynamic task oﬄoading in software deﬁned fog for IoT applications,” IEEE Journal on Selected Areas in Communica- tions, vol. 37, no. 5, pp. 1159–1166, 2019.
11. L. Lei, H. Xu, X. Xiong, K. Zheng, and W. Xiang, “Joint computation oﬄoading and multi-user scheduling using approximate dynamic program- ming in NB-IoT edge computing system,” IEEE Internet of Things Journal, vol. 6, no. 3, pp. 5345–5362, 2019.
12. Y. Mao, J. Zhang, S. H. Song and K. B. Letaief, ”Power-Delay Tradeoﬀ in Multi-User Mobile-Edge Computing Systems,” 2016 IEEE Global Com- munications Conference (GLOBECOM), 2016, pp. 1-6, doi: 10.1109/GLO- COM.2016.7842160.
13. Y. Chen, N. Zhang, Y. Zhang, X. Chen, W. Wu and X. Shen, ”Energy Eﬃcient Dynamic Oﬄoading in Mobile Edge Computing for Internet of Things,” in IEEE Transactions on Cloud Computing, vol. 9, no. 3, pp. 1050-1060, 1 July-Sept. 2021, doi: 10.1109/TCC.2019.2898657.
14. Y. Gao, W. Tang, M. Wu, P. Yang and L. Dan, ”Dynamic Social-Aware Computation Oﬄoading for Low-Latency Communications in IoT,” in IEEE Internet of Things Journal, vol. 6, no. 5, pp. 7864-7877, Oct. 2019, doi: 10.1109/JIOT.2019.2909299.
15. D. Zhang et al., ”Near-Optimal and Truthful Online Auction for Compu- tation Oﬄoading in Green Edge-Computing Systems,” in IEEE Transac- tions on Mobile Computing, vol. 19, no. 4, pp. 880-893, 1 April 2020, doi: 10.1109/TMC.2019.2901474.
16. E. Cuervo, A. Balasubramanian, Dae-ki Cho, A. Wolman, S. Saroiu,

R. Chandra, and P. Bahl. 2010. MAUI: making smartphones last longer with code oﬄoad. In Proceedings of the 8th international conference on Mobile systems, applications, and services (MobiSys ’10). Associa- tion for Computing Machinery, New York, NY, USA, 49–62. https://doi- org.uc.idm.oclc.org/10.1145/1814433.1814441

1. K. Kumar, and Y. H. Lu, Cloud computing for mobile users: can oﬄoading computation save energy?, Computer, vol. 43, no. 4, pp. 51-56 (2010).
2. D. Zhang et al., ”Resource Allocation for Green Cloud Radio Ac- cess Networks With Hybrid Energy Supplies,” in IEEE Transactions on Vehicular Technology, vol. 67, no. 2, pp. 1684-1697, Feb. 2018, doi: 10.1109/TVT.2017.2754273.
3. L. Burgstahler and M. Neubauer, New modiﬁcations of the exponential moving average algorithm for bandwidth estimation, in Proc. of the 15th ITC Specialist Seminar (2002).
4. R. G. Gallager, Principles of Digital Communication. Cambridge University Press, 2008.
5. Y. Kim, J. Kwak and S. Chong, ”Dual-Side Optimization for Cost- Delay Tradeoﬀ in Mobile Edge Computing,” in IEEE Transactions on Vehicular Technology, vol. 67, no. 2, pp. 1765-1781, Feb. 2018, doi: 10.1109/TVT.2017.2762423.
6. M. J. Neely, “Stochastic network optimization with application to com- munication and queueing systems,” Synthesis Lectures on Communication Networks, vol. 3, no. 1, pp. 1–211, 2010.
7. A. Leon-Garcia, Probability, Statistics, and Random Processes for Electrical Engineering, 3rd ed. Pearson Education, 2017.
8. C. de Boor, *A Practical Guide to Splines,* Springer, New York (1978).
9. P.L. Chebyshev, *Sur les fonctions qui diﬀ`erent le moins possible de z`ero* J. Math. Pures Appl., 19 (1876), pp. 319-346.
10. R.K. Beatson and H. Wolkowicz, *Post-processing piecewise cubics for mono- tonicity* SIAM J. Numer. Anal., 26 (2) (1989), pp. 480-502.
11. G.M. Nielson, *Some piecewise polynomial alternatives to splines under ten- sion,* R. Barnhill, R. Riesenfeld (Eds.), Computer Aided Geometric Design, Academic Press, New York (1974), pp. 209-235.
12. S. Pruess, *Properties of splines in tension,* J. Approx. Theory, 17 (1976), pp. 86-96.
13. S. Pruess, *Alternatives to the exponential spline in tension,* Math. Comput., 33 (1979), pp. 1273-1281.
14. D. Schweikert, *An interpolation curve using splines in tension,* J. Math.

Phys., 45 (1966), pp. 312-317.

1. H. Sp¨ath, *Exponential spline interpolation Computing,* 4 (1969), pp. 225- 233.
2. D. McAllister, E. Passow and J. Roulier, *Algorithms for computing shape preserving splines interpolations to data,* Math. Comput., 31 (1977), pp. 717-725.
3. R. Delbourgo and J. Gregory, *Shape preserving piecewise rational interpo- lation,* SIAM J. Sci. Statist. Comput., 6 (1985), pp. 967-976.
4. J. Gregory and R. Delbourgo, *Piecewise rational quadratic interpolation to monotonic data,* IMA J. Numer. Anal., 2 (1982), pp. 123-130.
5. Y. Dai, D. Xu, S. Maharjan and Y. Zhang, ”Joint Computation Oﬄoading and User Association in Multi-Task Mobile Edge Computing,” in IEEE Transactions on Vehicular Technology, vol. 67, no. 12, pp. 12313-12325, Dec. 2018, doi: 10.1109/TVT.2018.2876804.
6. K. Cheng, Y. Teng, W. Sun, A. Liu and X. Wang, ”Energy-Eﬃcient Joint Oﬄoading and Wireless Resource Allocation Strategy in Multi-MEC Server Systems,” 2018 IEEE International Conference on Communications (ICC), 2018, pp. 1-6, doi: 10.1109/ICC.2018.8422877.
7. C. -F. Liu, M. Bennis, M. Debbah and H. V. Poor, ”Dynamic Task Oﬄoad- ing and Resource Allocation for Ultra-Reliable Low-Latency Edge Comput- ing,” in IEEE Transactions on Communications, vol. 67, no. 6, pp. 4132- 4150, June 2019, doi: 10.1109/TCOMM.2019.2898573.
8. J. Ren, G. Yu, Y. Cai and Y. He, ”Latency Optimization for Resource Allocation in Mobile-Edge Computation Oﬄoading,” in IEEE Transactions on Wireless Communications, vol. 17, no. 8, pp. 5506-5519, Aug. 2018, doi: 10.1109/TWC.2018.2845360.
9. L. Cui et al., ”Joint Optimization of Energy Consumption and La- tency in Mobile Edge Computing for Internet of Things,” in IEEE In- ternet of Things Journal, vol. 6, no. 3, pp. 4791-4803, June 2019, doi: 10.1109/JIOT.2018.2869226.
10. Z. Kuang, L. Li, J. Gao, L. Zhao and A. Liu, ”Partial Oﬄoading Schedul- ing and Power Allocation for Mobile Edge Computing Systems,” in IEEE

Internet of Things Journal, vol. 6, no. 4, pp. 6774-6785, Aug. 2019, doi: 10.1109/JIOT.2019.2911455.

1. P. Paymard, N. Mokari and M. Orooji, ”Task Scheduling Based on Priority and Resource Allocation in Multi-User Multi-Task Mobile Edge Computing System,” 2019 IEEE 30th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC), 2019, pp. 1-7, doi: 10.1109/PIMRC.2019.8904174.
2. Y. Wu, Y. Wang, F. Zhou and R. Qingyang Hu, ”Computation Eﬃciency Maximization in OFDMA-Based Mobile Edge Computing Networks,” in IEEE Communications Letters, vol. 24, no. 1, pp. 159-163, Jan. 2020, doi: 10.1109/LCOMM.2019.2950013.
3. M. Li, S. Yang, Z. Zhang, J. Ren and G. Yu, ”Joint subcarrier and power allocation for OFDMA based mobile edge computing sys- tem,” 2017 IEEE 28th Annual International Symposium on Personal, In- door, and Mobile Radio Communications (PIMRC), 2017, pp. 1-6, doi: 10.1109/PIMRC.2017.8292207.
4. X. Yang, X. Yu, H. Huang and H. Zhu, ”Energy Eﬃciency Based Joint Computation Oﬄoading and Resource Allocation in Multi-Access MEC Sys- tems,” in IEEE Access, vol. 7, pp. 117054-117062, 2019, doi: 10.1109/AC- CESS.2019.2936435.
5. C. You, K. Huang, H. Chae and B. -H. Kim, ”Energy-Eﬃcient Resource Allocation for Mobile-Edge Computation Oﬄoading,” in IEEE Transactions

on Wireless Communications, vol. 16, no. 3, pp. 1397-1411, March 2017, doi: 10.1109/TWC.2016.2633522.

1. P. Singhal and A. Yadav, “Congestion detection in wireless sensor net- work using neural network,” in International Conference for Convergence for Technology-2014, April 2014, pp. 1–4.
2. University of Southern California Information Sciences In- stitute. The Network Simulator - ns-2. [Online] Available: https://sourceforge.net/projects/nsnam/ﬁles
3. T. Issariyakul and E. Hossain, Introduction to Network Simulator NS2, 1st ed. Springer Publishing Company, Incorporated, 2010.
4. J. B. Madalgi and S. A. Kumar, “Congestion detection in wireless sensor networks using mlp and classiﬁcation by regression,” in 2017 3rd Interna- tional Conference on Applied and Theoretical Computing and Communica- tion Technology (iCATccT), Dec 2017, pp. 226–231.
5. J. R. Quinlan, “Learning with continuous classes,” in Proceedings of the 5th Australian Joint Conference on Artiﬁcial Intelligence (AI ’92), pp. 343–348, World Scientiﬁc, Singapore, 1992.
6. J. B. Madalgi and S. A. Kumar, “Development of wireless sensor network congestion detection classiﬁer using support vector machine,” in 2018 3rd International Conference on Computational Systems and Information Tech- nology for Sustainable Solutions (CSITSS), Dec 2018, pp. 187–192.
7. M. Devare and A. Kumar, “Clustering and classiﬁcation (time series analy- sis) based congestion control algorithm: Data mining approach,” in IJCSNS

International Journal of Computer Science and Network Security, vol. 7, no. 9, September 2007, pp. 241–246.

1. N. Zhang, F. Jaafar, and Y. Malik, “Low-rate dos attack detection using psd based entropy and machine learning,” in 2019 6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/ 2019 5th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom), June 2019, pp. 59–62.
2. A. Ahmed and E. Ahmed, “A Survey on Mobile Edge Computing,” in Proc. of International Conference on Intelligent Systems and Control (ISCO), 2016.
3. Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, “Mobile Edge Computing - A Key Technology towards 5G,” White Paper, ETSI, 2015.
4. E. Li, Z. Zhou, and X. Chen, “Edge intelligence: On-demand deep learning model co-inference with device-edge synergy,” in Proceedings of the 2018 Workshop on Mobile Edge Communications, MECOMM@SIGCOMM 2018, Budapest, Hungary, August 20, 2018, 2018, pp. 31–36.
5. D. Xenakis, N. Passas, L. Merakos and C. Verikoukis, ”Mobility Man- agement for Femtocells in LTE-Advanced: Key Aspects and Sur- vey of Handover Decision Algorithms,” in IEEE Communications Sur- veys & Tutorials, vol. 16, no. 1, pp. 64-91, First Quarter 2014, doi: 10.1109/SURV.2013.060313.00152.
6. X. Chen, L. Pu, L. Gao, W. Wu and D. Wu, ”Exploiting Massive D2D Collaboration for Energy-Eﬃcient Mobile Edge Computing,” in IEEE

Wireless Communications, vol. 24, no. 4, pp. 64-71, Aug. 2017, doi: 10.1109/MWC.2017.1600321.

1. X. Chen, W. Li, S. Lu, Z. Zhou and X. Fu, ”Eﬃcient Resource Alloca- tion for On-Demand Mobile-Edge Cloud Computing,” in IEEE Transac- tions on Vehicular Technology, vol. 67, no. 9, pp. 8769-8780, Sept. 2018, doi: 10.1109/TVT.2018.2846232.
2. C. Shen, C. Tekin and M. van der Schaar, ”A Non-Stochastic Learning Approach to Energy Eﬃcient Mobility Management,” in IEEE Journal on Selected Areas in Communications, vol. 34, no. 12, pp. 3854-3868, Dec. 2016, doi: 10.1109/JSAC.2016.2612038.
3. J. W. Jiang, T. Lan, S. Ha, M. Chen and M. Chiang, ”Joint VM placement and routing for data center traﬃc engineering,” 2012 Proceedings IEEE INFOCOM, 2012, pp. 2876-2880, doi: 10.1109/INFCOM.2012.6195719.
4. F. Xu, F. Liu, H. Jin and A. V. Vasilakos, ”Managing Performance Overhead of Virtual Machines in Cloud Computing: A Survey, State of the Art, and Future Directions,” in Proceedings of the IEEE, vol. 102, no. 1, pp. 11-31, Jan. 2014, doi: 10.1109/JPROC.2013.2287711.
5. A. Fischer, J. F. Botero, M. T. Beck, H. de Meer and X. Hesselbach, ”Virtual Network Embedding: A Survey,” in IEEE Communications Sur- veys & Tutorials, vol. 15, no. 4, pp. 1888-1906, Fourth Quarter 2013, doi: 10.1109/SURV.2013.013013.00155.
6. A. Nadembega, A. S. Haﬁd and R. Brisebois, ”Mobility prediction model- based service migration procedure for follow me cloud to support QoS

and QoE,” 2016 IEEE International Conference on Communications (ICC), 2016, pp. 1-6, doi: 10.1109/ICC.2016.7511148.

1. S. Wang, R. Urgaonkar, K. Chan, T. He, M. Zafer and K. K. Leung, ”Dynamic service placement for mobile micro-clouds with predicted fu- ture costs,” 2015 IEEE International Conference on Communications (ICC), 2015, pp. 5504-5510, doi: 10.1109/ICC.2015.7249199.
2. A. Aissioui, A. Ksentini, A. M. Gueroui and T. Taleb, ”On Enabling 5G Au- tomotive Systems Using Follow Me Edge-Cloud Concept,” in IEEE Trans- actions on Vehicular Technology, vol. 67, no. 6, pp. 5302-5316, June 2018, doi: 10.1109/TVT.2018.2805369.
3. A. Rudenko, P. Reiher, G. J. Popek, and G. H. Kuenning. ”Saving portable computer battery power through remote process execution”. SIGMOBILE Mob. Comput. Commun. Rev. 2, 1 (January 1998), 19–26. https://doi- org.uc.idm.oclc.org/10.1145/584007.584008
4. G. Huertacanepa and D. Lee, “An adaptable application oﬄoading scheme based on application behavior,” in 22nd International Conference on Ad- vanced Information Networking and Applications Workshops, 2008.
5. Changjiu Xian, Yung-Hsiang Lu and Zhiyuan Li, ”Adaptive computation oﬄoading for energy conservation on battery-powered systems,” 2007 In- ternational Conference on Parallel and Distributed Systems, 2007, pp. 1-8, doi: 10.1109/ICPADS.2007.4447724.
6. M. R. Rahimi, N. Venkatasubramanian, S. Mehrotra and A. V. Vasilakos, ”MAPCloud: Mobile Applications on an Elastic and Scalable 2-Tier Cloud

Architecture,” 2012 IEEE Fifth International Conference on Utility and Cloud Computing, 2012, pp. 83-90, doi: 10.1109/UCC.2012.25.

1. D. Huang, P. Wang and D. Niyato, ”A Dynamic Oﬄoading Al- gorithm for Mobile Computing,” in IEEE Transactions on Wireless Communications, vol. 11, no. 6, pp. 1991-1995, June 2012, doi: 10.1109/TWC.2012.041912.110912.
2. R. Wolski, S. Gurun, C. Krintz and D. Nurmi, ”Using bandwidth data to make computation oﬄoading decisions,” 2008 IEEE International Symposium on Parallel and Distributed Processing, 2008, pp. 1-8, doi: 10.1109/IPDPS.2008.4536215.
3. Y. Wen, W. Zhang and H. Luo, ”Energy-optimal mobile application ex- ecution: Taming resource-poor mobile devices with cloud clones,” 2012 Proceedings IEEE INFOCOM, 2012, pp. 2716-2720, doi: 10.1109/INF- COM.2012.6195685.
4. L. Yang, J. Cao, Y. Yuan, T. Li, A. Han, and A. Chan. ”A framework for partitioning and execution of data stream applications in mobile cloud computing”. SIGMETRICS Perform. Eval. Rev. 40, 4 (March 2013), 23–32. https://doi-org.uc.idm.oclc.org/10.1145/2479942.2479946
5. M. R. Rahimi, N. Venkatasubramanian and A. V. Vasilakos, ”MuSIC: Mobility-Aware Optimal Service Allocation in Mobile Cloud Computing,” 2013 IEEE Sixth International Conference on Cloud Computing, 2013, pp. 75-82, doi: 10.1109/CLOUD.2013.100.
6. S. Barbarossa, S. Sardellitti and P. Di Lorenzo, ”Joint allocation of com- putation and communication resources in multiuser mobile cloud com- puting,” 2013 IEEE 14th Workshop on Signal Processing Advances in Wireless Communications (SPAWC), 2013, pp. 26-30, doi: 10.1109/S- PAWC.2013.6612005.
7. H. Al-Shatri, S. Mu¨ller and A. Klein, ”Distributed algorithm for energy eﬃcient multi-hop computation oﬄoading,” 2016 IEEE Inter- national Conference on Communications (ICC), 2016, pp. 1-6, doi: 10.1109/ICC.2016.7510809.
8. C. Funai, C. Tapparello and W. Heinzelman, ”Computational Oﬄoading for Energy Constrained Devices in Multi-Hop Cooperative Networks,” in IEEE Transactions on Mobile Computing, vol. 19, no. 1, pp. 60-73, 1 Jan. 2020, doi: 10.1109/TMC.2019.2892100.
9. Z. Hong, H. Huang, S. Guo, W. Chen and Z. Zheng, ”QoS-Aware Coop- erative Computation Oﬄoading for Robot Swarms in Cloud Robotics,” in IEEE Transactions on Vehicular Technology, vol. 68, no. 4, pp. 4027-4041, April 2019, doi: 10.1109/TVT.2019.2901761.
10. Z. Hong, W. Chen, H. Huang, S. Guo and Z. Zheng, ”Multi-Hop Cooperative Computation Oﬄoading for Industrial IoT–Edge–Cloud Computing Envi- ronments,” in IEEE Transactions on Parallel and Distributed Systems, vol. 30, no. 12, pp. 2759-2774, 1 Dec. 2019, doi: 10.1109/TPDS.2019.2926979.
11. Z. Ning, P. Dong, X. Kong and F. Xia, ”A Cooperative Partial Computation Oﬄoading Scheme for Mobile Edge Computing Enabled Internet of Things,”

in IEEE Internet of Things Journal, vol. 6, no. 3, pp. 4804-4814, June 2019, doi: 10.1109/JIOT.2018.2868616.

1. Y. Wang, M. Sheng, X. Wang, L. Wang and J. Li, ”Mobile-Edge Computing: Partial Computation Oﬄoading Using Dynamic Voltage Scaling,” in IEEE Transactions on Communications, vol. 64, no. 10, pp. 4268-4282, Oct. 2016, doi: 10.1109/TCOMM.2016.2599530.
2. Y. Jin, J. Jin, A. Gluhak, K. Moessner and M. Palaniswami, ”An Intelligent Task Allocation Scheme for Multihop Wireless Networks,” in IEEE Trans- actions on Parallel and Distributed Systems, vol. 23, no. 3, pp. 444-451, March 2012, doi: 10.1109/TPDS.2011.172.
3. Y. Tian and E. Ekici, ”Cross-Layer Collaborative In-Network Processing in Multihop Wireless Sensor Networks,” in IEEE Transactions on Mobile Com- puting, vol. 6, no. 3, pp. 297-310, March 2007, doi: 10.1109/TMC.2007.39.
4. L. Chen, S. Zhou and J. Xu, ”Computation Peer Oﬄoading for Energy-Constrained Mobile Edge Computing in Small-Cell Networks,” in IEEE/ACM Transactions on Networking, vol. 26, no. 4, pp. 1619-1632, Aug. 2018, doi: 10.1109/TNET.2018.2841758.
5. A. Munir, T. He, R. Raghavendra, F. Le, and A. X. Liu, “Network schedul- ing aware task placement in datacenters,” in Proc. 12th Int. Conf. Emerg. Netw. EXperiments Technol., 2016, pp. 221–235.
6. Y. Zhao, C. Tian, J. Fan, T. Guan and C. Qiao, ”RPC: Joint Online Reducer Placement and Coﬂow Bandwidth Scheduling for Clusters,” 2018 IEEE 26th

International Conference on Network Protocols (ICNP), 2018, pp. 187-197, doi: 10.1109/ICNP.2018.00028.

1. Q. Pu, G. Ananthanarayanan, P. Bodik, S. Kandula, A. Akella,

P. Bahl, and I. Stoica. ”Low Latency Geo-distributed Data Analyt- ics”. In Proceedings of the 2015 ACM Conference on Special Inter- est Group on Data Communication (SIGCOMM ’15). Association for Computing Machinery, New York, NY, USA, 421–434. https://doi- org.uc.idm.oclc.org/10.1145/2785956.2787505

1. L. Rupprecht, W. Culhane, and P. Pietzuch, “SquirrelJoin: Network-aware distributed join processing with lazy partitioning,” Proc. VLDB Endow- ment, vol. 10, no. 11, pp. 1250–1261, 2017.
2. E. Bastug, M. Bennis, E. Zeydan, M. A. Kader, I. A. Karatepe, A. S. Er, and M. Debbah, “Big data meets telcos: A proactive caching perspective,” Journal of Communications and Networks, vol. 17, no. 6, pp. 549–557, Dec. 2015.
3. T. X. Tran, A. Hajisami, P. Pandey and D. Pompili, ”Collaborative Mobile Edge Computing in 5G Networks: New Paradigms, Scenarios, and Chal- lenges,” in IEEE Communications Magazine, vol. 55, no. 4, pp. 54-61, April 2017, doi: 10.1109/MCOM.2017.1600863.
4. E. Zeydan et al., ”Big data caching for networking: moving from cloud to edge,” in IEEE Communications Magazine, vol. 54, no. 9, pp. 36-42, September 2016, doi: 10.1109/MCOM.2016.7565185.
5. W. Fan, Y. Liu, B. Tang, F. Wu and H. Zhang, ”TerminalBooster: Collabo- rative Computation Oﬄoading and Data Caching via Smart Basestations,” in IEEE Wireless Communications Letters, vol. 5, no. 6, pp. 612-615, Dec. 2016, doi: 10.1109/LWC.2016.2605694.
6. G. Lee, W. Saad and M. Bennis, ”An Online Optimization Framework for Distributed Fog Network Formation With Minimal Latency,” in IEEE Transactions on Wireless Communications, vol. 18, no. 4, pp. 2244-2258, April 2019, doi: 10.1109/TWC.2019.2901850.
7. T. X. Tran, P. Pandey, A. Hajisami and D. Pompili, ”Collaborative multi- bitrate video caching and processing in Mobile-Edge Computing networks,” 2017 13th Annual Conference on Wireless On-demand Network Systems and Services (WONS), 2017, pp. 165-172, doi: 10.1109/WONS.2017.7888772.
8. W. Jiang, G. Feng and S. Qin, ”Optimal Cooperative Content Caching and Delivery Policy for Heterogeneous Cellular Networks,” in IEEE Trans- actions on Mobile Computing, vol. 16, no. 5, pp. 1382-1393, 1 May 2017, doi: 10.1109/TMC.2016.2597851.
9. H. Hsu and K. -C. Chen, ”A Resource Allocation Perspective on Caching to Achieve Low Latency,” in IEEE Communications Letters, vol. 20, no. 1, pp. 145-148, Jan. 2016, doi: 10.1109/LCOMM.2015.2499193.
10. Z. Tan, X. Li, F. R. Yu, L. Chen, H. Ji and V. C. M. Leung, ”Joint Access Selection and Resource Allocation in Cache-Enabled HCNs with D2D Communications,” 2017 IEEE Wireless Communications and Networking Conference (WCNC), 2017, pp. 1-6, doi: 10.1109/WCNC.2017.7925732.
11. C. Wang, C. Liang, F. R. Yu, Q. Chen and L. Tang, ”Computation Of- ﬂoading and Resource Allocation in Wireless Cellular Networks With Mobile Edge Computing,” in IEEE Transactions on Wireless Communications, vol. 16, no. 8, pp. 4924-4938, Aug. 2017, doi: 10.1109/TWC.2017.2703901.
12. R. Huo et al., ”Software Deﬁned Networking, Caching, and Computing for Green Wireless Networks,” in IEEE Communications Magazine, vol. 54, no. 11, pp. 185-193, November 2016, doi: 10.1109/MCOM.2016.1600485CM.
13. J. Chakareski. ”VR/AR Immersive Communication: Caching, Edge Com- puting, and Transmission Trade-Oﬀs”. In Proceedings of the Workshop on Virtual Reality and Augmented Reality Network (VR/AR Network ’17). Association for Computing Machinery, New York, NY, USA, 36–41. https://doi-org.uc.idm.oclc.org/10.1145/3097895.3097902
14. Y. Cui, W. He, C. Ni, C. Guo and Z. Liu, ”Energy-Eﬃcient Resource Allocation for Cache-Assisted Mobile Edge Computing,” 2017 IEEE 42nd Conference on Local Computer Networks (LCN), 2017, pp. 640-648, doi: 10.1109/LCN.2017.112.
15. L. A. Belady, “A study of Replacement Algorithms for a Virtual-Storage Computer,” IBM Systems Journal, vol. 5, no. 2, pp. 78–101, 1966.
16. G. S. Paschos, G. Iosiﬁdis, M. Tao, D. Towsley and G. Caire, ”The Role of Caching in Future Communication Systems and Networks,” in IEEE Journal on Selected Areas in Communications, vol. 36, no. 6, pp. 1111-1125, June 2018, doi: 10.1109/JSAC.2018.2844939.
17. C. Aggarwal, J. L. Wolf, and P. S. Yu, “Caching on the World Wide Web,” Trans. Knowledge Data Eng., vol. 11, no. 1, pp. 94–107, 1999.
18. J. Kangasharju, J. Roberts, and K. Ross, “Object Replication Strategies in Content Distribution Networks,” Computer Communications, vol. 25, no. 4, pp. 376–383, 2002.
19. N. Golrezaei, K. Shanmugam, A. G. Dimakis, A. F. Molisch and G. Caire, ”FemtoCaching: Wireless video content delivery through distributed caching helpers,” 2012 Proceedings IEEE INFOCOM, 2012, pp. 1107-1115, doi: 10.1109/INFCOM.2012.6195469.
20. N. Golrezaei, A. F. Molisch, A. G. Dimakis and G. Caire, ”Femtocaching and device-to-device collaboration: A new architecture for wireless video distribution,” in IEEE Communications Magazine, vol. 51, no. 4, pp. 142- 149, April 2013, doi: 10.1109/MCOM.2013.6495773.
21. D. D. Sleator and R. E. Tarjan. ”Amortized eﬃciency of list update and paging rules.” Commun. ACM 28, 2 (Feb. 1985), 202–208. https://doi- org.uc.idm.oclc.org/10.1145/2786.2793
22. P. R. Jelenkovi´c and X. Kang. ”Characterizing the miss sequence of the LRU cache,” SIGMETRICS Perform. Eval. Rev. 36, 2 (September 2008), 119–121. https://doi-org.uc.idm.oclc.org/10.1145/1453175.1453203
23. D. Lee, J. Choi, J. -H. Kim, S. H. Noh, S. L. Min, Y. Cho, and C.

S. Kim. ”On the existence of a spectrum of policies that subsumes the

least recently used (LRU) and least frequently used (LFU) policies,” SIG- METRICS Perform. Eval. Rev. 27, 1 (June 1999), 134–143. https://doi- org.uc.idm.oclc.org/10.1145/301464.301487

1. S. Traverso, M. Ahmed, M. Garetto, P. Giaccone, E. Leonardi, and S. Niccolini. ”Temporal locality in today’s content caching: why it matters and how to model it,” SIGCOMM Comput. Commun. Rev. 43, 5 (October 2013), 5–12. https://doi-org.uc.idm.oclc.org/10.1145/2541468.2541470
2. F. Olmos, B. Kauﬀmann, A. Simonian, and Y. Carlinet, “Catalog dynam- ics: Impact of content publishing and perishing on the performance of a LRU cache,” in Proc. of ITC, 2014.
3. M. Leconte, G. Paschos, L. Gkatzikis, M. Draief, S. Vassilaras and S. Chouvardas, ”Placing dynamic content in caches with small population,” IEEE INFOCOM 2016 - The 35th Annual IEEE International Confer- ence on Computer Communications, 2016, pp. 1-9, doi: 10.1109/INFO- COM.2016.7524380.
4. S.-E. Elayoubi and J. Roberts, “Performance and Cost Eﬀectiveness of Caching in Mobile Access Networks,” in Proc. of ICN, 2015.
5. S. O. Somuyiwa, A. Gy¨orgy and D. Gu¨ndu¨z, ”A Reinforcement-Learning Approach to Proactive Caching in Wireless Networks,” in IEEE Journal on Selected Areas in Communications, vol. 36, no. 6, pp. 1331-1344, June 2018, doi: 10.1109/JSAC.2018.2844985.
6. A. Sadeghi, F. Sheikholeslami, and G. B. Giannakis, “Optimal and Scalable Caching for 5G Using Reinforcement Learning of SpaceTime Popularities,” IEEE J. Select. Areas Commun., vol. 12, no. 1, pp. 180–190, 2018.
7. S. Geulen, B. V¨ocking, and M. Winkler, “Regret Minimization for Online Buﬀering Problems Using the Weighted Majority Algorithm,” in Proc. of COLT, 2010.
8. G. S. Paschos, A. Destounis, L. Vigneri and G. Iosiﬁdis, ”Learning to Cache With No Regrets,” IEEE INFOCOM 2019 - IEEE Conference on Computer Communications, 2019, pp. 235-243, doi: 10.1109/INFO- COM.2019.8737446.
9. Y. Li, T. S. Salem, G. Neglia, and S. Ioannidis. ”Online Caching Networks with Adversarial Guarantees,” Proc. ACM Meas. Anal. Com- put. Syst. 5, 3, Article 35 (December 2021), 39 pages. https://doi- org.uc.idm.oclc.org/10.1145/3491047
10. T. Si Salem, G. Neglia and S. Ioannidis, ”No-Regret Caching via Online Mirror Descent,” ICC 2021 - IEEE International Conference on Communi- cations, 2021, pp. 1-6, doi: 10.1109/ICC42927.2021.9500487.
11. R. Bhattacharjee, S. Banerjee, and A. Sinha. ”Fundamental Limits on the Regret of Online Network-Caching,” Proc. ACM Meas. Anal. Comput. Syst. 4, 2, Article 25 (June 2020), 31 pages. https://doi- org.uc.idm.oclc.org/10.1145/3392143
12. D. Paria and A. Sinha, “LeadCache: Regret-Optimal Caching in Net- works,” in Proc. of NeurIPS, 2021.
13. S. Ghosh and D. P. Agrawal, ”Optimizing the energy envelope in the In- ternet of Things using predictive cubic splines,” 2020 IEEE International Conference on Sensing, Communication and Networking (SECON Work- shops), 2020, pp. 1-6, doi: 10.1109/SECONWorkshops50264.2020.9149775.
14. S. Ghosh and D. P. Agrawal, ”A decision support framework to en- hance user experiences in resource constrained devices,” 2020 IEEE Latin- American Conference on Communications (LATINCOM), 2020, pp. 1-6, doi: 10.1109/LATINCOM50620.2020.9282304.
15. S. Ghosh and D. P. Agrawal, ”Creating a more eﬃcient Internet of Things through predictive analysis of resources,” 2020 IEEE Latin- American Conference on Communications (LATINCOM), 2020, pp. 1-6, doi: 10.1109/LATINCOM50620.2020.9282339.
16. S. Ghosh and D. P. Agrawal, ”A high performance hierarchical caching framework for mobile edge computing environments,” 2021 IEEE Wireless Communications and Networking Conference (WCNC), 2021, pp. 1-6, doi: 10.1109/WCNC49053.2021.9417323.
17. S. Ghosh and D. P. Agrawal, ”Prioritized computation oﬄoading and resource optimization for networks with strict latency,” 2021 IEEE 7th World Forum on Internet of Things (WF-IoT), 2021, pp. 686-691, doi: 10.1109/WF-IoT51360.2021.9595680.
18. S. Ghosh and D. P. Agrawal, ”The optimization game: Mobility and Edge computing in dense networks,” 2021 IEEE 7th World Forum on

Internet of Things (WF-IoT), 2021, pp. 692-697, doi: 10.1109/WF- IoT51360.2021.9595890.

1. S.R. Yang, Y.-J. Tseng, C.-C. Huang, and W.-C. Lin, “Multi-access edge computing enhanced video streaming: Proof-of-concept implementation and prediction/QoE models,” IEEE Trans. Veh. Tech., vol. 68, no. 2, pp. 1888–1902, 2018.
2. P. Porambage, J. Okwuibe, M. Liyanage, M. Ylianttila, and T. Taleb, “Sur- vey on Multi-Access Edge Computing for Internet of Things Realization,” IEEE Commun. Surveys Tut., vol. 20, no. 4, pp. 2961–2991, 2018.
3. Y. Chen, N. Zhang, Y. Zhang, X. Chen, W. Wu, and X. S. Shen, “Energy Eﬃcient Dynamic Oﬄoading in Mobile Edge Computing for Internet of Things,” IEEE Trans. Cloud Comput.
4. Y. Dai, D. Xu, S. Maharjan, and Y. Zhang, “Joint computation oﬄoading and user association in multi-task mobile edge computing,” IEEE Trans. Veh. Tech., vol. 67, no. 12, pp. 12 313–12 325, 2018.
5. K. Cheng, Y. Teng, W. Sun, A. Liu, and X. Wang, “Energy eﬃcient joint oﬄoading and wireless resource allocation strategy in multi-MEC server systems,” in Proc. IEEE ICC, 2018, pp. 1–6.
6. C.F. Liu, M. Bennis, M. Debbah, and H. V. Poor, “Dynamic task oﬄoading and resource allocation for ultra-reliable low-latency edge computing,” IEEE Trans. Commun., vol. 67, no. 6, pp. 4132–4150, 2019.
7. J. Ren, G. Yu, Y. Cai, and Y. He, “Latency optimization for resource allocation in mobile-edge computation oﬄoading,” IEEE Trans. Wireless Commun., vol. 17, no. 8, pp. 5506–5519, 2018.
8. L. Cui, C. Xu, S. Yang, J. Z. Huang, J. Li, X. Wang, Z. Ming, and N. Lu, “Joint optimization of energy consumption and latency in mobile edge computing for Internet of Things,” IEEE Internet Things J., vol. 6, no. 3, pp. 4791–4803, 2018.
9. Z. Kuang, L. Li, J. Gao, L. Zhao, and A. Liu, “Partial oﬄoading scheduling and power allocation for mobile edge computing systems,” IEEE Internet Things J., vol. 6, no. 4, pp. 6774–6785, 2019.
10. P. Paymard, N. Mokari, and M. Orooji, “Task Scheduling Based on Priority and Resource Allocation in Multi-User Multi-Task Mobile Edge Computing System,” in Proc. IEEE PIMRC, 2019, pp. 1–7.
11. Y. Wu, Y. Wang, F. Zhou, and R. Q. Hu, “Computation Eﬃciency Max- imization in OFDMA-Based Mobile Edge Computing Networks,” IEEE Commun. Lett., vol. 24, no. 1, pp. 159–163, 2020.
12. M. Li, S. Yang, Z. Zhang, J. Ren, and G. Yu, “Joint subcarrier and power allocation for OFDMA based mobile edge computing system,” in Proc. IEEE PIMRC, 2017, pp. 1–6.
13. X. Yang, X. Yu, H. Huang, and H. Zhu, “Energy eﬃciency based joint com- putation oﬄoading and resource allocation in multi-access MEC systems,” IEEE Access, vol. 7, pp. 117 054–117 062, 2019.
14. C. You, K. Huang, H. Chae, and B.-H. Kim, “Energy eﬃcient resource allocation for mobile-edge computation oﬄoading,” IEEE Trans. Wireless Commun., vol. 16, no. 3, pp. 1397–1411, 2017.
15. Y. Wei and R. Lui, “Dual Methods for Nonconvex Spectrum Optimiza- tion of Multicarrier Systems,” IEEE Trans. Commun., vol. 54, no. 7, pp. 1310–1322, 2006.
16. J. Li and P. Mohapatra, *Analytical modeling and mitigation techniques for the energy hole problem in sensor networks,* Pervasive and Mobile Comput- ing, vol. 3, no. 3, Jun. 2007, pp. 233–254.
17. E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, *Maui: Making smartphones last longer with code oﬄoad,* in Proc. of ACM MobiSys, 2010.
18. B. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, *Clonecloud: Elastic execution between mobile device and cloud,* in Proc. of ACM EuroSys, 2011.
19. M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, *The case for vm- based cloudlets in mobile computing,* IEEE Pervasive Computing, vol. 8, pp. 14–23, Oct. 2009.
20. C. K. Tham and R. Chattopadhyay, *A load balancing scheme for sens- ing and analytics on a mobile edge computing network,* in Proc. of IEEE WoWMoM, 2017.
21. F. Bonomi, R. Milito, Z. Zhang, and S. Addepalli, *Fog computing and its role in the internet of things,* in Proc. of MCC, 2012.
22. A. Dou, V. Kalogeraki, D. Gunopulos, T. Mielikainen, and V. H. Tuu- los, *Misco: A mapreduce framework for mobile systems,* in Proc. of ACM PETRA, 2010.
23. M. Y. Arslan, I. Singh, S. Singh, H. V. Madhyastha, K. Sundaresan, and S.

V. Krishnamurthy, *Computing while charging: Building a distributed com- puting infrastructure using smartphones,* in Proc. of ACM CoNEXT, 2012.

1. H. Wang and L.-S. Peh, *Mobistreams: A reliable distributed stream pro- cessing system for mobile devices,* in Proc. of IEEE IPDPS, 2014.
2. S. Fan, T. Salonidis, and B. Lee, *A framework for collaborative sensing and processing of mobile data streams,* in Proc. of ACM MobiCom, 2016.
3. D. O’Keeﬀe, T. Salonidis, and P. Pietzuch, *Network-aware stream query processing in mobile ad-hoc networks,* in IEEE Military Communications Conference, pp. 1335–1340, Oct 2015.
4. T. Salonidis, G. Sotiropoulos, R. Guerin, and R. Govindan, *Online opti- mization of 802.11 mesh networks,* in Proc. of ACM CoNEXT, 2009.
5. S. K. Saha, P. Deshpande, P. P. Inamdar, R. K. Sheshadri, and D. Kout- sonikolas, *Power-throughput tradeoﬀs of 802.11n/ac in smartphones,* in Proc. of IEEE INFOCOM, 2015.
6. S. Yang, Y. Tahir, P. y. Chen, A. Marshall, and J. McCann, *Distributed optimization in energy harvesting sensor networks with dynamic in-network data processing,* in Proc. of IEEE INFOCOM, 2016.
7. M. A. Bagheri and Q. Gao, *An eﬃcient ensemble classiﬁcation method based on novel classiﬁer selection technique,* in Proc. of ACM WIMS, 2012.
8. T. Lan, D. Kao, M. Chiang, and A. Sabharwal, *An axiomatic theory of fairness in network resource allocation,* in Proc. of IEEE INFOCOM, 2010.
9. S. Boyd and L. Vandenberghe, *Convex Optimization.* New York, NY, USA: Cambridge University Press, 2004.
10. *ns-3: A discrete-event network simulator for Internet Systems.* Accessed on: April 30, 2020. [Online]. Available: h[ttps://www.nsnam.org.](http://www.nsnam.org/)
11. V. Gueant, *iPerf - The ultimate speed test tool for TCP, UDP and SCTP,*

Accessed on April 30, 2020. [Online]. Available: https://iperf.fr.

1. *Tesseract Open Source OCR Engine,* Accessed on April 30, 2020. [Online].

Available: https://github.com/tesseract-ocr/tesseract.

1. M. G. Kendall, *A new measure of rank correlation,* Biometrika, vol. 30, no.

1/2, pp. 81–93, 1938.

1. Z. -S. Feng, B. -R. Sun, J. Ding and Z. Liang, ”Spline Interpolation Uesed in Electronchemical Impedance Spectroscopy,” 2008 International Conference on Apperceiving Computing and Intelligence Analysis, 2008, pp. 16-19, doi: 10.1109/ICACIA.2008.4769961.
2. H. Li, G. Shou, Y. Hu and Z. Guo, ”Mobile Edge Computing: Progress and Challenges,” 2016 4th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud), 2016, pp. 83-84, doi: 10.1109/MobileCloud.2016.16.
3. U. Drolia et al., ”The Case for Mobile Edge-Clouds,” 2013 IEEE 10th In- ternational Conference on Ubiquitous Intelligence and Computing and 2013 IEEE 10th International Conference on Autonomic and Trusted Computing, 2013, pp. 209-215, doi: 10.1109/UIC-ATC.2013.94.
4. S. Barbarossa, S. Sardellitti and P. Di Lorenzo, ”Joint allocation of com- putation and communication resources in multiuser mobile cloud com- puting,” 2013 IEEE 14th Workshop on Signal Processing Advances in Wireless Communications (SPAWC), 2013, pp. 26-30, doi: 10.1109/S- PAWC.2013.6612005.
5. M. Satyanarayanan, P. Bahl, R. Caceres and N. Davies, ”The Case for VM- Based Cloudlets in Mobile Computing,” in IEEE Pervasive Computing, vol. 8, no. 4, pp. 14-23, Oct.-Dec. 2009, doi: 10.1109/MPRV.2009.82.
6. L. Lamport. 2019. The part-time parliament. Concurrency: the Works of Leslie Lamport. Association for Computing Machinery, New York, NY,

USA, 277–317. https://doi-org.uc.idm.oclc.org/10.1145/3335772.3335939

1. L. R. Ford Jr and D. R. Fulkerson. 2015. ”Flows in networks”. Princeton university press.
2. X. Ma, S.Zhang, W. Li, P. Zhang, C. Lin and X. Shen, “Cost-eﬀective workload scheduling in cloud assisted mobile edge computing,” Proc. 2017 IEEE/ACM International Symposium on Quality of Service (IWQoS), 2017, pp. 1-10.
3. J. Zhang, W. Xia, Z. Cheng, Q. Zou, B. Huang, F. Shen, F. Yan, and L. Shen, “An evolutionary game for joint wireless and cloud resource allocation

in mobile edge computing,” Proc. 2017 9th International Conference on Wireless Communications and Signal Processing (WSCP), 2017, pp. 1–6.

1. J. Zhang, W. Xia, F. Yan and L. Shen, ”Joint Computation Oﬄoading and Resource Allocation Optimization in Heterogeneous Networks With Mo- bile Edge Computing,” in IEEE Access, vol. 6, pp. 19324-19337, 2018, doi: 10.1109/ACCESS.2018.2819690.
2. M. Aazam and E. -N. Huh, ”Fog Computing: The Cloud-IoTIoE Middle- ware Paradigm,” in IEEE Potentials, vol. 35, no. 3, pp. 40-44, May-June 2016, doi: 10.1109/MPOT.2015.2456213.
3. J. Zhao, Y. Chen and Y. Gong, ”Study of Connectivity Probability of Vehicle-to-Vehicle and Vehicle-to-Infrastructure Communication Systems,” 2016 IEEE 83rd Vehicular Technology Conference (VTC Spring), 2016, pp. 1-4, doi: 10.1109/VTCSpring.2016.7504493.
4. Y. Zhang, R. Yu, M. Nekovee, Y. Liu, S. Xie and S. Gjessing, ”Cogni- tive machine-to-machine communications: visions and potentials for the smart grid,” in IEEE Network, vol. 26, no. 3, pp. 6-13, May-June 2012, doi: 10.1109/MNET.2012.6201210.
5. R. Nishtala et al. ”Scaling Memcache at Facebook,” Proceedings of the 10th USENIX conference on Networked Systems Design and Implementation, Apr. 2013 pp. 385–398.
6. E. Cuervo, A. Balasubramanian, D. K. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, MAUI: Making Smartphones Last Longer with Code Oﬄoad, ACM MobiSys (2010).
7. K. Kumar, and Y. H. Lu, Cloud computing for mobile users: can oﬄoading computation save energy?, Computer, vol. 43, no. 4, pp. 51-56 (2010).
8. Y. Xiao and M. Krunz, “QoE and power eﬃciency tradeoﬀ for fog com- puting networks with fog node cooperation,” in Proceedings of IEEE IN- FOCOM, 2017.
9. Y. Wang, X. Tao, X. Zhang, P. Zhang, and Y. T. Hou, “Cooperative task oﬄoading in three-tier mobile computing networks: An ADMM framework,” IEEE Transactions on Vehicular Technology, vol. 68, no. 3, pp. 2763–2776, 2019.
10. L. Liu, Z. Chang, and X. Guo, “Socially-aware dynamic computation of- ﬂoading scheme for fog computing system with energy harvesting devices,” IEEE Internet of Things Journal, vol. 5, no. 3, pp. 1869–1879, 2018.
11. S. Misra and N. Saha, “Detour: Dynamic task oﬄoading in software deﬁned fog for IoT applications,” IEEE Journal on Selected Areas in Communica- tions, vol. 37, no. 5, pp. 1159–1166, 2019.
12. L. Lei, H. Xu, X. Xiong, K. Zheng, and W. Xiang, “Joint computation oﬄoading and multi-user scheduling using approximate dynamic program- ming in NB-IoT edge computing system,” IEEE Internet of Things Journal, vol. 6, no. 3, pp. 5345–5362, 2019.
13. S. Mu, Z. Zhong, D. Zhao, and M. Ni, “Joint job partitioning and collabo- rative computation oﬄoading for Internet of Things,” IEEE Internet Things J., vol. 6, no. 1, pp. 1046–1059, 2018.
14. S. Bi, J. Ying, and Zhang, “Computation Rate Maximization for Wireless Powered Mobile-Edge Computing with Binary Computation Oﬄoading,” IEEE Trans. Wireless Commun., vol. 17, no. 6, pp. 4177–4190, 2018.
15. Y. Jong, “An eﬃcient global optimization algorithm for nonlinear sum-of- ratios problem,” Optimization Online, 2012.
16. P. Richtrik and M. Tak, “Iteration complexity of randomized block- coordinate descent methods for minimizing a composite function,” Math. Program., vol. 144, no. 1-2, pp. 1–38, 2014.
17. M. Patel, B. Naughton, C. Chan, N. Sprecher, S. Abeta, A. Neal et al., “Mobile-Edge Computing Introductory Technical White Paper,” White Pa- per, Mobile-Edge Computing (MEC) Industry Initiative, 2014.
18. T. Soyata, R. Muraleedharan, C. Funai, M. Kwon, and W. Heinzelman, “Cloud-vision: Real-time face recognition using a mobilecloudlet-cloud ac- celeration architecture,” in IEEE Symposium on Computers and Commu- nications (ISCC). IEEE, 2012, pp. 000 059–000 066.
19. J. Cohen, “Embedded speech recognition applications in mobile phones: Status, trends, and challenges,” in IEEE International Conference on Acous- tics, Speech and Signal Processing. IEEE, 2008, pp. 5352–5355.
20. E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chan- dra, and P. Bahl, “MAUI: making smartphones last longer with code of- ﬂoad,” in the 8th international conference on Mobile systems, applications, and services. ACM, 2010, pp. 49–62.
21. “Cisco Global Cloud Index: Forecast and Methodology, 2014–2019,” White Paper, Cisco, 2013.
22. B. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, “Clonecloud: elastic execution between mobile device and cloud,” in the sixth conference on Computer systems. ACM, 2011, pp. 301–314.
23. Y. Wen, W. Zhang, and H. Luo, “Energy-optimal mobile application ex- ecution: Taming resource-poor mobile devices with cloud clones,” in IEEE INFOCOM. IEEE, 2012, pp. 2716–2720.
24. M. V. Barbera, S. Kosta, A. Mei, and J. Stefa, “To oﬄoad or not to oﬄoad? the bandwidth and energy costs of mobile cloud computing,” in IEEE INFOCOM, vol. 2013, 2013.
25. A. Ahmed and E. Ahmed, “A Survey on Mobile Edge Computing,” in Proc. of International Conference on Intelligent Systems and Control (ISCO), 2016.
26. Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, “Mobile Edge Computing - A Key Technology towards 5G,” White Paper, ETSI, 2015.
27. E. Li, Z. Zhou, and X. Chen, “Edge intelligence: On-demand deep learning model co-inference with device-edge synergy,” in Proceedings of the 2018 Workshop on Mobile Edge Communications, MECOMM@SIGCOMM 2018, Budapest, Hungary, August 20, 2018, 2018, pp. 31–36.
28. D. Xenakis, N. Passas, L. Merakos, and C. Verikoukis, “Mobility Manage- ment for Femtocells in LTE-Advanced: Key Aspects and Survey of Handover

Decision Algorithms,” IEEE Communications Surveys & Tutorials, vol. 16, no. 1, pp. 64–91, 2014.

1. X. Chen, L. Pu, L. Gao, W. Wu, and D. Wu, “Exploiting Massive D2D Collaboration for Energy-eﬃcient Mobile Edge Computing,” IEEE Wireless Communications, vol. 24, no. 4, pp. 64–71, 2017.
2. X. Chen, W. Li, S. Lu, Z. Zhou, and X. Fu, “Eﬃcient resource allocation for on-demand mobile-edge cloud computing,” IEEE Transactions on Vehicular Technology, pp. 1–1, 2018.
3. C. Shen, C. Tekin, and M. van der Schaar, “A Non-Stochastic Learning Approach to Energy Eﬃcient Mobility Management,” IEEE Journal on Se- lected Areas in Communications, vol. 34, no. 12, pp. 3854–3868, 2016.
4. J. W. Jiang, T. Lan, S. Ha, M. Chen, and M. Chiang, “Joint vm placement and routing for data center traﬃc engineering.” in INFOCOM, vol. 12, 2012, pp. 2876–2880.
5. F. Xu, F. Liu, H. Jin, and A. V. Vasilakos, “Managing Performance Over- head of Virtual Machines in Cloud Computing: A Survey, State of the Art, and Future Directions,” Proceedings of the IEEE, vol. 102, no. 1, pp. 11–31, 2014.
6. A. Fischer, J. F. Botero, and M. T. Beck, “Virtual Network Embedding: A Survey,” IEEE Communications Surveys & Tutorials, vol. 15, no. 4, pp. 1888–1906, 2013.
7. A. Nadembega, A. S. Haﬁd, and R. Brisebois, “Mobility prediction model- based service migration procedure for follow me cloud to support qos and

qoe,” in Communications (ICC), 2016 IEEE International Conference on. IEEE, 2016, pp. 1–6.

1. S. Wang, R. Urgaonkar, T. He, K. Chan, M. Zafer, and K. K. Leung, “Dy- namic Service Placement for Mobile Micro-Clouds with Predicted Future Costs,” IEEE Transactions on Parallel and Distributed Systems, vol. 28, no. 4, pp. 1002–1016, 2017.
2. A. Aissioui, A. Ksentini, A. Gueroui, and T. Taleb, “On enabling 5g au- tomotive systems using follow me edge-cloud concept,” IEEE Transactions on Vehicular Technology, pp. 1–1, 2018.
3. A. Rudenko, P. Reiher, G. J. Popek, and G. H. Kuenning, “Saving portable computer battery power through remote process execution,” ACM SIGMO- BILE Mobile Computing and Communications Review, vol. 2, no. 1, pp. 19–26, 1998.
4. G. Huertacanepa and D. Lee, “An adaptable application oﬄoading scheme based on application behavior,” in 22nd International Conference on Ad- vanced Information Networking and Applications Workshops, 2008.
5. C. Xian, Y. Lu, and Z. Li, “Adaptive computation oﬄoading for energy conservation on battery-powered systems,” in International Conference on Parallel and Distributed Systems, vol. 2. IEEE, 2007, pp. 1–8.
6. M. R. Rahimi, N. Venkatasubramanian, S. Mehrotra, and A. V. Vasilakos, “Mapcloud: mobile applications on an elastic and scalable 2-tier cloud archi- tecture,” in the 2012 IEEE/ACM Fifth International Conference on Utility and Cloud Computing, 2012.
7. D. Huang, P. Wang, and D. Niyato, “A dynamic oﬄoading algorithm for mobile computing,” IEEE Transactions on Wireless Communications, vol. 11, no. 6, pp. 1991–1995, 2012.
8. R. Wolski, S. Gurun, C. Krintz, and D. Nurmi, “Using bandwidth data to make computation oﬄoading decisions,” in IEEE International Symposium on Parallel and Distributed Processing. IEEE, 2008, pp. 1–8.
9. L. Yang, J. Cao, Y. Yuan, T. Li, A. Han, and A. Chan, “A framework for partitioning and execution of data stream applications in mobile cloud computing,” ACM SIGMETRICS Performance Evaluation Review, vol. 40, no. 4, pp. 23–32, 2013.
10. M. R. Rahimi, N. Venkatasubramanian, and A. V. Vasilakos, “Music: Mobility-aware optimal service allocation in mobile cloud computing,” in IEEE International Conference on Cloud Computing (Cloud), July 2013.
11. S. Barbarossa, S. Sardellitti, and P. Di Lorenzo, “Joint allocation of com- putation and communication resources in multiuser mobile cloud comput- ing,” in IEEE 14th Workshop on Signal Processing Advances in Wireless Communications (SPAWC). IEEE, 2013, pp. 26–30.
12. R. Urgaonkar, S. Wang, T. He, M. Zafer, K. Chan, and K. K. Leung, “Dynamic Service Migration and Workload Scheduling in Edge-Clouds,” Performance Evaluation, vol. 91, pp. 205–228, 2015.
13. P. Mach and Z. Becvar, “Mobile Edge Computing: A Survey on Archi- tecture and Computation Oﬄoading,” IEEE Communications Surveys & Tutorials, vol. 19, no. 3, pp. 1628–1656, 2017.
14. P. Singhal and A. Yadav, “Congestion detection in wireless sensor net- work using neural network,” in International Conference for Convergence for Technology-2014, April 2014, pp. 1–4.
15. University of Southern California Information Sciences In- stitute. The Network Simulator - ns-2. [Online] Available: https://sourceforge.net/projects/nsnam/ﬁles
16. T. Issariyakul and E. Hossain, Introduction to Network Simulator NS2, 1st ed. Springer Publishing Company, Incorporated, 2010.
17. J. B. Madalgi and S. A. Kumar, “Congestion detection in wireless sensor networks using mlp and classiﬁcation by regression,” in 2017 3rd Interna- tional Conference on Applied and Theoretical Computing and Communica- tion Technology (iCATccT), Dec 2017, pp. 226–231.
18. J. R. Quinlan, “Learning with continuous classes,” in Proceedings of the 5th Australian Joint Conference on Artiﬁcial Intelligence (AI ’92), pp. 343–348, World Scientiﬁc, Singapore, 1992.
19. J. B. Madalgi and S. A. Kumar, “Development of wireless sensor network congestion detection classiﬁer using support vector machine,” in 2018 3rd International Conference on Computational Systems and Information Tech- nology for Sustainable Solutions (CSITSS), Dec 2018, pp. 187–192.
20. M. Devare and A. Kumar, “Clustering and classiﬁcation (time series analy- sis) based congestion control algorithm: Data mining approach,” in IJCSNS International Journal of Computer Science and Network Security, vol. 7, no. 9, September 2007, pp. 241–246.
21. N. Zhang, F. Jaafar, and Y. Malik, “Low-rate dos attack detection using psd based entropy and machine learning,” in 2019 6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/ 2019 5th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom), June 2019, pp. 59–62.
22. J. D. Case, M. Fedor, M. L. Schoﬀstall, and J. Davin, “Simple network management protocol (snmp),” RFC Editor, United States, RFC 1157, 05 1990. [Online]. Available: h[ttps://www.rfc-editor.org/rfc/rfc1157.txt](http://www.rfc-editor.org/rfc/rfc1157.txt)
23. R. Sommer and A. Feldmann, “Netﬂow: Information loss or win?” in Pro- ceedings of the 2Nd ACM SIGCOMM Workshop on Internet Measurment, ser. IMW ’02. New York, NY, USA: ACM, 2002, pp. 173–174. [Online].

Available: <http://doi.acm.org/10.1145/637201.637226>

1. P. Phaal, S. Panchen, and N. McKee, “Inmon corporation’s sﬂow: A method for monitoring traﬃc in switched and routed networks,” RFC Editor, United States, RFC 3176, 09 2001. [Online]. Available: h[ttps://www.rfc-editor.org/rfc/rfc3176.txt](http://www.rfc-editor.org/rfc/rfc3176.txt)
2. C. So-In, “A survey of network traﬃc monitoring and analysis tools,” p. 24, 01 2006. [Online]. Available: h[ttps://www.cse.wustl.edu/](http://www.cse.wustl.edu/)∼jain/cse567-

06/ftp/net traﬃc monitors3.pdf

1. C. Schmoll and S. Zander, “Netmate-user and developer manual,” 01 2004. [Online]. Available: https://github.com/DanielArndt/ netmate-ﬂowcalc
2. “dstat(1) - Linux man page.” [Online]. Available: https://linux.die.net/man/1/dstat
3. Y. Bengio and Y. Grandvalet, “No unbiased estimator of the vari- ance of k-fold cross-validation,” J. Mach. Learn. Res., vol. 5, pp. 1089–1105, Dec. 2004. [Online]. Available: [http://dl.acm.org/citation.](http://dl.acm.org/citation) cfm?id=1005332.1044695
4. S. Hochreiter and J. Schmidhuber, “Long short-term memory,” Neural Comput., vol. 9, no. 8, pp. 1735–1780, Nov. 1997. [Online]. Available: <http://dx.doi.org/10.1162/neco.1997.9.8.1735>
5. G. S. Paschos, G. Iosiﬁdis, M. Tao, D. Towsley, and G. Caire, “The role of caching in future communication systems and networks,” IEEE Journal on Selected Areas in Communications, vol. 36, no. 6, pp. 1111–1125, 2018.
6. Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, “A survey on mobile edge computing: The communication perspective,” IEEE Commu- nications Surveys & Tutorials, vol. 19, no. 4, pp. 2322–2358, 2017.
7. A. Ferdowsi, U. Challita and W. Saad, ”Deep Learning for Reliable Mobile Edge Analytics in Intelligent Transportation Systems: An Overview,” in IEEE Vehicular Technology Magazine, vol. 14, no. 1, pp. 62-70, March 2019.
8. C. Fricker, P. Robert, and J. Roberts, “A versatile and accurate approx- imation for lru cache performance,” in 2012 24th International Teletraﬃc Congress (ITC 24). IEEE, 2012, pp. 1–8.
9. P. R. Jelenkovi´c and A. Radovanovi´c, “Optimizing lru caching for variable document sizes,” Combinatorics, Probability and Computing, vol. 13, no. 4-5, pp. 627–643, 2004.
10. G. Neglia, D. Carra, and P. Michiardi, “Cache policies for linear utility maximization,” IEEE/ACM Transactions on Networking, vol. 26, no. 1, pp. 302–313, 2018.
11. E. Bastug, M. Bennis, E. Zeydan, M. A. Kader, I. A. Karatepe, A. S. Er, and M. Debbah, “Big data meets telcos: A proactive caching perspective,” Journal of Communications and Networks, vol. 17, no. 6, pp. 549–557, Dec. 2015.
12. T. X. Tran, A. Hajisami, P. Pandey, and D. Pompili, “Collaborative mobile edge computing in 5G networks: New paradigms, scenarios, and challenges,” IEEE Communications Magazine, vol. 55, no. 4, pp. 54–61, 14 Apr. 2017.
13. E. Zeydan, E. Bastug, M. Bennis, M. A. Kader, I. A. Karatepe, A. S. Er, and M. Debbah, “Big data caching for networking: Moving from cloud to edge,” IEEE Communications Magazine, vol. 54, no. 9, pp. 36–42, 16 Sep. 2016.
14. W. Fan, Y. Liu, B. Tang, F.Wu, and H. Zhang, “Terminalbooster: Collab- orative computation oﬄoading and data caching via smart base stations,” IEEE Wireless Communications Letters, vol. 5, no. 6, pp. 612–615, 02 Sep. 2016.
15. G. Lee, W. Saad, and M. Bennis, “An Online Optimization Framework for Distributed Fog Network Formation With Minimal Latency,” IEEE Trans- actions on Wireless Communications, vol. 18, no. 4, pp. 2244-2258, 05 Mar. 2019
16. T. X. Tran, P. Pandey, A. Hajisami, and D. Pompili, “Collaborative multi- bitrate video caching and processing in mobile-edge computing networks,” in Proceedings of 13th IEEE Annual Conference on Wireless On-demand Network Systems and Services (WONS), 21-24 Feb. 2017 (Jackson, WY, USA), pp. 165–172.
17. W. Jiang, G. Feng, and S. Qin, “Optimal cooperative content caching and delivery policy for heterogeneous cellular networks,” IEEE Transactions on Mobile Computing, vol. 16, no. 5, pp. 1382–1393, 03 Aug. 2016.
18. H. Hsu and K.-C. Chen, “A resource allocation perspective on caching to achieve low latency,” IEEE Communications Letters, vol. 20, no. 1, pp. 145–148, 09 Nov. 2015.
19. Z. Tan, X. Li, F. R. Yu, L. Chen, H. Ji, and V. C. Leung, “Joint access selection and resource allocation in cache-enabled HCNs with D2D commu- nications,” in Proceedings of IEEE Wireless Communications and Network- ing Conference (WCNC), 19-22 Mar. 2017 (San Francisco, CA, USA), pp. 1–6.
20. Y. Zhou, F. R. Yu, J. Chen, and Y. Kuo, “Resource allocation for information-centric virtualized heterogeneous networks with in-network caching and mobile edge computing,” IEEE Transactions on Vehicular Tech- nology, vol. 66, no. 12, pp. 11 339–11 351, 09 Aug. 2017.
21. C. Wang, C. Liang, F. R. Yu, Q. Chen, and L. Tang, “Computation of- ﬂoading and resource allocation in wireless cellular networks with mobile

edge computing,” IEEE Transactions on Wireless Communications, vol. 16, pp. 4924–4938, 16 May 2017.

1. R. Huo, F. R. Yu, T. Huang, R. Xie, J. Liu, V. C. Leung, and Y. Liu, “Software deﬁned networking, caching, and computing for green wireless networks,” IEEE Communications Magazine, vol. 54, no. 11, pp. 185–193, 15 Nov. 2016.
2. J. Chakareski, “VR/AR immersive communication: Caching, edge com- puting, and transmission trade-oﬀs,” in Proceedings of the ACM Workshop on Virtual Reality and Augmented Reality Network, 21-25 Aug. 2017 (Los Angeles, CA, USA), pp. 36–41.
3. Y. Cui, W. He, C. Ni, C. Guo and Z. Liu, ”Energy-Eﬃcient Resource Allocation for Cache-Assisted Mobile Edge Computing,” 2017 IEEE 42nd Conference on Local Computer Networks (LCN), Singapore, 2017, pp. 640- 648, doi: 10.1109/LCN.2017.112.
4. B. Jiang, P. Nain, and D. Towsley, “Lru cache under stationary requests,” ACM SIGMETRICS Performance Evaluation Review, vol. 45, no. 2, pp. 24–26, 2017.
5. Center for Research and Specialized Technology in micro and nanotech- nologies, “WSNet - An event-driven accurate and realistic network simula- tor written in C/C++ for wireless networks on a large scale” November. 2021 [Online]. Available: https://github.com/CEA-Leti/wsnet.
6. D. S. Berger, R. K. Sitaraman, and M. Harchol-Balter, “Adaptsize: Or- chestrating the hot object memory cache in a content delivery network,” in

14th USENIX Symposium on Networked Systems Design and Implementa- tion (NSDI 17), 2017, pp. 483–498.

1. R. Gallager, “A minimum delay routing algorithm using distributed com- putation,” IEEE transactions on communications, vol. 25, no. 1, pp. 73–85, 1977.
2. Y. Xi and E. M. Yeh, “Node-based optimal power control, routing, and congestion control in wireless networks,” IEEE Transactions on Information Theory, vol. 54, no. 9, pp. 4081–4106, 2008. pp. 73–85, 1977.
3. H. Al-Shatri, S. Muller, and A. Klein, “Distributed algorithm for Energy eﬃcient multi-hop computation oﬄoading,” in Proc. IEEE Int. Conf. Com- mun., 2016, pp. 1–6.
4. C. F. Funai, C. Tapparello, and W. Heinzelman, “Computational oﬄoading for energy constrained devices in multi-hop cooperative networks,” IEEE Trans. Mobile Comput., vol. 19, no. 1, pp. 60–73, Jan. 2020.
5. Z. Hong, H. Huang, S. Guo, W. Chen, and Z. Zheng, “QoS-aware coop- erative computation oﬄoading for robot swarms in cloud robotics,” IEEE Trans. Veh. Technol., vol. 68, no. 4, pp. 4027–4041, Apr. 2019.
6. Z. Hong, W. Chen, H. Huang, S. Guo, and Z. Zheng, “Multi-hop co- operative computation oﬄoading for industrial IoT-edge-cloud computing environments,” IEEE Trans. Parallel Distrib. Syst., vol. 30, no. 12, pp. 2759–2774, Dec. 2019.
7. Z. Ning, P. Dong, X. Kong, and F. Xia, “A cooperative partial computation oﬄoading scheme for mobile edge computing enabled Internet of Things,” IEEE Internet Things J., vol. 6, no. 3, pp. 4804–814, Jun. 2019.
8. Y. Wang, M. Sheng, X. Wang, L. Wang, and J. Li, “Mobile-edge comput- ing: Partial computation oﬄoading using dynamic voltage scaling,” IEEE Trans. Commun., vol. 64, no. 10, pp. 4268–4282, Oct. 2016.
9. Y. Jin, J. Jin, A. Gluhak, K. Moessner, and M. Palaniswami, “An intelli- gent task allocation scheme for multihop wireless networks,” IEEE Trans. Parallel Distrib. Syst., vol. 23, no. 3, pp. 444–451, Mar. 2012.
10. Y. Tian and E. Ekici, “Cross-layer collaborative in-network processing in multihop wireless sensor networks,” IEEE Trans. Mobile Comput., vol. 6, no. 3, pp. 297–310, Mar. 2007.
11. L. Chen, S. Zhou, and J. Xu, “Computation peer oﬄoading for energy- constrained mobile edge computing in small-cell networks,” IEEE/ACM Trans. Netw., vol. 26, no. 4, pp. 1619–1632, Aug. 2018.
12. A. Munir, T. He, R. Raghavendra, F. Le, and A. X. Liu, “Network schedul- ing aware task placement in datacenters,” in Proc. 12th Int. Conf. Emerg. Netw. EXperiments Technol., 2016, pp. 221–235.
13. Y. Zhao, C. Tian, J. Fan, T. Guan, and C. Qiao, “RPC: Joint online reducer placement and coﬂow bandwidth scheduling for clusters,” in Proc. IEEE 26th Int. Conf. Netw. Protocols, 2018, pp. 187–197.
14. Q. Pu, G Ananthanarayanan, P. Bodik, S. Kandula, A. Akella, P. Bahl and
    1. Stoica, “Low latency geo-distributed data analytics,” ACM SIGCOMM Comput. Commun. Rev., vol. 45, no. 4, pp. 421–434, 2015.
15. L. Rupprecht, W. Culhane, and P. Pietzuch, “SquirrelJoin: Network-aware distributed join processing with lazy partitioning,” Proc. VLDB Endow- ment, vol. 10, no. 11, pp. 1250–1261, 2017.
16. K. Kamran, E. Yeh, and Q. Ma, “Deco: Joint computation, caching and forwarding in data-centric computing networks,” in Proceedings of the Twentieth ACM International Symposium on Mobile Ad Hoc Networking and Computing, 2019, pp. 111–120.
17. J. Kleinberg, “The small-world phenomenon: An algorithmic perspective,” in Proceedings of the thirty-second annual ACM symposium on Theory of computing, 2000, pp. 163–170.
18. X. He, R. Jin, and H. Dai, “Multi-hop task oﬄoading with on-the-ﬂy computation for multi-uav remote edge computing,” IEEE Transactions on Communications, 2021.
19. D. Bertsekas, E. Gafni, and R. Gallager, “Second derivative algorithms for minimum delay distributed routing in networks,” IEEE Transactions on Communications, vol. 32, no. 8, pp. 911–919, 1984.
20. B. Liu, Y. Cao, Y. Zhang, and T. Jiang, “A distributed framework for task oﬄoading in edge computing networks of arbitrary topology,” IEEE Transactions on Wireless Communications, vol. 19, no. 4, pp. 2855–2867, 2020.
21. F. Chierichetti, R. Kumar, and S. Vassilvitskii. Similarity Caching. In Pro- ceedings of the Twenty-eighth ACM SIGMOD-SIGACT-SIGART Sympo- sium on Principles of Database Systems, PODS ’09, pages 127– 136, New York, NY, USA, 2009. ACM.
22. S. Pandey, A. Broder, F. Chierichetti, V. Josifovski, R. Kumar, and S. Vas- silvitskii. Nearest-neighbor Caching for Content-match Applications. In Pro- ceedings of the 18th International Conference on World Wide Web, WWW ’09, pages 441–450, New York, NY, USA, 2009. ACM
23. F. Falchi, C. Lucchese, S. Orlando, R. Perego, and F. Rabitti. A Metric Cache for Similarity Search. In Proceedings of the 2008 ACM Workshop on Large-Scale Distributed Systems for Information Retrieval, LSDS-IR ’08, pages 43–50, New York, NY, USA, 2008. ACM.
24. Y. Dong, Z. Chen, S. Liu, P. Fan, and K. B. Letaief, “Age-upon-decisions minimizing scheduling in internet of things: To be random or to be deter- ministic?,” IEEE Internet of Things Journal, vol. 7, no. 2, pp. 1081–1097, 2020.
25. Z. Bao, Y. Dong, Z. Chen, P. Fan, and K. B. Letaief, “Age-optimal ser- vice and decision processes in internet of things,” IEEE Internet of Things Journal, vol. 8, no. 4, pp. 2826–2841, 2021.
26. L. A. Belady, “A study of Replacement Algorithms for a Virtual-Storage Computer,” IBM Systems Journal, vol. 5, no. 2, pp. 78–101, 1966.
27. G. S. Paschos, G. Iosiﬁdis, M. Tao, D. Towsley, and G. Caire, “The Role of Caching in Future Communication Systems and Networks,” IEEE J. Select. Areas Commun., vol. 36, no. 6, pp. 1111–1125, 2018.
28. C. Aggarwal, J. L. Wolf, and P. S. Yu, “Caching on the World Wide Web,” Trans. Knowledge Data Eng., vol. 11, no. 1, pp. 94–107, 1999.
29. J. Kangasharju, J. Roberts, and K. Ross, “Object Replication Strategies in Content Distribution Networks,” Computer Communications, vol. 25, no. 4, pp. 376–383, 2002.
30. K. Shanmugam, N. Golrezaei, A. G. Dimakis, A. F. Molisch, and G. Caire, “Femtocaching: Wireless Content Delivery Through Distributed Caching Helpers,” IEEE Trans. Inform. Theory, vol. 59, no. 12, pp. 8402–8413, 2013.
31. N. Golrezaei, A. F. Molisch, A. G. Dimakis, and G. Caire, “Femtocaching and Device-to-device Collaboration: A New Architecture for Wireless Video Distribution,” IEEE Commun. Mag., vol. 51, no. 4, pp. 142–149, 2013.
32. D. D. Sleator and R. E. Tarjan, “Amortized Eﬃciency of List Update and Paging Rules,” Commun. ACM, vol. 28, no. 2, pp. 202–208, 1985.
33. P. R. Jelenkovic and X. Kang, “Characterizing the Miss Sequence of the LRU Cache,” SIGMETRICS Perform. Eval. Rev., vol. 36, no. 2, pp. 119–121, 2008.
34. D. Lee, J. Choi, J.-H. Kim, S. H. Noh, S. L. Min, Y. Cho, and C. S. Kim, “On the Existence of a Spectrum of Policies That Subsumes the Least Recently Used (LRU) and Least Frequently Used (LFU) Policies,” SIG- METRICS Perform. Eval. Rev., vol. 27, no. 1, pp. 134–143, 1999.
35. S. Traverso, M. Ahmed, M. Garetto, P. Giaccone, E. Leonardi, and S. Niccolini, “Temporal Locality in Today’s Content Caching: Why It Matters and How to Model It,” SIGCOMM Comput. Commun. Rev., vol. 43, no. 5, pp. 5–12, 2013.
36. F. Olmos, B. Kauﬀmann, A. Simonian, and Y. Carlinet, “Catalog dynam- ics: Impact of content publishing and perishing on the performance of a LRU cache,” in Proc. of ITC, 2014.
37. M. Leconte, G. Paschos, L. Gkatzikis, M. Draief, S. Vassilaras, and S. Chouvardas, “Placing Dynamic Content in Caches with Small Population,” in Proc. of IEEE INFOCOM, 2016.
38. S.-E. Elayoubi and J. Roberts, “Performance and Cost Eﬀectiveness of Caching in Mobile Access Networks,” in Proc. of ICN, 2015.
39. S. O. Somuyiwa, A. Gy¨orgy, and D. Gu¨ndu¨z, “A Reinforcement-Learning Approach to Proactive Caching in Wireless Networks,” IEEE J. Select. Ar- eas Commun., vol. 36, no. 6, pp. 1331–1344, 2018.
40. A. Sadeghi, F. Sheikholeslami, and G. B. Giannakis, “Optimal and Scalable Caching for 5G Using Reinforcement Learning of SpaceTime Popularities,” IEEE J. Select. Areas Commun., vol. 12, no. 1, pp. 180–190, 2018.
41. S. Geulen, B. V¨ocking, and M. Winkler, “Regret Minimization for Online Buﬀering Problems Using the Weighted Majority Algorithm,” in Proc. of COLT, 2010.
42. G. S. Paschos, A. Destounis, L. Vigneri, and G. Iosiﬁdis, “Learning to Cache With No Regrets,” in Proc. of IEEE INFOCOM, 2019.
43. Y. Li, T. Si Salem, G. Neglia, and S. Ioannidis, “Online Caching Networks with Adversarial Guarantees,” Proc. ACM Meas. Anal. Comput. Syst., vol. 5, no. 3, 2021.
44. T. Si Salem, G. Neglia, and S. Ioannidis, “No-Regret Caching via Online Mirror Descent,” in Proc. of ICC, 2021.
45. R. Bhattacharjee, S. Banerjee, and A. Sinha, “Fundamental Limits on the Regret of Online Network-Caching,” Proc. ACM Meas. Anal. Comput. Syst., vol. 4, no. 2, 2020.
46. D. Paria and A. Sinha, “LeadCache: Regret-Optimal Caching in Net- works,” in Proc. of NeurIPS, 2021.

## Appendix A: Lyapunov optimization and drift

Considering a network operating in discrete time denoted in unit time slots *t* ∈ {0*,* 1*,* 2*, ...*}, we can describe the network as a collection of backlogged queues ***Q***(*t*) as [3–6]:

***Q***(*t*) = {*Q*1(*t*)*, Q*2(*t*)*, ..., QK*(*t*)}*,*

where, *K* ≥ 0. In each time slot *t*, some action occurs that aﬀects the arrivals and departures of the queues creating a collection of real valued attribute vectors designated as ***x***(*t*), ***y***(*t*) and ***e***(*t*):

***x***(*t*) = {*x*1(*t*)*, x*2(*t*)*, ..., xM* (*t*)}*, M* ≥ 0

***y***(*t*) = {*y*0(*t*)*, y*1(*t*)*, ..., yL*(*t*)}*, L* ≥ 0

***e***(*t*) = {*ei*(*t*)*, e*2(*t*)*, ..., eJ* (*t*)}*, J* ≥ 0

The attributes represent the rewards and penalties associated with each time slot

*t* including power expenditures, distortions and packet losses. The attributes can

174

be generalized as:

*xm*(*t*) = *x*ˆ{*α*(*t*)*, ωt*}*,* ∀ *m* ∈ {1*,* 2*, ..., M* }

*yl*(*t*) = *y*ˆ{*α*(*t*)*, ωt*}*,* ∀ *l* ∈ {0*,* 1*, ..., L*}

*ej*(*t*) = *e*ˆ{*α*(*t*)*, ωt*}*,* ∀ *j* ∈ {1*,* 2*, ..., J*}

where, *α*(*t*) is the action undertaken on time slot *t* such as packet transmissions and *ω*(*t*) is a random event occurring during timt slot *t*. The action *α*(*t*) is chosen within an abstract set *Aω*(*t*) which in turn depends on *ω*(*t*).

Considering, *x*¯*m*, *y*¯*l*, *e*¯*j* to represent the averaged time of *xm*(*t*), *yl*(*t*) and *ej*(*t*) under a speciﬁc control sequence respectively, we would attempt to solve the following:

*Minimize* : *y*¯0

*such that* : 1*. y*¯*l* ≤ 0*,* ∀ *l* ∈ {1*,* 2*, ..., L*}

2*. e*¯*j* = 0*,* ∀ *j* ∈ {1*,* 2*, ..., J*}

3*. α*(*t*) ∈ *Aω*(*t*)*,* ∀ *t*

1. *All network queues are stable*

(1)

In general, we are aiming to optimize the convex functions of time averages. We consider *f* (***x***)*, g*1(***x***)*, ..., gL*(***x***) are the convex functions from R*M* to R, and *X* to be a closed convex subset of R*M* . Now, let us consider *x*¯ = {*x*¯1*, x*¯2*, ..., x*¯*M* } to

be the vector of the time averages of *xm*(*t*) attributes under the speciﬁc control

sequence. Under these circumstances, our problem reduces to the following:

*Minimize* : *y*¯0 + *f* (***x***¯)

*such that* : 1*. y*¯*l* + *gl*(***x***¯) ≤ 0*,* ∀ *l* ∈ {1*,* 2*, ..., L*}

2*. e*¯*j* = 0*,* ∀ *j* ∈ {1*,* 2*, ..., J*}

3*.* ***x***¯ ∈ *X*

4*. α*(*t*) ∈ *Aω*(*t*)*,* ∀ *t*

1. *All network queues are stable*

(2)

A solution to this problem would choose control sequences over time in reaction to the current state of the network so that all constraints are satisﬁed while ensuring that the quantity to be minimized is as small as possible. If, the control decisions taken are ineﬃcient, it results in a larger queue backlog. A good algorithm should be able to look at this backlog queue to determine the control decision in the next time slot *t*. This adaptive mechanism decouples algorithms from the knowledge of probabilities associated with random network events *ω*(*t*).

Formally, the constraints of the problem are analyzed and virtual queues are constructed that can meet the desired constraints. Virtual queues can ensure that the required time averaged constraints are satisﬁed even if the original problem does not specify any underlying physical queues. Now, the Lyapunov function *L*(*t*) is deﬁned as the sum of squares of all backlogs in the physical and virtual queues on a particular time slot *t*. This is a scalar measure of the network congestion. If the value of *L*(*t*) is small it means that all the queues in the system are small. However, if the value is large, it means that at least one

of the associated queues in the system has a large backlog. Now, the diﬀerence in the Lyapunov function Δ(*t*) is deﬁned as:

Δ(*t*) = *L*(*t* + 1) − *L*(*t*)

In order to maintain the network stability, decisions can be taken at every time slot *t* in order to greedily minimize Δ(*t*) which ensures that the backlogs are moved to a state with lower overall congestion. This method of minimizing the value of Δ(*t*) at every time slot *t* is known as minimizing the Lyapunov drift and helps to ensure that the required constraints for the problem are satisﬁed. The objective function for the problem is mapped to a function penalty and the problem reduces to minimize the drift and penalty expression denoted as:

Δ(*t*) + *V* · *penalty*(*t*)*, V* ≥ 0

where, *V* is the control parameter. If *V* = 0, changes the problem to minimizing the drift while for *V >* 0 introduces the weighted penalty term in the decision making. This term allows for a trade oﬀ between reducing the queue backlog and minimizing the penalty.

## Appendix B: Estimating the optimal oﬄoading decision

The Lyapunov function *L*(***Q***(*t*)) is deﬁned as [27]:

*L*(***Q***(*t*)) 1 Σ Σ {*Q*(*e,β*)(*t*)}2

2

*i*

*i*∈*N β*∈*i,l,o*

(3)

+ 1 Σ {*Q*(*c,β*)(*t*)}2

Σ

*i*

2

*j*∈*M β*∈*i,l,o*

The drift-plus penalty Δ*vL*(***Q***(*t*)) is denoted as:

Δ*vL*(***Q***(*t*)) E[*L*(***Q***(*t* + 1))

— *L*(**(***Q*)(*t*))|***q***(*t*)]

+ *V* E{*P* (*t*)|***Q***(*t*)}*,*

*V* ≥ 0

(4)

178

Now, from Equation 3, there exists a constant *θ*, where *θ* ≥ 0 and:

*L*(***Q***(*t* + 1)) − *L*(***Q***(*t*)) ≤ *θ* +

Σ *Q*(*e,a*)(*t*) *Ai*(*t* + *Wi*) + *b*(*e,l*)(*t*) − *b*(*e,o*)(*t*) +

*i*∈*N i*Σ∈*N*

*i*

*Q*(*e,l*)(*t*)

*i*

*b*(*e,l*)(*t*) −

*i i*

(*e*)

*τ f* (*t*)0

*i* +

(*e*)

*L*

*i*

Σ *Q*(*e,o*)(*t*) *b*(*e,o*)(*t*) − Σ *Ri,j*(*t*) +

*i*∈*N*

*j*∈*Mi*

*i*

*i*

*i*

(5)

Σ *Q*(*c,a*)(*t*) Σ *Ri,j*(*t*) − *b*(*c,l*)(*t*) − *b*(*c,o*)(*t*) +

*j*∈*M*

*j*Σ∈*M*

*j*

*Q*(*c,l*)(*t*)

*j*

*i*∈*Nj*

*b*(*c,l*)(*t*) −

*j*

*j j*

(*c*)

*τ f* (*t*)0

*j* +

(*c*)

*L*

*j*

Σ *Q*(*c,o*)(*t*) *b*(*c,o*)(*t*) − *Dj*(*t*)

*j*∈*M*

*j*

*j*

Now, E *Ai*(*t* + *Wi*) = *λi*, and substituting Equation 5 into Equation 4 we have:

Δ*V L*(***Q***(*t*)) ≤ *θ* +

*V* E{*Ai*(*t* + *Wi*) | ***Q***(*t*)} +

Σ *Q*(*e,a*)(*t*) E,*λi* − *b*(*e,l*)(*t*) + *b*(*e,o*)(*t*) | ***Q***(*t*), +

*i*∈*N i*Σ∈*N*

*i*

*Q*(*e,l*)(*t*) E

*i*

*i*

*b*(*e,l*) *t* −

*i*

*i*

*τ*0*f* (*e*)(*t*) (*e*)

*i*

*L*

*i*

| ***Q***(*t*) +

Σ *Q*(*e,o*)(*t*) E,*b*(*e,o*)(*t*) − Σ *R*

*i*

*i*

*i,j*

Σ *Q*(*c,a*)(*t*) E, Σ *Ri,j*(*t*) − *b*(*c,l*)(*t*) + *b*(*c,o*)(*t*) | ***Q***(*t*), +

*i*∈*N*

*j*∈*Mi*

(*t*) | ***Q***(*t*), +

(6)

*j*

*j*∈*M*

Σ

*Q*(*c,l*)(*t*) E

*j*

*j*∈*M*

*i*∈*Nj*

*j*

*j j*

*τ*0*f* (*c*)

*b*(*c,l*)(*t*) −

*j*

*c j*

| ***Q***(*t*)

+

*L*

Σ *Q*(*c,o*)(*t*) E *b*(*c,o*)(*t*) − *Dj*(*t*) | ***Q***(*t*)}

*j*∈*M*

*j*

*j*

Using Equation 6 we can now compute the total transmission capacity and the power consumption from the access node *i* to the computing node *j* as:

Δ*V L*(***Q***(*t*)) ≤ *θ* +

Σ *Q*(*e,a*)(*t*)E{*Ai*(*t* + *Wi*) | ***Q***(*t*)} +

*i*

*i*∈*N*

Σ E *Q*(*e,l*)(*t*) − *Q*(*e,a*)(*t*) *b*(*e,l*)(*t*) ***Q***(*t*) +

*i*∈*N*

*i*

*i*

*i*

Σ E *Q*(*e,o*)(*t*) − *Q*(*e,a*)(*t*) *b*(*e,o*)(*t*) ***Q***(*t*) +

*i*∈*N*

Σ

*fi* (*t*) | ***Q***(*t*)

+

*i*∈*N*

*i i*

(*e*) 3

E

*V τ*0*ς*

*fi* (*t*)

*i*

*τoQ*(*e,l*)(*t*)

−

*i*

(*e*)

*i*

*L*

(*e*)

Σ E *Q*(*c,l*)(*t*) − *Q*(*c,a*)(*t*) *b*(*c,l*)(*t*) ***Q***(*t*) +

*j*

*j*

*j*

Σ E *Q*(*c,o*)(*t*) − *Q*(*c,a*)(*t*) *b*(*c,o*)(*t*) ***Q***(*t*) +

*j*∈*M*

*j*

*j*∈*M*

Σ

Σ Σ , ,

E

*V τ*0*ς*

*j*∈*M*

*j j*

3 *τoQ*(*c,l*)(*t*)

*j*

*L*

*j*

*f* (*c*)(*t*)

−

*j*

(*c*)

*j*

*f* (*c*)(*t*) | ***Q***(*t*)

+

E *V τ*0*pi,j*(*t*) − *τ*0*mi,j*(*t*) log2(1 + *li,j*(*t*)*pi,j*(*t*)) | ***Q***(*t*) −

*i*∈*N j*∈*Mi*

Σ *Q*(*c,o*)(*t*)E{*Dj*(*t*) | ***Q***(*t*)}

*j*

*j*∈*M*

(7)

where,

*mi,j*(*t*) *Q*(*e,o*)(*t*) − *Q*(*c,a*)(*t*)*B and*

*li,j*

*i j*

(*t*) *Hi,j* (*t*)

*N*0*B*

∀ *i* ∈ *N, j* ∈ *Mi*

A solution to this problem can be obtained by minimizing the upper bound

of Δ*V L*(***Q***(*t*)) for each time slot *t*. However, this is a hard problem and an approximate solution can be obtained by approximation as:

*minimize* Σ *Q*(*e,l*)(*t*) − *Q*(*e,a*)(*t*) *b*(*e,l*) +

*i*∈*N*

*b,f,p*

*i*

*i*

*i*

Σ *Q*(*e,o*)(*t*) − *Q*(*e,a*)(*t*) *b*(*e,o*) +

*i*∈*N*

*i*Σ∈*N*

*i*

*i*

*V τ*0*ς*

*i*

*f* (*e*) 3 −

*i*

*i*

*τ*0*Q*(*e,l*)(*t*)

*i*

(*e*)

*L*

*i*

*f* (*e*) +

Σ *Q*(*c,l*)(*t*) − *Q*(*c,a*)(*t*) *b*(*c,l*) +

*j*∈*M*

*j*

*j*

*j*

(8)

Σ *Q*(*c,o*)(*t*) − *Q*(*c,a*)(*t*) *b*(*c,o*) +

*j*∈*M j*Σ∈*M*

*j*

*V τ*0*ς*

(*c*)

*j*

*f*

*j j*

*τ*0*Q*(*c,l*)(*t*)

3

*j*

— (*c*)

*L*

*j*

Σ Σ

(*c*)

*j*

*f*

+

*V τ*0*pi,j* − *τ*0*mi,j*(*t*) log2(1 + *li,j*(*t*)*pi,j*)

*i*∈*N j*∈*M*

Equation 8 can be solved by reducing it into smaller independent sub problems for estimating the oﬄoading decision ***b***(*t*), the CPU frequencies ***f*** (*t*) and the transmission power ***p***(*t*) for every time slot *t*.

## Appendix C: Cubic Splines

A cubic spline *f* (*x*) interpolating in the range *x*0 *< x*1 *< ... < x*(*n*−1) is deﬁned as a function for which *f* (*xk*) = *yk* [7, 8, 171]. This is a piecewise polyno- mial function that consists of *n* − 1 cubic polynomials *fk* deﬁned in the ranges

(*xk, x*(*k*+1)). Each *fk* is attached at *xk* for *k* = 1*, ..., n* − 2, such that *y*∗

*k*

= *f* ∗ (*xk*)

and *y*∗∗ = *f* ∗∗ (*xk*) are continuous. Figure 1 shows a cubic spline passing through

*k*

seven data points.

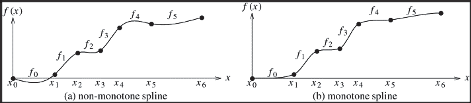


Figure 1: Non-monotonic and monotonic splines.

The *kth* polynomial curve is deﬁned over the interval (*xk, xk*+1), and can be represented by the cubic form:

*fk*(*x*) = *ak*(*x* − *xk*)3 + *bk*(*x* − *xk*)2 + *ck*(*x* − *xk*) + *dk* (9)

where,

*ak* = 1Δ*x*2 − 2Δ*yk*Δ*xk* + *y*∗ + *y*∗

*k*

*k*

*k*+1

*,* (10)

182

*bk* = 1Δ*xk*3Δ*yk*Δ*xk* − 2*y*∗ − *y*∗

*k*

*k*+1

*,* (11)

*ck* = *y*∗ *,* (12)

*k*

*dk* = *yk* (13)

In Equations 10 and 11:

Δ*xk* = *x*(*k*+1) − *xk , and*

Δ*yk* = *y*(*k*+1) − *yk , for k* = 0*, ..., n* − 2

The expressions for the cubic polynomial coeﬃcients are stated in terms of po- sition data and derivatives. The derivatives can be obtained from the position data by computing a tridiagonal system of equations that map the unknown derivatives to the known position data [1, 2].

Equation 9 can be rewritten as:

*fk*(*x*) =*H*0*x* − *xk*Δ*xkyk* +

*H*1*x* − *xk*Δ*xky*(*k*+1) +

(14)

Δ*xH*2*x* − *xk*Δ*xky*∗ +

*k*

Δ*xH*3*x* − *xk*Δ*xky*(*k*+1)∗

where, *H*0*, H*1*, H*2 *and H*3 are the cubic *Hermite* basis functions deﬁned over the interval 0 ≤ *u* ≤ 1 as [9, 10, 37]:

*H*0*u* = 2*u*3 − 3*u*2 + 1 (15)

*H*1*u* = −2*u*3 + 3*u*2 (16)

*H*2*u* = *u*3 − 2*u*2 + *u* (17)

*H*3*u* = *u*3 − *u*2 (18)

The Hermite basis functions have been derived from Equations 10 and 11 by

rearranging the terms to determine the weights associated with *yk, y*(*k*+1)*, y*∗

*k*

and

*y*(*k*+1)∗ . Therefore, the Hermite functions represent the cubic spline function as

a linear combination of position and derivative values. Note that Equation 9 represented the cubic spline curve as a linear combination of powers of x with

the position and derivative values in the coeﬃcients.

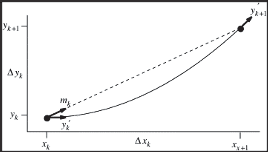


Figure 2: A single cubic polynomial segment.

Figure 2 represents a cubic polynomial segment with four constraints: the po- sition vectors (*xk, yk*) and (*x*(*k*+1)*, y*(*k*+1)) and their derivatives *yk*∗ and *y*(*k*+1)∗ .

The derivative slope *mk* = Δ*yk*

Δ*xk*

is shown with a dotted line. The derivatives are

related to the slope *mk* as:

*yk*∗ = *αkmk* (19)

*y*(*k*+1)∗ = *βkmk* (20)

for *αk* ≥ 0 and *βk* ≥ 0. The cubic curve in Figure 2 has *αk* = 0 and *βk* = 2.

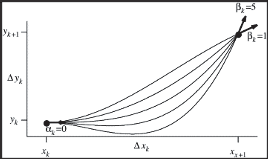


Figure 3: A set of ﬁve interpolating cubic polynomials.

In order to determine the best possible spline for a given set of data points, the derivatives can be changed to yield a large number of interpolating cubic splines. Splines related to such higher order derivatives can be used to extrapolate a given set of points better as long as the monotonicity of the spline is not changed.

Figure 3 shows a set of ﬁve cubic splines with *y*∗

*k*

= 0 and increasing values

of *y*(*k*+1)∗ . In Figure 3, *αk* = 0 and 1 ≤ *β* ≤ 5 for integer values of *βk*. The monotonicity is changed when *βk >* 3 since it creates a local minima.

## Appendix D: Suﬃciency for global optimality

When *b* = 0, the computation result size is directly proportional to the input data size:

*γt*(*gi*(*d, t*)) = *atgi*(*d, t*)

The increase in cost *D*, due to an increase in the data ﬂow *ri*(*d, t*) is due to:

* The increase in cost to forward the extra data to *i*’s neighbors *j* ∈ *o*(*j*).

This includes the increase in cost on the link (*i, j*) and the increased cost at the next hop *j*.

* The cost of assigning the extra computation that needs to be performed at

*i*

186

Therefore, the change in cost at *i* is given by:

*j*∈*o*(*i*)

*i,j*

*i,j*

*δD* = Σ

*δri*(*d, t*)

*i,j*

*φD* (*d, t*) *D*j (*F*

) + *δD*

(21)

+ *φD*(*d, t*) *δCi*(*Gi*) + *a δD*

*δrj*(*d, t*)

*i*

*i*

where,  *δD* is the increase in the cost arising from the increase in the traﬃc

*δtR*(*d,t*)

*i*0

*δgt*

*t δtR*(*d, t*)

*i*

ﬂow in *i*. The equivalent changes in cost for the result data is given by:

*δD* = Σ

*δtR*(*d, t*)

*i,j*

*i*

*φR* (*d, t*) *D*j(*F*

) + *δD*  (22)

*i*

Equations 21 and 22 can be rewritten as:

*j*∈*o*(*i*)

*i,j*

*δtR*(*d, t*)

*δD*

*i,j*

⎧⎪*tD*(*d, t*) *D*j

(*Fi,j*) + *δD , ifj* /= 0

= ⎨ *i*

*i,j*

*δrj* (*d,t*)

(23)

*δφD* (*d, t*)

*i*

*δgt*

*i*

⎪⎪⎩*tD*(*d, t*)

*δCi*(*Fi*) + *a*

*i*

*δD*

*t δtR*(*d,t*)

*, ifj* = 0

⎪

*δD D*j (*F*

*δφR* (*d, t*)

*i,j*

*i,j*

*i,j*

) + *δD*  (24)

Now, a possible solution is:

*j*

*δtR*(*d, t*)

*L*(*φ, λD, λR*) = *D*

* Σ Σ *i,*(*d,t*) Σ

*λ*

*D*

*φD* (*d, t*) − 1

(25)

*i*∈*V* (*d,t*)∈*S j*∈0∪*o*(*i*)

*i,j*

* Σ Σ *i,*(*d,t*) Σ

*λ*

*R*

*φR* (*d, t*) − 1*i*/=*d*

*i*∈*V* (*d,t*)∈*S j*∈*o*(*i*)

*i,j*

such that *φD* (*d, t*) ≥ 0 and *φR* (*d, t*) ≥ 0. Setting the derivative of *L* to 0 and

*i,j*

*i,j*

considering *b* = 0 and *φD* and *φR* as the globally optimal solutions:

⎧⎪= *min*

*δD*

*D*

⎨

*k*∈0∪*o*(*i*)

*δD , if φD* (*d, t*) *>* 0*,*

*δφD* (*d,t*)

*i,j*

*i,k*

*δφi,j* (*d, t*)

⎪

⎩ *k*

= *min*

∈0∪*o*(*i*)

*δD D*

*i,k*

⎧⎪= *min*  *δD , if φR* (*d, t*) *>* 0*,*

*δφD* (*d,t*) *, if φi,j* (*d, t*) = 0

*δD*

*R*

⎨

*k*∈*o*(*i*)

*δφR* (*d,t*)

*i,j*

*i,k*

*δφi,j* (*d, t*)

⎪

*δD R*

*k*∈*o*(*i*)

*i,k*

⎩= *min δφR* (*d,t*) *, if φi,j* (*d, t*) = 0

∀*i* and (*d, t*) and ∀*j* ∈ 0 ∪ *o*(*i*) and ∀*j* ∈ *o*(*i*) for the data and the result traﬃc ﬂows respectively.

Now, if *b* = 0 and *φD* and *φR* are feasible solutions then we have:

= *min*

*i,j*

⎨

*k*∈0∪*o*(*i*) *i,k*

*i,j*

*δD* (*d, t*)*, if φD* (*d, t*) *>* 0

*δD* (*d, t*)

≥ *min*

*i,j*

*k*∈0∪*o*(*i*) *i,j*

*δD* (*d, t*)*, if φD* (*d, t*) = 0

= *min δR* (*d, t*)*, if φR* (*d, t*) *>* 0

*i,j*

⎨

*k*∈*o*(*i*) *i,k*

*i,j*

*δR* (*d, t*)

≥ *min δR* (*d, t*)*, if φR* (*d, t*) = 0

*k*∈*o*(*i*) *i,j*

*i,j*

∀*i* and (*d, t*) and ∀*j* ∈ 0 ∪ *o*(*i*) and ∀*j* ∈ *o*(*i*) for the data and the result traﬃc

ﬂows respectively. *δD* (*d, t*) and *δR* (*d, t*) are the augmented marginals and are

*i,j i,j*

deﬁned as:

*D*

⎧⎪⎨

*Di*j*,j*

(*Fi,j*

) + *δD , if j* ∈ *o*(*i*)*,*

*δi,j* (*d, t*) =

⎩⎪

*δCi*(*Gi*) + *a δD , ifj* = 0

*δrj* (*d,t*)

*δgt*

*t δtR*(*d,t*)

*i*

*i*

*R*

*δ*

*i,j*

= *Di*j*,j*

(*Fi,j*

*δD*

) + *δtR*(*d, t*)

*j*

which makes *φD, φR* the globally optimal solution. The detailed proof is provided in Appendix E.

## Appendix E:

## Proof of global optimality

Considering Σ*j*

*R*

*φ*

*i,j*

= 1*,* ∀*i* in *non-destination* nodes:

*δD* = Σ

*δri*(*d, t*)

*φD* (*d, t*)*δD* (*d, t*)

*i,j*

*i,j*

*j*∈0∪*o*(*i*)

= Σ *φD* (*d, t*)*λD*

*j*:*φi,j>*0

*i,j*

*idt*

*D*

= *λ*

*idt*

Therefore, for the data and result traﬃc ﬂows in the network we have:

*δD* (*d, t*) ≥ *δD ,* ∀*i inV, j* ∈ 0 ∪ *o*(*i*)*,* ∀(*d, t*) ∈ *S,* (26)

*i,j*

*δri*(*d, t*)

*δR* (*d, t*) ≥ *δD ,* ∀*i inV, j* ∈ *o*(*i*)*,* ∀(*d, t*) ∈ *S* (27)

*i,j*

*i*

*δtR*(*d, t*)

Now, let *φ*j = (*φD*j*, φR*j) =/ *φ* and the corresponding forwarding and computation ﬂows *Fi*j*,j,* ∀(*i, j*) ∈ *E* and *Gi*j *,* ∀*i* ∈ *V* . Then, we have:

*min*

*f D,f R,g*

(*i*Σ*,j*)∈*E*

*Di,j*(*Fi,j*) + *Ci*(*Gi*) (28)

Σ

*i*∈*V*

190

since both *φ* and *φ*j are valid and (*Fi,j, Gi*) and *Fi*j*,j, G*j*i* are feasible. Now, in order for the equations to be valid:

*gi*(*d, t*) ≥ 0*,* ∀*i* ∈ *V, k* ≤ *T,* (*d, t*) ∈ *S*

⎪⎧ *D*

⎪⎨*f R* (*d, t*) ≥ 0*,*

*fi,j* (*d, t*) ≥ 0*,*

⎪⎪⎩

*i,j*

∀(*i, j*) ∈ *E,* (*d, t*) ∈ *S*

As the feasible set of results is convex, for any *μ* ∈ [0*,* 1], (1 − *μ*)*Fi,j* + *Fi*j*,j,* (1 −

*μ*)*Gi* + *μG*j*i* is also feasible for Equation 28. Considering *D*(*μ*) to be:

*D*(*μ*) =

Σ

(*i,j*)∈*E*

*Di,j* (1 − *μ*)*Fi,j* + *μFi*j*,j*

+ *Ci* (1 − *μ*)*Gi* + *μG*j*i*

*i*∈*V*

renders *D*(*μ*) to be convex in *μ* as *D* is convex in *Fi,j* and *Gi*. This proves the suﬃciency if *dD*(*μ*) ≥ 0 at *μ* = 0:

Σ

*μ*

*dD*(*μ*)

*dμ*

= Σ *D*j (*F*

*μ*=0

(*i,j*)∈*E*

)(*F* j − *F* )

+ Σ Σ *δCi*(*Gi*)(*gt*j − *gt*)

.

.

*i,j*

*i,j*

*i,j*

*i,j*

(29)

*δgt*

*i*

*i*

*i*∈*V t*≤*T*

*i*

Considering data traﬃc, multiplying both sides of Equation 26 by *φD*j(*d, t*) and

*i,j*

summing over *j* ∈ 0 ∪ *o*(*i*), gives us:

*δCi*(*Gi*) *φD*j(*d, t*) + Σ *D*j (*F*

*i*

)*φD*j(*d, t*) ≥ *δD*

*δgt i*0

*i,j i,j i,j*

*j*∈*o*(*i*)

*δri*(*d, t*)

(30)

* *a δD φD*j(*d, t*) − Σ

*t δtD*(*d, t*)

*i*0

*j*∈*o*(*i*)

*δrj*(*d, t*)

*i*

*δD φd*j (*d, t*)

Multiplying both sides by *tD*j(*d, t*) = Σ

*i*

*j*∈*I*(*i*)

*i,j*

*f D*j(*d, t*) + *ri*(*d, t*):

*δCi*(*Gi*) + Σ *D*j (*F*

*j,i*

*δgt*

*i,j*

*i,j*

*i*

*j*∈*o*(*i*)

)*f D*j(*d, t*) ≥ *tD*j(*d, t*) *δD*

* *a δD tR*j(*d, t*)*φD*j(*d, t*)

*i,j*

*i*

*δri*(*d, t*)

*t δtR*(*d, t*) *i i*0

*i*

* Σ *δD tD*j(*d, t*)*φD*j(*d, t*)

*j*∈*o*(*i*)

*δrj*(*d, t*) *i*

*i,j*

Summing over (*d, t*) ∈ *S* and *j* ∈ *V* :

Σ Σ *δCi*(*Gi*) *gt*j + Σ

*δgt*

*i*

(*i,j*)∈*E*

*i*∈*V t*≤*T*

*i*

*D*j (*F*

*i*∈*V* (*d,t*)∈*S*

)*FD*j

≥ Σ Σ

*tD*j(*d, t*) *δD*

* Σ Σ

*i,j*

*i,j*

*i,j*

*i*

*δri*(*d, t*)

*i*∈*V* (*d,t*)∈*S*

*a δD tD*j(*d, t*)*φD*j(*d, t*)

*i*

* Σ Σ Σ

*t δtR*j(*d, t*) *i*

*i*0

(31)

*δrj*(*d, t*) *i*

*i,j*

*δD tD*j(*d, t*)*φD*j(*d, t*)

(*d,t*)∈*S i*∈*V j*∈*o*(*i*)

where *FD*j = Σ

*i,j*

(*d,t*)∈*S*

*f D*j(*d, t*). Now we have:

Σ *tD*j(*d, t*)*φD*j(*d, t*) = *tD*j(*d, t*) − *rj*(*d, t*)

*i*∈*I*(*j*)

*i,j*

*i*

*i,j*

*j*

∀*j* ∈ *V* and (*d, t*) ∈ *S* by the law of traﬃc ﬂow conservation. Substituting this into Equation 31:

Σ Σ *δCi*(*Gi*) *gt*j + Σ

*i*

*D*j (*F*

)*FD*j ≥ Σ

*δD*

*r* (*d, t*)

Σ

*i*∈*V t*≤*T*

*δgt*

*i i,j*

(*i,j*)∈*E*

*i,j i,j i*

*i*∈*V* (*d,t*)∈*S*

*i δtR*(*d, t*)

*δri*(*d, t*)

(32)

— Σ Σ

*i*∈*V* (*d,t*)∈*S*

*t*

*a gt*j *δD*

*i*

Similarly, solving for the result traﬃc ﬂow by multiplying Equation 27 with

*φR*j(*d, t*) and summing over *j* ∈ *o*(*i*):

*i,j*

Σ *D*j (*F*

)*φR*j(*d, t*) ≥ *δD*

*i*

*j*∈*o*(*i*)

*i,j*

*i,j*

*i,j*

*δtR*(*d, t*)

(33)

— Σ *δD* + *φR*j(*d, t*)

*j*

*j*∈*o*(*i*)

*δtR*(*d, t*)

*i,j*

Multiplying by *tR*j(*d, t*) = Σ *f R*j(*d, t*)+ *atd*j(*d, t*) and summing over (*d, t*) ∈

*i j*∈*I*(*i*)

*S* and *j* ∈ *V* :

Σ *D*j (*F* )*FR*j ≥ Σ

*j,i*

Σ

*i*

*tR*j(*d, t*) + *δD*

*i*

(*i,j*)∈*E*

*i,j*

*i,j*

*i,j*

*i*

*i*∈*V* (*d,t*)*inS*

*δtR*(*d, t*)

(34)

— Σ Σ

*j*

*i*

*δtR*(*d, t*)

*i,j*

Σ *tR*j(*d, t*) *δD φR*j(*d, t*)

(*d,t*)∈*S i*∈*V j*∈*o*(*i*)

Now we have:

Σ *tR*j(*d, t*)*φR*j(*d, t*) = *tR*j(*d, t*) − *atg*j (*d, t*)

*i*

*i,j*

*j*

*j*

*i*∈*I*(*j*)

∀*j* ∈ *V* and (*d, t*) ∈ *S*. Substituting this in Equation 34, we have:

*t*

*i δtR*(*d, t*)

Σ *D*j (*F*

*i,j*

*i,j*

*i,j*

(*i,j*)∈*E*

)*FR*j ≥ Σ Σ

*a gt*j *δD*

*i*

(35)

*i*∈*V* (*d,t*)∈*S*

Summing Equations 32 and 35 we have:

Σ Σ *δCi*(*Gi*) *gt*j + Σ

*i*

*D*j (*F*

)*F* j

*i*∈*V t*≤*T*

*δgt*

*i i,j*

(*i,j*)∈*E*

*i,j i,j*

(36)

≥ Σ Σ

*δD*

*r* (*d, t*)

*δri*(*d, t*)

*i*

*i*∈*V* (*d,t*)∈*S*

Substituting *φ*j with *φ* we have a similar relation to Equation 36:

Σ Σ *δCi*(*Gi*) *gt* + Σ

*i*

*D*j (*F* )*F*

*i*∈*V t*≤*T*

*δgt*

*i i,j*

(*i,j*)∈*E*

*i,j i,j*

(37)

≥ Σ Σ

*δD*

*r* (*d, t*)

*δri*(*d, t*)

*i*

*i*∈*V* (*d,t*)∈*S*

since the equalities are valid from Equations 30 and 33. Therefore, Equation 37 can be abstracted from Equation 36 and completes the proof.

## Appendix F:

## The optimal cooperative strategy is NP-hard

Considering *N* edge servers in the cooperation block for an allocated cache of size *Ps*. The capacities for these edge servers are denoted by the set *Ca* = *Ca*1 *, Ca*2 *, ..., CaN* . The unit cache is denoted by *UPs* . Therefore, the cache *Ps* can

be split into *m* parts wherein:

*Ps*

*m* =

*UPs*

(38)

Assuming *m >> N* , for each edge server, the number of requests received for an unit of cache *nr* could be greater than 1 0 ≤ *nr* ≤ *m*. Now, the ratio of the

caches allocated to the edge server *ei* is:

*β* = *n rei .*

*i m*

This ensures that the caches are allocated in parallel to *N* edge servers which

minimizes the maximum latency incurred by these servers. Considering two 195

*Appendix F: The optimal cooperative strategy is NP-hard* 196

possible solutions for the cache allocation problem:

*nr*1 = ,*nr*1 *, nr*2 *, ..., nrN* , *and*

1

1

1

*nr*2 = ,*nr*1 *, nr*2 *, ..., nrN* ,*,*

2

2

2

the latencies for *nr*1 and *nr*2 are:

*nr*1∗ = max,*nr*1 *, nr*2 *, ..., nrN* , *and*

1

1

1

*nr*2∗ = max,*nr*1 *, nr*2 *, ..., nrN* ,

2

2

2

Now, if *nr*1∗ *< nr*2∗ , then the solution for *nr*1 is considered to be better than the solution for *nr*2 .

For the problem expressed in Equation 7.12, the total number of edge servers in the cooperation block in unknown which requires that the cache allocation strategy needs to be distributed. Hence, while the problem expressed in Equation 7.12, is more diﬃcult than the study mentioned earlier, the solution to Equation

7.12 is also a solution for the situation mentioned earlier. Therefore, an optimal solution of Equation 7.12 is also an optimal solution of the study mentioned earlier. This the problem in Equation 7.12 is NP-hard.

## Appendix G:

## The relationship between processing delay and the cooperation block

Edge servers in the cooperation block serve to extend the capacities of caches

at the edge and reduce latencies. In Figure if the values of *q*

and *qtf* + *qrf* at

*e e e*

the edge servers *e*1 and *e*2 are same and the capacity of *e*1 is less than *e*2, then

2 2 2 2

the edge server should join *e*1 in order to extend the capacity and reduce the

2

latencies of *e*1.

2

Conversely, in Figure if the capacities of *e*1 and *e*2 are *c*1 and *c*2*,* (*c*1 *< c*2)

2 2

respectively, the caches allocated are *P*1 and *P*2*,* (*P*1 *< P*2) respectively. This

can be expressed as:

*c*1 = *P*1

*c*2 *P*2

197

*Appendix G: The relationship between processing delay and the cooperation block* 198

Therefore, the capacity of *P*2 can be expressed as:

*P* = *c*1 *P*

2 *c*2 1

The cache allocation and the forwarding latency for both servers is considered to be *L*. Now, the latency in server *e*3 can be expressed as:

3

*l* = *P*3 + *L*

*c*2

Therefore, the latency in server *e*3 is based on the size of the cache allocated to

3

the server (*P*3). Now, if *e*3 is cooperating with *e*1 then:

⎪⎧ 1

3 2

*c*3 3 1

⎨*P*3 = *c*1+*c*3 *P , if e*3 ∈ *e*2*,*

1

*P* 2 = *c*3 *P*2 = *c*3 · *c*1 *P*1 *if e*3 ∈ *e*2

3

*c*2+*c*3

*c*2+*c*3

*c*2

3

2

wherein, *e*3 ∈ *e*1 denotes that *e*3 is a cooperating server of *s*1, and *e*3*ine*2 denotes

3

2

3

2

3

2

that *e*3 is a cooperating server of *e*2. Therefore, the ratio of the allocated cache

3 2

sizes can be expressed as:

*P* 1

3

2

*P*

3

= *c*1(*c*1 + *c*3) *c*2(*c*2 + *c*3)

(39)

As, *c*

*P* 1

*< c* , the ratio becomes 3

*<* 1, leading to *P* 1 *< P* 2. This means that

1 2 2 3 3

*P*

3

the processing delay of *e*3 is smaller when it is the cooperating server of *e*1 as

3 2

compared to *e*2.

2

## Appendix H:

## Tuning for optimal cooperation

The value of *α* determines the amount of cache allocated in the mobile device 0 ≤ *α* ≤ 1. From Equation 7.25:

*fm* 1 − *rf i*

1

*cm ce*1

*α* = 1

*ci* 1

*m e*1

*e*1

+ *fm* 1 − *cf ci*

Now, when *α* ≥ 0, two situations can arise as shown below.

If 1 −

*rf i*

*i* 1

1

1

*m*

*rf i*

1

*cm ce*1

≥ 0, then *ce*1 *<*

*cf* . Now, since 1 − *cm ce*1

≥ 0, the condition

*fm* 1 − *rf i* holds. Therefore, 0 ≤ *α* ≤ 1 as *ci*

*e*

*e*

*>* 0.

*cm c* 1

1

Conversely, if 1 −

*rf i*

*e*

*cm c* 1

1

*m*

*cm c* 1

1

*e*

≤ 0, then *ci*

1

1

1 *>* 0. Now, since *f*

≥

*c*

1

1

*f m*

1 −

*rf i*

*e*

≤ 0,

then *ci* + *f*

*e*

1

1

1 − *cf ci* , if *α* ≥ 0. Therefore, *ci >  fm* , wherein *f*

*m m*

*c*

*cf >* 1,

since *ci*1

*e*1

*f >* 0. Therefore, the condition 0 ≤ *α* ≤ 1 holds as

1

*f*

*e*

*e*

*m*

≥ *c*

*m*

*m* 1

1

1

1

*f*

*mcf* −1

1

*m*

.

*m*

*fm fmcf* −1

*m*

199

## Appendix I:

## Bounds on allowed cache access latencies

In order to determine the higher bound of the latency in the centralized caching mechanism we consider the following:

* The values of the cache allocation and result forwarding latencies are the

same as *Lrf*

max

*t*

+ *L*

*f*

max.

* The queueing latencies for cache allocation in all cooperation servers are

*μ*max.

* The latencies of all the servers in the centralized mechanism are similar.

Therefore, the value of *L*∗ can be computed as:

*L*∗ = *μ*

max

+ (*n* − 1) *Lrf*

*tf*

+ *L*

*,*

max

and is the higher latency bound in the centralized caching mechanism.

max

200

The lower latency bound in the centralized caching mechanism can be similarly determined:

* The values of the cache allocation and result forwarding latencies are the

same as *Lrf*

min

*t*

+ *L*

*f*

min.

* The queueing latencies for cache allocation in all cooperation servers are

*μ*min.

* The latencies of all the servers in the centralized mechanism are similar.

Therefore, the value of *L*∗ can be computed as:

*L*∗ = *μ*min + (*n* − 1) *Lrf*

min

*tf*

+ *L*

*,*

min

and is the lower latency bound in the centralized caching mechanism.

Therefore, Δ = *Lm*

∗*L*

is bounded by:

*Lm*  *Lm*

*<* Δ *<*

(40)

*μmax*

max

+ *L*

min

+ *L*

+ (*n* − 1) *Lrf*

*tf*

max

*μ*min

+ (*n* − 1) *Lrf*

*tf*

min

*Lm*

Since Δ ≤ *α*, the mechanism is

*μ*min+(*n*−1) *L*min*rf* +*Lmin*

Now, from the termination condition in 7.33:

*tf* − *approximation*.

*μ*min

+ (*n* − 1) *Lrf*

*tf*

+ *L*

min

*< Lm*

*< μ*max

+ (*n* − 1) *Lrf*

*tf*

+ *L*

max

min

Considering, the boundary conditions deﬁned above:

min

max

max

max

max

*,* (41)

+ (*n* − 1) *Lrf*

*μ* + (*n* − 1) *Lrf*

max

min min min *<* Δ *<*

+ (*n* − 1) *Lrf*

*tf*

max

*μ*

*max*

+ *Ltf*

+ *L*

*μ* + (*n* − 1) *Lrf*

*μ*

*min*

+ *Ltf*

+ *L*

*tf*

min

which can be written as:

(2*n* − 1)*L*min

(2*n* − 1)*L*max

*<* Δ *<* (2*n* − 1)*L*max *,* (42)

(2*n* − 1)*L*min

wherein, *L* = min,*Lrf*

min

min*, L*min

*tf ,μ*

, and *L*

= max,*Lrf*

*tf ,μ* ,,

which reduces Equation 42 to:

min

max

max*, L*max

max

*L*min *L*max

*L*max

*<* Δ *<*

*L*min

(43)

Now, since the approximation ratio *α* is greater than 1 and *L*max *>* 1, the value

*L*

*min*

of *α* is bounded within:

1 *α L*max *L*min

≤ ≤

(44)